0

11th National Bureau of Standards/National Computer Security Center National Computer Security Conference

PROCEEDINGS

17-20 October 1988

"COMPUTER SECURITY... *Into The Future*"

WELCOME

The National Computer Security Center and the Institute for Computer Sciences and Technology are pleased to welcome you to the Eleventh Annual National Computer Security Conference. The past ten conferences have stimulated the sharing of information and the application of this new technology. We are confident the Eleventh NCS Conference will continue this tradition.

This year's conference theme--Computer Security: Into the Future--reflects the growth of computer security awareness and a maturation of the technology. Our next major challenge is to understand how to build secure applications on trusted bases. The efforts of the National Computer Security Center,the Institute for Computer Sciences and Technology, computer users, and the computer industry have all contributed to the advances in computer security over the past few years. We are committed to a vibrant partnership between the Federal Government and private industry to further the state of the art in computer security.

Our challenge is to build upon the foundations we have established so that secure applications emerge. We must understand and record how we build on these foundations in order to secure user-based systems. To be successful, we need your help as you take back to your places of work an increased awareness of where we are, where we must go, and how to get there.

JAMES H. BURROWS
Director
Institute for Computer Sciences

PATRICK R. GALLAGHER, JR.
Director
National Computer Security Center

TABLE OF CONTENTS

Models

Modeling Integrity

Risk Management

Audit and Intrusion Detection

Applying Security Techniques

Communications Security

Verification

How Soon for Code Level Verification

Data Base Management Security

Networking

Local Area Networks

Vendor Activities

System Security Requirements

Automated Tools

Security Architectures

User Perspectives

A MULTILEVEL SECURITY MODEL FOR OBJECT-ORIENTED SYSTEMS

T.F. Keefe

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

W.T. Tsai

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

M.B. Thuraisingham

Honeywell
Corporate Systems Development Division
Golden Valley, MN 55427

Abstract - This paper describes a security model for a Multilevel Secure Object-Oriented System. The model is posed in terms of an object-oriented computation model incorporating distributed co-operating objects. The model supports a data sensitivity level classification appropriate for use in Multilevel Secure Database Management Systems (MLS/DBMS). This security model allows a subject to act with the lowest clearance level necessary to accomplish a task and thus avoid over-classification of data. The paper discusses the security properties of the model, including the safety of message passing and the existence of covert channels.

Index Terms - Multilevel Security, Multilevel Secure Database Management Systems, Security Model, Object-Oriented Systems

1. Introduction

MultiLevel Secure Database Management Systems (MLS/DBMSs) allow users with different clearance levels to share a database consisting of data having varying sensitivity levels. MLS/DBMSs achieved prominence at the Air Force Summer Study of 1982 [AIRF82] as a method of preventing DBMS security violations. During the study various designs for MLS/DBMSs were proposed. One design based on a near-term set of requirements incorporated off-the-shelf concepts in its solution and another based on a long-term set of requirements including content, context and dynamic classification and a solution to the inference and aggregation problem. The committee members defined a partial solution and outlined further research.

Recently much research is devoted to the design of Multilevel Secure Relational DBMS [DENN87b, DILL86, DWYE87]. Techniques to deal with the inference and aggregation problems are also being investigated [HINK88, MORG88, SUOZ87, THUR87, THUR88].

The relational data model is well defined and generally applicable to a wide range of data modelling problems. For some problem domains involving Multimedia DBMS and CAD/CAM, object-oriented systems present a more suitable data model and have become popular for use in these domains.

Object-oriented systems began as programming systems and are only now dealing with issues such as data models, predicate based queries [CHEN87], schema evolution, version control [BANE87], transactions and controlled sharing of data [FISH87]. Resolving these issues paves the way for more useful object-oriented DBMS and generates a need for security.

Object-oriented DBMSs unify a data model and a computational model setting them apart from relational systems. The relational algebra does not deal with the subject of updating or creating new relations even though most relational DBMS do provide this capability. The fact that the object-oriented computational model allows for creation and modification of data as well as data access forces a security model to deal with the problem of data modification.

The computational model also defines objects as isolated computational entities communicating explicitly with other objects through messages. This naturally leads to distributed security enforcement rather than the centralized enforcement possible with relational queries.

Previous work on security in object-oriented systems has been done to enforce discretionary and mandatory security policies. [ANCI83] describes a protection mechanism and defines how it may be embedded in an object-oriented concurrent programming language. The protection mechanism is based on capabilities and allows for static access control. The protection mechanism implements discretionary but does not address mandatory security.

Mandatory security is investigated in [MIZU87]. Security is enforced with a combination of compile-time and run-time checks. The security model classifies variables as having a fixed or indeterminate sensitivity level. The indeterminate levels are meant to deal with indeterminate information flows and must be checked at run-time. The security model does not support the classification of data according to its content and does not support a separate classification for aggregate data objects.

When classifying data in a database two factors are considered, the type of data that has been created and the sensitivity level of the data which is used to create it. Security constraints attempt to model the correlation between types of data and corresponding sensitivity levels. In many systems the subject's security clearance level is assumed to be the sensitivity level of data used in creating a new datum. This is based on the fact that the subject's clearance level represents the most sensitive datum the subject has access to. This leads to over-classification, since this clearance level will always dominate the actual sensitivity level of data incorporated in the result. [WOOD87] discusses the classification of information based on its composition. Data is marked with sensitivity labels which track the least upper bound of all data in the composite object. A covert channel is identified which exists when a higher clearance subject causes an object to become unreadable by a subject with a lower clearance. To avoid this channel, the labels are used in an advisory manner and not in the enforcement of mandatory security. Separate Mandatory Access Control Levels (MACLs) are attached to data objects for this purpose. This approach does not solve the over-classification problem with respect to mandatory access, since the MACLs do not represent the highest sensitivity level of data known by the process which created the object but the highest sensitivity level of data the subject is allowed to know. The model described in [WOOD87] assumes that the sensitivity level of an object is independent of other objects' values and sensitivity levels. This assumption is not consistent with requirements for security in DBMSs.

We propose a security model for a Multilevel Secure Object-Oriented System with the following advantages. It is posed in terms of an object-oriented computation model incorporating distributed co-operating objects. Each object is assumed to be a self-contained computing element whose only interaction with other objects is through sending and receiving messages. The model supports a mandatory security policy with extensions to support the data classification necessary for use in MLS/DBMS. This security model allows a subject to act with the lowest security classification level necessary to accomplish a task and thus avoids over-classification of data in the presence of updates. The model does this without introducing the covert channel as discussed in [WOOD87]. This allows data classification to follow a set of security constraints defined on the database schema and not the security clearance level of users making the updates.

The organization of this paper is as follows: Section 2 describes the essential points of MLS/DBMS. Section 3 gives an overview of object-oriented systems. Section 4 describes a multilevel security model for object-oriented systems and Section 5 discusses the security properties of the model. Finally, Section 6 concludes this paper with future considerations.

2. MLS/DBMS

A MLS/DBMS is different from a conventional DBMS in at least the following ways:

1. Every data item in the database has associated with it one of several classifications or sensitivities, that may need to change dynamically over time.

2. A user's access to data must be controlled based upon the user's authorization with respect to these data classifications.

Providing a MLS/DBMS on current computing systems presents many problems. The granularity of classification in a DBMS is generally finer than a file and may be as fine as a single data element. Another problem that is unique to databases is the necessity to classify data based on content, time, aggregation and context. DBMSs are also vulnerable to inference attacks where a user infers unauthorized information from legally obtained data.

A solution proposed to overcome some of these problems in relational database management systems is to use security constraints to associate classification levels with all data in a database [DENN87a, DWYE87]. The constraints provide the basis for a versatile and powerful classification policy because any subset of data can be specified and assigned a level.

Simple constraints provide for the classification of the entire database, as well as the classification by relation and by attribute. Constraints that classify by **content** provide the mechanism for classification by tuple and by element. **Context-based** constraints classify relationships among data. In addition, the results of applying a function to an attribute in all or a subset of tuples in a relation, such as sum, average, and count can be assigned different classification levels than the underlying data. Finally, the classification levels of the data can change dynamically based upon changes in time, content, or context.

A constraint consists of a data specification and a classification. The data specification defines any subset of the database using relational algebra and the classification defines the classification level of this subset. For example, consider a database which consists of a relation EMP(NAME, SALARY, SOC_SEC#) with SOC_SEC# as the key.[1]

The **content**-based constraint, using the notation proposed in [DWYE87], which classifies the names of all employees who earn more than 50K as Secret is expressed as:

> LEVEL(PROJECT[NAME] (SELECT[SALARY>50K] EMP)) = SECRET

and the **context**-based constraint which classifies all names and salaries taken together as Secret is expressed as:

> LEVEL(PROJECT[NAME, SALARY] EMP) = SECRET

The **simple** constraints which classifies all names and salaries taken individually as Secret is expressed as:

> LEVEL(PROJECT[NAME] EMP) = SECRET
>
> LEVEL(PROJECT[SALARY] EMP) = SECRET

3. Object-Oriented Systems

This section gives a brief background on object-oriented systems. There is a wide variation in what is meant by "object-oriented". Most of our interpretation comes from SMALLTALK-80 [GOLD83]. Variations on this object-oriented model are given in [STEF86]. The object-oriented model as defined by SMALLTALK was intended as a programming system. Our definition of an object-oriented system also stems from our desire to incorporate database considerations such as data models, predicate based queries, schema evolution, version control, transactions and controlled sharing of data. Our understanding of these issues comes from [BANE87], [FISH87] and [YOON87].

In an object-oriented system everything is represented as an object. An object is made up of private state information and a set of actions which represent the only way to access or modify this state information. The state information is represented as a set of instance variables whose values are objects each of which contains its own state information and methods. The actions defined on an object are called **methods**. A method carries out its action by sending messages. A **message** consists of a method selector, which is the name of the method to be invoked, followed by a list of objects to be used as arguments to the method. Sending a message to an object causes a method to be executed. Objects are passive entities which store information. A method is also passive and represents a function which can be performed on an object. A message combined with an object yields a method activation. Method activations are active and perform the computation in the system.

Primitive objects represent their state directly without using other objects, examples of these primitive objects are numeric values, strings and identifiers. Primitive methods represent actions carried out directly by the virtual machine without sending messages, examples are adding numeric values and reading the value of an instance variable.

Each object has a type or class it belongs to. All objects in a class are equivalent computationally. Each may have a different state but the type of computation which can be performed on an object is uniform throughout the class. The class defines what methods are available in instances of the class and what instance variables are included in the instance objects. The class of an object is also an object. A class object responds to messages to create new instance objects. A class object defines a type by specifying the types which it specializes. These types are referred to as its super-types. An object inherits methods and access to instance variables from its class object and each super-type of the class object all the way up the lattice to the root, OBJECT.

An object represents a *distributed* computation element. Methods are specified such that only data contained in the object receiving the message can be modified directly. A method activation has no knowledge about the states of other objects unless it explicitly queries them and it can not affect the state of other objects except through requests to them. Each method activation performs an independent computation except where it explicitly communicates by sending a message.

For the most part, methods are described informally in the text. When we wish to be more precise we will use notation similar to that in [GOLD83]. A method specification consists of a message pattern and a sequence of expressions separated by periods. The message pattern determines the message selector the method will be used for and assigns names to the formal parameters of the method. An example of a message pattern is shown below:

> spend: amount on: reason

The message selector for this method is 'spend:on:'. The two formal parameters in this method are 'amount' and 'reason'. The expressions which make up the body of the method consist of message expressions with an optional assignment. Message statements are described briefly below:

Unary Messages

A unary message consists of the name of the receiver object followed by the selector of the method to be executed. The statement below sends the message consisting of a selector named 'salary' and no parameters to the object 'Emp01':

> Emp01 salary

Keyword Messages

A message can be constructed from parts of the selector or keywords alternated with arguments. The following message sends the object 'HouseHoldFinances' the selector spend:on: along with objects representing the real number 30.45 and the string 'food'.

> HouseHoldFinances spend: 30.45 on: 'food'

A message expression returns an object as a result which represents the value of the expression. This object can be assigned to an instance variable. This is done by preceding the message expression with the name of the variable and the assignment symbol '←' as in the example below:

> TotalFinances ← TotalFinances + (HouseHoldFinances totalSpentFor: 'food')

Blocks

A block is similar to a function in a traditional programming language. It takes a list of arguments and produces a result. A block is similar in form to a method. It is enclosed in square brackets and begins with a list of parameters. Separated from the parameters by a '|' is a list of expressions which form the body of the block. The block shown below is a function of one argument 'ObjectToClassify' and returns a boolean result:

> [:ObjectToClassify | (ObjectToClassify salary) > 100000]

The block sends its argument the message 'salary' and to the resulting object it sends the message with selector '>' and argument 100000. A block is an object and can be used as an argument to a method.

4. Security Model

This section proposes a security model posed in terms of the object-oriented computing model. The model combines the use of security constraints for data classification with mandatory access control. Security constraints allow the

[1] The notation used in our discussion of database concepts and relational algebra is based on [ULLM82].

automatic classification of data objects by their type and by their relation to other data. Classification by security constraints conflicts with classification by information flow A newly created object has two classifications, one by an applicable security constraint and another from the current security classification of the user creating the object, (a user can only write objects with sensitivity levels dominating their current security classification level). A distinguishing aspect of this model is the emphasis placed on classification derived from security constraints.

[DENN87c] uses security constraints to classify newly entered data. Once classified, the MACLs are fixed and do not respond to changes in related data. Since the levels are fixed, after some updates to the database the levels assigned to data may not be consistent with the levels assigned by the constraints. Consider the security constraint which classifies, the names of employees as Secret when the employee's salary is greater than $100,000.00 and the name is Unclassified otherwise. When an employee's salary is increased to over $100,000.00 the sensitivity level of the name remains Unclassified by this model. In this model the sensitivity level of a datum does not depend solely on the applicable security constraints and therefore there can be more than one sensitivity level for a datum. Since the key value does not determine the sensitivity level of an entity, polyinstantiation [DENN87b] is used to disallow a low level user from overwriting higher level invisible data without opening a covert channel.

The opposite extreme is a model which insures that security constraints are always maintained. The level of each piece of data is completely determined by the applicable security constraints. This model allows only those modifications to the database which maintain the security classification determined by the security constraints and adhering to information flow restrictions. In the case of the classification discussed above, if a Secret user created an employee with a salary over $100,000.00, the data would be inserted in the database. If it were attempted by a Top Secret user it would be rejected since the data would have to be classified Top Secret to dominate the clearance level of the user which is in conflict with the level Secret assigned by the security constraint. In this model, the security level of an object is completely determined by security constraints. If the security constraints are conditioned only on the key value of an entity, the key value completely determines the sensitivity level and polyinstantiation is unnecessary.

The proposed model is somewhere between the other two. It allows a subject to act with the lowest authority possible so that data can more often be classified in accordance with security constraints. It applies the security constraints in a dynamic fashion changing the classification of a piece of data when the security constraint derived level changes. For example, if the names of employees are classified Secret when the employee's salary is greater than $100,000.00 and Unclassified otherwise, then when an employee's salary is increased to over $100,000.00 the sensitivity level of the name is also changed. This model insures that an object's assigned sensitivity level always dominates the sensitivity level determined by security constraints. This model must rely on polyinstantiation since the sensitivity level of an object is not determined solely by security constraints. Modifying the security classification level of subjects and objects dynamically can open covert storage channels and so need to be done with caution. The proposed model allows these level changes in only those cases where a covert channel can not exist.

The elements of the proposed security model are discussed in the next section. It describes the role of each object-oriented element in the security model. This is followed by a discussion of the type of security constraints included in the model and their representation. Finally, there is a description of the model restrictions.

4.1. Security Entities

This section identifies the role played by each entity in the object-oriented computation model in the security model. The portions of the object-oriented model discussed are: objects, methods, messages and method activations. The object-oriented model requires certain conceptual extensions to support mandatory security; these are discussed as well.

Objects — An object is a collection of passive data with an associated sensitivity level. The protected data is the object's instance variables and it is disclosed by reading one or more of the variables.

Methods — A method is a function defined for execution on the data of a particular object type. It is a passive entity. When a message is sent to an object a particular method is selected and executed in a method activation and this method activation is an active entity.

Messages — A message is sent on behalf of a security subject. It is sent to an object requesting execution of a selected method with the authority of the security subject which the message represents. A message is an object and therefore is protected by the security system. Messages are labelled with two security classification levels. The first is the clearance level, L_{Sclear}, of the security subject originating the message. The second level is the current security classification level, $L_{Scurrent}$, of the originating subject. These two levels act as an upper and lower bound on the classification level of the new method activation.

Method Activation — Method activations are the only active entities in the model and therefore represent security subjects. Each method executes in a separate context described by an activation. The execution is carried out by sending messages to objects. Sending messages is not a security relevant action, for two reasons. First, because the message carries with it boundaries on the authority of the method activation it creates, which are encompassed by the boundaries of the subject sending the message. Secondly, the data sent in messages is in the form of protected objects. These points will be discussed more fully in the section describing model properties. Certain primitive actions such as reading an instance variable, writing an instance variable, carrying out a conditional action or creating a new object are carried out directly by the method activation without sending any messages. These actions are security relevant since they directly access and modify information in the method activation and instance variables of the object.

4.2. Security Constraints

This section discusses the type of security constraints supported by the model. The first section explains the security constraint mechanism and how it can be used to represent **simple**, **content**-based and **context** security constraints. The next section defines a method used to enter the constraints and shows specific examples of its use to register **simple**, **content**-based and **context** security constraints.

Both sections demonstrate how the classification mechanism works through the use of examples on the database described in the next two figures. Figure 1 gives the schema of a sample database. The schema is for a database containing personnel information for a company. There are two types of complex objects in the database, Employee type objects and Department type objects. Each Employee object has a field (instance variable) for the social security number, name and salary of the employee and one which is filled by a Department type object which describes the department the employee is a part of. Each Department object has a field for the department name (Dname) and project name (Project) of the project the employees of the department are working on and a field which is filled by the Employee type object representing the manager (Mgr) of the department.

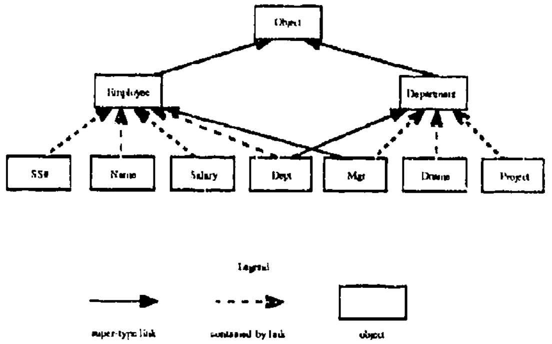


Figure 1 - Sample Schema Diagram

Figure 2 depicts objects in a database following the schema shown in Figure 1. In the figure, boxes represent instance objects, arrows point to the value of the instance variable, the class of the object is given in the upper left corner of the object and the upper right hand corner contains an identifier to reference the objects in the following discussion.

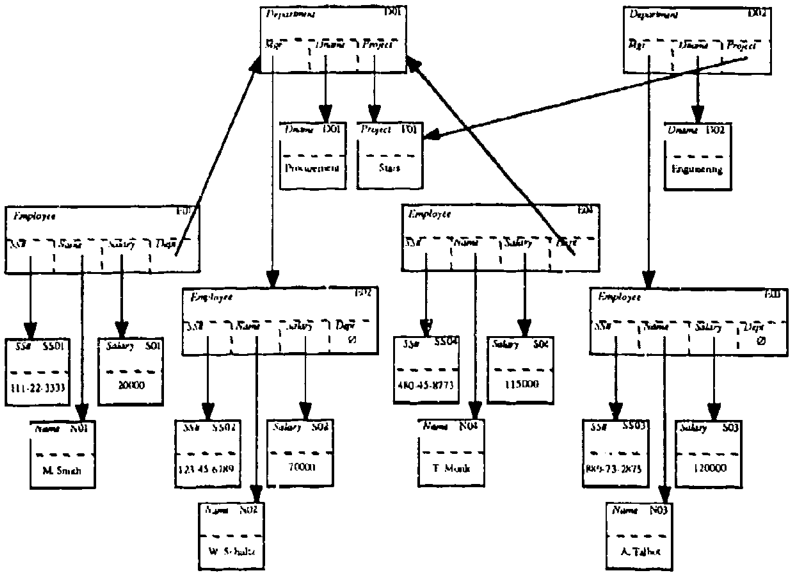


Figure 2 - Sample Database

4.2.1. Assigning Classification Levels

Every object has a sensitivity level, L_C, determined by a set of object classification functions. Each function groups objects into sets called classification sets and gives each set a sensitivity level L_C. The meaning is that the sensitivity level for disclosing *all* objects in the classification set is L_C. Each object individually can be disclosed without regard to L_C, however, the last object disclosed must be classified at a level which dominates L_C. L_C is the sensitivity level of the object determined by the security constraints in force. This is only one factor used to determine an object's sensitivity level L_O which is used by the reference monitor to determine the allowability of an access.

An object is considered disclosed to a subject S_1 if another subject S_2 which can write objects visible to S_1 has read it. In other words the object is read with respect to a subject with clearance L_{S1} if it was read by a subject with clearance L_{S2} such that $L_{S2} \leq L_{S1}$. This definition is very restrictive. It is required to protect classification sets in the case of one subject reading a member of the set and writing the information into a new object of a different type. For example, consider the context constraint which classifies names and salaries together as Secret and otherwise Unclassified. If an Unclassified user reads the name of an employee and stores the name in an object of type 'string', the context constraint will no longer relate this name to the salary of the employee. The definition of who has read the name object must include any other user who is allowed to read the special name object of type 'string'. It must include all subjects with current classifications which dominate the current classification of the subject when the object was read.

This mechanism allows the expression of **simple**, **content** and **context** security constraints as described in [DWYE87]. A **simple** constraint classifying all objects of class Project as Secret is represented by placing each member of 'project' in a separate classification set with a classification of Secret. Since each set consists of only one object, the object will immediately receive a classification level L_C of Secret. When this classification is applied to the sample database shown in Figure 2 the classification in Figure 3 results. This constraint produces only one classification set. The set contains the object 'P01'. Since it is the only object in the Secret set, its sensitivity level, L_C, immediately becomes Secret.

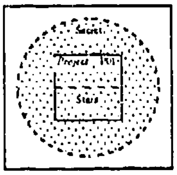


Figure 3 - Classification of Objects of Class Project as Secret for Sample DB

A **content**-based constraint specifies a set of objects by means of a predicate based on the values of some objects and classifies each with the same classification. For example, classify the names of all employees whose salary is greater than $100,000.00 as Secret. This type of constraint is represented the same as a **simple** constraint. Each 'name' which has a corresponding 'salary' greater than $100,000.00 is placed in a classification set by itself and the set is classified Secret. The two classification sets which result from applying this constraint to the database of Figure 2 is shown in Figure 4. This constraint produces two classification sets, one containing 'N04' and the other containing 'N03'. Since each object is the only object in its classification set, it takes on the sensitivity level of its classification set, Secret.

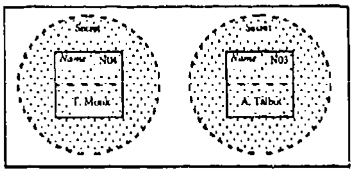


Figure 4 - Classification of the Names of all Employees Whose Salary is Greater than $100,000.00 as Secret for Sample DB

A **context** constraint matches this security constraint mechanism exactly. Related objects are grouped into classification sets and given the sensitivity level L_C meaning that the sensitivity level for disclosing *all* objects in a set is L_C. Figure 5 shows an example of a **context** constraint classification. This constraint classifies the Project and the Name of any Employee working on the Project, taken together as Secret. The constraint creates four classification sets, { 'N01', 'P01' }, { 'N02', 'P01' }, { 'N03', 'P01' } and{ 'N04', 'P01' }. Each of these sets share the object 'P01'. This means that as soon as 'P01' is read, all of the Name objects become Secret and if any Name object is read, 'P01' becomes Secret.

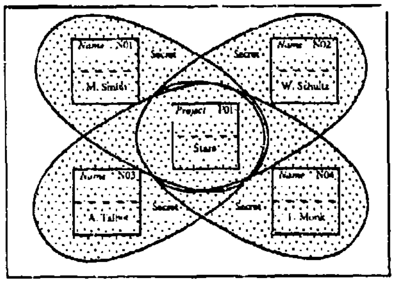


Figure 5 - Classification of Project and the Name of any Employee Working on the Project, Taken Together as Secret for Sample DB

The security model is enforced by cooperating autonomous objects. This affects the way in which security is enforced and in particular how security constraints are specified. In this *distributed* model each object is given responsibility for insuring the security of its own data. Security constraints must be reevaluated when a change is made to the database. For each security constraint, one or more objects must be chosen to be responsible for doing this reevaluation whenever it is necessary. This responsibility is split between the objects which are members of the classification set and what is called the **anchor object** for the constraint. This anchor object is not classified by the constraints but is used as a reference point for evaluating the constraint. The responsibility of the anchor object is to alert objects when they are classified by the security constraint. The anchor object in turn depends on objects which the constraint is conditioned on to alert it to changes in their values. This mechanism allows the burden of the constraint maintenance to be shared among many objects.

In a **simple** or **content**-based constraint the anchor object is chosen to be the class object which the classified objects are instances of. For example in the constraint,

Name in Employee where Salary > $100,000.00 is Secret

the anchor object is the class object Name. The set of objects to be classified is specified with respect to this anchor object. The class object Name is responsible for alerting each Name object with a Salary over $100,000.00 that it is classified by the security constraint. Whenever a new Name object is created which satisfies the predicate the anchor object must alert it to its new classification. Consider a Name object created with a Salary of $50,000.00, the constraint will not apply but the corresponding Employee and Salary object will be made responsible to report changes in their values to the anchor object, Name. Later if the Salary is updated to $110,000.00 the object will report this to the class object Name and the anchor object will alert the Name instance object of its new classification.

Simple and **content**-based constraints classify single objects not sets of objects as do **context** constraints. A **context** constraint must specify a classification set. Each object in the set allows itself to be read only when at least one other member of the set is still unread. The last unread object in the set must increase its sensitivity level to that specified in the **context** constraint before it is read. Instead of maintaining the constraint specified classification set, the set of specified objects which have not yet been read can be maintained. The classification set is then specified and maintained as an ordered sequence of objects. The anchor object is responsible for alerting each first object that it is the first object in a **context** constraint. The object is also given the specification of the rest of the ordered sequence. Each object in the sequence then acts as an anchor object for the next object in the sequence, alerting it that it is included in the constraint and passing on the specification of the rest of the ordered sequence of objects. Consider the constraint,

Name in Employee and Salary in Employee taken together are Secret.

The anchor object for this constraint is the class object Employee, (this is an arbitrary decision). 'Employee' is required to alert each object which fills the Name slot of one of its instances that it is part of a **context** constraint. The specifications for the rest of the objects are passed on with this notification. The Name object which receives this information then uses the specification of the rest of the ordered sequence to alert the prospective next objects in the sequence. In this example the Name object determines its containing Employee object and then the Salary object contained within. This Salary object is alerted that it is part of the context constraint. The Salary object is the last in the sequence and so doesn't need to alert any further objects.

Each **context** constraint creates one or more classification sets. In the example above there is one set of objects for each Employee object in the database. If this constraint is applied to the database shown in Figure 2, the resulting classification sets are shown in Figure 6. This constraint produces four classification sets, { 'N01', 'S01' }, { 'N02', 'S02' }, { 'N03', 'S03' } and { 'N04', 'S04' }. Each set has a sensitivity level of Secret. This classification has no immediate effect on the sensitivity level, L_C, of any of the objects, if however object 'S03' is read this constraint will cause L_C of 'N03' to become Secret.

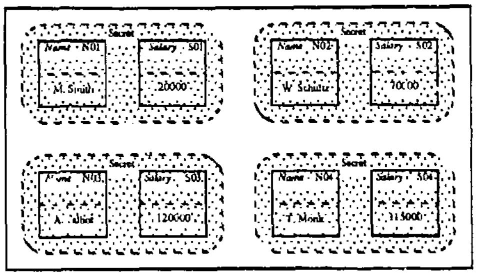


Figure 6 - Classification of Name and Salary Taken Together as Secret for Sample DB

4.2.2. Specifying Security Constraints

A set of methods are defined in all objects, securityConstraintf1:Level:, securityConstraintf1:f2:Level:, etc., which take an ordered collection of functions and an aggregate sensitivity level L_C as arguments. The functions $\{f_1 \dots f_n\}$ are defined as follows:

f_1 : anchor_object → object*

$f_{2\text{-}n}$: object → object*

Where object* represents a set of zero or more objects. The functions $f_1 \dots f_n$ are used in the following way to define a set S of classification sets.

$$S_1 = \{ x \mid x \in f_1(O_{Anchor}) \}$$
$$S = \{ \{ y_1, y_2, \dots, y_n \} \mid y_1 \in S_1 \wedge y_2 \in f_2(y_1) \wedge \dots \wedge y_n \in f_n(y_{n-1}) \}$$

Where O_{Anchor} is the anchor object, the object receiving the classification message. The first function, f_1, applied to the anchor object produces the first set S_1 of objects in the classification sets. The next object in the classification set results from applying f_2 to one of the objects in S_1. This is carried out for all n functions to create each element of S.

The following are examples of how security constraints can be represented using the above classification scheme. We are expressing the constraints in a notation similar to SMALLTALK-80 [GOLD83] as described in Section 3. First we will describe some of the methods used in the example:

Object Class	Method	Description
object	fillsSlot:In:	This is a predicate. When used as 'fillsSlot: name In: employee'. It returns **True** if the receiving object is the value of the 'name' instance variable in an 'employee' object.
object	containingObject	This method returns the object which uses the receiver as the value of one of its instance variables. For example, if the object 'S01' received the message the result would be 'E01', the employee' object which 'S01' is contained in.
class	instancesOf	Returns the objects which are instances of this class.
class	with:	This message is used as 'Set with: a'. It is sent to the class object 'Set' and creates a new set containing the object 'a'.

set	select:	This method takes one argument which is a predicate. It returns a new set which contains all of the members of the original set for which the predicate is true.
set	collect:	This method takes one argument which is a function and applies it to each member of the receiving set object. The objects returned by the function applications are collected into a new set which is the result.
employee	name	Returns the value of the 'name' instance variable.
employee	salary	Returns the value of the 'salary' instance variable.

Below are the sample security constraints:

Simple Security Constraints

Constraint: project in department is Secret

```
project  securityConstraintf1:
            [:Object | ( Object instancesOf ) select:
                [:ObjectToClassify | ObjectToClassify   fillsSlot: project
                                                        In: department]
            ]
        Level: Secret.
```

This constraint classifies all objects which fill the 'project' role in 'department' objects as Secret. The constraint is established by sending the anchor object 'project' the block shown and the sensitivity level Secret. The block f1 first computes the set of instances of its argument. Elements of this set are then selected for inclusion in the result based on the block which takes an object as an argument and returns **True** if the object fills the slot named project in a department object. This constraint would classify object 'P01' from Figure 2 as Secret.

Content-Security Constraints

Constraint: name in employee where salary > 100000 is Secret

```
name       securityConstraintf1:
                  [:Object | ( Object instancesOf ) select:
                      [:ObjectToClassify |
                          (ObjectToClassify     fillsSlot: name
                                                In: employee)
                              and:
                          (((ObjectToClassify containingObject)salary)
                                  > 100000)
                      ]
                  ]
           Level:  Secret.
```

This constraint classifies all objects which fill the 'name' role in 'employee' objects if the corresponding 'salary' is greater than 100000. The constraint is established by sending the class object 'name' f1 and the sensitivity level Secret. The anchor object is the class object name. The block f1 returns the set of instances of its argument which satisfy the following block. The block combines two predicates using the ':and' message. The first is **True** if the object fills the 'name' slot in 'employee'. The second determines the corresponding 'salary' object and tests to see if its value is greater than 100000. The effect of the above constraint would be to classify the object 'N03' as Secret.

Context-Security Constraints

Constraint: name in employee and salary in employee taken together are Secret

```
employee  securityConstraintf1:
              [:Object | (Object instancesOf) collect: [:each | each name] ]
          f2:
              [:p | Set with: ((p containingObject) salary)]
          Level. Secret.
```

This constraint groups objects into two element classification sets and assigns the set a sensitivity level of Secret. The constraint requires of a user reading more than one object in any set to have at least a Secret clearance. The class object 'employee' is the anchor and is sent f1, f2 and the sensitivity level Secret. The class object Employee is used as the argument to f_1 to compute the first elements in each classification set. The block first creates a set of all of the instances of 'employee', in this example the set {E01, E02, E03, E04}. From this set it creates a new set by applying the block [:each | each name] to each member. This block returns the 'name' of the employee object. The resulting set is {N01, N02, N03, N04}. Given an argument in {N01, N02, N03, N04}, f_2, maps it to the second element in the classification set. The function finds the containing object and then requests of it the salary object. It then creates a set from these objects. The resulting sets obtained in this way are shown in Figure 6.

These methods allow a common method for defining **simple**, **content**-based and **context** constraints. Simpler methods could be developed if each type of constraint were considered separately. For example, a **simple** constraint can be specified by supplying only the class name of the objects to be classified. A **content**-based constraint needs in addition a predicate to be evaluated by the objects to be classified.

4.3. Model Restrictions

This section describes the security model restrictions. The restrictions define a set of allowable object accesses. There are four parts to the model. The first part describes which object accesses are allowed based on the sensitivity level of the object and the current security level of the method making the request. The second part describes allowable assignments and allowable changes to object sensitivity levels. The next section describes allowable assignments and allowable changes to security classification levels for methods. The final section discusses the effect of security-inconsistent database states on mandatory security.

4.3.1. Object Access

A method activation executing with a current security classification level $L_{S_{current}}$ is allowed to:

(1.1) Read the instance variables of an object with sensitivity level L_o such that $L_o \leq L_{S_{current}}$.

(1.2) Modify the instance variables of an object with sensitivity level L_o such that $L_{S_{current}} \leq L_o \leq L_{S_{clear}}$.

In addition, pointer references are restricted as follows:

(1.3) A pointer to an unreadable object behaves exactly as a null object pointer.

Rules (1.1) and (1.2) by themselves do not insure the simple-security property or the *-property [BELL76] since the levels of objects and methods are allowed to change and these changes have not yet been defined. The maintenance of these properties can be insured only after examining the modification policy for sensitivity levels for objects and classification levels for methods. This is discussed in Section 5.

4.3.2. Object Sensitivity Levels

Security classification rules determine sensitivity levels for all objects at all times. In the interest of maintaining mandatory security some of these derived sensitivity levels can not be used. The following rules describe the way assignments are made, taking into account the sensitivity level derived from the security constraints and concerns for information flow restriction.

(2.1) Objects are assigned the lowest sensitivity level L_O at object creation time such that L_O dominates all sensitivity levels L_1, ..., L_n imposed by applicable security constraints and L_O dominates the security level, $L_{S_{current}}$, of the method activation creating the object. In other words $L_O = L_{S_{current}} \sqcap L_1 \sqcap \ldots \sqcap L_n$.[1]

(2.2) The security level of an object can only be increased. An object classified with a sensitivity level of L_O can be changed to level

[1] $\sqcap$ represents the least upper bound defined on the security classification lattice by the partial ordering $\leq$.

L_O' if and only if $L_O \leq L_O$'. Downgrading of objects must be done by trusted method activations.

(2.3) The security level L_O of an object can be affected only by method activations executing with a current classification level $L_{Scurrent}$ such that $L_{Scurrent} \leq L_O$. If this were not the case a covert channel would exist since a higher level subject could signal information to a lower level subject by increasing the sensitivity level of an object originally readable by the lower level subject, thus making it unreadable. This channel is pointed out in [WOOD87]. The model restriction allows a subject with a clearance level $L_{S2clear}$ to make modifications to the security level of an object which is visible to a subject with a clearance level $L_{S1clear}$ even when $L_{S1clear}$ is strictly dominated by $L_{S2clear}$, as long as $L_{S1current} \geq L_{S2current}$. This approach decreases the amount of over-classification and at the same time eliminates the covert channel.

4.3.3. Method Activation Security Levels

A method activation executes with a security classification level $L_{Scurrent}$ determined by two quantities. The first is the clearance level L_{Sclear} of the security subject which initiated the computation. L_{Sclear} is the security clearance level of a user and applies to all methods which are executed on the user's behalf. The second quantity which determines $L_{Scurrent}$ is the current security classification level $L_{Soriginator}$ of the method activation which started this method by sending a message. Both of these quantities are at least conceptually carried by the message. A passive method is combined with a passive message to create a method activation which executes with a security classification level determined by $L_{Soriginator}$ and L_{Sclear} as obtained from the message. Below is a set of rules determining the current security classification of a method activation.

(3.1) The login method begins execution with classification level $L_{Scurrent}$ = System Low.

(3.2) A method activation begins with a classification level $L_{Scurrent} = L_{Soriginator}$.

(3.3) If an attempt to read an object with sensitivity level L_O such that $L_O \leq L_{Sclear}$ fails, the classification level of the method will be modified to $L_{Scurrent}$' such that $L_{Scurrent}' = L_{Scurrent} \sqcap L_O$.

(3.4) A method activation object O_{ma1} is only visible to another activation object O_{ma2} and vice versa if either:
(*i*) O_{ma1} originated execution of O_{ma2}.
(*ii*) O_{ma1} originated execution of O_{ma3} and O_{ma3} is visible to O_{ma2}.

Rules (3.1) through (3.3) insure that L_{Sclear} will always dominate $L_{Scurrent}$. $L_{Scurrent}$ starts at System Low and if $L_{Scurrent} \leq L_{Sclear}$ neither (3.2) or (3.3) will make $L_{Scurrent} > L_{Sclear}$. Rule (3.4) states that method activation objects are only visible to other objects in the same calling graph.

4.3.4. Model Enforcement and Security-Consistent Database States

Security constraints are used to classify consistent entities only. At times during the creation or update of an object an entity can become inconsistent. When this happens it is not possible to immediately classify some of the objects involved. This complicates security enforcement since it becomes impossible to determine immediately if an operation can be allowed. The problem is illustrated in the following example.

This example is interested in trying to create an 'employee' object and place it in the database. The employee object is 'E03' from the sample database shown in Figure 2. Assume 'Name' objects with corresponding 'Salary' objects greater than 100K are Secret and all other objects are Unclassified. The subject's current classification level is Unclassified and its clearance level is Secret. The steps in the object creation are listed in Table 1 along with the sensitivity of the object being created or modified and $L_{Scurrent}$, the current classification level of the method activation.

Step	Action	Object Sensitivity	$L_{Scurrent}$
1.	Create employee object 'E03'	Unclassified	Unclassified
2.	Store 'E03' in department object 'D02'	Unclassified	Unclassified
3.	Create social security object 'SS03'	Unclassified	Unclassified
4.	Store 'SS03' in employee object 'E03'	Unclassified	Unclassified
5.	Create salary object 'S03'	Unclassified	Unclassified
6.	Store 'S03' in employee object 'E03'	Unclassified	Unclassified
7.	Create name object 'N03'	Unclassified	Unclassified
8.	Store 'N03' in employee object 'E03'	Secret	Secret

Table 1 - Steps in Creating an Employee Object

In steps 1 through 7, the database is not consistent. According to our assumptions, only the sensitivity levels of Name objects are affected by a relation to another object provided only in consistent objects. Once the 'Salary' object is stored the correspondence between 'N03' and 'S03' is established and 'A. Talbot' becomes Secret. At this time the subject must change $L_{Scurrent}$ to Secret. There is a time between steps 7 and 8 when 'A. Talbot' has been entered in the system but not yet classified Secret.

This problem stems from the fact that security constraints are applied after each change to an object and not when a consistent object has been created. The security constraints in the example classify names with corresponding high salaries as Secret and otherwise they are assumed to be Unclassified. In step 7 there is still no corresponding salary for the name 'N03' and so it is assumed Unclassified. In fact, the sensitivity level of 'N03' is *unknown* because no specific security constraint applies to the object when it is inconsistent.

We are still investigating this problem. Our approach is to do these modifications inside a transaction. A transaction [DATE84] groups individual operations carried out on a database to be considered as one atomic change. A transaction has two possible outcomes. It can be committed in which case the transaction completes and its affect on the database is made permanent. It can be aborted in which case the database is restored to its state previous to the beginning of the transaction. The transaction allows the individual modifications needed to get to a security-consistent state to be considered one unit of change. It is described further below:

1. If an object is modified outside of a transaction it must go immediately to a consistent state, where each object is classified by a security constraint. The modification must not cause the classification of any object to become unknown.

2. If an object is modified inside a transaction the classification of an object can go through unknown states. When a change causes an object to go from an unknown classification to a known classification, the validity of the intervening operations is checked. If security is violated the transaction is aborted and the database state is restored to its previous state.

This method is outlined in Table 2.

The table outlines the actions involved in creating an employee object. There is one extra column in this example which represents the conditions under which the action is allowed by the security model. This condition is based on the as yet unknown sensitivity levels. In the table L_{N03} represents the unknown sensitivity level of the object 'N03'. Once step 9 is complete L_{N03} is found to be equal to Secret, the condition on step 9 is not satisfied and the transaction must be aborted.

5. Model Properties

This section discusses properties of the security model. We don't attempt formal proofs of these properties but rather use informal arguments to demonstrate the properties. In the future we hope to develop a formal model and prove these properties at that time

5.1. Simple Security Property

The simple security property states that a subject with a current security classification level L_S is not allowed to read an object with a sensitivity level L_O such that $L_O > L_S$. In the notation used in this model it is, a subject with clearance level L_{Sclear} is not allowed to read an object with sensitivity level L_O if $L_O > L_{Sclear}$. This is ensured by restriction (1.1) from the previous section along with the fact that at all times $L_{Scurrent} \leq L_{Sclear}$. This follows from restrictions (3.1), (3.2) and (3.3).

Step	Action	Object Sensitivity	$L_{Scurrent}$	Allowed on Condition
1.	Start Transaction			
2.	Create employee object 'E03'	Unclassified	Unclassified	
3.	Store 'E03' in department object 'D02'	Unclassified	Unclassified	
4.	Create social security object 'SS03'	Unclassified	Unclassified	
5.	Store 'SS03' in employee object 'E03'	Unclassified	Unclassified	
6.	Create salary object 'S03'	Unclassified	Unclassified	
7.	Store 'S03' in employee object 'E03'	Unclassified	Unclassified	
8.	Create name object 'N03'	L_{N03}	L_{N03}	$L_{N03} \leq$ Secret
9.	Store 'N03' in employee object 'E03'	L_{N03}	L_{N03}	$L_{N03} \leq$ Unclassified
10.	Abort Transaction			

Table 2 - Steps in Creating an Employee Object with Deferred Classification

5.2. *-Property

The *-property states that a subject with current security classification level L_S can not write objects with sensitivity level L_O such that $L_O < L_S$. The proposed model allows a subject to write objects with sensitivity levels below L_{Sclear} as long as the subject does not have information from objects whose level strictly dominates the object written. Evidence the model enforces this is based on two facts, the first that $L_{Scurrent}$ dominates the sensitivity level of all information the method has obtained and second the method activation can not write or create objects such that $L_{Scurrent} > L_O$ (from (1.2)).

The information accessible to a method activation can come from its instance variables, information about its calling context and information available about the existence of unreadable objects. The information accessible from instance variables is covered by point (3.3), $L_{Scurrent}$ dominates the sensitivity level of all objects which are directly read by a method activation.

Information read by the calling method activation can be passed on by the mere fact that the method is executed. For example, in the computation below,

```
SecretObject ifTrue: [ UnclassifiedObject at: Answer put: True]
```

the execution of the true block is predicated on the information in SecretObject. This method activation is restricted to start execution at the classification level of its originator by restriction (3.2), Secret in this case. This ensures that $L_{Scurrent}$ dominates the level of its originating activation level and thus it dominates the sensitivity level of all information its execution could be predicated on. This also address the problem of information being transferred when the SecretObject is False, since the program can not store information when the value is True and it doesn't attempt to when the value is False, this program will not pass information about SecretObject.

Information about the existence of objects is given to a method activation when it can distinguish between null objects and objects it is not allowed to read. This transfer of information is disallowed by (1.3).

5.3. Message Safety

Sending and receiving messages can not violate mandatory security. This will be discussed in two parts. Sending a message to begin execution of a method is discussed first, followed by a discussion of the object returned on completion of the method execution.

A message is sent by an active method activation, M_1, to a passive object causing another method activation, M_2, to begin execution. M_1 is executing with a clearance level of L_{Sclear} and a current classification level of $L_{S1current}$. From restrictions (3.2) and (3.3) it can be seen that the method activation M_2 is started with the same current authority level and the same clearance level. Any information which is transferred to the method activation M_2 by beginning its execution is acceptable since both methods execute with the same current classification level.

Restriction (3.3) places the upper bound for $L_{S2current}$ to be L_{Sclear}. Thus the upper bound on any object returned to M_1 by M_2 is also L_{Sclear}, by (3.3) and (1.2). This object can always be read by M_1 because of (3.3) and the fact that the same level for L_{Sclear} applies to both method activations. Security can only be violated if M_2 can return higher level information to M_1 and M_1 does not increase its current classification level to match that of M_2. If M_1 attempts to read the object returned it will raise its classification level according to (3.3) and security will not be violated. If M_1 does not read the object it will not receive the information and security will again not be violated.

5.4. Storage Channels

This section will discuss covert storage channels. The main threat of a covert channel in this model comes from covert signalling using the sensitivity levels of objects. This problem can exist in security models which allow the sensitivity levels of objects to change. The signalling is done by allowing a high level subject to modify the sensitivity levels of objects, making them either visible or invisible to a lower level subject. We have added restrictions to the model to disallow this signalling. Method activations are objects but have different restrictions on them than normal objects. First the restrictions for normal objects will be discussed and then the special case of method activations is discussed.

To disallow signalling through the sensitivity level of normal objects, restriction (2.3) was added. This forbids a high level subject from modifying the sensitivity level of an object visible to a lower level subject indirectly. This can also take place directly if the subject tries to modify a lower level object, and is disallowed by (1.2). It takes place indirectly when a change to a higher level object causes the security constraint derived sensitivity level L_C to change. This is a natural restriction if the security level of the object is actually recorded in the object and the method activation making a change to an object supplies the authority to update all changed sensitivity levels. This ensures that a method activation M_1 can only change the visibility of an object visible to another method activation M_2 if $L_{S1clear} \leq L_{S2clear}$. This transfer of information is legitimate and does not violate security.

Method activations violate the above restriction. Restriction (3.3) allows the change of a method's security level conditioned on the existence of an object with a higher classification level. This can allow a covert storage channel if another method activation can monitor the classification level of the method activation. A method activation is an object which changes its visibility to other method activations depending on its sensitivity or classification level. Restriction (3.4) was added to eliminate this possible channel. Method activations are allowed to see other method activation objects in the same calling graph since this may be necessary in practice. This does not cause a channel since the method activation object of method activations in the same calling graph is always visible to another method activation in the same tree and will cause $L_{Scurrent}$ of an observing method to rise to the sensitivity level of the activation being observed. This is because they share the same value for L_{Sclear}, (see discussion of message safety above).

6. Conclusion

We have proposed a security model for a Multilevel Secure Object-Oriented System. The model is posed in terms of an object-oriented computation model incorporating distributed co-operating objects. Each object is assumed to be a self-contained computing element whose only interaction with other objects is through sending and receiving messages.

The model contains extensions to support the data classification necessary for use in MLS/DBMS. This security model allows a subject to act with the lowest classification level necessary to accomplish a task and thus avoid over-classification of data in the presence of updates. This allows data classification to follow a set of security constraints defined on data containers and not the security clearance level of the subject making the updates.

One distinct advantage of our approach is that the object-oriented computation model provides a uniform treatment for all objects in the system. This simplifies the statement of a security model and the subsequent design.

There are many issues which remain to be examined. Although covert storage channels in the proposed security model have been considered we have not as yet performed a formal analysis of these storage channels. The practicality of some of the methods proposed, such as the deferred enforcement for security-inconsistent database states need to be determined.

We intend to further develop this security model and analyze its security properties more formally. At that time we will consider the problem of system complexity and verification. We also intend to implement the model in an object-oriented system to investigate the feasibility of the model and performance issues related to its implementation.

7. References

[AIRF82] "Multilevel Data Management Security", Committee on Multilevel Data Management Security, Air Force Studies Board Commission on Engineering and Technical Systems, National Research Council, National Academy Press, Washington D.C., 1983.

[ANCI83] P. Ancilotti, M. Boari, and N. Lijtmaer, "Language Features for Access Control", *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 1, January 1983, pp. 16-25.

[BANE87] J. Banerjee, H.T. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou, and H.J. Kim, "Data Model Issues for Object-Oriented Applications", *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, January 1987, pp. 3-26.

[BELL76] D.E. Bell, and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretations", Technical Report MTR-2997, Mitre Corp., March 1976.

[CHEN87] Q. Chen, "An Extended Object-Oriented Database Approach", *Proceedings of COMPSAC 87 International Conference*, 1987, pp. 625-631.

[DATE84] C.J. Date, *An Introduction to Database Systems Volume II*, Addison-Wesley, Reading, MA, 1984.

[DENN87a] D.E. Denning, S.K. Akl, M. Heckman, T.F. Lunt, M. Morgenstern, P.G. Neumann, and R.R. Schell, "Views for Multilevel Database Security", IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, February 1987, pp. 129-140.

[DENN87b] D.E. Denning, T.F. Lunt, R.R. Schell, M. Heckman, and W.R. Shockley, "A Multilevel Relational Data Model", in *Proceedings of the 1987 Symposium on Security and Privacy*, April 1987, pp. 220-234.

[DENN87c] D.E. Denning, T.F. Lunt, R.R. Schell, M. Heckman, and W.R. Shockley, "Secure Distributed Data Views: The Sea View Formal Security Policy Model", A003: Interim Report, SRI International, July 20, 1987.

[DILL86] B.B. Dillaway and J.T. Haigh, "A Practical Design for Multilevel Security in Secure Database Management Systems", *Aerospace Security Conference*, December 1986.

[DWYE87] P.A. Dwyer, G.D. Jelatis, and M.B. Thuraisingham, "Multilevel Security in Database Management Systems", *Computers & Security Journal*, Vol. 6, #3, June 1987, pp. 252-260.

[FISH87] D.H. Fishman, D. Beech, H.P. Cate, E.C. Chow, T. Connors, J.W. Davis, N. Derret, C.G. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M.A. Neimat, T.A. Ryan, and M.C. Shan, "IRIS: An Object-Oriented Database Management System", *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, January 1987, pp. 48-69.

[GOLD83] A. Goldberg, and D. Robson, *SMALLTALK-80 : The Language and Its Implementation*, Addison-Wesley, Reading, MA, 1983.

[HINK88] T.H. Hinke, "Inference Aggregation Detection in Database Management Systems" in *Proceedings of the 1988 Symposium on Security and Privacy*, April 1988.

[MIZU87] M. Mizuno, and A.E. Oldehoeft, "Information Flow Control in a Distributed Object-Oriented System with Statically Bound Object Variables" in *Proceedings of 10th NBS/NCSC National Computer Security Conference*, 1987, pp. 56-67.

[MORG88] M. Morgenstern, "Controlling Logical Inference in Multilevel Database Systems" in *Proceedings of the 1988 Symposium on Security and Privacy*, April 1988, pp. 245-255.

[STEF86] M. Stefik, and D.G. Bobrow, "Object-Oriented Programming: Themes and Variations", *AI Magazine*, Vol. 6, No. 4, Winter 1986, pp. 40-62.

[SUOZ87] T. Su and G. Ozsoyoglu, "Data Dependencies and Inference Control in Multilevel Relational Database Systems" in *Proceedings of the 1987 Symposium on Security and Privacy*, April 1987, pp. 202-211.

[THUR87] M.B. Thuraisingham, "Security Checking in Relational Database Management Systems Augmented with Inference Engines", *Computers & Security Journal*, Vol. 6, #6, 1987.

[THUR88] M.B. Thuraisingham, W.T. Tsai, and T.F. Keefe, "Secure Query Processing using AI Techniques", *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, 1988.

[ULLM82] J.D.Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, Maryland, 1982.

[WOOD87] J.P.L. Woodward, "Exploiting the Dual Nature of Sensitivity Labels", in *Proceedings of the 1987 Symposium on Security and Privacy*, April 1987, pp. 23-30.

[YOON87] B.D. Yoon, F. Suzuki, H. Ishikawa, and A. Makinouchi, "Experimental Multimedia DBMS Using an Object-Oriented Approach", *Proceedings of COMPSAC 87 International Conference*, 1987, pp. 632-641.

AN INTERNET SYSTEM SECURITY POLICY AND FORMAL MODEL

J. W. Freeman, R. B. Neely
Ford Aerospace Corporation
10550 State Hwy. 83
Colorado Springs, Colorado 80921

G. W. Dinolt
Ford Aerospace Corporation
3939 Fabian Way
Palo Alto, California 94304

This research was sponsored in part by the USAF Rome Air Development Center under the Multinet Gateway Program, contract number F30602-86-C-0138.

A security policy and a formal policy model for the security properties of an internet system are presented. The model is a result of the resolution of specific system design issues, environmental attributes, security requirements and the desire to formally specify and verify the internet system design with respect to specific security constraints. Although the modeling approach is general and applicable to many systems, the actual resulting model is system-specific.

Introduction

In this paper, we document a security policy and formal policy model for an internet system. We give a rationale for the model and its development with respect to related requirements from the *DoD Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD [1]. The model provides a view of the internet system as a whole and not as a collection of components.

Several kinds of security models have been described in recent papers [2,3,4]. The specific kind of security model one would use is driven by the functionality of the target system [5]. Such systems include operating systems and their kernels, network components with specific functional requirements, networks themselves and data base systems. These are being analyzed from a *formal* modeling point of view. Adaptation of any single security model, such as the Bell-LaPadula model [6], for all targets may not be appropriate because of the variety of analyses and particular requirements of interest.

Many models [2,3,4,6] describe system security in terms of states (or state-transitions) of the system. The use of a state oriented model forces an order on the events of a system. In tne case of a system that provides a datagram service, one cannot depend on the order of the arrival and departure of datagrams at an individual component or at the system as a whole. The security properties of the components of the datagram system as well as the datagram system itself must therefore be independent of these aspects. The model we present is independent of the order of the datagrams as they pass through the system.

Background

At the fall 1987 SIGSAC conference at UCLA, J. Millen summarized reasons for modeling systems and what system models are to accomplish [7]. First, models are constructed to provide a descriptive capability that can be used to identify the important concepts. Second, models are constructed to provide a general mechanism to analyze these important concepts. Third, models are constructed to provide a mechanism for obtaining specific solutions. They are to be used to answer questions about the system.

The following applies those observations to security models. First, models are used to describe the security properties of the system. Second, they are used to provide a means to analyze these security properties. Third, they are used to provide a mechanism to answer questions about the security of the system. Security models also are used to establish the basis for the formal verification of the system security design. In this paper we present an internet formal policy model and illustrate these modeling observations. We provide a general modeling approach and offer a specific internet policy model.

The Multinet Gateway System security policy model provides a description of the security properties of a system of packet switch nodes as a whole system. The model does not deal just with a node within the system, nor just the software portion of the corresponding Trusted Computing Base[1]. This is because a user of an internet system is interested in what the *entire* system will do with his information, from visible interface to visible interface. His interest will not be satisfied merely by telling him about the properties of some piece of software embedded deeply within the internet system. The focus of the formal policy model is protection against compromise together with specific integrity constraints that support protection against compromise.

The formal policy model defines, as important from a security point of view, the notions of *information units*, their *acceptance* into the system, the associated internal processing (termed *derivation*, which includes *information unit* isolation by security label) and their *delivery* out of the system. A definition of system security is then made in these terms. The model formulation is expressed in terms of what information is allowed to flow. It is not expressed in terms of states and state-transitions (see Sections 3, 4). This formulation defines a general mechanism for the specification and analysis of the security properties of an internet system. By making specific choices within the components of the formal policy model, distinct policies can be specified and implemented. This includes a portion of a DoD policy expressed as a "dominance" [1] relation on sensitivities. The model has been used to provide a means to establish consistency among the security properties.

Multinet Gateway System and Environment Considerations

An internet system is a collection of gateways interconnected by networks that provides a datagram service to Hosts. To set the framework for the policy itself, a brief discussion is provided of the Multinet Gateway System (MGS), its environment and related security concerns. The reader is encouraged to read [8] for additional background information. The internet system, security policy and formal policy model are described and illustrated in this paper by direct usage of the MGS concepts and terminology.

The purpose of the MGS is to increase inter-operability and survivability of DoD communications networks and to provide secure communications. Increased interoperability is achieved by allowing Hosts on different networks, with different network protocols, to exchange data without resorting to exceptional procedures. Survivability is achieved by providing the capability to use public networks as transfer mechanisms to reestablish DoD internet connectivity. Secure communication is achieved by a combination of label-based access control mechanisms, information isolation and processing separation. Encryption is provided where necessary.

In Figure 1, we show a configuration of a MGS, the attached networks and their Hosts. The configuration consists of a MGS, together with Hosts and End Networks external to the MGS. Hosts are connected to the MGS via End Networks. Neither End Networks nor Hosts are under the control of the MGS. The system boundary of the MGS is identified in the figure. It is necessary to identify and describe the security characteristics expected of the MGS by the Hosts and interconnecting networks, as well as the characteristics expected of the Hosts and interconnecting networks by the MGS. Hence, these characteristics and assumptions form the basis for the MGS security policy.

The MGS consists of MG NODES and Transport Networks connecting the MG NODES. Two aspects of the figure are to be noted for modeling purposes. First, the MGS, not just a node, is to be identified by the formal model as a single entity. Second, the Transport Networks provide a private subnet that is to be viewed as inside the MGS and hence under its control. One can achieve this result by actually placing the Transport Networks inside a physical boundary completely under the control of the MGS or one can use some means, say encryption, to guarantee that MGS traffic across some resource shared with other systems (i.e., the Transport Networks) is isolated from those other systems. This is the basis for a secure channel within the MGS. A secure channel is a generalization of the "trusted path" concept as described in [1]. A secure channel is realized by specific mechanisms that allow the communication of sensitive information, both within a Multinet Gateway Node and among gateway nodes. A Host connected to a Transport Network does not have access to this secure channel.

Finally, in providing the datagram service for end-users, additional information is required to be handled by the MGS that is not end-user data. Examples include specific protocol information or control information. The system, therefore, needs to distinguish between these two types of information.

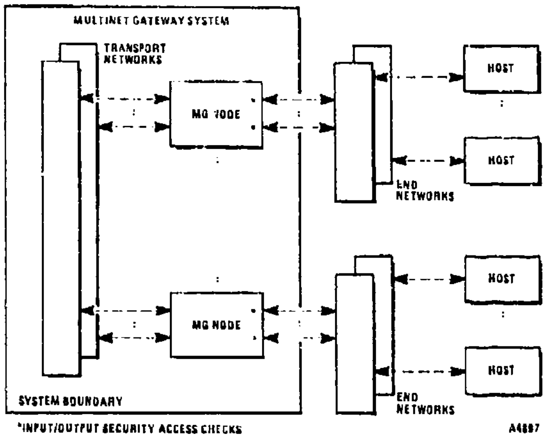


Figure 1. The Multinet Gateway System As an Internet

Security Policy

Perimeters and Policy

In the specification of the security policy, security responsibilities are allocated to the components of MGS, End Networks and Hosts. We use the notion of a perimeter enclosing various components to bound the security properties of the components. There are two perimeters of importance for the Multinet Gateway System: the Security Perimeter and the Certification Perimeter.

The Security Perimeter of the MGS extends to the Internet Protocol (IP) layer of protocol on a Host. This is because the MGS provides an IP datagram service, and because neither End Networks nor Hosts are under the control of the MGS. For example, a Host and an End Network are to provide the correct security sensitivity of each datagram sent to the MGS. The security perimeter extends, therefore, beyond the system boundary of the MGS. These associated assumptions and security characteristics are to be included in the internet security policy, and consequently, included in the formal specification.

The Certification Perimeter encompasses the internet security relevant functions. The Certification Perimeter is contained within the Security Perimeter. For the MGS Certification Effort, security assertions are made and being proven (verified) concerning security relevant functions that are within the Certification Perimeter. Security assumptions are made also about security relevant functions outside the Certification Perimeter, but within the Security Perimeter.

The MGS assumes security responsibility for Host data at a MGS Port. This Port is the interface between the End Network and the MGS. The collection of MGS Ports establishes the MGS Certification Perimeter, which is the system boundary and identified in Figure 1. The MGS Security Policy, as seen by Hosts, is defined with respect to this Certification Perimeter.

The MGS implements a security policy based on the DoD Security Policy. The enforcement of this security policy depends upon a combination of administrative procedures and technical enforcement mechanisms. Administrative procedures are necessary to determine and validate the associated security attributes of each Host and assign those security attributes to the appropriate MGS Port.

Technical enforcement mechanisms are then used to ensure that all data exchanged via the MGS is always mediated against these security attributes and that the security attributes are protected against unauthorized modification. Hosts are expected to have a wide range of security attributes Of concern here are those Host security attributes related to the exchange of data through the MGS. These Host specific security attributes must be converted into a uniform set of *Sensitivity* levels, to ensure that there is consistency in *Sensitivity* level naming conventions.

Multinet Gateway System Security Policy

The MGS Security Policy encompasses *Protection Against Compromise*, *Integrity*, *Provision of Service*, and *Accountability*. The full policy statement is given in [9]. In particular, the Protection Against Compromise Policy is one of assuring the secrecy of the information within datagrams handled by the MGS. Related to this are the integrity considerations that are in direct support of the maintenance of that secrecy. The organization of the statements emphasize the relationship between a restricted form of integrity and the overall policy for Protection Against Compromise. Since the scope of the formal model is only on the Protection Against Compromise together with specific integrity constraints, we do not document the full Accountability Policy or the Provision of Service Policy in this paper.

Protection Against Compromise: The intent of the MGS Security Policy for Protection Against Compromise is that information flowing through the MGS will not be sent to Hosts and End Networks that are not allowed to see that information. The exchange or transport of information between Hosts and the MGS shall be either end-user information or non end-user information. The policy of Protection Against Compromise consists of the following rules:

DATA SECRECY

a. The security policy shall provide for the control of information within datagrams based on the labeling of information.

b. This policy refers to end-user information at the MGS certification perimeter.

c. The unit of data exchange between Hosts and the Multinet Gateway System for end-user information shall be termed an *information unit*.

d. The unit of data exchange between Hosts and the Multinet Gateway System for non end-user information shall be termed a *non information unit*.

e. There shall be the notion of a *Sensitivity* level associated with each *information unit* that is to reflect the *Sensitivity* of the *information unit*. The *Sensitivity* level shall be realized via a *security label*. The *security label* shall consist of a security classification and a set of security categories.

f. Hosts may be authorized to send and receive data at more than one *Sensitivity* level. Associated with each Multinet Gateway System Port is one or more *security labels* authorized for the Hosts connected to that port via an End Network. There may be separate sets of *security labels* for incoming and outgoing ports.

g. It shall be possible to associate a *security label* with each *information unit* as it enters a port.

h. The *security label* associated with an *information unit* accepted into the System shall be one of the *security labels* associated with the port on which the unit was received. Otherwise, the *information unit* will not be accepted.

i. *Information units* may be transformed as they pass through the system. The transformations will be limited, however, so that data from one or more *information units* are combined into one *information unit* via a transformation only when the associated *security labels* are equal. The *security label* of the result is to equal the *security label* of the *information units* being transformed. The resulting unit is said to be *Derived From* the associated *information units* being transformed.

j. If an *information unit* is delivered to a port for transmission to a Host, then (1) it was *Derived From information units* accepted into the system and (2) the *security label* associated with this unit is one of the *security labels* associated with the *Destination* port. Otherwise, it will not be delivered.

k. *Information units* enter and leave the MGS only via End Networks.

l. *Non information units* received by the MGS will not compromise any information in *information units* and no *non information unit* sent out a MGS port will contain information from an *information unit*.

The above policy statements can be summarized as follows: The *security label* associated with an *information unit* is not to be changed while the unit is inside the system, the association between *security label* and data is to be maintained throughout the system, and data from two different *information units* can be combined inside the system only when the associated *security labels* are the same. An *information unit* will be allowed to enter (leave) the system only if an associated port possesses a *security label* set compatible with the *security label* of the *information unit*. Further, there shall be no mixing of *information units* with *non information units*.

The policy itself is quite general and can be used to describe a number of policies for potentially different applications. We give a summary of examples of this in Section 5.

INTEGRITY IN SUPPORT OF DATA SECRECY

The Multinet Gateway Integrity Policy is based on the notion of *information unit* described above. The Integrity Policy requires that information flowing out of the MGS is equivalent to information read into the MGS. The Integrity Policy provides the explicit definition of the *Derived From* relationship mentioned above. The terms used in the policy are the same as those for Protection Against Compromise. The Integrity Policy consists of the following rules:

a. This policy refers to information at the MGS certification perimeter.

b. Such information is embodied in *information units*, as specified in the policy for Protection Against Compromise.

c. An *information unit* delivered from an MGS must be *Derived From* at least one *information unit* accepted into the MGS. (Note that this statement is related to statements (h) and (i) of the Protection Against Compromise Policy.)

d. The delivered *information unit* must satisfy one of the following properties:

1. It must be the same as an *information unit* accepted into the MGS,

2. Its contents must be contained in an *information unit* accepted into the MGS, or

3. Its contents are a combination of *information units* accepted into the MGS.

The MGS is defined to be **Secure** with respect to Protection Against Compromise, if at all times, every *information unit* ever sent by the system to a port for delivery to a Host satisfies the *Data Secrecy* and *Integrity* statements above.

A distinct integrity model is not provided. Only these statements related to integrity are formally modeled and verified and then only within the context of the model for protection against compromise.

Accountability: The MGS Accountability Policy refers only to events that take place inside the MGS. The events that will be monitored are those that either affect the security of the system or represent an attempted security violation. These events will be logged in a protected fashion and made accessible to appropriate operators of the MGS. Since the MGS node is an Internet Device acting at the IP level, by its nature it is a *best effort* datagram forwarding device. The associated policy statement is that the MGS, within the limits of the IP protocol, will provide a best effort datagram forwarding service in getting the accountability information to the appropriate audit gathering facility.

Policy Statement: MGS Security: The MGS is said to be **SECURE** if is it secure with respect to both the Policy on Protection Against Compromise and the Policy on Accountability.

It is important to note that the formal model (given in Section 4) as well as the formal specification and its verification is the basis for increased assurance at the A1 level [1] that the running MGS satisfies the Policy on Protection Against Compromise.

MGS Model of Policy: Narrative

In this section, we present a narrative description of the Multinet Gateway System Security Policy Model. The model is one one of an external view of the system. The model is based on identified terms, a collection of security assertions about these terms, and specific relationships among them.

The formal description of the properties of the system, on which this narrative description is based, is given in Section 4. A discussion of several consequences of the formal model and how various security policies can be described using the model are given in Section 5.

Primitive Terms

The motivation and security policy specification of the previous sections have been given in rather concrete terms. The formal model is presented in more abstract terms to better describe the important concepts. The following list identifies terms in the model. They are given with a brief description of the intended semantics. The list also provides a means to associate the abstract terms with the concrete terms used in the previous sections.

INTERNAL_SYSTEM

The *INTERNAL_SYSTEM* is the system under discussion. This is the system that is being modeled. This section describes the security model of the *INTERNAL_SYSTEM*, which relates to the MGS discussed in the previous sections.

EXTERNAL_SYSTEM

The *INTERNAL_SYSTEM* provides data transfer services for the *EXTERNAL_SYSTEMs*. Although security properties of the *EXTERNAL_SYSTEMs* are not being demonstrated, assumptions about these security properties will be modeled. *EXTERNAL_SYSTEMS* relate to the Hosts, which use the services of the MGS.

i_wire

The *INTERNAL_SYSTEM* receives information from the *EXTERNAL_SYSTEMs* via *i_wires*. An *i_wire* represents an incoming connection to an END NETWORK.

o_wire

The *INTERNAL_SYSTEM* sends information to the *EXTERNAL_SYSTEMs* via *o_wires*. An *o_wire* represents an outgoing connection to an END NETWORK.

information_unit

There is a set *IU* of *information_units*. They are used to carry end-user information among *EXTERNAL_SYSTEMs* by way of the *INTERNAL_SYSTEM*. Note that we use the term *information_unit* here rather than Protocol Data Unit or datagram for the sake of generality and historical reasons. The two names refer to the same concept.

security_label

There is a set *SL* of *security_labels*. A single *security_label* is used to mark the *Sensitivity* of an *information_unit*. Each *i_wire* (*o_wire*) is associated with a set of *security_labels*. An *information_unit* is accepted for transfer from an *i_wire* (or transfer to an *o_wire*) only if its *security_label* is an element of the set of *security_labels* associated with the *i_wire* (or *o_wire*). Note that it is not necessary at this time to describe an inner structure or components of a *security_label* in order to define the various functions on a *security_label*, or to define what is meant by the term **SECURE** system. A *security_label* encompasses the notion of a security attribute as used in the previous sections.

Derived_From

There is a notion of one *information_unit* being *Derived_From* one or more other *information_units*. The *Derived_From* operation permits the description of the security properties of fragmentation, assembly, transfer from one location to another, encryption and decryption.

Figure 2 illustrates this more general setting and is to aid the understanding of these terms when compared with the terms used to describe the formal model. Again, it is important to note that the policy model views the *INTERNAL_SYSTEM* as a black box.

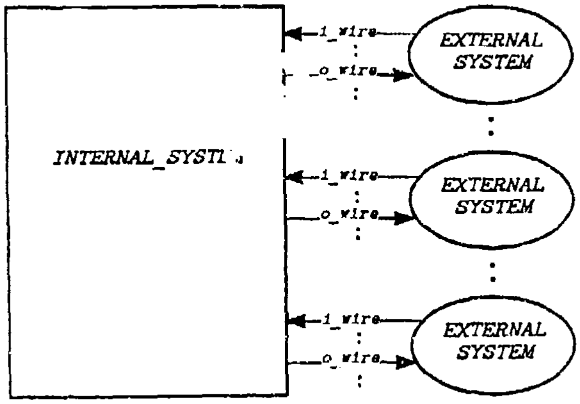


Figure 2. A Pictorial Description of the Model

Security Assertions

The following assertions are to be satisfied by the system. By communication we mean the transfer of *information_units*.

a. *i_wires* and *o_wires* always connect *EXTERNAL_SYSTEMs* to the *INTERNAL_SYSTEM*

b. The only communication into and out of the *INTERNAL_SYSTEM* is via *i_wires* and *o_wires*.

c. All communication across *i_wires* and *o_wires* consists of *information_units*.

d. Each *information_unit* has an associated *security_label*.

e. Each *i_wire* and each *o_wire* has an associated set of *security_labels*.

f. The *INTERNAL_SYSTEM* accepts an *information_unit* only from *i_wires* and only if the *security_label* of the *information_unit* is an element of the set of *security_labels* associated with the *i_wire* bearing the *information_unit*.

g. The *INTERNAL_SYSTEM* delivers an *information_unit* only to *o_wires* and only if the *security_label* of the *information_unit* is an element of the set of *security_labels* associated with the *o_wire* to which the *information_unit* is delivered.

h. If a given *information_unit* is delivered to an *o_wire*, then it was *Derived_From information_units* accepted from *i_wires*. Additionally, the *security_label* of each such accepted *information_unit* must equal the *security_label* of the delivered *information_unit*.

There are three major points addressed by these assertions. First, there is an acceptance criterion (Assertion f). Secondly there is a *Derived_From* criterion (Assertion h), and finally there is a delivery criterion (Assertion g). The other assertions are there to guarantee that the *INTERNAL_SYSTEM* has the appropriate relationship with *EXTERNAL_SYSTEMs*. Figure 3 illustrates the acceptance into , derivation and delivery out of the MGS.

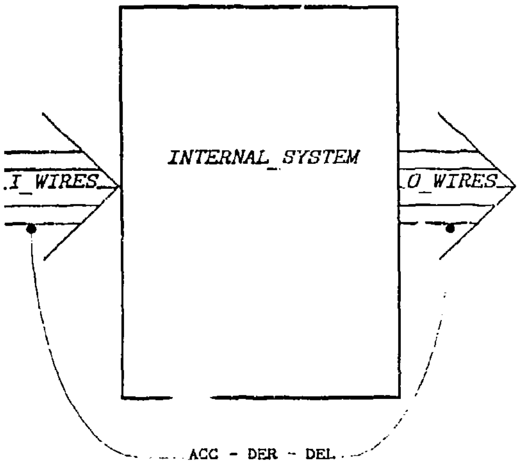


Figure 3. Acceptance, Derivation and Delivery of the Model

MGS Model of Policy: Mathematical Description

This section presents the formal model of the security policy on Protection Against Compromise, which is given in Section 2.

The MGS Security Policy Model is a structure consisting of three components. The first component, specified in section 4.1, is a collection of sets. The second component, specified in section 4.2, is a collection of functions using these sets. These functions are termed "primitive" because they are the basis of all the security relationships being specified. The third component, specified in section 4.3, is a collection of boolean-valued functions which specify the necessary relationships among the various functions. These are the security assertions. Using these relationships of the model, an expression is then given that specifies what it means for a system to be **SECURE** and based on the model.

Throughout this section the notation *PS (M)* is used to denote the POWER SET of a given set M. For a set M, the power set, *PS (M)*, is the set of all subsets of the set M. Additionally, for two arbitrary sets, A and B, A × B denotes the cartesian product of the sets.

Underlying Sets For the Policy Model

Let *I_WIRE*, *O_WIRE*, *SL*, *IU* and *DU* be non-empty sets. Let elements of these sets be called *i_wires*, *o_wires*, *security_labels*, *information_units* and *data_units*, respectively. No assumptions are made about the sets other than that they are finite and no two of them have a common element. Further, let *IU* contain two sets, IU_{in} and IU_{out}. Elements of IU_{in}, (IU_{out}) represent *information_units* coming into (leaving) the *INTERNAL_SYSTEM*, respectively. Let *DU* contain a distinguished element, termed the *null_data_unit*. Let *INTERNAL_SYSTEM* be a single object. These sets model the primitive terms in the previous section except for the term *Derived_From*.

Primitive Functions

The following functions are the basis for the security assertions identified in the policy section. Let functions be specified as follows:

Derived_From:

$$\mathit{Derived\_From} : IU_{out} \rightarrow PS\ (IU_{in}) \tag{1}$$

The function *Derived_From* associates with each *information_unit*, *iu*, leaving the *INTERNAL_SYSTEM*, a subset of *information_units* that enter. The *Derived_From* function can be used to discuss the necessary security properties of fragmentation, assembly, transport, encryption and decryption. This discussion is illustrated by presenting, in Section 5.2, the specific relationship between fragmentation and assembly with *Derived_From*.

Is_Received:

$$\mathit{Is\_Received} : IU_{in} \times \mathit{I\_WIRE} \rightarrow \{\ T\ ,\ F\ \} \tag{2}$$

The function *Is_Received* associates with each *information_unit* and *o_wire* pair a boolean value. If *Is_Received (iu, i_wire)*, then the *iu* was actually received on that *i_wire* by the *INTERNAL_SYSTEM*.

Is_Delivered:

$$\mathit{Is\_Delivered} : IU_{out} \times \mathit{O\_WIRE} \rightarrow \{\ T\ ,\ F\ \} \tag{3}$$

The function *Is_Delivered* associates with each *information_unit* and *o_wire* pair a boolean value. If *Is_Delivered (iu, o_wire)*, then the *iu* was actually sent to that *o_wire* by the *INTERNAL_SYSTEM*.

Sensitivity:

$$\mathit{Sensitivity} : IU \rightarrow SL \tag{4}$$

The *Sensitivity* function associates with each *information_unit* a *security_label*. The *security_label* associated with each *iu* is used to control the acceptance and delivery of *information_units* on particular *i_wires* and *o_wires*.

I_Wire_Allow:

$$\mathit{I\_Wire\_Allow} : \mathit{I\_WIRE} \rightarrow PS\ (SL) \tag{5}$$

The function *I_Wire_Allow* associates with each *i_wire* a subset, possibly null, of *security_labels*. An *iu* can be accepted into the *INTERNAL_SYSTEM* only if an appropriate relationship exists between the *Sensitivity* of the *iu* and the set of *security_labels* associated with the *i_wire*. The function *I_Wire_Allow* allows one to describe that relationship, which is given by the function *Is_Securely_Accepted*, and specified in the Security Assertions subsection (4.3).

O_Wire_Allow:

$$\mathit{O\_Wire\_Allow} : \mathit{O\_WIRE} \rightarrow PS\ (SL) \tag{6}$$

The function *O_Wire_Allow* associates with each *o_wire* a subset, possibly null, of *security_labels*. An *iu* can be delivered from the *INTERNAL_SYSTEM* to the *o_wire* only if an appropriate relationship exists between the *Sensitivity* of the *iu* and the set of *security_labels* associated with the *o_wire*. The function *O_Wire_Allow* allows one to specify that relationship, which is given by the function *Is_Securely_Delivered*, and specified in the Security Assertions subsection also (4.3).

Data:

$$\mathit{Data} : IU \rightarrow DU \tag{7}$$

The function *Data* associates with each *information_unit* a *data_unit*. It represents the data portion of the *information_unit*.

Is_Part_Of:

$$\mathit{Is\_Part\_Of} : DU \times DU \rightarrow \{\ T\ ,\ F\ \} \tag{8}$$

The function *Is_Part_Of* define a relation on *data_units*. Let it be reflexive and transitive.

Data_Combine:

$$\mathit{Data\_Combine} : PS\ (DU) \rightarrow DU \tag{9}$$

The function *Data_Combine* permits one to describe the bringing together of *data_units* into a single *data_unit*. Let the image of the empty set, (which is an element of *PS (DU)*), be the distinguished element of *DU*, the *null_data_unit*.

Data_Accounted_For:

$$\mathit{Data\_Accounted\_For} : IU \times PS\ (IU) \rightarrow \{\ T\ ,\ F\ \} \tag{10}$$

Data_Accounted_For is defined as:

$$\begin{gathered} \mathit{Data\_Accounted\_For}\ (iu,\ X) \\ \textbf{iff} \\ \left\{ \begin{array}{l} \text{There exists a set } P,\ P \subseteq DU, \text{ such that} \\ \mathit{Data}\ (iu) = \mathit{Data\_Combine}\ (P)\ \ \& \\ p \in P \ \text{ implies there exists } x_p \in X,\ \mathit{Is\_Part\_Of}\ (p,\ \mathit{Data}\ (x_p)\,)\ \ \& \\ x \in X \ \text{ implies there exists } p_x \in P,\ \mathit{Is\_Part\_Of}\ (p_x,\ \mathit{Data}\ (x)\,) \end{array} \right\} \end{gathered} \tag{11}$$

This function describes what it means for a given *data_unit* to be related to other *data_units*.

Security Assertions

Security assertions identified in the narrative description are expressed in mathematical terms next.

Is_Securely_Accepted: The following definition specifies what it means to be accepted into the *INTERNAL_SYSTEM*.

$$\mathit{Is\_Securely\_Accepted} : IU_{in} \times \mathit{I\_WIRE} \rightarrow \{\ T\ ,\ F\ \} \tag{12}$$

Is_Securely_Accepted must have the property:

$$\begin{gathered} \mathit{Is\_Securely\_Accepted}\ (iu,\ w) \\ \textbf{iff} \\ \mathit{Sensitivity}\ (iu) \in \mathit{I\_Wire\_Allow}\ (w)\ \ \& \\ \mathit{Is\_Received}\ (iu,\ w) \end{gathered} \tag{13}$$

The function *Is_Securely_Accepted*, is related to the functions *Sensitivity*, *I_Wire_Allow* and *Is_Received*. The necessity for such a relationship is that in an actual system the *INTERNAL_SYSTEM* may read in *information_units* from an *i_wire* and may not be able to determine the *Sensitivity* of the particular *information_unit* until after it has been read in. Once the *Sensitivity* has been determined, it is then possible to say whether it is permissible to process it further. If *Is_Securely_Accepted (iu, w)*, then the *iu* is a candidate for further processing.

Is_Securely_Derived:

$$\mathit{Is\_Securely\_Derived} : IU_{out} \rightarrow \{\ T\ ,\ F\ \} \tag{14}$$

Is_Securely_Derived must have the property:

$$\begin{gathered} \mathit{Is\_Securely\_Derived}\ (iu) \\ \textbf{iff} \\ \left\{ \begin{array}{l} \text{for every } iu' \in \mathit{Derived\_From}\ (iu) \\ \quad \mathit{Sensitivity}\ (iu) = \mathit{Sensitivity}\ (iu')\ \ \& \\ \quad \text{for some } \mathit{i\_wire}\ \ w,\ \mathit{Is\_Securely\_Accepted}\ (iu',\ w)\ \ \& \\ \mathit{Data\_Accounted\_For}\ (iu,\ \mathit{Derived\_From}\ (iu)\,) \end{array} \right\} \end{gathered} \tag{15}$$

The function *Is_Securely_Derived* is related to other functions. This relationship is specified in expression (15).

Specifically, a given *information_unit* is determined to be securely derived if and only if three conditions are satisfied. First, the particular *information_unit* was ***Derived_From*** a set of *information_units* that they themselves were securely accepted. Second, each of these securely accepted *information_units* have the same ***Sensitivity*** as the derived *information_unit*. Third, the data portion of derived *information_unit* equals the combination of ***data_units*** that are themselves part of the data of the incoming *information_units*. Note that the ***Sensitivity*** of the derived *iu* is the same as the ***Sensitivity*** of the ***accepted ius***. If an *information_unit* is determined to be securely derived, then it is a candidate for further processing.

Is_Securely_Delivered: The next definition specifies the conditions that must be met before an *information_unit* can be placed on a particular *o_wire*.

$$\mathit{Is\_Securely\_Delivered}: IU_{out} \times O\_WIRE \rightarrow \{\ T\ ,\ F\ \} \qquad (16)$$

Is_Securely_Delivered must have the property:

$$\begin{gathered} \mathit{Is\_Securely\_Delivered}\ (iu\ ,\ w) \\ \textbf{iff} \\ \left\{ \begin{array}{l} \mathit{Sensitivity}\ (iu) \in \mathit{O\_Wire\_Allow}\ (w)\ \ \& \\ \mathit{Is\_Securely\_Derived}\ (iu) \end{array} \right\} \end{gathered} \qquad (17)$$

The function *Is_Securely_Delivered* is a boolean function. Just as there is a specific relationship between the function *Is_Securely_Accepted* and some other functions, there is to be a specific relationship with the function *Is_Securely_Delivered* and additional functions. Expression (17) gives that relationship. Specifically, a particular *information_unit* is said to be securely delivered if and only if two conditions are satisfied. First, the set of ***security_labels*** associated with that particular *o_wire* must have as an element, the ***security_label*** of the particular delivered *information_unit*. Second, the *information_unit* must have been securely derived. These are the conditions specifying what it means to deliver an *information_unit* to a given *o_wire* securely.

Definition of a Secure System

The definition of a secure ***system*** is given in this subsection. As presented, the MGS Security Policy Model is a structure consisting of three components. Even though it allows for considerable flexibility, it is intended that the meaning of security be the same no matter how one may choose to use the flexibility provided. The flexibility is allowed by letting the actual choice of the sets (first component) and the functions defined using them (second and third components) be left to a given design and implementation approach.

In order to accurately describe what it means for a ***system*** to be **Secure** based on the model, two additional notions need to be described. First, definitions of what an instance of the model is and what security means within that particular instance of the model need to be identified. Second, a means of relating a system to a particular instance of the model needs to be identified.

An *INSTANCE_OF_MODEL* is defined to be a particular choice for each of the five sets and a particular choice of the primitive functions, e.g., ***Derived_From***, ***Is_Received***, ***Is_Delivered***, ***Sensitivity***, ***I_Wire_Allow***, etc. that satisfy the specified expressions (1)-(17).

Note that the functions, *Is_Securely_Accepted*, ***Is_Securely_Derived*** and ***Is_Securely_Delivered***, are defined in terms of the previous functions and no arbitrary choice can be made for them.

An *INSTANCE_OF_MODEL* is defined to be **SECURE** if, for the corresponding choices made in determining the particular *INSTANCE_OF_MODEL*, the following statement is true:

$$\begin{gathered} \text{for every } iu \in IU \\ \textbf{if} \\ \mathit{Is\_Delivered}\ (iu\ ,\ w) \\ \textbf{then} \\ \mathit{Is\_Securely\_Delivered}\ (iu\ ,\ w) \end{gathered} \qquad (18)$$

Note that the expression in (18) is the only place where the function ***Is_Delivered*** is related to the function ***Is_Securely_Delivered***.

Consider an arbitrary system and a given instance of the model, denoted by *INSTANCE_OF_MODEL*. An association of the *INSTANCE_OF_MODEL* to the entities of the system is defined to be a mapping from the particular components of *INSTANCE_OF_MODEL* to the system's entities.

A ***system*** is said to be **SECURE and Based on the Model** if the following conditions are satisfied. First, there exists an instance of the model, *INSTANCE_OF_MODEL*. Second, there exists an association of the instance of the model, *INSTANCE_OF_MODEL*, to the system's entities. Third, suppose the system actually delivers an entity that corresponds to an ***information_unit*** under the particular association and model instance. Then the expression numbered (18), when interpreted within the system via the same association, is to evaluate to true if and only if the expression (18) within *INSTANCE_OF_MODEL* evaluates to true.

MGS Model of Policy: Discussion

The policy stated as the MGS Security Policy in Section 2 and the associated formal model given in Section 4 are very general. Depending on given instances of the model, including the definition of ***security_labels*** and the association of ***security_labels*** with the ports of a system, all sorts of flows are possible. To enforce DoD policy, one expects to use the DoD security labeling scheme. One expects to not permit the ***write down*** of information; that is, labeling a port with a ***high*** label when all the *EXTERNAL_SYSTEMs* connected to it are at ***low*** level. The associated ***wires*** should not be permitted to carry the high data. The model, in fact, disallows this. This is one of the places where the administrative actions in determining the ***security_labels*** associated with ports becomes crucial to enforcing DoD policy.

Specific observations about the formal model and the type of policies one can obtain from the model are summarized below.

Observations On the Formal Policy Model

There is neither a "destination" nor "source" function defined on the set of information units. While it is tempting to introduce such concepts, it is not necessary for this particular environment, and would probably impose additional difficulties for the formal verification of a system based on such a model. Additionally, the formal model does not explicitly incorporate (at this time) ***non_information_units*** although the security policy does identify such entities. The focus of the formal modeling has been on handling end-user information rather than on modeling, for example, protocol control messages. Such information would not have the same acceptance or delivery checks. Further, such non end-user information leaving the system needs to be "derived from" only non end-user and system initialization information and not be mixed with end-user information.

Note that the empty set (or null label) can not be associated with an *information unit.* A particular *i_wire* or *o_wire,* however, may have the empty set associated with it (via the functions *I_Wire_Allow* and *O_Wire_Allow*). If it is an *i_wire,* then no *information_units* will be accepted from that *i_wire.* If it is an *o_wire,* no *information_units* will be delivered to it.

There is nothing in the assertions that guarantees the delivery of *information_units* once they are accepted into the system. It may turn out that all *security_label* sets associated with *o_wires* do not contain the *security_label* of an accepted *information_unit.*

If an *information_unit* was delivered from the *INTERNAL_SYSTEM,* then there was at least one *information_unit* accepted into the *INTERNAL_SYSTEM.* The information contained in the delivered *iu* was *Derived_From* the accepted *information_unit*(s).

There is no connection in the model between an *i_wire* and an *o_wire.* In a particular application, each *i_wire* may be paired with an *o_wire.* But the *security_label* sets attached to the *i_wires* and *o_wires* are independent of each other. Hence in an *i_wire / o_wire* pair, the *i_wire* and *o_wire* may have different *security_label* sets associated with them. Also, an *INTERNAL_SYSTEM* can securely deliver an *iu* to more than one *o_wire.*

An *EXTERNAL_SYSTEM* may be connected to the *INTERNAL_SYSTEM* by a single *i_wire* or *o_wire.* That is, the *EXTERNAL_SYSTEM* may be only a source or a sink for information with respect to the *INTERNAL_SYSTEM.*

Further, the data will not necessarily stay constant throughout its traversal of an *INTERNAL_SYSTEM.* In fact, it may change because of fragmentation, encryption, decryption, etc. Consequently, there is no reason to expect any verification regarding the possibility of no change to the data or some portion of the data.

The security policy and formal model allow transformations on information units that relate, under a given condition, two or more units to another single information unit (refer to policy statement (h) of Protection Against Compromise and the notion of *derivation*). This may potentially permit data aggregation, where, by the model scheme, a resulting information unit would have a sensitivity label possibly lower than the aggregation of the set of associated information units. There are two aspects to this aggregation. First, the aggregation is achieved "outside" the system. Namely, the system being modeled is a datagram service and the aggregation would be related to the higher level transport services using the datagram service of the system being modeled. Second, if, within the modeled system, one assembles information units, they are assembled from a related entity that was previously fragmented. The assembly is actually the re-assembly of a previously fragmented datagram. Therefore, in this second case there would be no aggregation.

Finally, the concept of *Derived_From* is distinct from the concept of *Data* and the combination of data. The latter is introduced to describe the necessary properties of fragmentation and assembly. The concepts are brought together only in the meaning of *Is_Securely_Derived.*

Assembly, Fragmentation and Derived_From

This subsection shows how the function *Derived_From* models both assembly and fragmentation. The upper portion of Figure 4 illustrates the concept of the assembly and fragmentation of *information_units* across the *INTERNAL_SYSTEM.*

Now consider the function *Derived_From.* As defined, the image of an *information_unit* under *Derived_From* is a set of incoming *information_units.* Observe that the sense of direction of the function, *Derived_From,* considers an outgoing *information_unit* and "looks back" at what the given *information_unit* is derived from among the incoming *information_units* (refer to lower portion of Figure 4). In this way assembly is represented directly by the function *Derived_From.*

To explicitly relate fragmentation to *Derived_From,* one additional definition is needed. For a given *information_unit, iu* belonging to IU_{in}, define the following set:

$$FRAGMENTS\ (iu) = \{\ f \mid f \in IU_{out},\ iu \in Derived\_From\ (f)\ \}$$

The set, *FRAGMENTS (iu),* identifies all those outgoing *information_units* that are related to the given *iu* by the function *Derived_From.* In this way, fragmentation is modeled by the function *Derived_From.*

Two properties are associated with *Derived_From* (ref: expression 15 of Section 4). They are the security properties for fragmentation as well as assembly. First, for a given outgoing *information_unit, iu,*

$$\text{For all } x \in Derived\_From\ (iu).\ Sensitivity\ (iu) = Sensitivity\ (x)$$

Second, all portions of data in *iu* are to be accounted for by data in the *information_units* of the set, *Derived_From (iu).* This concept is captured by *Data_Accounted_For* of Section 4.

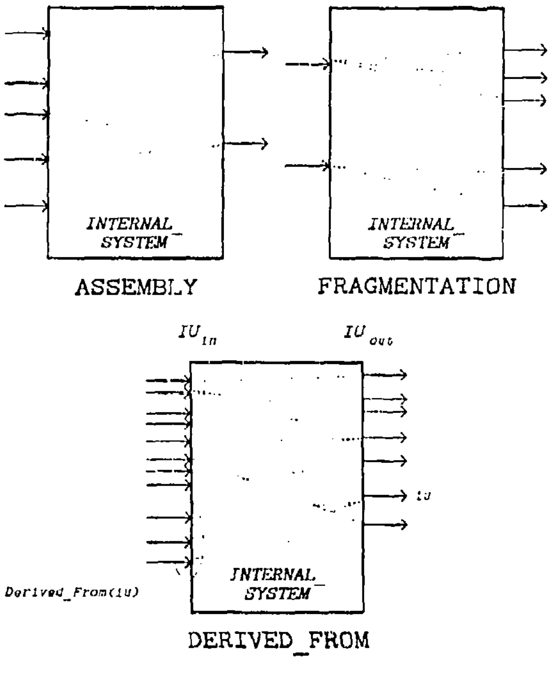


Figure 4. Assembly, Fragmentation and Derived_From

An incoming *information_unit* that does not leave the *INTERNAL_SYSTEM* is modeled by *Derived_From* by having that incoming *information_unit* not be an element of the image of any outgoing *information_unit* under fragmentation are adequately modeled as described, then the approach is symmetric. Specifically, referring to the lower portion of Figure 4 and "looking forward," rather than "backward," one could define a function "Derive_For," which would associate a set of outgoing *information_units* with an incoming *information_unit*. Fragmentation and assembly would be modeled in an analogous manner. Since we are interested in what goes out of a system based on what comes into it, the choice was made as given.

Security Assertions as Mathematical Relationships

Certain assertions within the policy are represented within the formal model as specific relationships among functions. Specifically, one can think of the *INTERNAL_SYSTEM* as a device that reads *information_units* from *i_wires*, produces new *information_units* from the accepted ones, and delivers *information_units* to *o_wires*.

The security assertions (a)-(c) of subsection 3.2 above must actually be satisfied by both physical and hardware limitations of the *INTERNAL_SYSTEM*. It is assumed that the *INTERNAL_SYSTEM* is connected only to packet switch networks. It is assumed that all input and output of *information_units* to the *INTERNAL_SYSTEM* occur through only these networks. It is assumed that these networks are connected only to external *INTERNAL_SYSTEMs* and in fact that the *information_units* transferred to the *INTERNAL_SYSTEM* have security_labels. That is, that it is possible for the *INTERNAL_SYSTEM* to determine the *security_label* of all *information_units* received.

The collection of functions, *Derived_From*, *Is_Received*, *Is_Delivered*, *Sensitivity*, *I_Wire_Allow*, and *O_Wire_Allow*, are primarily for description. They either describe properties of the *information_units* or of the *i_wires* and *o_wires*.

The last three security assertions stated in section 3.2 are described mathematically in sections 4.3.1-4.3.3. At any instant of time, the *INTERNAL_SYSTEM* must be secure in the sense of the definition in 4.4. That is, all *information_units* actually sent to an *o_wire* up to that point must satisfy the conditions given in 4.4.

Implementing a System-Specific Security Policy

To implement a particular security policy within a system based on the model, several steps are required.

a. Determine the set of *security_labels*.

b. Determine the set of *information_units* to be managed by the *INTERNAL_SYSTEM*. This set will change with time.

c. Determine the mechanism(s) in the *INTERNAL_SYSTEM* for determining the *Sensitivity* of *information_units*.

d. There must be databases, or some other mechanisms, in the *INTERNAL_SYSTEM* so that the functions *I_Wire_Allow* and *O_Wire_Allow* can be implemented. These databases must be securely initialized and protected from unauthorized changes. The content of these databases determines the acceptable flow of *information_units* from *i_wires*, through the *INTERNAL_SYSTEM*, and finally to *o_wires*. Various specific policies can be implemented depending on the content of the databases.

e. The *INTERNAL_SYSTEM* must have a notion of *Derived_From*, which has the identified properties.

f. It must then be verified that the system maintains the definition of **SECURE** for every *information_unit* actually sent out on an *o_wire*.

The model supports a number of specific security policies. The security policy in force for a particular implementation of the model depends on the *security_label* set and the distribution of the subsets of *security_labels* to the various *i_wires* and *o_wires*. For example, DoD policy would be that if an *i_wire* or *o_wire* could carry top secret information, it could also carry secret, confidential and unclassified information, unless explicitly stated otherwise. This can be modeled by having the *security_label* set of the *i_wire* or *o_wire* contain all four classifications. This allows the tie to the familiar "dominance" relation identified in [1].

Another policy might be that the *i_wire* or *o_wire* connected to *EXTERNAL_SYSTEMs* should only allow secret information. This can be described in the model by having the *security_label* set associated with the *i_wire* and *o_wire* that connect the *EXTERNAL_SYSTEMs* to the *INTERNAL_SYSTEM* to contain only the element secret. *Information_units* with other than a secret *security_label* cannot then be carried on the particular *i_wire*. Other specific policies can be realized via this general policy model.

Model Validations

It is often required to validate a given model in two ways. First, validate that the model actually represents the concepts and statements within a given security policy. Call this an *external* validation. For the Multinet Gateway System, this *external* validation is based on the Protection Against Compromise Policy. Second, validate, by some reasonable means, that the model is consistent within itself. Call this an *internal* validation. Both validations of the formal model are given in [9]. The internal validation gives our interpretation of the requirement stating ". . . a formal model of the security policy supported by the TCB shall be maintained . . . that is proven consistent with its axioms [1, para. 4.1.3.2.2]."

Conclusions

This paper has presented an internet security policy and formal security policy model. The scope of the security policy is the collection of the security properties of a packet-switched internet system providing a datagram service. The formal model focuses on protection against compromise. The paper has summarized the approach taken to show informally that the model is a representation of an appropriate portion of the policy and that the model itself has an internal consistency. It illustrates a way of modeling the security attributes of an internet system and provides a specific example of one such security model. The model and approach outlined here has been used in the production of the formal specification, in Gypsy, of the Multinet Gateway System, and in the formal verification of that specification.

The authors gratefully acknowledge the careful review provided by Jim Williams, Don Good, Mike Smith and Max Heckard of previous drafts during the development of this paper.

References

[1] DoD Computer Security Center, "Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, Dec., 1985.

[2] J. McLean, C. Landwehr, C. Heitmeyer, "A Formal Statement of the MMS Security Model," *Proceedings of the 1984 Symposium on Security and Privacy*, April 29-May 2, 1984, Oakland, Calif. pp. 188-194.

[3] J. Glasgow, G. MacEwen, "A Two-Level Security Model For a Secure Network," *Proceedings of the 8th National Computer Security Conference*, Sept. 30-Oct. 3, 1985, Gaithersburg, Md., pp.56-63.

[4] J. Goguen, J. Meseguer, "Security Policies and Security Models," *Proceedings IEEE Symposium on Security and Privacy*, April 1982, pp. 11-22.

[5] D. Nessett, "Factors Affecting Distributed System Security," *Proceedings IEEE Symposium on Security and Privacy*, April 7-9, 1986, Oakland, Calif., pp. 204-222.

[6] D. Bell, L. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," Technical Report, MITRE Corp., 1971, Bedford, MA.

[7] J. Millen, Introductory Remarks, Session on "Toward a Theory of Computer Security," ACM SIGSAC Symposium on Distributed Systems & Local Networks, UCLA, Nov. 14, 1987.

[8] Baker, P. C., Dinolt, G. W., Freeman, J. W., Krenzin, M. D., and Neely, R. B., "A1 Assurance For an Internet System: Doing the Job", *Proceedings of the 9th National Computer Security Conference*, Sept. 15-18, 1986, Gaithersburg, Md., pp. 130-137.

[9] Ford Aerospace Corporation, "Security Model Task Report: Policy and Formal Model," (Rev. to Final) July, 1988, CSD-TR1711.

ULYSSES: A Computer-Security Modeling Environment

Tanya Korelsky, Bill Dean, Carl Eichenlaub,
James Hook, Carl Klapper, Marcos Lam,
Daryl McCullough, Clay Brooke-McFarland, Garrel Pottinger,
Owen Rambow, David Rosenthal, Jonathan P. Seldin,
and D. G. Weber

Odyssey Research Associates, Inc.
301A Harris B. Dates Drive
Ithaca, New York 14850-1313 *

Abstract

This paper presents an overview of the Ulysses computer security modeling environment. Ulysses is a design environment in which models of systems can be described formally, properties of those models can be verified, and in which specialized security analysis is supported by a formal theory of security. The theory of security is motivated by non-deducibility and non-interference concerns, and it also permits the security analysis of complex designs by decomposing them into interacting parts. Graphical and textual specification languages allow users to describe these design decompositions in an intuitive manner, while remaining grounded in the formal theory of security. A natural-language component generates English descriptions of user-created models. A library facility allows re-use of secure models. The use of this environment requires extensive theorem-proving and heuristic support; this is provided by a powerful mathematical engine, incorporating a meta-language facility.

1 Introduction

Ulysses is a collection of tools that assist in the design and verification of secure computer systems. It is being developed at Odyssey Research Associates (ORA) in Ithaca, New York. It provides a rich environment in which both new and previously defined secure systems and secure system components can be dynamically examined and incorporated into a system design. The design methodology supported by Ulysses uses the same principles of modularity and reusability that characterize modern programming development environments. Because Ulysses supports the verification of security properties, it includes an automated theorem proving engine and tools for constructing proofs. This paper is intended to give an overview of the important ideas and tools incorporated by the system.

From a security standpoint, the most important feature of Ulysses is the capability of producing a complete and formal proof of security. A *security methodology* is a definition of security together with a collection of theorems which aid in constructing a proof of security for particular models. These theorems are often expressed as conditions for deducing the security of a whole system from the properties of its components. With such a methodology, the task of proving security of an entire system reduces to the smaller tasks of showing the particular properties on only parts of the system. When a methodology can be carried out formally we say that it is the basis for a *formal security analysis.* One instance of such a security methodology is the noninterference security definition and theory of [McC88b]. In this case, whenever all of the components are shown to be secure then one can conclude the system is also secure. This sort of property is called *composable* or a "hook-up" property. Composable properties are particularly easy to work with. The more general security methodologies are often constructed to be applied to particular classes of models (e.g. a process connected to a buffer). The Ulysses environment is one which aids in both the development and the formal application of security methodologies.

As a design tool, Ulysses was influenced by the experience of the MASCOT project [Sta86] and the hierarchical design abstraction of Moriconi's PegaSys system [MH85]. These pictorial system description schemes are similar to the graphical specification language that Ulysses users will be given to describe systems. The design process, which Moriconi calls refinement, is primarily "top-down". A user begins with a diagram representing the entire system, and refines it by dividing it into sub-systems, each represented by an icon. Connections between sub-systems are also specified as icons. The meaning of each icon is given formally in the theory of security, and the user may also associate other information (documentation, other formal specifications) with the icons. The design is "grounded" by associating formal textual specifications in the theory of security with atomic icons

*This work was supported by the Air Force Systems Command at Rome Air Development Center under Contract No. F30602-85-C-0098. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Air Force or the U.S. Government.

(i.e., icons representing components that have not been further graphically refined). If each atomic component is proved secure (in the sense of [McC87]), then the "hook-up theorem" can be used to infer the security of the entire system.

One of the more innovative aspects of the Ulysses system is its mathematical foundation. Ulysses is being developed in a formal system based on a constructive type theory which is also capable of expressing classical mathematics. The advantages of this foundation fall under two heads—improved support for security modeling and exciting prospects for future extensions.

Security modeling support is enhanced because the logical basis allows for a more rigorous treatment of modeling than previous bases. This enhancement has two main aspects. First, for many security theories, it is possible to formalize the relation between the theory and the semantics of the specification language used to describe systems. Second, the economy with which the underlying logic is formulated and its known consistency allay doubts about correctness of the implementation and about correctness of the logic itself.

The logic also allows the modeling of polymorphic typing, which is of great interest in current discussion of programming languages, and, due to its constructive character, includes a powerful model of computation. We believe that these features will make it possible to extend Ulysses to include a system development environment which is both sound and robust.

The paper is organized as follows. Section 2 explains in greater detail the primary theory of security being used in Ulysses. Next, in section 3, various ways in which the system can be used are described. The implementation of the type theory mentioned above is discussed in section 4. Finally, in section 5, we conclude with a few remarks about the software implementation of Ulysses.

2 Security Analysis

Secure design in Ulysses depends on flexible and sound theoretical foundations. To develop such foundations we examined previous formalisms for security, particularly the pioneering work of Bell and LaPadula in access control[BL76], the non-interference model of Goguen and Mesegner[GM82], and the information flow theory of Sutherland [Sut86].

Our investigations convinced us that these previous models of security were, for the purposes of secure design in Ulysses, lacking in some respects. Some of the problems we found among these formalisms were

- they were not based on observable behavior
- they were not sufficiently implementation-independent
- they could only be applied to completed systems, and therefore could not be used for the incremental development of a secure design
- they only applied at one level of abstraction
- they were only suitable for deterministic systems

The biggest problem, however, was that there was no research on the interactions of trusted systems and processes—in particular, it was not known to what extent security was preserved when one connected several trusted systems into a distributed system. The primary security formalism used by Ulysses is based on this previous work, but it goes beyond it in that it is intended to be useful in *design* as well as in implementation. In contrast with the preceeding formalisms, the Ulysses security formalism can be used to analyze the security of isolated components and partially fleshed-out system designs, whose implementations are still undetermined. This gives the designer greater flexibility, allowing him to

- reuse off-the-shelf secure components
- discover the security flaws of a design early so as to minimize wasted effort
- freely substitute components with equivalent security characteristics

A formal definition of secure processes that had many of the desirable features mentioned above has been developed by McCullough ([McC88b]). One of the properties derivable from this theory is the "hook-up" property, which provides the basis for a formal security analysis. The theory is formulated in terms of state-machines. Each state transition corresponds to a possible input, output or internal event of a system. Non-deterministic choice between different transitions is allowed. Within this framework, a security property can be defined. We call this property 'flow security'. It is a noninterference property which limits the effect that transitions associated with high security levels can have on transitions at lower security levels. These limits formalize the intuitive notion that information should not flow from high level users to low level users. It is a composable property, meaning that if system A and system B are each flow-secure, then the combined system of A composed with B will also be. It must be noted that other security properties often turn out not to be composable [McC88a].

Using Ulysses to prove that components are flow-secure will permit us to incrementally verify the security of a system. Once the flow-security of all atomic components is verified, the flow-security of the entire system is assured.

Although a particular theory of processes and a particular theory of security are used in Ulysses, they are neither fixed nor "built in". The theory of security may be expanded by proving (within Ulysses) new facts about the hook-up of processes. For example, the hook-up of several processes, none of which is secure, may form a combined process that is secure [WL87]. Ulysses allows the formulation of such new theories of security. These alternate theories could then be used in proving new hook-up theorems as well as properties of systems. The mechanisms for packaging new theories and referencing them are outlined in section 4.

3 How Ulysses Will Be Used

The user may interact with the system through several specially designed interfaces. They are:

- *The graphical system design interface:* graphical system descriptions, in which icons have formal meaning in the theory of security, are used to describe the design of a model from its secure components
- *The natural language component:* brief summaries of the design and its security characteristics may be generated automatically

- *The verification and textual specification interface:* textual specifications of components can be built and verified, new security theories can be added to the system, and tactics (heuristics) for proving security can be built
- *The library browsing interface:* models and other information stored in the Ulysses system may be reviewed and updated

In the next few sections we provide a more detailed description of these interfaces.

3.1 Operations

User interactions fall into three main divisions: adding information to the system, retrieving information from the system and deriving new information within the system. The subject of most end user interactions will be descriptions of computer systems. These are most naturally presented in a graphical manner, allowing the user to visualize the subject system. The graphic language employed presents objects hierarchically, much like the PegaSys system[MH85]. Within this context of graphical representations the following operations are typical of what the Ulysses user interface supports:

- Retrieving Information:
  - view an archived system or component
  - show (or hide) the sub-components of an icon representing a process
  - show the textual specification associated with an icon
  - show the textual specification associated with the hook-up of two icons
- Adding Information:
  - load an archived component into the current system design
  - create (or delete) a hook-up (i.e., a communication channel) between two processes
  - refine an icon representing a system by adding or changing icons representing its sub-components
  - associate formal textual specifications and other properties with an icon
- Synthesizing Information:
  - prove that component specifications imply the specifications of the systems they form
  - give general mathematical facts to assist in proving security
  - ask the system to assist heuristically in developing a secure system

Some of these operations involve interfaces besides the strictly pictorial or graphical one suggested above. For example, the natural language interface produces English text associated with graphical displays of the system. Textual specifications for atomic icons are produced by interacting with an editor. Retrieving archived systems may involve browsing through the library of stored specifications. Generating proofs involves using the interface to the theorem-prover and its tactics (heuristics).

3.2 The Graphics Interface – An Example

The Ulysses graphical design environment allows the designer to select an icon representing a component, (using a mouse-driven "point-and-click" scheme), and cause Ulysses to open up the component so that its internal structure can be seen. The components can themselves be made up of lower-level components. The lowest level may either be left pending or contain a link to a textual process specification.

We describe the way the graphics system works by using a simple example. We will model some aspects of a secure distributed operating system (SDOS). (This example is a simplification of the design described in [V+88].) A system is distributed if it is composed of a network of computers with no shared memory. An operating system is distributed if it can service requests so that the location of the resources used to handle those requests is transparent to the user. In this example, we will model how messages can flow securely through such a system. Since it is distributed, requests from a user may have to be serviced by a different host than the one the request was made on. Hence, not only must a message be routed to the right location, but sometimes the location may have to be found first. We will call the task which actually does the routing of messages the 'message switch' and the task which makes the determination of a host to handle the message the 'locator'.

We first need to construct an overview of the system consisting of a collection of interconnected hosts and users (figure 1). This might be done by the following sequence of steps. First we create and connect icons representing a particular host (called an SDOS node) and a set of users. Now we create a virtual component to represent that any arbitrary set of interconnections between nodes is to occur and then connect it to the SDOS-users icons.

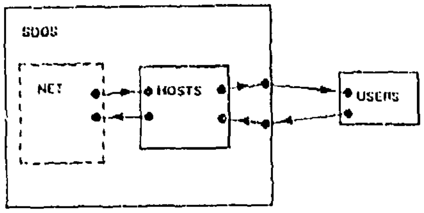


Figure 1: Multiple Hosts with Multiple Users

We can then provide a more detailed description of each SDOS node (figure 2). We use the refinement operation to open up the SDOS node and proceed to add more detail inside of it. In this case we have decided to break up the functions of SDOS into four categories. TIP's are the processes which handle communications with the users, and the NET is the process which handles communication with the actual network protocol. The Kernel handles the most essential operations of the system. The other kinds of processes are divided into two boxes representing other managers and processes. (Note that we must explicitly perform an operation which connects the SDOS users to the TIP processes.) In

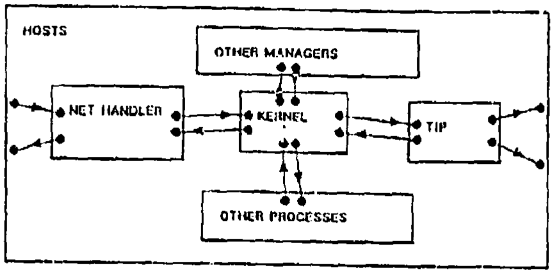


Figure 2: A particular host

this presentation we only care about the system operations which relate to message handling, and these are contained within the Kernel. We may consider the other process icons as place holders for possible future refinement of the model.

We can further refine the picture by describing the structure of the Kernel (figure 3). In particular the Kernel contains the message switch and other management tasks. One of these management tasks is the locator. Finally we connect the message switch icon to the other processes outside of the Kernel.

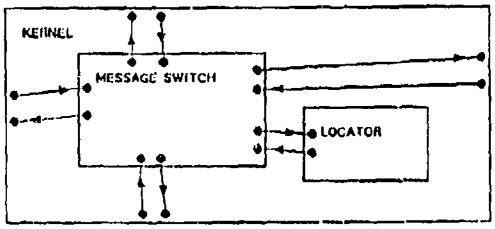


Figure 3: The Kernel

The graphics system not only constructs and displays the model, but it also constructs the security theory of the example. It uses assumptions or theorems about the underlying components to infer what is true about the systems which contain them. It also uses information about the components to enforce the requirement that only the appropriate connections are made. In the example, the Ulysses system can infer that the entire model is flow-secure once it has been verified that each of the pieces is secure.

3.3 Textual Specification Interface

Atomic processes are ones that are not subdivided further in the graphical representation. The graphics system generates conditions implying that all atomic processes and larger subsystems have been legally hooked-up; these conditions are usually trivial and can be discharged automatically by the theorem-prover. If all the connections between components are legal, then by composability we can verify that the composite system is flow secure by establishing that each atomic component is flow secure. For some simple cases, the proof that a component is flow secure can be done automatically; in general, though, it is necessary to examine or refine the component using the textual specification interface.

The security of a component depends on its functionality. The textual interface allows us to describe the functionality of a component by giving a state machine representation of its behavior. This representation consists of the definitions of the data types involved in the internal state parameters of the component and in the messages that the component uses to communicate, together with axioms describing the possible state transitions of the component and the security levels associated with messages and internal parameter information. From the state machine representation of a component, the statement of flow security for that component can be generated automatically.

The state-machine model, the theory of security, and tactics for proof of security for particular processes, are all connected via the textual interface to the logical formalism described in section 4. Flow security, as well as other properties of components, can thus be verified by reasoning in the underlying logic.

3.4 Natural Language

The text generation component of Ulysses is intended to serve two purposes: First, to provide annotations and comments to aid the designer during the design process; second, to produce an overview of the system, including its security characteristsics, once the design has been completed. As practical experience with the design process is still limited, efforts have concentrated on the second application.

The design of a system in Ulysses is determined graphically and (in the case of atomic components) by the textual formal specification. Typically, such designs would be accompanied by manually written annotations. Annotations complement diagrams and formal specifications by giving an informal rationale behind the design and its structure. Annotations summarize the functionality of the design components and explain them by appealing to concepts shared by the designer and the reader of the annotations. Such annotations become indispensable in the context of a secure design; these usually involve some compromise between functional requirements and security considerations that need explanation or justification.

The text generation component of Ulysses is a pioneering attempt to generate annotations automatically. During the first stage of our work, the emphasis is on producing texts that describe the security features of the system with less attention paid to functionality.

The information about how security is enforced in the system is derivable from the history of the security proof for the component. Thus a user who is not familiar with a given design would have to consult three different sources – graphical design, formal specification, security proof – and synthesize an understanding of the system himself. It is this job of synthesis that the natural language generation component performs.

The area of text generation of system documentation has not yet been studied by either linguists or computer scientists. However, highly promising and effective systems exist for other domains (for example McKeown's interface to a naval data base, [McK85]). Such work has allowed us to build on existing general techniques and concentrate on specific problems arising in our domain of annotations of secure designs. At present, the generation component of Ulysses produces multi-paragraph texts about systems such as the Secure Distributed Operating System (SDOS) [V+88].

The text generation requires computation at three levels: text planning, sentence planning and sentence generation, described below.

Text planning assembles a series of conceptual representations that determine the contents and organization of the text. Formally, the approach used is inspired by that of McKeown: schemata encode recurring textual patterns and access the available knowledge. However, there is no homogenous knowledge representation in Ulysses that the text generator can use. Instead, it uses any information that is available such as:

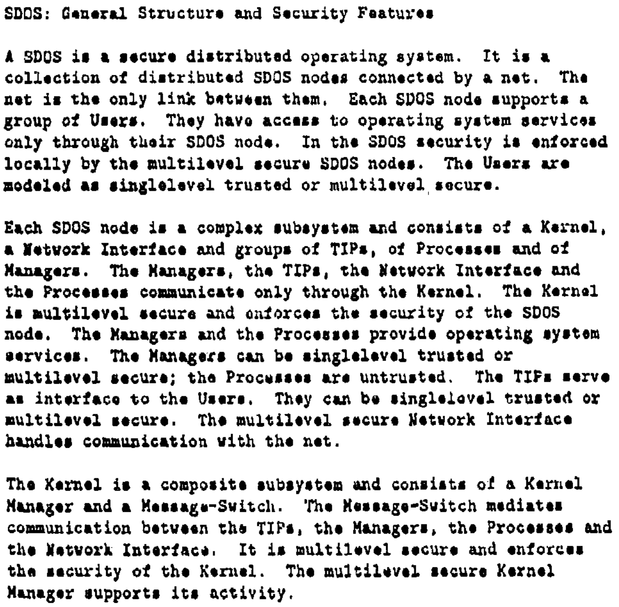
- the decomposition of the system into communicating components, as defined by the user through the graphical interface
- the security characteristics of individual components as they are determined during the proof, and the proof strategies used
- domain-specific knowledge about different types of systems (operating systems, gateways, LANs, etc.)
- certain information the user has entered after being prompted by Ulysses

However, this information about the system is not yet in a format that could be accessed by a general text planner; instead, the available information needs knowledge-based interpretation in order to serve as the basis for informative and meaningful texts. This is particularly true of the description of the system's security features. Certain typical security strategies need to be inferred from the structure of the system and the level of security of the components. For example, component which serve as mediators of communication between other components must figure more prominently in the security analysis. The knowledge needed for interpretation is encoded directly in the schemata, which makes text planning efficient but restricts it to the domain of secure system design.

Sentence planning takes the sequence of conceptual representations and transforms it into a sequence of sentence representations. The transformation involves message combination (determining sentence boundaries), syntactic decisions (determining sentence structure) and lexicalization (choosing English words for the concepts).

Sentence generation produces an English sentence. The generation component is based on Meaning-Text Theory [Mel81]. It defines a series of transformations from the sentence representation (the deep-syntactic representation) to the surface string, thus localizing linguistic decisions at particular levels.

Figure 4 shows an annotation of SDOS generated by the system.



```
SDOS: General Structure and Security Features

A SDOS is a secure distributed operating system.  It is a
collection of distributed SDOS nodes connected by a net.  The
net is the only link between them.  Each SDOS node supports a
group of Users.  They have access to operating system services
only through their SDOS node.  In the SDOS security is enforced
locally by the multilevel secure SDOS nodes.  The Users are
modeled as singlelevel trusted or multilevel secure.

Each SDOS node is a complex subsystem and consists of a Kernel,
a Network Interface and groups of TIPs, of Processes and of
Managers.  The Managers, the TIPs, the Network Interface and
the Processes communicate only through the Kernel.  The Kernel
is multilevel secure and enforces the security of the SDOS
node.  The Managers and the Processes provide operating system
services.  The Managers can be singlelevel trusted or
multilevel secure; the Processes are untrusted.  The TIPs serve
as interface to the Users.  They can be singlelevel trusted or
multilevel secure.  The multilevel secure Network Interface
handles communication with the net.

The Kernel is a composite subsystem and consists of a Kernel
Manager and a Message-Switch.  The Message-Switch mediates
communication between the TIPs, the Managers, the Processes and
the Network Interface.  It is multilevel secure and enforces
the security of the Kernel.  The multilevel secure Kernel
Manager supports its activity.
```

Figure 4: The SDOS Text

3.5 Library of Models

The function of the library is to provide and organize information useful in the design and verification of systems. The library contains three major kinds of information:

1. *System descriptions.* Both the specifications of atomic components and interconnections of complex system designs built from the components are stored in the library. The status of what has been proven about the security of the components and of the systems is also maintained.
2. *Security Theory and other Mathematical facts.* The library maintains a store of information describing the security theory and other relevant mathematical facts. Also included are definitions of the tactics and theories used by the theorem prover.
3. *Graphical presentation.* The system can record various facts about the graphical layout of designs. This information is in addition to the design information of the theories.

The library will contain a variety of designs of generic, commonly used software systems ranging from very small components, such as buffers or queues, to complex ones, such as a Database Management System. The list of secure designs includes some generic trusted processes (multilevel buffers, secure separators, secure schedulers), a Local Area Network (several designs for different medium access control), a Multinet Gateway, a Database Management System, a Distributed Operating System, and several others. The user will be able to study library designs and their associated documentation and to use them in his own designs. The user can either use library designs as "building blocks" and rely on proofs done by Ulysses' developers, or change library designs according to the requirements of his or her system.

Ulysses will support browsing through the library for components or other theories that meet given criteria.

The theory of security currently used in Ulysses is merely a default. It is one particular set of theorems about the hook-up of components into systems. Other theories of security are possible. For example, a precise theory of components that are "almost secure" might be definable, and facts about their hook-up proved. Users may add such new theories to the library.

4 The Mathematical Component

The goal of the Ulysses project is to understand security at the design level and to automate that understanding in a logically coherent formal setting. We believe that a general mathematical theorem-proving environment based on type theory is a good foundation for this task.

We will explain what this assertion means and what our reasons for believing it are in the following way. We begin with a general sketch of the mathematical component's design. Then we discuss its antecedents (subsection 4.1), explain how mathematics will be expressed within this framework (subsection 4.2), and say why this setting is especially suited to work on security modeling (subsection 4.3). The remainder of the section will be devoted to some brief remarks about technical aspects of the design—the underlying logic (subsection 4.4), the core component of the logic's implementation (subsection 4.5), and theory management (subsection 4.6).

Within the mathematical component, the word "theory" has a technical sense. But this technical usage reflects accurately many important facets of the term's ordinary meaning. Intuitively, a theory is a collection of related facts and notations for expressing them. The facts collected in theories may be related in a number of ways — they may be stated in a common language, may depend on common axioms, and may support one another in various fashions in the sequential development of a theory.

Facts and notation may be incorporated into a theory by relying on a previously developed theory, by introducing a notational abbreviation, by introducing a definition, by postulating a new axiom, by stating and proving a theorem, and by introducing all of the facts and notations from some other theory after showing that all of its assumptions are satisfied in the theory being developed.

The mathematical component provides a rigorous setting within which all of these features of our informal conception of theories are represented precisely and usefully.

Theories have two main parts: a precis, which identifies the linguistic dependencies and states the postulates of a theory, and a body, which contains the development of new facts and notation introduced by the theory. In the context of a library of theories, the precis determines the initial environment of a theory. That is, it determines the collection of facts and notation imported from other theories directly. In addition, it contains the axioms and new symbols characteristic of the theory under development. The body extends the environment of facts, assumptions and notation defined by the precis. Essentially, the meaning of a theory is the incremental extension of the initial environment which the body provides.

Ulysses is based on a version of type theory capable of expressing both classical and constructive mathematics. Within the mathematical component of the system, the relationship between theories of security and the semantics of their specification languages is expressed rigorously, for theories which permit it. In particular, this has been done for flow security. This formal foundation will reduce the consistency question for the logic and the correctness question for the theorem prover, to the consistency and correct implementation of the six rules of the underlying type theory.

4.1 History

The idea of using type theory for the expression of mathematics in a theorem proving environment was first championed by de Bruijn in the AUTOMATH system [Bru80]. The aim of AUTOMATH was to verify mathematical "books". The system was very batch oriented, originally reading the "book" as a deck of punched cards.

In the early eighties, Constable and his students at Cornell University began another major project to express mathematics in type theory called the "prl" project [C+86]. ("prl" is short for "Proof Refinement Logic" and is pronounced "pearl".) Their work was inspired by the AUTOMATH project and by the work of the logician Per Martin-Löf [Mar82]. Their aim was not just to provide an environment for the verification of mathematics, but to assist users in developing mathematical theories interactively.

The key idea that made this possible was the concept of a refinement style proof editor [Bat79]. Such a proof editor allows the user to state a theorem and then construct a proof interactively by manipulating subderivations displayed on the screen. This can be done either by directly invoking the primitive rules of the system or by invoking *tactics* which direct the machine to do these micro-inferences automatically.

The tactic mechanism has proved to be a vital feature of the systems developed during the prl project, and it plays an equally important role in Ulysses. The meta-language of the prl systems is **ML** [Mil78], which was developed to provide the meta-language of **LCF** [GMW79]. The same is true of our system. Tactics are segments of **ML** code which extend the primitive inferential apparatus of the logics on which these systems are based. The systems' proof checking mechanisms insure that these extensions are sound.

In principle, it would be possible to write a general theorem proving program and rely on it as one's sole tactic. But experience with the prl systems has shown that it is more productive to design tactics for specific circumstances encountered in developing mathematical theories. The tailoring of tactics to the special requirements of security modeling is one of the most important features of the mathematical component of Ulysses.

Although the design of the mathematical component draws heavily from the experience of the prl project, there are two primary differences. The first is the choice of the underlying logical system and the second is the addition of a facility for modular theories. These two issues are related: the logical base we have chosen supports modularity much more easily than does the type theory on which prl is founded [Sel88].

In addition to these differences, there are several design differences that are expected to give Ulysses significantly better performance than the Nuprl system (the new and current version of prl) and allow the mathematical component to be integrated with other system components. One of these is the use of graph reduction to handle necessary computations in the underlying lambda calculus (subsectio.. 4.5). Another is the treatme:.t of definitions. We discuss this briefly now and return to the topic in subsection 4.6.

For Ulysses to be used successfully, the mathematics expressed in theories must be visually similar to mathematics as it is ordinarily written, and, when the mathematical component is used as part of a domain specific system, domain specific information must be presented to the user in a recognizable form. Consequently, there must be a very powerful mechanism for extending the not·.tion of the system. This feature is called the definition facility, and the concerns expressed in the first sentence of this paragraph played a major role in shaping its design.

4.2 Expressing Mathematics and Security Theories

Once the foundation of the system is laid, the next step is to express something in it. That requires developing familiar mathematical concepts, such as elementary arithmetic, simple set theory, a theory of sequences, and some simple computational models within the system. These theories will be included in the system library. The theory manager enforces a presupposition structure specifying which other theories must be included in the current environment if a given theory is to be used.

Once all of these basic pieces of mathematics are in place, the theory of security is formulated and theories describing example systems are synthesized within Ulysses. This collection of theories forms an experimental testbed for the automation of security reasoning. The automation will be provided by combining the ML mechanism that provides assistance in proofs with supplementary code written in Common Lisp. Besides supplying assistance in proofs, the system will provide support for formulating appropriate security theorems and for integrating previously defined structures into the current environment.

We expect that once we have developed a library of verified, composably secure components, most Ulysses users will be able to view the system as a fully automated theorem prover, and not as a proof development environment. The sophisticated user will have the opportunity, however, to invoke any mathematical facts that can be developed within the system when arguing formally that a system, or system component, is secure. And the system designers will have confidence that all of the components they have supplied to the users are, in fact, secure according to the formal definition of security axiomatized within the system.

Besides providing a formal framework in which to pursue the current goals of the Ulysses project, the system design allows for the development of extended versions which will incorporate a code verification facility. We believe that, ultimately, this formal foundation will make Ulysses a trustworthy robust system development environment. Of course, whether we are right can be determined only by developing such an environment and bringing the result before the bar of experience for judgment.

4.3 Advantages for security modeling

A number of features of the mathematical component make it an especially useful tool for dealing with security modeling. Chief among these is the tactic mechanism. We are creating a library of tactics tailored to the demands of proving security results about the sorts of models most commonly dealt with. This library will greatly enhance the usefulness of Ulysses.

Quite often, it is possible to restrict attention to a class of security models considerably smaller and simpler than the class of all such models, and, for such models, it is fairly simple to prove what needs to be shown about the processes involved. In such cases, we are going to automate the proof process almost completely by writing appropriate tactics.

For example, many processes accept an input, emit some outputs, and then process the next input. For these kinds of processes, one can use a security tactic which converts the goal of proving flow security into simpler kinds of conditions. It suffices to show that (1) for any given input there will be only finitely many outputs, (2) the security level of an output is always greater than or equal to the level of the input, (3) the content of the output is based only on the input and information carried by parameters at or below the level of the input, and (4) high level inputs do not change the low level characteristics of these parameters. See [Ros88] for more details.

Another important characteristic of the mathematical component is its expressive power. With this system it is relatively easy to build new abstract data types. This makes descriptions of models easier to understand and allows for more accurate descriptions. Descriptive power also enables the user to formulate properties of a process more easily. For example, it is easy to assert within the language that a process halts. Also, the security theory is built directly into the system. This insures that proving that a system has the properties specified by the theory really does show it is secure, in the sense specified by the theory.

A third useful aspect of the proof development environment is that it allows security results to be proven about generic descriptions and not just particular instances (see subsection 4.4). This means that most adjustments to a model will require little if any work in reconstructing proofs of security for it. Often, the modeling environment will do all the necessary reconstruction, without intervention from the user — if you want to add new components to a model, demonstrating security may involve no direct effort on your part.

4.4 The logic

The logical system underlying Ulysses is the theory of constructions of Coquand and Huet [CH84], [CH86], [Hue87], [Sel88], [Pot87]. Besides providing a quite expressive logical system, the theory of constructions also includes a powerful model of computation. The model of computation is not important for the current aims of the Ulysses project, but we expect to rely on it in future extensions of the system.

There are only two built-in types in the theory, but there is a facility for reasoning in the context of type assumptions that describe both the operations and axioms of mathematical theories. This mechanism supports an abstract style of theory development. Suppose group theory has been developed in a context that ι.ecifies the operations and axioms characteristic of groups. It will be possible to instantiate this abstract theory on a particular structure by specifying which operations of the structure are to play the role of the group operations and proving that the group axioms are satisfied by the specified operations.

The theory of constructions contains a type which formally represents the type of all propositions. In turn, this type is contained in a type which satisfies very strong closure conditions. Consequently, the theory contains full, higher order logic — having specified a ground type by means of appropriate assumptions, one can refer to properties of the ground objects, properties of such properties, functions from properties of the latter sort to those of the former, and so on, without limit.

The logic provided by the theory of constructions is constructive. This is noteworthy for two reasons. First, it is the key reason why the theory includes a model of computation. Second, it means that the logic is *richer* than classical, two valued logic — constructivity is an enhancement, *not* a restriction.

The theory of constructions provides a model of computation in which, besides taking types as arguments, functions can return types as values. Furthermore, the type of the value returned by a function can depend on the argument, and not merely on the argument's type. Consequently, the theory is of great interest from the point of view of research on polymorphism in programming languages.

This model of computation is extremely powerful. Formal measures of its power, relying on the results of [Gir71,Gir72], show that it is strong enough to represent any computable (total) number theoretic function considered in ordinary mathematical practice, and this is a lower bound. An informative upper bound on the strength of the model of computation built into the theory of constructions has not yet been established.

As far as correctness is concerned, Coquand has shown that the logic embodied in the theory of constructions is consistent and that all computations in the model of computation terminate. Taken together with the features of the theory discussed above, this explains why the theory of constructions is interesting, both from a purely logical standpoint and from the point of view of theoretical computer science. It also led to the decision to base the mathematical component of Ulysses on the theory of constructions.

4.5 The Primitive Inference Engine

The core of the mathematical component is the Primitive Inference Engine (PIE). The PIE includes a proof checker for the theory of constructions and rudimentary (but extensible) tools for proof development. We are proceeding on the basis of a formulation of the theory of constructions which is especially suited to the character of the proof development tools and also allows for efficient proof checking [Pot88a].

The central computational problems involved in implementing this formal system have to do with handling substitution, reduction and conversion of terms. We have reduced these problems to their essence by representing terms of the theory of constructions in the simpler framework provided by the type-free lambda calculus and have done the same thing for the relations of reduction and conversion [Pot88a].

Recasting the computational problems in this way is, of course, only a beginning. An efficient implementation of the type-free lambda calculus will be produced by using graph reduction [Wad71, Tur79].

4.6 Theory management

As was remarked in subsection 4.1, in many important respects the mathematical component of Ulysses is modeled after Nuprl. We end this section by commenting briefly on two important differences.

It is reasonable to say that proofs of hook-up security require a small mathematical foundation, if "small" is understood in the sense of the term customary among mathematicians. But actually providing such a foundation requires building a complicated structure in the machine. Definitions are common and vital components of this structure, so it is important to have an efficient way of handling them. Our approach to this problem is quite different from the one taken by Nuprl, and we think it will turn out to be superior [Pot88b].

It is also certain that we must develop theories modularly, if Ulysses is to be practically useful. It should be clear from the discussion of this section that this concern is handled adequately in the system we are constructing. In contrast with this, Nuprl provides *no* mechanisms for modular theory development, and certain features of the logical system on which Nuprl is based make the project of introducing such mechanisms problematic. This reinforces our conviction that the theory of constructions, which directly supports modularity in developing theories, is a good choice for the logical basis for security modeling in Ulysses.

5 Implementation

A U ysses system that implements the functions described in previous sections is now being built at ORA. By October 1st, we expect to have a functional prototype. The prototype version will run under Symbolics Genera 7.1. However, we have placed great emphasis on portability even at early stages in the development. Most of the source code is written in Common Lisp or one of its object-oriented extensions (Symbolics Common Lisp or CLOS); the only exceptions are the tactics, which are written in a version of ML that is itself implemented in Common Lisp. Symbolics Common Lisp will be converted into CLOS and *v.v.* with a set of translation macros. As a result, only a minimum of effort should be involved in re-targeting Ulysses to any other architecture that supports Common Lisp; most of this effort will relate to the graphical interface.

6 Conclusion

The design of Ulysses incorporates ideas and techniques from a diverse collection of sources, including those in computer security, systems design, computational logic, and computational linguistics in order to create a modeling environment with both rigor in its theoretical foundations and flexibility in its use. Because of the nature of the theorem prover and the overall design of the system, it has the potential to significantly reduce the time and effort needed in constructing secure models.

References

[Bat79] Joseph L. Bates. *A Logic for Correct Program Development.* PhD thesis, Cornell University, 1979.

[BL76] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, Revision 2, MITRE Corp., Bedford MA, March 1976.

[Bru80] N. G. de Bruijn. A survey of the project automath. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, New York, 1980.

[C+86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.

[CH84] Thierry Coquand and Gérard Huet. A theory of constructions. Presented at the International Symposium on Semantics of Data Types, Sophia-Antipolis, June 1984.

[CH86] Thierry Coquand and Gérard Huet. Constructions: A higher order proof system for mechanizing mathematics. In *EUROCAL85*, volume 203 of *Lecture Notes in Computer Science*, pages 151–184, Berlin, 1986. Springer-Verlag.

[Gir71] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971. North-Holland.

[Gir72] Jean-Yves Girard. *Interprétation Fonctionnelle et Élimination des Coupures de L'Arithmétique d'Orde Supérieur.* PhD thesis, University of Paris VII, 1972.

[GM82] J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the Symposium on Security and Privacy.* IEEE, 1982.

[GMW79] M. J. Gordon, J. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Springer Verlag, 1979. Lecture Notes in Computer Science 78.

[Hue87] Gérard Huet. A uniform approach to type theory. Notes distributed at the University of Texas Institute on Logical Foundations of Functional Programming., June 1987.

[Mar82] Per Martin-Löf. Constructive mathematics and computer science. In L. J. Cohen, J. Los, H. Pfeiffer, and K.-P. Podewski, editors, *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North-Holland Publishing Company, Amsterdam, 1982.

[McC87] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proceedings of the Symposium on Security and Privacy*, pages 161–166. IEEE, 1987.

[McC88a] D. McCullough. Noninterference and the composability of security properties. In *Proceedings of the Symposium on Security and Privacy.* IEEE, 1988.

[McC88b] D. McCullough. The theory of security. Technical report, RADC, 1988.

[McK85] Kathleen McKeown. *Text Generation.* Cambridge University Press, Cambridge, 1985.

[Mel81] Igor Mel'čuk. Meaning-text models. *Annual Review of Anthropology*, 10:27–62, 1981.

[MH85] Mark Moriconi and Dwight F. Hare. The PegaSys system: Three papers. Technical Report 145, SRI Computer Science Lab, Menlo Park, California, September 1985.

[Mil78] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Science*, 17:348–375, 1978.

[Pot87] Garrel Pottinger. Strong normalization for terms of the theory of constructions. Technical Report TR 11-7, Odyssey Research Associates, February 1987.

[Pot88a] Garrel Pottinger. Ulysses: Logical and computational foundations of the primitive inference engine. Technical Report TR 11-8, Odyssey Research Associates, January 1988.

[Pot88b] Garrel Pottinger. Ulysses: Logical foundations of the definition facility. Technical Report TR 11-9, Odyssey Research Associates, January 1988.

[Ros88] D. Rosenthal. An approach to increasing the automation of the verification of security. paper submitted to the Computer Security Foundations Workshop, June 1988.

[Sel88] Jonathan P. Seldin. MATHESIS: The mathematical foundation for ULYSSES. Technical report, RADC, 1988.

[Sta86] Staff. *The Official Handbook of MASCOT.* Systems Designers, 1986. Version 3.1.

[Sut86] David Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, September 1986.

[Tur79] David Turner. A new implementation technique for applicative languages. *Software-practice and Experience*, 9:31–49, 1979.

[V+88] S. Vinter et al. The secure distributed operating system project. Technical Report 6144, BBN Labs and Odyssey Research Associates, February 1988.

[Wad71] C. P. Wadsworth. *Semantics and Pragmatics of the Lambda-calculus.* PhD thesis, Oxford University, 1971.

[WL87] D.G. Weber and Bob Lubarsky. The SDOS project – verifying hook-up security. In *Third Aerospace Computer Security Conference*, pages 7–15. AIAA/ASIS/IEEE, 1987.

IMPLEMENTING THE CLARK/WILSON INTEGRITY POLICY USING CURRENT TECHNOLOGY

W.R.Shockley
Gemini Computers, Inc.
P.O. Box 222417, Carmel, CA 93922
(408)-373-8500

Abstract

In this paper, the integrity policy introduced by Clark and Wilson is taken as a set of valid requirements suitable for commercial and other data processing requirements that must be enforced with a high level of assurance. A methodology for converting a policy expressed in terms of the Clark/Wilson notation into a corresponding mandatory policy expressed in terms of a lattice of access classes, together with an appropriate supporting policies for identification and authentication is stated. The existence of such a methodology implies that the Clark/Wilson integrity requirements can be met by existing, appropriately-configured Trusted Computing Bases.

1. Introduction

The integrity policy presented by D.D.Clarke and D.R.Wilson in [1] has received a relatively high degree of attention as an accurate representation of what the business and commercial data processing community means by the term **integrity** with respect to data processing applications oriented toward commercial applications, just as the Bell and LaPadula formal security policy model [2] has served, in the past, as the technical basis for trusted computer systems enforcing a mandatory access control policy oriented toward military and government applications processing information classified under federal regulation.

Clark and Wilson state, as their two major conclusions, that "a lattice model is not sufficient to characterize integrity policies", and that "distinct mechanisms are needed to control disclosure and to provide integrity". The implication of these conclusions, if true, is that the Trusted Computing Base technology described in [3] are not applicable to the evaluation of systems designed to enforce the Clark/Wilson integrity policy.

In this paper, issue is taken with both of these conclusions. The argument has the following outline: starting with an arbitrary Clark/Wilson policy, an equivalent access control policy based upon a lattice of sensitivity labels is derived. Together with appropriate supporting controls for a security officer interface and identity-based subject activation, a policy interpretation compliant with the requirements of [3] can therefore be formulated. A TCB enforcing such a policy would satisfy the Clark/Wilson policy as well as the Criteria. As the originally chosen integrity policy was arbitrarily chosen from the family of Clark/Wilson policies, it follows that any Clark/Wilson policy can be enforced by an appropriately-configured TCB meeting the criteria stated in [3].

As the transformation is constructive, it shows how an arbitrary policy, expressed in terms of the Clark/Wilson model, can be reformulated as an equivalent combination of access controls based upon a lattice of access classes together with a discretionary component controlling access to the execution of transactions to the granularity of an individual.

The implication of this construction is that one could envision a TCB, designed to be evaluated under the criteria of the [3], that is also well-suited to the enforcement of controls expressed in terms of the Clark/Wilson model. In fact, it can further be observed that such a TCB is already available. In a later section, I will discuss how an existing TCB (Gemini Computer's GEMSOS) can be tailored to support a Clark/Wilson model.

1.1 Relationship to Previous Work

Several papers earlier than [1] are important in the study of the application of computer security technology to integrity issues. I have drawn freely from them in the work presented below. Biba, in [4] presents a "mandatory integrity policy" that is the mathematical dual of a mandatory secrecy policy based on a lattice of labels. Such a policy is often called a **Biba integrity policy.** Lipner, in [5] constructs a commercially-oriented policy from a combination of secrecy and mandatory integrity levels and categories. Shirley and Schell, in [6] introduce the notion of **program integrity,** a policy that is important when subjects that are "trusted with respect to integrity" exist in a system. They demonstrate, in addition, that a ring-based protection hierarchy, such as that found in Multics or GEMSOS, can be interpreted as a hierarchical system of subjects trusted (to various degrees) with respect to integrity, upon which the program integrity policy is enforced.

Boebert and Kain, in [7] introduce a system of **trusted pipelines,** enforceable by the Honeywell LOCK (formerly, SAT) TCB. They demonstrate (correctly) that a hierarchy of Biba integrity levels alone is insufficient to enforce a trusted pipeline. The generalization that a full lattice including Biba integrity categories is insufficient as well is not addressed by Boebert and Kain. This paper is an important predecessor to [1]: indeed, it could be fairly stated that the Clark/Wilson policy is an elaboration of the trusted pipeline idea.

In addition to these, the Clark/Wilson presentation has induced a number of additional papers, generally of the form "system X can enforce the Clark/Wilson policy." A recent paper by Lee, [8] presents a construction identical in many respects to the system presented here. The primary deficiency in Lee's paper is that one of the important Clark/Wilson constraints, requiring that controls be enforced at the granularity of a user/object/program **triple** appears to be inadequately addressed. Lee's work and mine are independent: drafts of both papers were presented concurrently as position papers at the invitational Workshop on Integrity Policies for Commercial Information Systems held at Bentley College, Waltham, Mass. The notion of what Lee calls a **partially trusted subject** upon which both of our systems depend is original with neither Lee nor myself: it is discussed by Schell et al. in [9] and by Bell [10] as a part of this "updated" version of the Bell and LaPadula model.

Also noteworthy is a recent paper by Karger [11] that provides a capability-oriented perspective on the Clark/Wilson requirements and raises a number of interesting design and implementation issues, as well as featuring a review of background papers and reports more extensive than that given here.

1.2 Acknowledgements

Preparation of this paper was supported under contract F30602-85-C-0243 by the U.S. Air Force, Rome Air Development Center as part of the SeaView project to design a Class A1 secure relational DBMS. I am also grateful for the numerous comments and suggestions that I received from various individuals and the referees, which have allowed the content of this paper to be simplified and its length reduced. Any remaining flaws are, of course, my own.

1.3 Overview

This paper provides an overview of the basic construction that I have defined for translating an arbitrary abstract system meeting the Clark/Wilson requirements into an equivalent system based upon a label-based access control policy with integrity and disclosure categories and "partially trusted" subjects. Rather than presenting this transformation in abstract mathematical terms, I have chosen in this paper to provide a more understandable overview, together with a concrete example. For those who may be interested, the more abstract (and precise) presentation is available as a Technical Report in [12].

A short section after the technical overview addresses the ability to implement an instance of the transformation using a currently available TCB, the GEmini Multi-Processing Secure Operating System (GEMSOS).

2. Technical Summary

In this section, I will first review selected technical capabilities provided by a typical commercially-available TCB (GEMSOS) that will be important in constructing the transformation from a Clark/Wilson set of access control requirements to an equivalent set of policy requirements, stated in terms of discretionary, non-discretionary, and application policy controls. (I have chosen to use the term "non-discretionary access controls" in place of the usual "mandatory access controls", as originally defined by Salzer and Schroeder [13], in order to avoid Clark and Wilson's complaint that the use of the term "mandatory" can be confusing to those unfamiliar with the jargon of the Trusted Computing Base technical community.) It should similarly be understood that by identifying certain of the controls described in the system below as "discretionary", I mean simply that the control is based on an individual **user identifier** (as opposed to an access class or clearance) and represents an authorization for that individual user to perform some security-relevant action (represented by access to a directly or interpretively accessible object.)

In order to illustrate the system concretely, I will develop a "toy" system as the summary proceeds. We will imagine a system comprised of four data types, A, B, C, and D, with each data type comprised of an indefinite number of distinct data objects. (For example, data type A might include data objects A1, A2, A3, etc.) We will suppose that there are defined three transactions that transform data from one type to another: AtoB, BtoC, and BtoD. We will also suppose that there is defined a verification procedure ValidateAB that determines whether the objects of type A and B are mutually consistent (without modifying them). (These transactions are simply executable programs.) The example will be extended as needed below.

2.1 Non-Discretionary Mechanisms

The purpose of this section is to review the mechanisms assumed available for the enforcement of the non-discretionary components of the policy and their application in terms of a strongly-typed, transaction-oriented system such as that described by Clark and Wilson. It is assumed that the system is comprised of objects (passive information repositories) and subjects (active entities that can read and/or write objects.) It is important to note that we distinguish between a program (which is an object) and a **subject** (which is typically a program in execution, acting on behalf of a particular user). The abstraction of a subject is implemented by the security kernel. The distinction between a program and a subject is important because the single label on a program represents its sensitivity as a data repository, while the pair of labels (explained later) on a subject, which are related only incidentally to the label on the program it is executing, represents the accesses allowed to the subject. It is similarly important to distinguish between the notion of a subject and of a user. A subject is an entity internal to the computer system, which executes on behalf of a user (who is external to the computer system). Again, the distinction will be important because a given subject may well have a pair of labels only incidentally related to the user's clearance.

The set of all possible access classes forms a lattice -- mathematically, a set of labels with a dominance relation that partially orders them, such that least upper and greatest lower bounds are uniquely defined. (It may be observed that when integrity and/or disclosure categories exist, it is not necessary for all possible combinations of the categories to be defined in the set of labels to have a lattice -- a lattice that includes all possible combinations is called a distributive lattice [14]. Of importance in this paper is that the lattice is built from two essentially independent components: every label represents a sensitivity with respect to disclosure, and an independent component representing a sensitivity with respect to modification. Because these components are mathematically independent, we are able (in effect) to give each object (including programs) independent labels with regard to its disclosure and integrity sensitivities, give users independent clearances with respect to disclosure and integrity, and give subjects (programs in execution) independent authorizations with respect to read and write access.

Both the disclosure and Biba integrity components of a label may generally contain hierarchical levels and non-hierarchical categories. As it turns out, non-hierarchical categories alone are sufficient to implement the desired non-discretionary component of a Clark/Wilson policy. The following notation will be used to represent an arbitrary set of integrity and disclosure categories:

[a, b, c, . . .]{x, y, z . . .}

is that unique access class composed of integrity categories a, b, c, and so on, together with secrecy categories x, y, z, and so on. Thus, square brackets are used for a set of integrity categories, and curly braces for a set of disclosure categories. For arbitrary access classes, these sets may overlap.

For access classes composed of sets of integrity and disclosure categories alone, the dominance relation is simplified to the following: access class A dominates access class B if, and only if, the disclosure categories of A are a superset (proper or improper) of the disclosure categories of B, and the integrity categories of A are a subset (proper or improper) of the integrity categories of B.

Intuitively, a system of strongly typed objects may be constructed as follows: each data type is represented by an integrity category reserved for that type, (used to limit the subjects that will be allowed to modify objects of that type) and by a disclosure category reserved for that type (used to limit the subjects that will be allowed to observe objects of that type).

For our example system, the access class labels reserved for objects of type A, B, C, and D are [a]{a}, [b]{b}, [c]{c}, and [d]{d}, respectively.

Program objects are given special treatment. Because we wish to control the ability to execute transactions to the granularity of a single certified transaction, each transaction object (program) will be assigned an individual data type of its own. In addition, we will indicate that a transaction is certified to operate upon objects of a particular data type by including the integrity category for that data type in the access class of the transaction object.

For our example, certified program AtoB is certified to operate (either by reading, writing, or both) on objects of types A and B. In addition to the unique transaction categories t1 reserved for it, we add the integrity categories for both A and B to the access class for the object containing this program: viz., [t1,a,b]{t1}. Similarly, BtoC would have access class [t2,b,c]{t2}, BtoD would have access class [t3,b,d]{t3}, and ValidateAB would have access class [t4,a,b]{t4}.

It may appear surprising that the program for a transaction is given an integrity category for a data type it only needs to read. However, a read, in a real computer system, is useful only if the read allows a copy to be made of the data value (e.g., as a return value in a subject's stack.) This copy must be protected as having the same integrity as the original: therefore, in order to work, the program (when executed) must be able to write information (in order to make copies) of any integrity category it is required to read. It follows that the program itself must also be certified to write information of this integrity.

Subjects are given two labels, called the write label and the read label, one of which (the write label) serves to prevent the subject from writing objects of an unauthorized type, and the other (the read label) from reading objects of an unauthorized type. The precise rules enforced are as follows: a subject will be allowed to write an object only if the write label of the subject is mathematically dominated by the label of the object, and will be allowed to read an object only if the read label of the subject mathematically dominates the label of the object.

The reader may justifiably find it difficult to apply these rules when both disclosure and integrity categories are in use, particularly as the mathematical definition of dominance is abstract. In more intuitive terms, a subject may only write objects that have all of the secrecy categories in the subjects read label (or more) -- no write down with respect to disclosure. A subject may also only write objects that have no more integrity categories than the subject's read label -- no write up in integrity. The subject similarly may not read up in secrecy or read down in integrity. The abstract mathematical definitions allow the security kernel to enforce all four of these constraints by encoding them in two subject labels and one object label.

In order to capture the notion of strongly-typed objects, it turns out that the appropriate format for the write label of a subject is the set of all integrity categories for the data types it is certified to read and/or write, while the form of the read label is the set of all disclosure categories for these data types, together with the disclosure category for the transaction to be executable. (For our system, a subject must usually be confined to execute a single transaction in order to successfully be created.)

For our example (without knowing anything about user clearances yet) we suppose that some subject must be created to execute transaction AtoB. The indicated write and read labels for such a subject would then be [a,b] for the write label, and {a,b,t1} for the read label.

It should be observed that relative to objects with an assigned type, the labels on the subject correctly and precisely constrain it to manipulate data objects of the desired type only. For example, an object of type D, with label [d]{d}, cannot be read, because [d]{d} is not dominated by the subject's read label {a,b,c,t1,t2}. Similarly, the object [d]{d} cannot be modified by this subject because it does not dominate the subject's write label, [a,b,c]. Finally, transaction t2 cannot be executed by this subject (for example), because it will not be allowed to read or execute an object with disclosure category {t2}.

An important observation is that a subject labeled as described is "partially trusted" in that it may be able to write objects of different access classes, and may be able to write objects of an access class not dominating the access class of some object it can read. Therefore, it is important that the subject be limited to execute only those transactions certified to perform the type conversions it might be able to make. The program integrity rule, however, guarantees that this will be the case. The program integrity rule requires that any program executed by a subject have an access class with integrity categories that include all of the integrity categories of the subject's write label. (It may have more). This rule has a relatively intuitive interpretation in the context of strong typing: program integrity guarantees (as enforced by the security kernel) that a subject may only execute transactions that have been certified to operate correctly against all of the data types for data objects the subject is allowed to access. That is, enforcement of program integrity by the kernel means globally that every transaction that is executed will be certified to be executable against any of the typed objects accessed -- exactly what is wanted for any strongly typed system, including Clark/Wilson.

Users (who are distinct from subjects) are given a clearance that reflects their authorization to manipulate data of a given type by placing both its disclosure and modification categories in the user's clearance. Furthermore, a user is given authorization to execute a particular transaction by placing its disclosure category in the user's clearance. It is sufficient that the TCB constrain the read and write labels of a subject, executed on behalf of an authenticated user, to have a write class that is some subset of the user's integrity categories, and a read class that is some subset of the user's disclosure categories. A subject obeying this constraint will either have no transaction executable (i.e., the attempt to create the subject by the TCB aborts) or will end up executing a single transaction, authorized to the user, against objects of data types authorized to the user. As discussed above, program integrity guarantees that such a transaction will also be one that has been certified to operate on objects of the given type.

It is worth noting that although a transaction might be certified to operate on a variety of types (e.g., A, B, and C), an individual user might be authorized only to operate on a subset of these types (e.g., A and B). In such a case, the user will not be able to create a subject executing the transaction against an object of the forbidden type, even though the transaction itself is certified to do so. Many existing systems based upon granting access to "canned transactions" are unable to limit the authorizations independently for different users of the same transaction. If a user has access to a transaction, the user "inherits" any authorizations the transaction may have. (The "setuid" feature provided by UNIX works this way, for example.)

In contrast, the system described here allows the authorizations of different users to be controlled independently of the certifications recorded for each transaction. Thus, the certifying official (for transactions) need only concentrate on determining what a transaction is trusted to do correctly in selecting a label for the transaction, while the security administrator controls access by individual users to particular transactions and object types without, (in theory), needing to consider the certification a transaction has gained. If an attempt is made to give a subject "too much authority" with respect to the actual certification recorded for a transaction, the TCB, enforcing program integrity, will abort the subject before it begins because the transaction will be unexecutable.

2.2 Discretionary Mechanisms

The system summarized above does not yet capture the complete intent of the Clark/Wilson requirements with respect to access control. It might be characterized as "strong typing," with users cleared to execute particular transactions and (independently) to access objects of particular types. Clark and Wilson ask, in particular, for controls on which transactions a user can execute against which objects: that is, permission to access object A and to execute transaction T, should not automatically imply permission to execute T against A, even if T is certified for A.

In particular, Clark and Wilson require that the TCB maintain (either implicitly or explicitly), a list of **relations** listing authorized combinations of users, transactions, and objects (or data types). It is important to see how the system described so far does not meet this requirement.

In more complex systems than our example, it becomes possible, if there are no additional controls, to implicitly clear a user for an undesired transaction in order to make some combination of desired transactions available. Suppose, for example, that there are four data types (A, B, C, D), and two certified transactions, both "query" transactions, each certified to operate correctly on each of the data types. (We might imagine, for instance, that A, B, C, and D are disjoint collections of personnel records for four different divisions, t1 is a transaction for observing salaries, and t2 a transaction for observing training qualifications.)

We wish to give a particular user the authority to observe salaries in database A only, and training qualifications in any of the databases. In the system so far described, there is no way to do this without granting too much authorization. In order to make transaction t1 executable against A for the user, we must add {t1,a}[a] to the user's clearance. In order to make transaction t2 executable against A, B, C, and D, we must add {t2,a,b,c,d}[a,b,c,d] to the user's clearance. The user's single composite clearance is now {t1,t2,a,b,c,d}[a,b,c,d] and nothing prevents the user from executing either transaction against any of the data objects.

It can be argued that by restructuring the transactions and repartitioning the data, the desired effect could be attained. While this is true, it is hardly convenient or practical to have to repackage either transactions or data types in reaction to the addition of new users with novel clearances. In this matter, I concur whole-heartedly with Clark and Wilson: the maintenance of a table of relations between permitted user/transaction/object combinations (however it may be stored physically) is a practical necessity.

The approach I endorse is to treat the table of triples as a special form of discretionary controls maintained within the TCB. When a request is made to the TCB to create a new subject, this table will be consulted to determine whether the new subject is permitted. Sufficient information to do this is encoded in the requested read and write labels for the subject (plus the user identity, maintained within the TCB) if only single-transaction subjects are allowed: the integrity category for the subject, plus any data types accessed, is included in the write label requested for the subject.

I have preferred to separate this control from the underlying non-discretionary controls for the following reasons:

- it would appear that the control is intrinsically discretionary in nature: it is based (in part) on the actual user identity (not a clearance). The notion of the non-discretionary component of the system is that users are cleared to execute transactions and access data objects on a long-term basis: this is refined by a discretionary control granting access to particular combinations of transactions and objects on a more volatile basis.

- it should not be assumed that the mechanism is vulnerable simply because I have called it a "discretionary" mechanism, any more so than the management of group memberships (for example) is vulnerable. It would be possible, for instance, to make the table of relations modifiable only by a designated security administrator via a trusted TCB interface.

- the decomposition into three related, but distinct sets of controls (certification of transactions, clearance of users, and authorization by means of relations) would appear to simplify the problem of keeping everything straight. In particular, the certification of transactions would depend only upon their correctness (the certifier does not need to be concerned with the impact of a change in certification on user authorizations); the clearance of users I view as establishing long-term "damage control" boundaries, while the authorization of users to execute transactions against particular objects or data types in the relation table establishes a shorter-term "need to do".

- It should be understood that in order for execution of a transaction to commence, several things must match: 1) the transaction must be certified for execution against the selected data types, 2) the user must have a basic clearance both to access the data types, and to execute that transaction, 3) specific authorization in the relation table must exist for the user to execute the transaction against these specific objects. If any of these conditions fail, the transaction cannot begin.

3. Application to the Clark/Wilson Requirements

In the previous section, the emphasis was on presenting an overview of how the proposed system is to work. In this section, the requirements stated by Clark and Wilson in [1] are restated, with a short summary of how they are mapped into the proposed system.

3.1 Definitions

- **Constrained Data Item (CDI)** -- those data items within the system to which the integrity policy must be applied. -- A CDI corresponds to what has been called a **data type** in the preceding section. Note that a CDI may consist of many distinct objects, or, for important data objects, an object may be given a unique type: it is up to the application designer.

- **Integrity Verification Procedure (IVP)** -- a procedure, the purpose of which is to confirm that all of the CDI's in the system conform to the integrity specification at the time the IVP is executed. -- In my system the IVP, as designed, would be a transaction object certified to correctly perform the verification function over all represented data types. Note that my system also accommodates "smaller" IVP-like transactions for arbitrary subsets of the data types.

- **Transformation Procedure (TP)** -- a well-formed transaction that changes the set of CDIs from one valid state to another. -- In my system, a TP is an arbitrary transaction object (program) that has been certified to operate correctly on its designated data types.

- **Unconstrained Data Item (UDI)** -- a data item not covered by the integrity policy. -- In my system, a data type would be reserved for UDIs. Those TP's certified to correctly transform UDIs to CDIs (i.e., validate and move data into the system) would simply have the integrity and disclosure categories for the UDI data type added to their access class.

3.2 Enforcement Rules

- **C1:** All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run. -- Clark and Wilson identify this as a requirement imposed upon the certifier of the IVP, as it would be in the system I have described.

- **C2:** All TPs must be certified to be valid. . . For each TP, the certifier must specify a list of CDIs (called a relation) which the TP has been certified to manipulate correctly. -- In the system described, this list is embedded in the access class assigned to the TP program object.

- **E1:** the system must maintain the relation referred to in rule C2, and must ensure that the only manipulation of any CDI is by a TP, for which the CDI occurs in the relation for that TP. -- This rule is enforced indirectly by means of program integrity. The security kernel ensures by means of this constraint that the TP executed by a subject is certified for all CDI's accessible by the subject. It follows that any CDI manipulated by a TP in execution is one that the TP is certified to manipulate correctly.

- **E2: the system must maintain a list of relations associating triples of the form <UserID, TP, CDI> that identifies which users may cause which TPs to be executed to manipulate which CDIs. -- as discussed in the last section, this rule is enforced explicitly by the TCB as a discretionary policy. However, it is backed up by the additional requirement that the user be cleared for a given TP and list of CDIs in the non-discretionary sense.**

- **C3:** the list of relations in E2 must be certified to meet the separation of duty requirement. -- Clark and Wilson identify this as a rule to be enforced by the human administrators of the system.

- **E3:** the system must authenticate the identity of each user attempting to execute a TP. -- as this is a commonly met requirement for any high-assurance evaluated TCB, it would seem unnecessary to address this requirement in any detail.

- C4: all TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed. -- It might be noted that this is presented by Clark and Wilson as an application-dependent requirement, requiring review of the TP by the certifier. However, the intent of this requirement would be met in part by the security audit function of the underlying TCB, which would record, as a security-relevant event, the creation of a new subject, its associated user and access classes in the security audit log.

- C5: any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. -- Enforcement of this rule is also the province of the certifiers: the described system provides a means, however, for ensuring that a TP not certified to take a UDI as input in fact, cannot be executed with read access to a UDI.

- E4: only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, those associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity. -- This rule is to be enforced in several parts (outside the security kernel). First of all, in order to "certify" a TP one must be able to create a subject with a write label containing the integrity category [tn] reserved for that TP. It follows that a user with a clearance containing {tn}[tn] is a "certifier" for the TP. In order to execute the TP against a data type A, a user's clearance must contain {tn,a}[a]. Thus, there exists a clear-cut way to distinguish "certifiers" from "users" of a TP: only certifiers have a clearance containing [tn]. The rule that must be enforced by the TCB can then be restated as follows: no individual may be given a clearance that simultaneously contains the integrity category for a transaction and the integrity and/or secrecy category of any data type contained in the label for the transaction object. This rule would be most easily enforced by the TCB the time some user was given a clearance as a "certifier" by ensuring that the user was cleared for none of the reserved proposed for it oy the "certifying" individual).

4. Prospects for a Near-Term Implementation

In the material presented above, I have described the proposed system for supporting the Clark/Wilson requirements in the simplest mathematical terms I could find: the emphasis was on making a rather complex construction as clearly explained as possible without becoming mired in extraneous design issues. In particular, neither the efficiency nor the prospects for actually building the proposed system were considered. The purpose of this section is to address these issues briefly.

We might first list some of the characteristics a conventional TCB with a non-discretionary security kernel should have in order to support the construction given above:

- it should support both disclosure and Biba integrity policies;

- it should support partially trusted subjects with both write and read labels;

- it should support both hierarchical and non-hierarchical access classes;

- it should enforce a program integrity policy;

- it should be subsetted in such a way that the special requirements of the Clark/Wilson policy for constraining clearances and imposing controls on the creation of new subjects based on <UserId, TP, CDI> triples can be introduced without disturbing the security kernel itself.

The GEMSOS TCB has all of these properties. One issue raised by Karger in [11] is worth special mention: it should be apparent that a relatively large number of integrity and disclosure categories will be needed for a practical system. GEMSOS supports an access class label with over 90 bits available to represent the lattice of access classes. The commercial version of this system uses these bits to represent a distributed lattice conformant to the guidelines in [3]. However, the interpretation of these bits is confined internally to a single module which is easily modified. In order to support a Clark/Wilson policy (as only limited combinations of the categories will actually occur) this module can relatively easily be restructured to encode a much higher number of "data types". The remainder of the kernel depends, for its correct operation, only upon the fact that the policy is a lattice. (In particular, non-distributive lattices are accommodated.) Thus, making the required modification to the kernel is an issue primarily of routine software engineering.

The following changes, all relatively minor, would be needed to convert the

GEMSOS TCB into one supporting a Clark/Wilson policy in a practical way:

- the internal module interpreting access class labels would need to be modified, as discussed above;
- additions would have to be designed and coded for the discretionary access control manager to enforce the additional requirement to constrain subject creation based on a table of "triples". These would not disturb the existing discretionary and/or non-discretionary controls, but serve as a refinement to them.
- additions would have to be designed and coded to enforce rule E4.

All of these changes are relatively minor. Although a re-evaluation of the TCB would be induced, existing evidence could be substantially re-used. In particular, because of the high degree of structure in the existing GEMSOS design, and because no fundamental changes would be required to the design, the magnitude of effort required for such a re-evaluation would be lcw, and the risk of failure small.

A final point worth noting is that mathematically (and practically, as well) it is relatively easy to define a lattice that combines the lattice of "types" with conventional disclosure and integrity lattices (using the Cartesian product.) It follows that a policy combining the military and strongly-typed systems is immediately feasible.

5. Conclusions

In this paper I have presented a construction that maps an arbitrary Clark/Wilson policy to an equivalent "military" policy containing both non-discretionary and discretionary components. In particular, the most important elements of the Clark/Wilson requirements (execution of TPs only against CDIs they are certified for, and only by users authorized to execute the TP against these CDIs) can be en'orced with the strong assurances traditionally associated with non-discretionary policies. Moreover, because this construction points the way for utilizing existing technology, the prospects for a near-term implementation of a highly-assured Clark/Wilson system are promising.

However, this construction would appear to have some theoretical interest as well. Essentially, the construction shows that a security kernel supporting Biba integrity, with both hierarchical and non-hierarchical components, and enforcing program integrity, can serve as a strong type manager. As it would appear that Boebert and Kane have shown that the inverse transformation is also possible -- a strong type manager can enforce a lattice security policy -- in some sense, these views about integrity are two different ways of talking about the same things. Which way you select depends upon the things you want to talk about -- a "change of coordinates" into the other system is always possible.

REFERENCES

1. D.D.Clark and D.R.Wilson. "A Comparison of Commercial and Military/computer Security Policies." In **Proc. 1987 Symp. on Security and Privacy**. IEEE, April 1987.

2. D.E.Bell and L.J.LaPadula. **Secure Computer Systems: Unified Exposition and Multics Interpretation.** EDS-TR-75-306, The MITRE Corp., Bedford, Mass., March 1976.

3. **Department of Defense Trusted Computer Systems Evaluation Criteria. Dept. of Defense, National Computer Security Center.** Dec. 1985. DOD 5200.28-STD.

4. K.J.Biba. **Integrity Considerations For Secure Computer Systems.** USAF Electronic Systems Division, Bedford, Mass., ESD-TR-76-372, April 1977.

5. S.B.Lipner. "Non-Discretionary Controls for Commercial Applications." In **Proc. 1982 Symp. on Security and Privacy**. pages 2-10, IEEE, April 1982.

6. L.J.Shirley and R.R.Schell. "Mechanism Sufficiency Validation by Assignment." In **Proc. 1981 Symp. on Security and Privacy**. pages 26-32, IEEE, April 1981.

7. W.E.Boebert and R.Y.Kain. "A Practical Alternative to Hierarchical Integrity Policies." In **Proc. 8th National Computer Security Conference.** pages 18-27, October 1985.

8. T.M.P.Lee. "Using Mandatory Security to Enforce "Commercial" Security." In **Proc. 1988 Symp. on Security and Privacy**. pages 26-32, IEEE, April 1988.

9. R.R.Schell, T.F.Tao and M.Heckman. "Designing the GEMSOS Security Kernel for Security and Performance." In **Proc. 8th National Computer Security Conf.** pages 108-119, 1985.

10. D.E.Bell. **Secure Computer Systems: A Network Interpretation. In Proc. Second Aerospace Computer Conf.: Protecting Intellectual Property.** pages 2-4, Dec. 1986.

11. P.A.Karger. "Implementing Commercial Data Integrity with Secure Capabilities." In **Proc. 1988 Symp. on Security and Privacy**. pages 130-139, IEEE, April 1988.

12. W.R.Shockley. **Implementing the Clark/Wilson Integrity Policy Using Current Technology.** Technical Report GCI-88-2-01, Gemini Computers, Inc., Jan. 1988.

13. J.H.Saltzer and M.R.Schroeder. "The Protection of Information in Computer Systems." In **Proc. IEEE** Vol. 63, No. 9, (September 1975), pp. 1278-1308.

Formalizing Integrity using Non-Interference

Paul A. Pittelli

Department of Defense

Advanced Computer Security Research Division

27 November 1987

ABSTRACT

Currently, an important concept in computer security is data integrity. As early as 1977, there existed formal models which incorporated integrity in an access control policy [Biba]. In 1982 Goguen and Meseguer provided the modelling world with their non-interference theory which develops assertions based upon pairs of users [G+M]. Furthermore Clark and Wilson [C+W] discussed the concept of data integrity and the differences between an integrity policy and those policies controlling access to sensitive information. This paper takes advantage of the flexibility of non-interference to define a security policy for data integrity.

Introduction

A major concern of computer security is the concept of data integrity. Integrity considers the ability of a computer system to assure its users that the information it stores is not corrupted. This is different from the access control policies [B+L] used for protecting sensitive information which have been studied intensely for the past decade. Unlike access control policies which restrict access to data objects based on classification and clearances, an integrity policy should discuss properties of the system which protect the soundness and completeness of the stored information. We do not concern ourselves with a discussion of the differences in the two policies, but this point is clearly brought out in a paper by David Clark and David Wilson [C+W]. However we use certain of the concepts from their paper as motivation for our integrity policy which is expressed in non-interference theory.

Motivation

The model by Clark and Wilson [C+W] consists of a finite state machine which has a set of constrained data items (CDI's) representing those elements for which integrity must be provided and similiarly a set of unconstrained data items (UDI's). Also included in the model are two types of procedures. The first type of procedure is called a Transformation Procedure (TP) which can be viewed as the typical state transition functions. The second type of procedure is a Integrity Verification Procedure (IVP), whose purpose is "to confirm that all of the CDI's in the system conform to the integrity specification at the time the IVP is executed." [C+W p.189]. With a given set of procedures, they also define a set of nine rules, partitioned into certification and enforcement rules, to which the procedures must adhere if the system is to provide data integrity.

A key concept described by these rules is that of separation of duty. Separation of duty refers to a system in which a state transition cannot be fully executed by one user but requires the cooperation of two or more users to complete. A typical example of separation of duty is a business procurement process. That is, one user requests an object, another user authorizes the request, a third actually purchases and receives, until finally the original user who requested the object receives it. In this example a company handling multi-million dollar objects certainly would not want <u>one</u> person to have the capability of performing all the functions in the procurement process. It is this concept of separation of duty upon which we will build our definition of an integrity policy.

Definitions

Let U be the set of all system users and C the set of state changing commands. For every user $u \in U$ let $a_u \in U$ be the user who is designated as the "authorizer" of any command issued by u. Informally, our definition of an integrity preserving system is as follows:

<u>**Def**</u>: A system (finite state machine) preserves integrity provided that for every user $u \in U$ and command $c \in C$, the command c, when issued by u does not effect the system until a_u authorizes c.

The phrase "u does not effect the system" can be restated as "whatever u does is not visible by any other user" or, better yet "u does not interfere with v" where $v \in U \setminus \{u, a_u\}$. The last phrasing indicates a relation to non-interference theory. However we will see that the clause "until a_u authorizes the command" will be represented in a definition for a purgeable user-command pair. We formalize our definition using the notation of Goguen and Meseguer in [G+M]. Recall that w is a finite sequence of user-command pairs $w = (u_1,c_1),(u_2,c_2),\ldots,(u_n,c_n)$ where $u_i \in U$ and $c_i \in C$. The family of all possible input sequences is denoted by $(UxC)^*$. The state of the machine determined by w from the universal initial state is denoted by [[w]]. The non-interference assertion that we develop for integrity differs somewhat from the assertions that Goguen and Meseguer created. The difference lies in the definition of purgeability of a user-command pair which in turn creates a difference in the purge function.

A purging function is the key tool in formalizing non-interference assertions. Given a finite sequence of user-command pairs w, the purge of the sequence w is simply a certain subsequence of w. In our case, this subsequence of w is the one where all commands issued by u are authorized. That is, any command by u that is not authorized is deleted (purged) from the sequence w. Formally we say:

<u>**Def 1**</u>: A user command pair $(u_i,c_i) \in w$ is purgeable with respect to u iff $u = u_i$ and there is not a user-command pair $(u_j,c_j) \in w$ with $u_j = a_u$ and $c_j =$ "authorization command for u" and $i < j$.

Using this definition of a purgeable user-command pair we get a recursive definition for the purge function:

<u>**Def**</u>: Let $w = (u_1,c_1),(u_2,c_2),\ldots,(u_n,c_n)$. Then for $i = 1,2,\ldots,n$ we have $\mathrm{Purge}_u\!:(UxC)^* \Rightarrow (UxC)^*$ where $\mathrm{Purge}_u(\emptyset) = \emptyset$ and $\mathrm{Purge}_u((u_i,c_i),\ldots,(u_n,c_n)) =$

$$\mathrm{Purge}_u((u_{i+1},c_{i+1}),\ldots,(u_n,c_n))$$

if (u_i,c_i) is purgeable with respect to u

or

$$(u_i,c_i),\mathrm{Purge}_u((u_{i+1},c_{i+1}),\ldots,(u_n,c_n))$$

otherwise

Informally $\mathrm{Purge}_u(w)$ will delete from the sequence w any command issued by u that is not later authorized by a_u. Notice that this definition of purging is different from the purging performed in the non-interference definition in [G+M p.79]. Their purging function simply removes all user-command pairs issued by u or removes those where the command is from a subset of C. Our definition of purge is dependent upon the rest of the sequence in determining the purgeability of a user-command pair.

Even though the defintion of our purge function is different from Goguen and Meseguer's, with an extra assumption we can derive a purge function which performs conditional non-interference. The assumption is: let any authorization command issued by a_u authorize all previous commands by u. With this additional hypothesis, our purge function behaves exactly like that of Goguen and Meseguer's conditional non-interference purging. That is, $\text{Purge}_u(w) = w_1w_2$ where $w = w_1w_2^1$ and w_1 is the longest subsequence of w which ends in an authorization command and $w_2 = \text{Purge}_u(w_2^1)$. We will discuss more aspects of purging in a later section, for now let us continue with our developement of data integrity. Given the above definitions we can state the following:

<u>Policy 1</u>: A system (finite state machine) preserves integrity provided that for every u ε U and v ε U\{u,a_u} we have u does not interfere with v modulo Purge_u written

$$u :| v \text{ mod } \text{Purge}_u$$

$$|||$$

$$\text{out}([[w]],v,r) = \text{out}([[\text{Purge}_u(w)]],v,r)$$

$$\text{for all } w \, \varepsilon \, (UxC)^*.$$

Generalizations

Upon examination of the above definition, there are several ways one could generalize to allow more flexibility. Each generalization will be summarized by stating a new definition for the purgeability of a user-command pair and also a revised policy statement. The purge function will also be different but that is implicit because of the new definition of purgeable. We have already stated a generalization earlier with the assumption that all previous commands issued by u can be authorized by a_u with a single command. The most obvious way to generalize is to say that more than one user needs to authorize a command issued by u. That is, let

A_u = {x ε U | x must be a member of the authorization process for u }.

We postulate that the members of the authorization set A_u constitute a tree as defined in digraph theory. This seems to be a natural structure to impose on the set because of the management hierarchy which exists in the corporate world. We know that in many instances a manager will not grant authorization for an action until various subordinates have given their approval. This concept simply says that there is a partial ordering on the set A_u. Thus A_u determines an "authorization tree for u" with the members of A_u as the nodes, u as the root, and a directed edge exists from u_i to u_j, with u_i,u_j ε A_u, iff u_j follows u_i in the authorization process for u. The example below will help to illustrate this point.

<u>Ex</u>: Suppose Paul, whenever he wants to publish a paper, has to get the following approvals; Will(technical advisor), Ted(division chief), John(office chief), and Fred(patent officer). If we assume that the jobs of Will,Ted, and Fred are independent and an office chief is above a division chief in the corporate architecture then A_{Paul} = {Will,Ted,John,Fred} and the authorization tree for Paul looks like Figure 1.

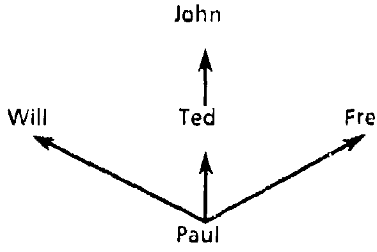


Figure 1

Incorporating A_u into our integrity concept yields the new purgeable definition and policy:

<u>Def 2</u>: A user command pair (u_i,c_i) ε w is purgeable with respect to u iff $u = u_i$ and there is not a subsequence of user-command pairs beginning at (u_i,c_i) which define the authorization tree for u.

<u>Policy 2</u>: A system (finite state machine) preserves integrity provided that for every u ε U and v ε U\({u}UA_u) we have u does not interfere with v modulo Purge_u written

$$u :| \, v \bmod \mathrm{Purge}_u$$

$$|||$$

$$\mathrm{out}([[w]],v,r) = \mathrm{out}([[\mathrm{Purge}_u(w)]],v,r)$$

$$\text{for all } w \, \varepsilon \, (UxC)^*.$$

Closely related to the previous generalization is the capability of providing "group" integrity protection. This refers to allowing individuals to interact with each other without worrying about integrity, in other words interfering. Groups arise naturally out of the common practice of partitioning a project among several people. In this case we would want all the people on the same project to be able to interact freely without always having to satisfy an authorization process. This is easily formalized by defining $G_u = \{u_j \mid \text{the integrity concern between } u \text{ and } u_j \text{ is void}\}$. This generalization does not effect the definition of a purgeable user-command pair, and the policy is defined by replacing $v \, \varepsilon \, U\backslash(\{u\} \cup A_u)$ in Policy 2 with $v \, \varepsilon \, U\backslash(\{u\} \cup A_u \cup G_u)$.

The last generalization we want to make concerns the actual command that a user issues. In particular, suppose that a user has to seek authorization from two different sources depending upon the command that he performs. The combination of the earlier examples illustrates this notion. That is, suppose Paul wants to purchase a Cray computer. The procurement process involves someone with the capability to authorize the use of corporate funds whereas the publishing process is independent of money matters. This suggests that a user u has an authorization tree for every different command that he can issue. Adding this concept leaves us with the final definition and policy statement:

Purgeable Def : A user command pair $(u_i,c_i) \, \varepsilon \, w$ is purgeable with respect to u iff $u = u_i$ and there is not a subsequence of user-command pairs beginning at (u_i,c_i) which define the authorization tree for u issuing command c_i.

Integrity Policy : A system (finite state machine) preserves integrity provided that for every $u \, \varepsilon \, U$ and $v \, \varepsilon \, U\backslash(\{u\} \cup A_{(u,c)} \cup G_u)$ we have u using command c does not interfere with v modulo Purge_u written

$$u,c :| \, v \bmod \mathrm{Purge}_u$$

$$|||$$

$$\mathrm{out}([[w]],v,r) = \mathrm{out}([[\mathrm{Purge}_u(w)]],v,r)$$

$$\text{for all } w \, \varepsilon \, (UxC)^*.$$

Remarks:

Clearly, we ought to consider the question: are all the concerns of data integrity emcompassed in our policy definition? The answer is probably no. For instance in [C+W] Clark and Wilson state nine rules which must be satisfied to provide data integrity. Some of the rules require procedures which certify state transitions and data items; others require procedures to identify and authenticate every user attempting to execute a transition. These rules, certainly germane, are not covered by our model.

Our definition has the advantage of formalizing, in our opinion, the notion of separation of duty, a concept of considerable current interest and concern as pointed out forcefully in [C+W]. To use the theory of non-interference seems to be a very natural mathematical environment in which to try to express the notion precisely. There is the additional advantage that the theory has been elegantly developed by Goguen and Meseguer, Haigh and Young, and Johnson and Thayer.

Moreover, we have described a different type of non-interference assertion. That is, a non-interference assertion that does not purge like that of Goguen and Meseguer, nor does it act like a conditional non-interference assertion. The reason for this lies in the definition of the function Purge_u. As stated before, Purge_u removes any occurrence of an unauthorized command issued by u, whereas the Goguen and Meseguer non-inteference purge function removes all occurrences of a command(in a certain set) issued by u and the conditional non-interference purge function purges only after a specific occurrence, thus allowing previous unauthorized events to be effective. Hence we see that the "power of a non-interference policy" (i.e. how much interference is allowed) is contingent upon the definition of the purge function.

For the moment, suppose that our purge function actually behaves like a conditional non-interference assertion. (Recall that the necessary assumption for this example is that any command issued by a_u authorizes all previous user-command pairs (u,c) in the sequence w). With this extra hypothesis, the integrity policy is directly related to the multi-domain policy(MDS) for SAT which is described in [H+Y]. The non-interference assertions in both policies look for an occurrence of a "channel", a path from the domain of user u to domain d in the MDS policy and an "authorization tree" for user u issuing command c for our integrity policy. It is interesting that these assertions not only act the same way on commands but are considered to comprise the mandatory part of the overall security policy.

Conclusion

In this paper we have developed a formalization of the concept of data integrity, the basis of which is separation of duty. Specifically we formalized the intuitive notion of an "authorization process" by defining a purgeable user-command pair. From this definition we created a purge function which in turn results in a policy for integrity. The use of non-interference theory as a mathematical environment in which to describe the policy, not only allows us the capability to enhance our defintion with aspects of integrity which may appear in the future as our understanding of the concept deepens, but also exhibits a relationship between integrity policies and multi-level security policies already developed in non-interference assertions.

References

[B+L] Bell,D.E. and LaPadula,L.J., "Secure Computer Systems",
ESD-TR-73-278 (Vol. I-III),
Mitre Corporation, Bedford, Ma., April 1974.

[Biba] Biba,K.J., "Integrity Considerations for Secure Computer Systems"'
Mitre TR-3153,
Mitre Corporation, Bedford, Ma., April 1977.

[C+W] Clark,D.D. and Wilson,D.R., "A Comparison of Commercial and Military Computer Security Policies", Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 1987.

[G+M] Goguen,J.A. and Meseguer,J., "Unwinding and Inference Control", Proceedings of the 1984 IEEE Symposium on Security and Privacy, April 1984.

[H+Y] Haigh,J.T. and Young,W.D., "Extending the Noninterference Version of MLS for SAT"', IEEE Transactions on Software Engineering,
Vol. SE-13, No. 2, Feb. 1987.

A RISK ANALYSIS MODEL FOR THE MILITARY ENVIRONMENT

Thomas W. Osgood
Manager, Security Assurance
Computer Sciences Corporation
3160 Fairview Park Drive
Falls Church, Va. 22046
(703) 876-1206

ABSTRACT

This paper describes a model for ADP Risk Analysis (RA) that was developed in response to the special requirements of the military data processing environment typified by the Defense Communications Agency's (DCA) Joint Data Systems Support Center (JDSSC) in the Pentagon. The reasons why more traditional RA models and methodologies have failed in this environment are identified. The special challenges faced by risk analysts in the military classified ADP environment are described. This paper considers the needs of security management officials for RA results in both a single-site single-system environment and the more typical multiple-systems multiple-sites environments faced by JDSSC and other military commands. Finally, a methodology for RA is presented that responds to these needs through the use of multiple metrics, a standardized threat nomenclature, and standardized reporting.

INTRODUCTION

During 1987, work sponsored by the DCA JDSSC ADP Security Office (C703) resulted in the development of a RA methodology for use by JDSSC ADP Security officials during RAs at World Wide Military Command and Control System (WWMCCS) sites and other installations operated and managed by JDSSC. The JDSSC RA Guide (RAG) incorporates a model and a methodology for RA that is appropriate to JDSSC's needs but somewhat different from other RA models being used in similar environments. While certainly not state-of-the-art given the science of RA, the JDSSC RAG was designed to provide practical guidance in the performance of an ADP RA, not to define new methods for analyzing diffuse risks.

Unlike many other RA methodology results, the JDSSC RAG RA results are combinable: RA efforts from distributed sites can be summarized over a large number of installations. This capability can be used to identify the types of "network security postures" JDSSC requires. Also, the JDSSC RAG RA results are abstractable, producing the level of RA reporting necessary for both low-level specific countermeasures and high-level JDSSC policy decisions and budget planning.

While results from the application of the JDSSC RAG are still tentative, they show great promise for the future. Current evolutionary plans for the JDSSC RAG include expansions in the guidance provided for Network RAs, automation of the model and the methodology, and the establishment of mechanisms for effective Risk Management in a military ADP environment.

BACKGROUND

Before beginning any discussion of the JDSSC RAG, it is appropriate to begin with a rapid review of why ADP RAs are performed, what is expected of them, and why current methods just don't seem to work.

RA is a Well Developed Science

What may still distinctly surprise many involved in the business of ADP RA is that RA, in its purest sense, is a well developed, sophisticated, and evolving science. However, the business of ADP RA is not well developed, sophisticated, or evolving. On the contrary. Little new or innovative work in ADP RA is occurring at all.

The science and the art of RA have been applied to other fields over a significant period of time. RAs by professional risk analysts against a wide variety of complex systems have been conducted. Notable among non-ADP RA efforts was a study conducted to determine the safety of commercial nuclear power stations [1]. The RA contained detailed examinations of the safety mechanisms incorporated into commercial reactor systems, and it plotted the failure rates of individual components (as well as related and dependent components in combination) against the potentials for measurable leakages of radiation. A highly quantified study, it has been widely cited as an illustration of what the process of RA should be.

As RA has evolved, organizations such as the Risk Analysis Society have greatly extended the types of problems which can be considered through quantitative methods. Non-bayesian techniques for evaluating risk have been developed and applied to a wide variety of problems not amenable to deterministic evaluation.

Origins of ADP RA - Basic Mandates

The business of ADP RA can be said to have begun with the publication of Transmittal Memorandum Number 1 to the Office of Management and Budget's Circular A-71 [2]. OMB A-71/TM#1 required that all executive branch departments and agencies develop and implement computer security programs. Within this original guidance, RAs were explicitly called for at all computer installations operated by or for the federal government "to provide a measure of the relative vulnerabilities at the installation so that security resources can effectively be distributed to minimize the potential loss." RAs were required for all installations each time a significant change occurred, or at least once every five years.

Both before and after the publication of OMB Circular A-71/TM#1, the Department of Commerce published a series of standards and guidelines to aid federal agencies in the performance of RAs [3] [4] [5]. The requirements and specific methodologies for RA have also been incorporated into a number of Department of Defense (DoD) regulations including [6], [7], and [8].

The original guidance from the OMB has recently been replaced as OMB A-130 [9]. The requirements in the current OMB Circular are only slightly more explicit than those originally contained in [2] - "The objective of a Risk Analysis is to provide a measure of the relative vulnerabilities and threats to an installation so that security resources can be effectively distributed to minimize potential loss." However, the current circular does allow for a variance in the formalism of the analysis based on the size of the installation - "Risk Analyses may vary from an informal review of a microcomputer installation to a formal, fully quantified risk analysis of a large scale computer system."

This softening in the requirement is perhaps in response to the realities of ADP RA state-of-the-practice. Many ADP managers perceive ADP RAs costly, time-consuming, and of questionable value to management planning. The next section of this paper reviews the problems with ADP RA in more detail.

REQUIREMENTS ANALYSIS

The development of the JDSSC RAG generally followed the stages of the standard product lifecycle, beginning with an an analysis of the requirements that must be satisfied by the JDSSC ADP RA methodology. Following this analysis, a number of ideas were prototyped for evaluation through their application to a live analysis effort.

Purpose of RA - Management Benefits

The primary management benefit of an ADP RA is that the quantified evaluation of risk (i.e., relative criticality based on some common metric - the basis of all RA efforts) is highly useful as a yardstick of relative need. During the process, management also gains an insight into problems faced at several system levels, many of which are normally hidden because of overall system complexity. Decisions to act based on RA results assure the best use of available funding, where best is defined by the metric employed. If the metric employed is dollars, then RA points towards the actions that make the best economic sense. As described earlier, however, dollars are not the only possible metric, and other methods for quantifying loss can be used in situations where fiscal economics are inappropriate.

A secondary benefit from ADP RA is the opportunity to reacquaint staff personnel with the importance of the data processing resource to overall mission objectives. Through the identification of real and potential losses, the extent of the reliance on an ADP resource is rediscovered. Most, if not all, user mission objectives are directly dependent upon the success of the ADP organization. In the absence of the ADP resource, no alternative means for user mission satisfaction are available. The true extent of DoD reliance on its ADP resources is only poorly understood by most data processing professionals involved in supporting these resources.

Another secondary benefit from the process of RA is the identification of important dependencies. Seemingly unimportant resources and functions can play paramount roles in overall system reliability. Within an ADP environment, the reliability of the entire ADP resource can be focused on individual pieces of equipment and specific personnel. Even minor failures can have major impacts on an entire installation. Ramifications of minor problems in the ADP environment can mean disasters for user organizations.

Problems with Existing ADP RA Methods and Models

Many models, methodologies, and tools supporting (some purporting to 'automate') ADP RA have been produced and employed by different federal department and agencies [7] [8] [10] [11] [12]. These models of ADP RA, and the particular methodologies supporting them, vary from agency to agency, from regulation to regulation, and from standard to standard. Seemingly, the only thing that all existing ADP RA methodologies, tools, and models have in common is their diversity. Nearly every existing methodology, tool, and model has its own positive and negative points [13]. Few are compatible with any other approach. Nearly all are based on a purely financial analysis of loss and cost-effectiveness of countermeasures.

Considerable resources have been invested over the past ten years in the performance of ADP RAs. A cottage industry in the performance of RAs has emerged to service the ADP RA needs of the federal marketplace. Unfortunately, the diversity and the impropriety of the methods for ADP RA being employed have raised serious doubts about the utility of the process to ADP management.

ADP RA results have been widely criticized for (1) their sizes - ADP RAs can produce volumes of detailed data of questionable accuracy or utility, (2) their diversity - managers are faced with a wide variety of RA results from different efforts, and (3) their nature - the types of issues considered by different RA methodologies and models are different, and ADP RA is highly dependent upon the personnel performing the analysis.

Within the military environment, experiences in the performance of ADP RAs have been much the same as for non-DoD agencies. Unfortunately for the DoD, where the greatest reliance on ADP exists, and where the most significant risks are faced, none of the methodologies for RA is at all appropriate. Without exception, these methodologies, models, and tools fail to properly appreciate the priorities of the military environment. These priorities include elements that are crucial considerations in ADP RA:

1. Unlike most other federal agencies, the DoD ADP systems process classified information that must be protected to the maximum extent possible. Military command and control systems process information vital to the national defense.

2. System failures in the military environment have implications for national security, not just finance. The eventual users of military systems include all military commands and elements. Failures of different military systems have differing levels of implications.

3. Policy decisions and budget allocations in the DoD are made centrally. Current ADP RAs are highly system- and environment-specific processes. Thus, policy decisions must be based on very detailed RA results.

4. An ADP RA requires so much time and associated resources that the known risk posture within a single authority or command (such as JDSSC) cannot be kept current. Practical means are unavailable for keeping ADP RA results up to date in a rapidly changing environment.

5. A significant level of expertise is required for the performance of a quality ADP RA. It is difficult to provide sufficient guidance in the performance of ADP RAs that inexperienced personnel can produce useful results.

Currently available tools JDSSC evaluated before the JDSSC RAG was developed were found to be universally difficult to use or tailor to specific environments, lacking in mechanisms for maintaining RA results, and unable to produce both detailed and abstracted results. The Los Alamos Vulnerability Analysis (LAVA) tool received from the National Computer

Security Center (NCSC) was evaluated in depth. The evaluation concluded that:

1. LAVA can be quite cumbersome to use. If questions within its automated questionnaire are answered incorrectly: no mechanisms exist for their specific modification.

2. While LAVA's extensive automated questionnaire quite well addresses the areas within its scope, no mechanism is provided to address issues outside of the defined areas. Non-addressed areas included TEMPEST, office automation, personal computers, Operations Security (OPSEC), and word processing.

3. The report LAVA produces is difficult to read, and conveys less insight to actual security conditions than does an annotated copy of the input questionnaire upon which the report is based. Vulnerability ratings are presented with no description of their basis.

4. LAVA is based on assumptions about the types of threats to which the data processing resource is exposed. For this to be reasonable, other assumptions must be made about the scope of the analysis LAVA is able to support.

JDSSC's analysis of LAVA, the National Aeronautics and Space Administration's (NASA) Self Analysis Guide (SAGUD), and Lance Hoffman's RISKCALC, among others, have resulted in the following conclusions about available ADP RA methodologies and tools:

1. None of the examined methodologies or tools provides sufficient comparison of RA results across different systems or installations.

2. None of the examined methodologies or tools adequately addresses the issues of mission satisfaction or information compromise.

3. The tools examined are not sufficiently flexible or expandable to be useful to JDSSC because of the dynamic nature of the JDSSC ADP environment.

4. None of the examined methodologies or tools allows risk analysis results to be collected and accumulated across installations for strategic planning and abstract analysis.

While the failings of particular tools and methodologies differ, no existing tool or methodology seems to solve some problems:

1. The value of classified information is difficult to quantify. No reliable method exists for determining the value of classified information in the general case. No formula is possible that factors in the real value of classified information to a potential adversary.

2. The values of assets are not identical in all instances. The value of classified information when compromised, for example, is much greater than the value of classified information unintentionally destroyed (e.g., in a fire). Valuation must be as a function of the threats an asset is exposed to, a point missed by most, if not all, established methodologies.

3. Losses experienced due to 'mission dissatisfaction' can include a decrease in U.S. defense readiness. Threats which might affect mission satisfaction are difficult to quantify realistically. As a result, RA findings that imply effects against mission satisfaction are typically under- or over-emphasized by the losses attributed to them.

4. None of the established methodologies allow for sufficient comparison or abstraction of ADP RA results. ADP RA results must be comparable across systems and installations and must be easily, intuitively, and quickly understood by laymen.

5. The maintenance of ADP RA results is not well supported in a very dynamic and networked environment. No provisions exist for rapid calculations based on 'what if' scenarios against risk analysis results, or for the dynamics of an environment with rapidly changing threats and assets.

6. Risk Management (the continuing identification of risks, and the corrective actions taken in response to identified risks) is not sufficiently emphasized by existing tools or methodologies. Some tools include no provisions for Risk Management whatsoever.

7. Even within a given methodology, RA results tend to vary, and even strong methodologies can result in ADP RA results that are inconsistent with prior studies in the same installation. Strong guidelines for the analysis techniques, scope and categories of investigation are needed to ensure consistent ADP RA results.

To a large extent, the methods employed cause the problems with ADP RAs. [14] stated that "The majority of computer security risk analyses have used annual loss expectancies (ALEs), a method well-suited to and used by insurance companies." Most methodologies examined compute the ALEs in terms of dollars. However, dollars are an inappropriate measure of many risks faced in the military environment. Losses of classified information, or of the implications inherent in potential failures of critical defense ADP systems, just cannot be stated in terms of "dollars lost" per instance or per year.

[14] also concludes that the science has been hampered by the lack of available, appropriate metrics to apply to intangible losses. [14] further identifies means to analyze diffuse and undefined risks. Both [14] and [15] discuss the need to better apply the true science of RA to the problem of ADP RA through non-bayseian techniques. However, in the analyses which led to the production of the JDSSC RAG, the problems with the techniques used to compute risk (ALEs) were seen as less indicative of why current models have failed to be useful than the problems obvious with the techniques used to compute specific loss. It was felt that dollars lost were an extremely inappropriate way to express the potentials involved in classified information compromise and in denial of service for crucial military ADP resources.

Qualitative versus Quantitative RA

Although highly quantified computer security RAs have tended to become quickly overbearing, unquantified analyses face other risks. Unless supported by some form of quantification, findings of vulnerability and recommendations for countermeasures and safeguards are reduced to opinion and conjecture.

It is in the quantification of risk that RA derives its benefits. Qualitative assessments of security (physical security, technical security within systems, and administrative controls, etc.) by experienced analysts usually identify many weaknesses for which remedial actions can be recommended and reasonably supported. Unfortunately, no budget is sufficient to allow implementation of every safeguard that looks attractive or which seems necessary. In the military environment, as elsewhere, many reviews have been conducted based on this "best guess" approach, resulting in recommendations which may not have been the best application of available funding.

[16] warns against the qualitative approach to computer security: "Security Measures are cost-effective only when the losses that are displaced are significantly greater than the [cost of the] security measures." Although individual problems are easy to evaluate on their own merits, the "common sense" approach quickly breaks down when applied to many concurrent problems. The problems facing management may also be extremely complex and require a deep understanding of the specific situation to appreciate the need for any remedial action. Only by somehow quantifying risk can different problems be realistically compared and decisions made about safeguard implementations really supported. [16] recommends that analysts "do a comprehensive job of problem definition and gross quantification before attempting the implementation of computer security measures."

Minimal Requirements

While several ADP RA methodologies are in use, most were intended for application to non-defense systems, where economics plays the major role in management decision making. JDSSC's special challenges are not satisfied through any methodology or tool demonstrated to date, in part because many of the situations it faces do not lend themselves to a purely financial analysis. Safeguards over classified information, for example, are difficult to justify by dollars saved. Defining the JDSSC RAG first required reviewing what the JDSSC environment required.

JDSSC manages systems critical to national security, distributed across a wide geographic region. Its mission includes support of (1) the National Military Command Center (NMCC) supporting the Organization of the Joint Chiefs of Staff (OJCS), the Office of the Secretary of Defense (OSD), and the National Command Authority (NCA); (2) the Alternate Military Command Center (ANMCC); and (3) a wide variety of smaller and more specialized operational, developmental, and research systems and networks (both local and wide area) supporting critical defense needs. In the near future, classified networks, office automation, and classified word processing systems will probably become even more prevalent than they are today. Due to this variety of support areas, JDSSC's most important requirement for ADP RA is for techniques sufficiently flexible for each of these diverse types of systems and which allow identification of the types of risks each faces. In practical terms, and because some of the automation security requirements for networks (as one example) are not fully defined today, the methodology must be expandable.

Some systems managed and operated by JDSSC are subject to security requirements based on their processing modes. JDSSC systems process classified information at various levels, and each level is associated with increasing requirements for computer security. The ADP RA methodology required by JDSSC must include provisions for these types of considerations.

Security management within JDSSC is centralized. Any recommendations must be based on identification of the most critical problems among all JDSSC systems so that decisions can be made about where action (new or revised policies, etc.) is most needed. A second important consideration for JDSSC is the need for a methodology that can allow security management officials to realistically compare problems across systems and installations and to make summary decisions at the policy level based on this information.

Personnel responsible for budget allocations have only a limited understanding of the details of each system supported by JDSSC. These officials must be provided with summary information that can be rapidly assimilated without reviewing voluminous reports or detailed calculations. Techniques for ADP RA results abstraction are needed to support high-level management decision making.

Prior ADP RAs performed against JDSSC systems have identified major risks. Recommendations for the implementation of countermeasures were based on the findings, and actions were assigned to different organizations to ensure that risks were mitigated. Follow-on reviews revealed, however, that in many cases ADP RA recommendations were not acted upon, and that risks identified during analyses were still present when the next analysis was performed. JDSSC needed mechanisms to provide for proper Risk Management. A system was needed to ensure that ADP RA results were acted upon in a timely manner and that dependent situations (where multiple actions were needed to respond to single risks) were successfully tracked.

In the past, JDSSC has attempted to perform RAs according to defined methodologies and guidelines published by various sources. There have been problems in applying standardized techniques to JDSSC systems, and the standardized techniques have not fully met all JDSSC requirements for ADP RA and risk management. These problems fall into four major categories:

1. Technology. JDSSC is involved in state-of-the-art application of available technology for secure networks, secure systems, office automation, and classified word processing systems. Mechanisms needed to evaluate these types of systems are not included, incomplete, or not expandable or modifiable.

2. Comparative Results. JDSSC management must be able to compare results obtained from analyses at one installation with those obtained at other installations. Methodologies not designed to support comparisons between installations are difficult to use for this purpose.

3. Results Abstraction. The budget allocation process must be supported by information that is brief, concise, rapidly understandable, and that does not require a detailed understanding of the systems or specific problems involved. In most of the tools and methodologies used, the results are presented in lengthy reports which contain detailed computations, none of which is suitable for JDSSC.

4. Risk Management. Mechanisms are needed to ensure that ADP RA results are acted upon in a timely manner and to track progress toward planned goals. No current tools or methodologies sufficiently provide for this need.

In general terms, the concepts involved in ADP RA are relatively simple. Problems are discovered, the assets involved are valued, the frequencies of occurrence are determined or estimated, the losses are computed, and countermeasures are postulated and analyzed. Problems often arise, however, in applying this relatively simple concept to the JDSSC environment. The problems arise from unique aspects of these systems and from shortcomings in a number of popular methodologies and tools.

DESIGN OF THE JDSSC ADP RA METHODOLOGY

The JDSSC ADP RA methodology's basic requirements are that new approaches be developed to account for the failings of the currently available methods. The JDSSC RAG is designed around an approach for quantifying "risk" that does not depend upon dollars as the sole measure of loss.

ADP RA Risk Model

Earlier, we reflected on the sophisticated work being done to analyze diffuse risks by professional risk analysts outside of the ADP field. Others have described how these methods might be more appropriate than the more simplistic model of risk nearly all ADP RA models employ. The ADP RA objective is not, however, the most accurate portrayal of the true extent of risks. Mandates require only the accurate ranking of relative risks to the ADP resource.

For the purposes of an ADP RA (done quickly, and with only a limited amount of time to quantify results), the model of risk most ADP RA methodologies employ may be the most appropriate and is certainly quite adequate:

RISK ALE = AFE x SLE

RISK ALE Annual Loss Expectancy. A measure of the extent of the danger from a given threat.

AFE: Annual Frequency Estimate. How often a given negative event is expected to occur.

SLE: Single Loss Estimate Some measure of exactly what the results of that negative event will be each time it occurs.

The model allows evaluation and relative ranking of negative events (threats). It is beyond the scope of this paper to debate the advantages of alternative models of risk. Suffice it to say that we believe that this model in its most general sense is sufficient to this application, and that its inaccuracies are well hidden by the fallacies inherent in any attempt to quantify threat freqencies or the true loss that will be experienced in any disaster

ADP RA Results Evaluation

Numbers are used in an ADP RA not to absolutely quantify the exact risk, but rather to relatively rank risks. As a result, and because of the inaccuracies built into any evaluation of risk, the process of ADP RA is at the same time highly qualitative (i.e., judgmental) and quantitative (i.e., based on numbers). Understanding exactly what the results of an ADP RA effort mean (and what they do not) and how these results can support risk mitigation is important. Without an appreciation of the inaccuracies of the process, misconceptions are probable

A major misconception can occur when risk is expressed as a figure, an ALE. For example, an ALE of $27,000.00 due to fires in the computer room must be taken with a truckload of salt. No installation (still standing) loses this much every year. Even a liberal interpretation (the figure divided by the likelihood of a fire yielding some figure for the potential damage a fire is likely to cause) is unrealistic. No study can exactly predict losses or the cost of recovering them. Postulations of potential losses are hypothetical at best. Real disasters are messy, worse-than-worst-case, and wholly unpredictable. An installation with an excellent fire safety program can be destroyed by fire immediately after receiving a clean bill of health from the local fire marshal.

What then do ALEs represent, if the real costs associated with disasters cannot be reliably established in advance? They represent the magnitude of the potential or risk. Problems or threats with high ALEs are more important than those with low ALEs. Only in this relative and qualitative ranking do the numbers employed in the process have their place.

Management has a limited budget and a limited opportunity for positive change. ADP RA techniques indicate where improvements are most needed and where resources can best be applied. That is all. In the situation described above, management would be wrong to assume that, by setting aside the ALE for fire every year, that they would be covered in the event of a fire disaster. They would also be wrong to assume that any safeguard costing less than 27K annually is cost-effective. This risk must be compared with others, and what is possible to mitigate those risks which appear most threatening must be postulated. The relative cost-effectiveness of countermeasures must be assessed against the most relatively serious risks. In many cases, doing anything to reduce either the likelihood or the potential impact of identified risks may not be cost-effective. Their identification is still important.

The value of the process lies not in the exactness of the figures employed but in their magnitudes, reasonableness, and in the relative ranking of problems based on their consistent application across a range of situations. There is a great desire for techniques and "truly scientific" methods to overcome the vagaries of the ADP RA process. These desires spring from fears that the actual numbers employed in ADP RAs are unrealistic. The fears are justified. Real numbers could never be produced in advance. Even close estimations are difficult at best. Experiences with well known threats (Courtney's five major sins, etc.) tend to support the contention that, through quantification, a reasonable qualitative ranking can be achieved.

Increases in the accuracy of the values for assets (and the other assumptions such as threat frequencies) do not increase the accuracy of the process. Quantified ADP RAs are performed to avoid the only alternative, a best-guess qualitative ranking of problems. Guessing (i.e., estimating asset values, threat frequencies, and relative degrees of exposure) is still required, but is performed in limited ways. Upper and lower bounds for the guesses are provided as, for example, statistics for threat frequencies and equipment purchase costs. The methods yield generally supportable rankings for problems that can be intuitively ranked, and they increase the confidence in the rankings of less easily understand problems.

Risk Analysis Metrics

In nearly every application of the traditional RA model to ADP RA, the SLE and the RISK ALE are demonstrated as dollars. During the analysis preceding the development of the JDSSC RAG, however, we wondered if other measures of "risk" might also be useful when dollars (as a measure of loss) were inappropriate. Those areas where financial analyses are most inappropriate are information compromise and system downtime. Alternative metrics were devised and applied to a live analysis effort as an evaluation of their utility.

Information Compromise: A metric for information compromise was based on a qualitative review of elements of the compromise threat. We decomposed the threat as the inherent risk associated with the classification level of information (Top Secret data is inherently more valuable than Confidential information), the extent of the compromise (need to know violations are less severe than a leakage from Top Secret to Unclassified), the extent of the loss (a little data is less valuable than a great volume of data if the other factors remain the same), and the utility of the compromised information (automated media at high density or high speed) is more useful than paper output.

For information compromise, the SLE is computed as a formula:

SLE = C x E x A x L

C A Multiplier based on the highest classification of data which could be exposed. Note 1.

E The percentage of the total volume of data (contained within the system being examined) that is exposed to the threat. Note 2.

A A multiplier based upon the avenue through which the information is exposed. Note 3.

L The number of classification levels over which information exposure occurs. Note 4.

Note 1. Classification multipliers were established on an order of magnitude scale to allow the formula to be biased approximately equally between a small volume of highly classified data and a large volume of less highly classified data due to the inferences possible through volume and the probability of classification through aggregation.

Note 2. Exposure, a factor applied to all formulas in the JDSSC RAG, is used to apply granularity within undefined assets, such as system information volumes.

Note 3. The Avenue multiplier was originally devised as a measure of the bandwidth of compromise (volume over speed). In practice, the data required to accurately compute bandwidth is generally unavailable or difficult to compute, and a rougher measure (the avenue multiplier is described below) was employed.

Note 4. The number of levels is a multiplier to describe the increasing loss potential as information is compromised across need to know (level 1) and classification level (Confidential to Unclassified is Level 2, Secret to Confidential is Level 3, etc.) boundaries.

Because ADP systems are vulnerable to compromise of information through various types of mechanisms, the metric includes an Avenue Multiplier to allow the speed of leakage to be considered:

10 - Information exposed over high-speed communications lines to remote installations.

9 - Information exposed to local automated processes on high speed media.

8 - Information exposed to local automated processes on lower speed media.

7 - Information exposed over low-speed communications lines to remote installations.

6 - Information exposed to high speed terminal devices with local storage capability.

5 - Information exposed to high speed terminal devices without local storage capability.

4 - Information exposed to low speed display terminals.

3 - Information exposed to high speed hard copy terminals.

2 - Information exposed to low speed hard copy terminals.

1 - Information exposed on paper only - not automated media.

The scale allows high risk exposures in an automated environment (i.e., high speed data leakage in a digital form away from the facility) to be afforded more importance than less inherently risky losses (i.e., improper handling of paper media). In practical terms, and given the high degree of "noise" present in possible attempts to glean useful information through the examination or monitoring of an automated system, the scale reflects variance in the potential that an adversary could gain sufficient data in an appropriate form for automated or manual analysis to actually discover something useful.

Through this metric, loss is expressed as an abstract number. The actual units (CEALs) were sufficiently obtuse that the term 'Abstracted Units' was employed in reporting the values computed for various situations. While the scale produced may not be uniform, since more serious problems may not result in a SLE value sufficiently high to reflect their true import, the formula has resulted in a reasonable relative ranking of problems, which was the intent. It also satisfies the basic objectives:

1. Risks associated with information compromise across both discretionary and mandatory controls can be computed and compared.

2. Situations that involve exposure of classified information to personnel with no need-to-know will rank lower (L=1) than situations associated with the compromise of information across levels (L.gt.1).

3. The greater the number of classification levels crossed, the greater the risk. The higher the bandwidth (as estimated via the 'avenue') the greater the risk.

4. Situations involving highly classified information will tend to have higher risk values than situations involving less highly classified information, unless the volume (as estimated via the exposure) of less highly classified information is sufficiently great to overcome the order of magnitude emphasis of classification level.

Mission Dissatisfaction: Some ADP RA methodologies attempt to place an overall value on the ADP organization, or on the overall value of the user organization. In the military environment, the approaches used to value the mission have been inappropriate, resulting in dollar values for "mission" that are much too high, while still missing the vital factors which must be evaluated.

Mission values are sometimes based on salaries (of all personnel), equipment costs, or annual budget allocations. All techniques in use to place a financial value on "mission satisfaction" as an asset result in enormous numbers. These numbers, in the presence of even relatively minor risks to ADP resource availability, result in potential loss values (based on the percentage of potential availability unrealized or percentage of mission unsatisfied) that can justify nearly any safeguard that at all reduces the potentials for system downtime. Vast savings appear possible through applying expensive countermeasures to reduce downtime potentials by minuscule amounts.

For commercial organizations, a case can be made for "mission satisfaction" valuation as a function of the overall organization's reliance on the ADP resource for revenue. In a military environment, however, the competition is not economic but strategic. The value of a command and control system cannot be estimated as dollars per hour nor can downtime be computed in terms of dollars lost. Downtime losses are much greater conceptually than in financially.

Any metric of mission satisfaction must consider system availability. Mission satisfaction for an ADP organization is best described as the highest degree of system availability and the lowest degree of downtime. Any metric which attempts to portray the "losses" associated with system downtime must appreciate the realities of such situations:

1. Downtime losses for systems differ according to the criticality of the resource being examined. In a computer room containing multiple resources, only a subset of these resources is absolutely critical. Others (development systems, etc.) could become unavailable for significant periods of time without appreciable impacts on the overall mission.

2. Downtime losses are not linear. A downtime of four days is much more than four times as damaging than a single day of system unavailability. Secondary losses begin to accrue as organizations which rely upon the resource are unable to satisfy their needs. Initial per-hour figures may escalate as the length of unavailability increases.

Our original thoughts led us to the following formula for losses associated with system downtime:

$$ALE = AFE * M * (D*D)$$

ALE = Annual "Risk"
AFE = Frequency of the situation involving downtime
M = A measure of the criticality of the system
D = Downtime length.

Real costs (personnel costs, etc.) were estimated as dollars lost with an appreciation for the secondary impacts of lengthy denials to various users:

$$SLE = AFE * D * (C + (C1 + C2 \ldots))$$

SLE = Dollar costs associated with downtime.
C = Cost per unit of downtime (Note 1.)
C1,2 = Cost escalation based on downtime length.

Note 1. Cost per unit of downtime must be computed based on the user population dependent upon the resource. This user population must be identified based on the users of information produced, not merely by the number of user accounts.

In actual use, however, the survey of user organizations required to actually quantify these loss potentials proved extremely complex and the analysis of potential per-hour or secondary costs much too time-consuming for manual tracking. Also, the rapid ADP RA performed to testbed this metric discovered risks applied equally across all surveyed resources. In a more detailed analysis, the use of both of the formulas described above may be possible and appropriate. In the rapid ADP RA performed, however, the metric for downtime losses was considerably simplified:

SLE (in hours) = D * E
ALE (in hours) = AFE * SLE
D = the downtime length possible (in hours)
E = the percentage of system resources (normally 100%) affected by the threat.

Downtime losses are computed as annual hours-lost figures for all situations involving the potentials for downtime.

Analysis using Multiple Metrics: In use, the use of dollars, "Abstracted Units" (for information compromise) and "hours" (for denial of service potentials) results in three rankings of problems discovered during an ADP RA. Problems can be ranked according to those with the greatest potential annual costs, those with the greatest potentials for information compromise, and those with the greatest potentials for system downtime. These rankings are useful both in isolation and in comparisons with one another.

Different problems will tend to be shown as most important according to each metric. Specific situations will entail losses in more than one metric. For example, in a fire the systems may need to be shut down (downtime), the components may burn (dollar losses), and unauthorized personnel will have to be granted access to the computer room (information compromise). When the relative rankings of these problems (according to the various metrics involved) are considered, however, the potentials for information compromise (based on a ranking in the face of other information compromise potentials) quickly diminish, while the potentials for denial of service and major dollar costs (again as relatively ranked within these scales) become apparent.

Countermeasure evaluations are also different in an ADP RA model which employs multiple metrics. The traditional dollars-saved per dollars-invested cost-benefit analyses can also be "abstracted units saved" per dollar invested or "hours of downtime saved" per dollar invested. Although the metrics employed make it more difficult to state with assurance that "countermeasure x is cost-effective," they do point out which countermeasures are more relatively cost-effective. Again, relative, not absolute, ranking is facilitated. Comparing problems or countermeasures across metrics is purely subjective. It is impossible to state that a problem in information compromise is more or less severe than a problem with availability or real costs. Each problem is important on its own merits.

Risk Management

An ADP RA is useful only within a program for managing risk. Risk Management is a responsibility of senior management in all ADP installations as a part of everyday business. A periodic ADP RA supports this process but cannot in isolation satisfy the need for a systematic program for Risk Management.

Risk Management is applied to any system that faces risks. In software development, for example, one of the major management programs that must be implemented is a Risk Management program to deal with the threats to the software design and development process [DoD-STD-2167]. Although the management of an ADP installation is a venture with significantly more inherent risks than those faced during software development, few installations have formalized their risk management approaches. As a result, management is quickly overwhelmed with problems, and a fire-fighting approach to ADP resource integrity and reliability management is inevitable.

Problem identification and evaluation occur within the context of specific disasters. The minimum actions absolutely necessary to resolve current situations are considered and acted upon without considering root causes or long-term effects. This approach to management is the state-of-the-practice in ADP organizations that face rapid change or a significant number of regular threats. Ironically, it is exactly this environment that would benefit most from a formalized risk management program.

Effective management actions in any system correspond to Risk Management. Management determines new programs, initiatives, and corrective actions based on informal perceptions of the severity of the problems addressed or averted. Even high-level budget decisions are based on an informal understanding of cost vs. benefits.

Risk Management is only the formalization of the process of effective management. Too often, problems discovered during one ADP RA remain to be rediscovered during the next. Too often, problems are qualitatively perceived to be minor until crises occur. Too little action is taken too late in response to these problems. In other cases, minor problems become lost in the system and are never dealt with or responded to. Problems or risks considered too minor will be ignored. Rapid evaluations of potential risks ignore some potential impacts. The extent of interdependencies within an ADP organization is generally accepted but poorly understood. In some cases, decisions are reached regarding the need to respond to needs, but effective actions to implement these decisions are not taken to the depth necessary for effective problem resolution. The details of implementing policy are much too voluminous for the current methods of control and monitoring. ADP RAs conducted at JDSSC installations have revealed numerous cases of incongruities between policy and actual practice, or between high level decisions and low level implementations.

Formalizing the existing management system of control in response to the volume of problems and the details of the implementation of responses is necessary and long overdue. To identify how that formalization can be achieved, we reviewed how Risk Management works in well-defined management controls such as those mandated for software development. Risk Management consists of the following steps, each of which is conducted within a formalized tracking system:

1. Risk Identification. Problems are identified several ways. ADP RAs identify many problems in a short time. Other risks are identified the hard way - after the fact. Finally, many problems are recognized by management during day-to-day operation.

2. Risk Evaluation. The probability of risk, its potential impacts, and any and all contributing factors must be identified as quickly and as completely as possible after a risk has been identified.

3. Risk Mitigation. A response to each identified risk should be decided based on an understanding of both the risk and the costs of alternative responses. Risk Mitigation is the development of appropriate resposes to known risks.

4. Risk Monitoring. After a response has been decided upon, its implementation and effectiveness in use must be monitored by management.

These steps remain the same for any system and should be quite familiar to anyone in Configuration Management. A problem reporting system is required; and the status of the analysis, review, approval, and implementation of countermeasures (corrective action) is regularly recorded and reported. Our analyses show no reason to modify this system, and we incorporated it directly into the JDSSC RAG.

Risk Management aids management not only in terms of what decisions and actions must be taken but also in terms of how those decisions are made. Courtney's [un]common sense recommendations for consideration of losses before countermeasures are implemented by such an approach. Once established, such a system facilitates control to a greater degree of detail than is humanly possible without formal tracking.

JDSSC RAG METHODOLOGY

Once the basic model of risk was established, the other required elements of the methodology were developed around it. Summarization methods were defined from both the defined defined and a standardized list of threats. ADP RA results combinations (the prelude to true network ADP RA techniques) were defined based on percentage of losses due to threats by metrics, an approach which allows different ADP RA scopes in different locations. Finally, the analysis stages were defined to allow both standard problems, those typically found or expected in nearly all ADP installations, and non-standard problems, those unique to the specific environment and which may have never before been encountered, to be identified and analyzed.

Planning

The elements of the scope of an ADP RA should be agreed upon in advance as should the schedule for interim reporting. The results of this planning should be in writing.

The first phase of performance defined in the JDSSC RAG is scope identification. The scope of an ADP RA has three elements; physical, technical, and administrative.

Within the physical scope, the specific facilities and areas within those facilities to be reviewed are identified. The list of external and well known threats to the facility in general are agreed upon in advance. Areas to remain unaddressed (e.g., overall facility problems, grounds, etc.) should be explicitly identified.

It is unproductive to repeat some analyses performed many times before. It is unlikely that moving an existing computer facility (the only possible response to some of the "risks" considered in many ADP RAs) can be justified based on external factors like the risk of flooding, earthquakes, volcanoes, or great hurricanes. Given the frequency of ADP RAs it is also unlikely that prior analysis results in these areas will need to be adjusted (for continental drift or the global greenhouse effect) very soon. It is only necessary that these factors be understuod once - before the facility is built. The JDSSC RAG recommends that prior analysis results be consulted for this information if it must be republished at all.

Within the technical scope, the actual systems to be reviewed are agreed upon, as is the depth of technical analysis to be applied against each system. Within JDSSC, other initiatives exist to review risks to ADP resources and to grade the vulnerabilities of technical security mechanisms. Within other agencies, programs for contingency planning, application certification, and ADP MIS may provide significant inputs to these types of analyses if they are necessary. In this area, it is imperative that scoping be performed based on an understanding of the materials available for analysis. Attempting to perform application certification, inventorying, or contingency planning within an ADP RA is inappropriate. Unless sufficient tracking mechanisms exist, reviews of the technical environment can be extremely time-consuming.

Finally, the organizations to be reviewed are agreed upon. In specific cases (e.g., the ADP Security organization, the Operations organization), the actual organizational structure and reporting mechanisms can become threats to the ADP resource. While many may disagree, the organization is itself an expensive (and continuing) asset, and it may itself be at risk based on threats management is exposed to.

Qualitative Review

Once the scope of an ADP RA has been established, the second stage of performance can begin. A qualitative review of the installation is performed. Questionnaires are distributed to site personnel, and the answers to those questionnaires should be available and reviewed prior to interviews. Site reviews and tours are required for all involved in an ADP RA.

To a large degree, the JDSSC RAG methodology draws from the already available successful and positive elements of other methodologies for ADP RA. LAVA's excellent questionnaire is incorporated, as are the questionnaires from AR 380-380, the WWMCCS RAG, and the NASA ADP Risk Analysis Guideline. The JDSSC RAG provides guidance as to the most appropriate audiences for each element of each questionnaire. The areas where contentions or differences exist between different copies of identical questionnaires can provide valuable insights about where problems exist within an ADP installation.

All involved in an ADP RA effort are required to tour the facility and make their feelings and impressions known to the other members in writing. While all ADP RA members can be expected to see similar things in these unstructured reviews, each will see each thing differently and will spot problems missed by all others who perform the same review. This 'touch and feel' element of an ADP RA cannot be eliminated and is a large part of the value provided through the analysis. ADP RA remains dependent upon the people performing the analysis. Personnel with the right backgrounds and level of generalized experience required are needed.

Structured interviews are conducted from the bottom up in all reviewed organizations (as well as within organizations not being specifically reviewed). Interviewing from the bottom up maximizes the productivity with personnel at higher points in an organizational structure where experience is concentrated. Interviews are conducted not to learn of the standard threats to the ADP resource, which should have been identified through the questionnaires, but to learn of other and non-standard threats the ADP resource is exposed to. The JDSSC RAG provides a starting point for this stage of the analysis, including a structure for interviewing that concentrates on the identification of duties, responsibilities, and reporting structure.

Qualitative reviews employ standard mandates, including the EDP Auditors Association's Control Objectives - 1980. Using established mandates limits the types of subjective judgements that can lead to contention. Within JDSSC, analyses are also made against established mandates including JCS Publication 22 and DoD-5200.28.

Quantification and Analysis

After the qualitative review is completed, the quantification process begins. Standardized threats and national statistics for those threats are employed. Most problems have well-known countermeasures. Those without well-known responses require more in-depth analysis.

Each problem is associated to a set of threats and recorded on standardized forms. The potential losses to each threat (in each appropriate metric) are computed and recorded.

Countermeasures are analyzed in terms of their impact on the losses to each threat in each metric. Note that the countermeasures are evaluated on their own merits and independently of problem-specific losses. Problem-specific loss estimation is useful only for problem ranking. The mapping between countermeasures and problems is less than exact; one problem may require multiple countermeasures, one countermeasure may apply to a number of specific problems.

Summarization and Abstraction

The set of quantified loss potentials and countermeasure analyses are input to the final stage of the analysis - abstraction and summarization. The results of this stage are used to report the ADP RA to management and to allow ADP RAs to be combined across installations.

Problem-specific losses are first combined to produce losses by threat within metric during the last phase of the analysis described above. Next, the percentage of loss attributable to each threat within metric is defined. This effort should also result in the identification of the most serious problems (for major threats) within each metric. The summarization report contains these percentage of loss by threat within metrics figures and illustrates them by summarized descriptions of the most serious problems for each metric. Countermeasures which are the most cost-effective are also contained in the summary report.

The summarization report should contain all of the information required for ADP RA results combination. In practice, the JDSSC RAG may need to be adjusted to increase or change the information contained in the summary report.

Pie charts were used in the testbed ADP RA to illustrate the losses attributable to major threats in each metric. A list of the percentage of loss by threat is listed to contain threat percentages too minor to be visible within thew pie chart. While graphical representation is seen as an important feature of the summary report, other representations of the information are still being researched.

FUTURE EXPANSIONS

The JDSSC RAG was delivered in December of 1987. It remains far from optimal, especially in terms of the level of guidance it provides to inexperienced personnel. Future efforts will concentrate on making the JDSSC RAG easier to use in locations where specific expertise in ADP RAs is not available, such as high security environments and remote locations. Other areas for research include the areas identified below.

Combinational ADP RA Results
During 1988 and 1989, JDSSC will begin to employ its RAG during other ADP RAs. As additional ADP RA results are produced, the means for combining and summarizing ADP RA results contained in the RAG will be revised and demonstrated. Combining ADP RA results is a capability prerequisite to effective ADP Security management within JDSSC.

Expansions to a Network ADP RA Model
However, the combining of specific ADP RA results is not the only prerequisite to the performance of Network ADP RAs. Networks are exposed to threats not applicable to ADP resources in isolation. Many of the elements of the JDSSC RAG (e.g., its standardized threat nomenclature, etc.) must be revised or extended before application to a network is possible.

Analyses of "risk" in an automated network environment based on the trustedness of its systems versus the clearance levels of its users (the NCSC's Yellow Books) must also be considered. It must be accepted, however, that networks incorporating all trusted elements remain in the future for JDSSC. In the largest part, both its networks and its systems in isolation remain unevaluated and uncertifiable. The lack of trusted components is not, however, in any way slowing the movement of agencies like JDSSC towards networking. Security mechanisms are being retrofitted into commercially available networking systems as an alternative to no security at all. As a result, the need for Network ADP RAs is extremely high.

Automation
Automation severely constrains the ability to modify a methodology in development. As a result, we have resisted the impulses to automate too much of the methodology too quickly. However, several elements of the methodology are now ready for such a process. We intend to begin by automating the best-known part of the methodology, quantification and summarization, along with an automated set of questions in specific areas. Implementation will concentrate on the ability to to transmit risks, problems, countermeasure implementation reports, and security postures between various field locations and a central controlling authority - the model of ADP security in the military environment.

CONCLUSIONS

The JDSSC RAG is still in its beginning and conceptual design stages. More work is needed before it becomes a useful tool for JDSSC ADP Security management. It provides some possible responses to some of the more difficult questions facing risk analysts in a classified ADP environment. In the absence of experienced ADP RA analysts, means must be found to communicate exactly what must be done to identify how results are to be produced. JDSSC remains committed, however, to seeking answers to these questions through the development of new methods. Existing methods and tools have not met their needs. The approach conceptualized above may provide the starting point for analyses to solve the critical problems facing ADP management officials in the military environment.

REFERENCES

1. Nuclear Regulatory Commission, Reactor Safety Study: An Assessment of Accident Risk in U.S. Commercial Nuclear Power Plants, NUREG-75/014, 1975.
2. Executive Office of the President, Office of Management and Budget, Security of Federal Automated Information Systems, Circular No. A-71, Transmittal Memorandum No. 1, 27 July 1978.
3. U.S. Department of Commerce, National Bureau of Standards, Guidelines for Automatic Data Processing Physical Security and Risk Management, FIPS PUB 31, June 1974.
4. U.S. Department of Commerce, National Bureau of Standards, Guideline for Automatic Data Processing Risk Analysis, FIPS PUB 65, 1 August 1979.
5. U.S. Department of Commerce, National Bureau of Standards, Audit and Evaluation of Computer Security, Special Publication Number 500-19, October 1977.
6. Department of the Army, Automation Security, Army Regulation 380-380, 1985.
7. Department of the Navy, Electronic Systems Command, Department of the Navy ADP Security Manual, OPNAVIST 5239.2.
8. Defense Communications Agency, Security Requirements for ADP Systems, DCA Instruction 630-230-19, 29 October 1985.
9. Executive Office of the President, Office of Management and Budget, Management of Federal Information Resources, Circular A-130, 12 December 1985.
10. EDP Audit Controls, NASA ADP Risk Analysis Guideline, July 1984.
11. Lance J. Hoffman, RISKCALC - A User's Reference Manual, 1985.
12. Department of Energy, Center for Computer Security, LAVA, An Application of the Los Alamos Vulnerability Assessment Methodology, Release 1.0, Los Alamos National Laboratory, 1986.
13. U.S. Department of Commerce, National Bureau of Standards, Technology Assessment: Methods for Measuring the Level of Computer Security, NBS Special Publication 500-133, October 1985.
14. Lance J. Hoffman, Risk Analysis and Computer Security: Bridging the Cultural Gaps, Proceedings of the 9th National Computer Security Conference, 15 September 1986.
15. Dr. R. Brown, Managing Diffuse Risks from Adversarial Sources (DR/AS) With Special Reference to Computer Security, Proceedings of the 9th National Computer Security Conference, 15 September 1986.
16. Robert H. Courtney, Good Computer Security Demands Realistic Problem Assessment, Proceedings of the Second Annual Symposium on Physical/Electronic Security - Philadelphia AFCEA, August 1986.

KNOWLEDGE-BASED MODELLING OF SYSTEM USAGE FOR RISK MANAGEMENT

H.N. Mayerfeld

Martin Marietta Laboratories
1450 South Rolling Road
Baltimore, Maryland 21227

E.F. Troy

Martin Marietta Information & Communications Systems
P.O. Box 1260
Denver, Colorado

Abstract

As the number and complexity of computer systems grow, the need for useful tools for performing risk assessments of these systems will become more pressing. In recent years, there have been several attempts to automate the risk assessment process through the use of questionnaires and menus. Some of these are implemented on personal computers for wide availability. Although these techniques offer an improvement over completely manual methods, they are often either cumbersome to use because of the wealth of information that must be laboriously extracted, or inadequate for deriving a sufficiently accurate risk assessment.

We have been investigating a new artificial intelligence-based approach to standardizing and automating the risk management process that will enable the analyst to produce risk assessments that are less costly, more uniform, and less prone to subjectivity. Central to our approach is the concept of determining the risk to information as it is used in the system, rather than the replacement cost of hardware and facilities. A four-level abstraction hierarchy for classifying system components and assets is used as the basis for constructing system models. We then determine risk to informational assets according to three primary criteria of security value: confidentiality, integrity, and availability. A model of information usage in the system is then developed to analyze the security risk for the complete information system.

Introduction

The development of an effective security program is critically dependent on the application of risk management to the initial design, subsequent modifications, and ongoing monitoring of a system. Therefore, the need for useful risk assessments and tools will become more urgent as the number and complexity of computer systems grow. Although a variety of methods have been proposed and are currently in use for performing risk analysis [1, 2], many are difficult to apply efficiently. In recent years, there have been several attempts to automate the risk assessment process through the use of computer-driven questionnaires and menus. Some of these, including RiskPAC, RiskCALC, RISKA, and LAVA/CS [3], are implemented on personal computers for wide availability. Although these techniques offer an improvement over completely manual methods, they are often either cumbersome to use because of the wealth of information that must be laboriously extracted via lengthy questionnaires, or inadequate for deriving a sufficiently accurate risk assessment due to their focus on component replacement cost.

With a goal of enabling risk assessments that are less costly, more uniform, and less subjective, we have been investigating a new approach to standardizing and automating the risk management process, which incorporates artificial intelligence techniques of representation and reasoning to model a computer system, its components, and the asset usage within the system. The approach draws on research in artificial intelligence, which has led to new methods of representing symbolic information at different levels of abstraction. Frame-based and object-oriented systems, in particular, are extremely powerful and versatile techniques for describing entities symbolically and embedding them into hierarchies of related entities. We are exploring the use of these methods for constructing representations of a computer system's components, so that we can model their interactions. In addition, we will be using

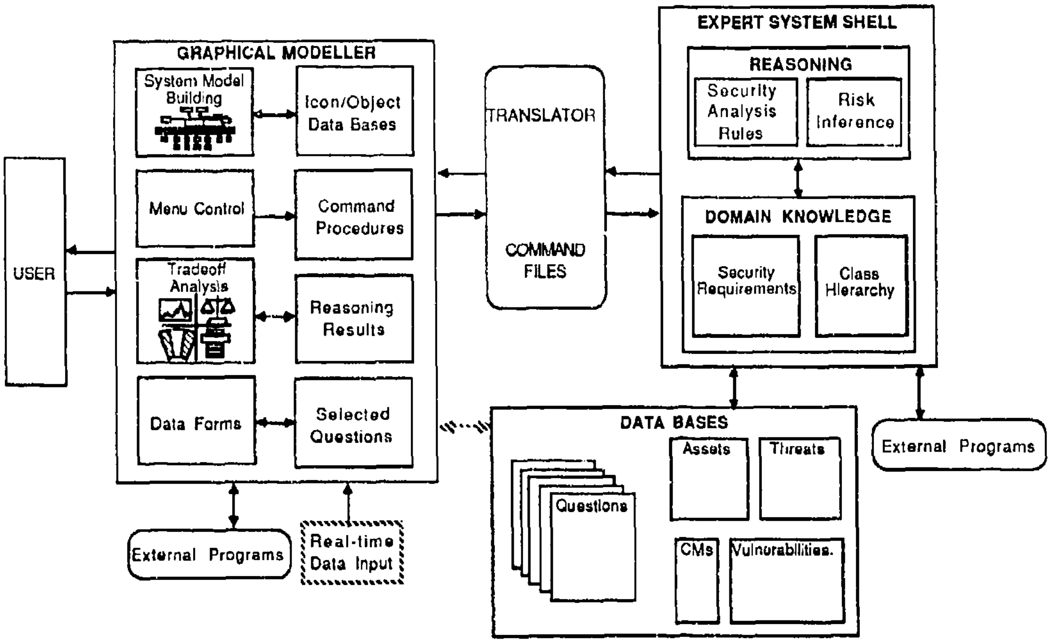


Figure 1. Architecture for a knowledge-based risk management system.

advanced techniques for reasoning about imprecise and uncertain information, such as the theory of fuzzy sets, to better describe the risk involved in a complex environment.

We have developed the concept for a knowledge-based expert system to assist in the risk management process. The current proposed architecture for our system, described in [5], is shown in Fig. 1. This approach has these primary features:

- It is based on multiple, high-level computer graphic models of the system, so that fewer detailed questions are required, many relationships can be derived automatically, and all of the input data can be checked for consistency.
- It minimizes the requirement for sophisticated risk management knowledge and leads to more uniform results.
- It considers all aspects of comprehensive risk management, drawing on multiple underlying knowledge bases for expertise about the domain.

Central to the design is the security schematic, which is a model of the security requirements and attributes of the system based on an underlying model of the risk management process. The requirements are broken down into the three basic or primary criteria of security value — confidentiality, integrity, and availability — and drive all of the system's reasoning. The underlying knowledge bases or hierarchical data bases contain taxonomies of risk entities, such as assets, threats, vulnerabilities, and countermeasures, as well as banks of questions, similar to the ones found in automated questionnaires such as those used by LAVA/CS. These questions may be selected dynamically by the system as needed, rather than mechanically through a laborious, step-by-step process. The reasoning module, or inference engine, controls the operation of the system, and includes the capacity for generating and analyzing security requirements; building and maintaining models; selecting appropriate parameters, questions, and data from the knowledge bases; and analyzing the trade-offs necessary for efficiently managing risk. Despite all this complexity, the user interface portion of the system presents a palatable set of views of the system's model and analysis, as well as dialog windows, which allow the option of querying or modifying any part of the knowledge bases textually or graphically.

Informational Assets

Traditionally, risk assessments have focused on the replacement value of the hardware and facilities of a system. Indeed, the risk assessment methodologies sanctioned by various government agencies, such as that described in FIPS PUB 65 [6], are also based on this approach. Although it is undeniably important to include direct physical losses in a comprehensive risk assessment, the greatest risks to any computer system by far, and those that are hardest to quantify, are the compromise of the informational content of the system, rather than the system components themselves. We have therefore concentrated on quantifying and expressing the risk to the *informational assets* of a computer system. We view the definition and evaluation of informational assets as central to the task of adequately assessing system risk.

Informational assets (which we shall sometimes refer to as simply assets) refer to the actual knowledge or information that is valuable to the organization, such as customer names and addresses, not the instantiations of that information, such as data files containing customer names and addresses. The distinction is a subtle but important one. We are concerned with ensuring the security of the information, rather than a particular instantiation of it. For instance, if a disk containing records of recent transactions crashes and is lost, the information may be recoverable from a backup copy, or by reconstruction of the lost records.

As mentioned above, the security requirements of an organization's assets can be classified into three *primary criteria of security value*: confidentiality (protecting an asset from harmful disclosure), integrity (protecting an asset from modification), and availability (assuring that information is available when needed). The value of a primary criterion of a particular instantiation of an asset, as it were, may or may not correspond to the value of the primary criterion of that asset itself. So, for instance, in the example used above, assurance of the availability of the particular data file containing recent transactions is not necessary in order to assure the availability of the information itself.

The intuitive inverse correlation of availability and confidentiality can be demonstrated clearly using a particular attribute of the instantiation of an asset according to our formulation. For example, if we consider the attribute of number of copies of an asset, we can show that the risk to confidentiality rises as the number of copies rises, while the risk to availability declines, as illustrated in Fig. 2a. Integrity risk has a more complex curve, shown in Fig. 2b. The integrity of an asset is at greater risk of compromise as the number of copies rises (in the absence of countermeasures, such as matching the copies against a master), yet the risk also increases as the number of instantiations approaches zero, since it cannot be lower than the risk to availability.

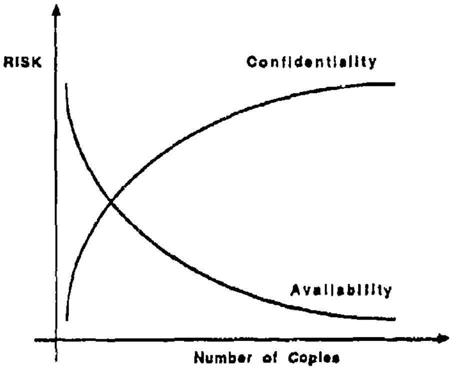


Figure 2a. Risk trade-offs based on number of software instantiations of an informational asset.

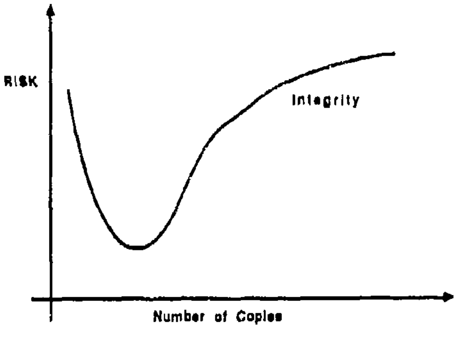


Figure 2b. Integrity risk tends to the maximum of confidentiality and availability risk.

An asset has a number of attributes that must be specified and understood clearly before its informational value can be established. These include attractiveness to threat agents, perceived value, possible outcomes (undesirable events that can befall it), and the sum of all other attributes, its actual compromise value, which is an expression of how much is lost if its security, as measured by one of the primary criteria, is compromised. To determine asset value, we must develop a methodology for considering these tightly interrelated attributes.

INFORMATION	Types:	Names, dates, figures, documents, ideas, programs, processes	Compromise Value Security requirements • Confidentiality • Integrity • Availability

SOFTWARE	Functions:	Transfer	Transformation	Storage
	Types:	Systems	Applications	Data Files Records

MEDIA	Functions:	Storage
	Types:	Tapes, hard disks, floppy disks, printouts, paper, punch cards

HARDWARE	Functions:	Transfer	Transformation
	Types:	Tape and disk drives, workstations, terminals, printers, cables, wires	Processors

Figure 3. Abstraction levels of assets and components.

Abstraction Levels of Assets and Components

Although informational assets are the primary entities needing protection, and drive the determination of security requirements, we cannot assess the risk to assets directly, nor protect them directly. Instead, we must consider the system and the environment in which the information is processed.

Accordingly, we have developed a four-level abstraction hierarchy for classifying assets and system components, illustrated in Fig. 3. At the lowest level are the hardware components of the system, such as the CPU, tape and disk drives, printers, and cables. These are generally fixed in place physically, and are the base on which everything else operates. The next level comprises media components, which sit on the hardware components, but tend to be less fixed. Examples of media components are tapes, disks, and printouts. The third level, software, includes files, databases, and programs, which exist in the environment provided by the hardware and media levels. At the highest, most abstract, level are the informational assets themselves. It is at this level that asset value and security requirements are determined.

Informational assets can have instantiations at each of the component levels. For instance, customer names and addresses (information) may be recorded in a database (software), stored on a disk (medium), and accessed through a disk drive (hardware). Threats and their actions operate in the environment of the component levels, and countermeasures are implemented there as well, although informational assets may be the ultimate targets of those threats.

It is also useful to classify system components according to functionality with respect to assets processing in the system. The functions performed by an information system can be divided into three broad categories: storage, transfer, and transformation. If we are to model asset usage in the system, it is essential to understand these three functions, the relationships and differences between them, and the ways in which they are performed by the system's components.

Figure 4 depicts a matrix showing the different functions associated with various components at the three lower abstraction levels. More information is contained here than is immediately apparent. For instance, although both hardware and media components are used for storage, hardware storage typically tends to be short term, whereas media storage implies a longer term. Storage in software, meanwhile, has a different meaning, because the software component used for storage resides in a

	Storage	Transfer	Transformation
Hardware	Memory Buffers	Drives Workstation, terminal, keyboards Printers Cables, wires, conduits	Processors
Media	Tapes Disks Printouts Paper		
Software	Data Files Records Buffers	System Software	Application Software

Figure 4. Functional component matrix.

hardware or media component. These distinctions are invaluable in modelling system usage and in assessing the risk associated with the system and the methods of minimizing that risk.

Modelling System Components

Based on the preceding discussion of assets and components, it is clear that an accurate, comprehensive risk assessment for informational assets must entail a model of the components in a computer system and their interactions, overlain by an asset usage model that describes the processing of information by the system. Likewise, an automated system for assisting in the risk management process should be capable of constructing and utilizing such models.

We are currently developing the framework for such a system. A required step is to create a component library, consisting of data structures that represent knowledge about system components. As the information contained in the library becomes richer, the skill of the system will improve. At a basic level, however, library entries must include the type and function of the component with respect to both the processing of assets and its links to other system components.

The user of such an automated system, as we envision it, would select predefined components from the library and link them together into a functional and physical model of the system using existing CAD/CAM tools, which provide graphic displays that facilitate interaction and enhance understanding. Alternatively, the user would be able to define novel components and include them in the model, as well as enter them into the library.

In addition to a library of system components, it will be necessary to develop data structures to represent the informational assets that need protection from compromise. With these, the user would be able to construct an information flow model to illustrate the processing of assets through the system, which would be presented as another graphical view of the system. The three graphical views described here are illustrated in Fig. 5.

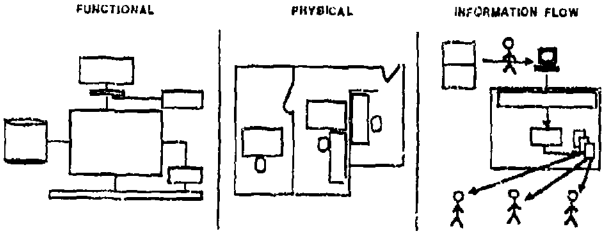


Figure 5. User views of system models.

The automated system would then apply the user's model of system design and asset usage, combined with its knowledge of the component characteristics and the security requirements of the assets, to identify component vulnerabilities with respect to the assets and to propose adequate countermeasures for dealing with them.

We summarize here the stages of risk management according to the above methodology:

1. *Building a model of the system under review* — this is done graphically, using schematics and other diagrams. In this stage, predefined components are selected from existing data bases, additional novel components that may be present are defined, and the components are structured into a complete system definition, which includes the functions of and relationships between components. The system is then checked for consistency and

2. *Identifying the assets processed by the system and their value, taking into account the consequential value of asset compromise* — the informational assets processed by the system are identified and assessed, and the outcomes of their compromise are specified. An important aspect of this activity is the identification of asset transfer and utilization. The information from this stage is used to derive asset compromise cost by component.

3. *Identifying vulnerabilities associated with the system components and countermeasures for neutralizing or minimizing those vulnerabilities* — component vulnerabilities that expose the assets they process are defined, and countermeasures (CM) are identified that can be used to reduce or eliminate asset exposure.

4. *Identifying threats to the system assets* — based on knowledge of threat agents and their actions, specific threats are identified that may exploit a vulnerability of the system to compromise the security of an asset. Included are both non-human or unintentional threats such as component failure, and intentional threat actions such as

5. *Analyzing the likelihood and severity of possible threat paths, and identifying the outcome of threat actions* — possible and likely paths by which threats could access and compromise assets are analyzed, along with the outcomes and consequences ensuing from each. From this analysis, overall risk of compromise to system assets can be asssessed.

6. *Presenting a summary of system risk that offers safeguard packages described in terms of costs and benefits* — the results of the risk analysis are presented to the user in the form of a risk summary and graphic descriptions of various safeguard alternatives with their cost/benefit trade-offs. Specific situations representing the highest risk are identified.

The stages of model-based risk management are portrayed in Fig. 6. In the next section, we present a description of a knowledge-based system that assists in this process, with some suggestions for its implementation.

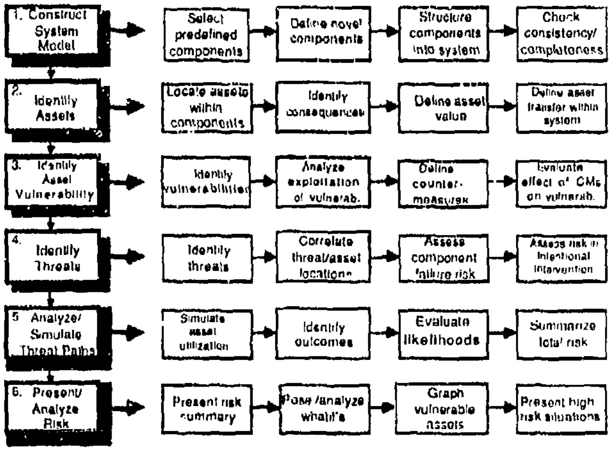

1. Construct System Model
Select predefined components
Define novel components
Structure components into system
Check consistency/ completeness
2. Identify Assets
Locate assets within components
Identify consequences
Define asset value
Define asset transfer within system
3. Identify Asset Vulnerability
Identify vulnerabilities
Analyze exploitation of vulnerab.
Define counter-measures
Evaluate effect of CMs on vulnerab.
4. Identify Threats
Identify threats
Correlate threat/asset locations
Assess component failure risk
Assess risk in intentional intervention
5 Analyze/ Simulate Threat Paths
Simulate asset utilization
Identify outcomes
Evaluate likelihoods
Summarize total risk
6. Present/ Analyze Risk
Present risk summary
Pose /analyze whatif's
Graph vulnerable assets
Present high risk situations

Figure 6. Model-based risk management.

A Knowledge-Based System for Modelling Asset Usage

Useful methods for symbolically representing knowledge have evolved from research in artificial intelligence. In an object-oriented representation, each entity is represented as an object with various attributes. But, each object may be a member of one or more classes of objects which have attributes of their own, and the object classes are also objects, and may thus be members of other classes, and so on. This formalism allows us to build hierarchies of objects, which can be constructed to correspond to actual hierarchies of entities in the domain being described. Objects may inherit attributes or values from their parent classes, and default values may be specified for the attribute values. Various processing methods can be used with objects, including triggering actions based on the value of the object's attributes.

We are currently designing an object-oriented system for representing and reasoning about system components and asset usage. Hardware, media, and software, as discussed above, are examples of object classes in such a representation. One attribute shared by the members of all of these classes is function, i.e., storage, transfer, or transformation. Conversely, certain classes, such as hardware, may have attributes, such as physical description, size, capacity, and location, that are not shared by other classes. The value of specific attributes also may vary within an object class. Those objects with similar attributes can be grouped into subsets.

The input and output ports are critical attributes in representing the transfer of assets within a system. At the hardware level, these may refer to actual hardware ports or terminals of the component, whereas at the media level, they refer to the hardware on which the media reside, and for software, to the input and output capabilities of the software component. Integrating the representation of the attributes of the various component levels is the key to creating an asset usage model of the system.

We are designing the system so that it will lead the user through the risk management process by constructing a model or set of models of the system under consideration, including the physical layout, functional, and information flow diagrams described above. Since the graphical objects in these views are representations of the underlying objects of the knowledge base, the user is actually building a model of the system in the computer's memory. The user would be able to switch from one view to another at will, and modify or query the knowledge base interactively from any view, while the system would guide the user through this model-building phase and check for missing or inconsistent information. The expert system would use these graphic models to derive information about the security of the system, inferring most relationships directly. It would then walk the user through a dialog requesting additional information not explicit in those views and suggest values for risk management parameters. This method ensures the accuracy and consistency of the analysis, facilitates modification, and closely resembles the method risk management professionals use to perform risk assessments.

Implementation Example

We now present a specific example to illustrate how the proposed system might work. The knowledge base may include a hierarchical structure, such as that in Fig. 7 showing the representation of knowledge about networks. Each of the specific network implementations (leaf nodes) would have a component library entry, and the higher level nodes would have library templates filled in only to the appropriate level of detail. Examples of portions of these library entries are shown in Figs. 8a, b, and c. Note that each successive node down the hierarchy inherits information from its parent node. Thus, the knowledge engineer who builds the hierarchy does not have to enter all the information for each node, reducing effort and the potential for entry errors. Additional and more specific information can be added for particular nodes, as shown.

TYPE OF NETWORK:
    Baseband
    Broadband

CHANNEL TYPE:
    Coaxial cable
    Twisted pair
    Radio
    Fiber optic

NUMBER OF STATIONS:

DATA RATE:
    1 Mbps
    5 Mbps
    10 Mbps
    20 Mbps

Figure 8a. Network object entry.

Suppose the user indicates, perhaps by clicking the mouse on a network icon on the main model-building display, that the system under review includes a network. The expert system could then present a menu of network types. It might even display this in the form shown in Fig. 7, with common defaults highlighted as indicated, and allow the user to browse through the tree and select a node.

If the user then selects the node labelled "Ethernet," the network specified by the user as being part of the system schematic model is now identified as an Ethernet, and is associated with the information contained in the component library about Ethernets, as well as the information about bus networks and networks in general.

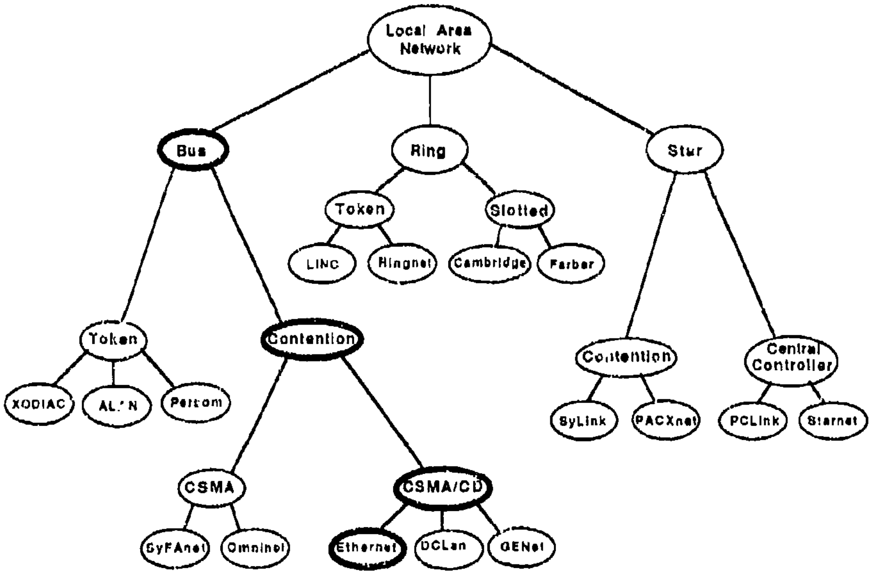


Figure 7. Network knowledge base hierarchy.

TYPE OF NETWORK:
Baseband
Broadband

CHANNEL TYPE:
Coaxial cable
Twisted pair
Radio
Fiber optic

NUMBER OF STATIONS:
Maximum of 1024

DATA RATE:
1 Mbps
5 Mbps
10 Mbps
20 Mbps

PACKET SIZE:
Must be between 56 and 1518 bytes

TOTAL LENGTH OF CABLE:
Max 2500 meters

LENGTH OF INDIVIDUAL CABLE SEGMENTS:
Max 500 meters

NUMBER OF CONNECTED CABLE SEGMENTS:
Max 5

MEDIUM ATTACHMENT UNITS PER SEGMENT:
Max 100 per indiv. cable segment

LENGTH OF MEDIUM TRANSCEIVER CABLE:
Max 50 meters

Figure 8b. CSMA/CD architecture network
(Carrier Sense, Multiple Access with Collision Detection
as defined in IEEE Standard 802.3).

TYPE OF NETWORK:
Baseband

CHANNEL TYPE:
Coaxial cable

ACCESS METHOD:
CSMA/CD

NUMBER OF STATIONS:
Maximum of 1024

DATA RATE:
10 Mbps

BACKOFF ALGORITHM:
Binary exponential backoff

INTERPACKET SPACING:
9.6 μs

PACKET SIZE:
Must be between 64 and 1518 bytes

TOTAL LENGTH OF CABLE:
Max 2500 meters

LENGTH OF INDIVIDUAL CABLE SEGMENTS:
Max 500 meters

NUMBER OF CONNECTED CABLE SEGMENTS:
Max 5

MEDIUM ATTACHMENT UNITS PER SEGMENT:
Max 100 per indiv. cable segment

LENGTH OF MEDIUM TRANSCEIVER CABLE:
Max 50 meters

DISTANCE BETWEEN 2 FARTHEST END NODES:
Max 2700 meters

Figure 8c. Ethernet object entry.

If the user were to then construct an information flow model showing the transmission of assets sensitive to disclosure (confidentiality requirement) along this network, the expert system would be able to infer a possible threat to confidentiality at this component, based on its knowledge of the vulnerability of coaxial cable bus networks to undetected wiretapping. It might then propose a countermeasure, such as the installation of fiber optic cable. The expert system would also know that a threat agent could, via a single component, gain access to the assets that are processed on the other components on that network, and consider this possibility in relation to the security requirements of the assets.

Status and Plans

Our system has been under design since February 1987. The initial conceptual design was completed in mid-1987 and reviewed internally and by a government team consisting of experts from the National Bureau of Standards and the National Computer Security Center. A static mockup of a sample walkthrough was constructed as part of the presentation. In early 1988, we implemented a portion of the system for proof of concept on a personal computer. Following the review of this implementation, we are continuing to develop an initial prototype of the full-scale system.

Our work thus far has revealed a number of areas that need more attention. It is clear that a lucid, comprehensive, workable model of risk management must be formulated as the basis for this work. We are interacting vigorously with major government and industry figures in this area [4]. In addition, taxonomies of the various components of such a model must be developed. It is especially important to create better quantitative and qualitative methods for measuring risk and analyzing trade-offs, and we intend to investigate the use of reasoning methods from artificial intelligence and traditional sources for this purpose. For example, estimates of the likelihood of a given threat action occurring are often necessarily imprecise. Fuzzy set theory provides tools developed specifically for reasoning with imprecise information, and can be utilized in this case. We also must design easy-to-use and representationally adequate user presentation and interface methods. We plan to pursue all of these issues in 1988 and 1989.

Acknowledgements

We would like to thank Bill Mitchell and Steve Maslen for providing management support for this project. We are grateful also for the input and impetus provided by Bruce Aldridge, John Lewis, and Ed Galavotti in the initial phases of this work. Stuart Katzke of the National Bureau of Standards and Sylvan Pinsky of the National Computer Security Center have been very supportive and helpful, and Ken Otwell's contributions are especially appreciated.

References

[1] B.A. Haugh, "A Survey of Standards, Methods, and Tools for Assessment and Reduction of Risk in Computer Systems," Report MML TR87-34, Martin Marietta Laboratories, 1987.

[2] P.S. Browne, "Survey of Risk Assessment Methodologies," No. 83-01-02 in *Auerbach Data Security Management*, New York, N.Y.: Auerbach Publishers, 1984.

[3] S.T. Smith et al., "LAVA for Computer Security (Release Version 1.01)," Los Alamos Lab document LA-UR-86-2942, Los Alamos National Laboratory, 1986.

[4] H.N. Mayerfeld, "Definition of Assets as the Basis of Risk Management," in *Proceedings of the 1988 Risk Model Builders Workshop*, Denver, CO, 1988.

[5] H.N. Mayerfeld, E.F. Troy, J.W. Lewis, and B.T. Aldridge, "M^2R_X: Model-Based Risk Assessment Expert," in *Proceedings of the Third Aerospace Computer Security Conference*, American Institute of Aeronautics and Astronautics, 1987.

[6] National Bureau of Standards, *Guideline for Automatic Data Processing Risk Analysis*, U.S. Department of Commerce, FIPS PUB 65, August 1979.

MODELING SECURITY RISK IN NETWORKS

Howard L. Johnson
Information Intelligence Sciences, Inc.
15694 E. Chenango
Aurora, CO 80015

J. Daniel Layne *
Computer Technology Associates, Inc.
7150 Campus Drive, Suite 100
Colorado Springs, CO 80920

Abstract

Distributed secure systems also have distributed security policy and unequal security risk. The n-squared problem (addressing security interface of n communicating nodes, not just the directly connected ones) and the cascading problem (creating greater risk by connecting systems of differing data exposure levels) are primary sources of difficulty in distributed system risk analysis. Landwehr and Lubbes described factors for determining Orange Book evaluation criteria in complex systems. This paper expands on their approach by adding network risk propagation rules. The model presented here is applicable to evaluation of sensitivity requirements (preventing unauthorized disclosure) and criticality requirements (preserving system integrity and availability) in heterogeneous networks. An automated analysis tool has been developed.

Background

The authors discussed issues of network and distributed system security at last year's conference [1]. There the ideas of Biba [2] and others were used to propose a criticality approach similar to that used when protecting sensitive information, which is the primary objective of current security policy and requirements (see Figure 1). Also discussed were techniques of system decomposition, an approach which deals individually and in combination with the elements of very large systems.

Criteria / Topic	Sensitivity (Existing Basis)	Criticality (Proposed Enhancements)
Protect	Classified data	Mission Data Control Data, Processes
Threat	Disclosure	Loss of Integrity Denial of Service
Levels	Unclassified Confidential Secret Top Secret (Compartments)	Noncritical Critical Highly-Critical (Compartments possible)
Control Goal	Need-to-Know	Need-to-Modify/Execute
Protection Mechanisms	Resistance	Resistance Detection/Recovery

Figure 1. Network Security Elements

This paper addresses the complex subject of risk analysis in a distributed system. The approach follows the lead of National Computer Security Center (NCSC) Yellow Book [3] guidance for assigning Orange Book [4] division and class and it also extends the ideas of Landwehr and Lubbes [5] to distributed, heterogenous environments. The recently available Trusted Network Interpretation (TNI, [6]) provides some guidance for evaluating and accrediting heterogenous networks, but TNI emphasis is on "single trusted systems." This paper thus describes a methodolgy for determining security (sensitivity and criticality) requirements in complex networks.

A system security policy must cover all of what is internal, plus external communications interfaces (logical as well as physical). This follows from and expands the Orange Book concept of the primary external interface as the human "user." The concept covers all "external subjects," including humans, computers (e.g., hosts), networks, other components or other systems.

The identification/authentication policy must cover each of these external subjects and, using access control capabilities, determine what controlled information can be received from and sent to each of them. There must be label consistency or a mapping technique must be defined that ensures proper protection and integrity. In some systems it will be necessary to maintain accountability to the user level, even though the user interface is with an external system. Sometimes the policy will require accountability only at the interfacing system level. The interface policy deals not only with the physical interconnectivity, but also with all pairs of communicating entities. This is the so called N-squared problem (Figure 2).

* Dan Layne's Current Address:

Martin Marietta Corporation
Information & Communications Systems
700 W. Mineral, MS XL5700
Littleton, CO 80120

This work was sponsored in part by USAF Space Command, under contract number F05605-85-C-0019 awarded to Computer Technology Associates and performed with Information Intelligence Sciences, Inc. The statements herein reflect the opinions of the authors and do not necessarily reflect the views or policy of the Air Force.

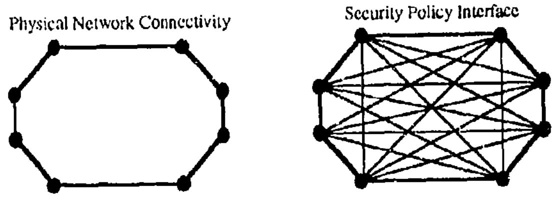


Figure 2. N-Squared Problem

Sometimes data are passed from one external system through the system of interest to another external system (Figure 3). Policy must ensure that required protection consistent with a mutual interconnection policy exists at the interface. If the systems are nodes of a network that receives and delivers encrypted data and if a mandatory sensitivity or criticality level separation or a discretionary "need-to-know" or "need-to-modify" exists, the appropriate security labels and access control lists must be shared between the two systems communicating data. The network need not necessarily be aware of these labels and lists.

Each network component has a unique security policy (even if it is no policy). This policy may be more strict or less strict than the policy of the other components. Inclusion into the system might increase the risk associated with a component due to the cascading problem (Figure 4), wherein the range of security levels in the network may be greater than the accreditation range of any component.

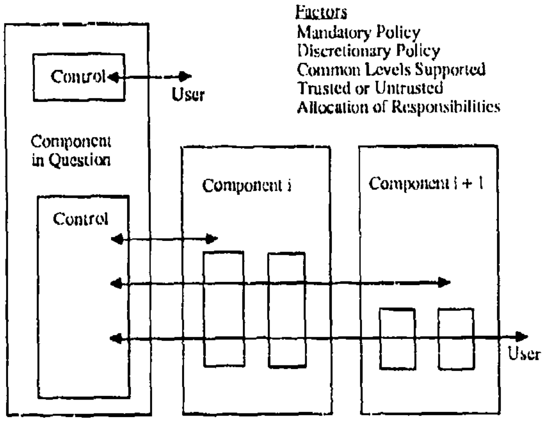
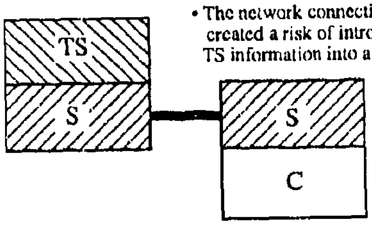


Figure 3. Interconnection Policy



• The network connection has created a risk of introducing TS information into a C-S system

Figure 4. Cascading Problem

When we consider the security policy from an overall system level, it must be assured that all component policies are supported throughout the system (including both physical and logical interface). Further, there may be policy dictated at the system level that is over and above the policy that exists at the individual component level, and this higher level policy must also be supported. Finally, there is policy at the system level which concerns the system's interface with the outside world, and it must be ensured that this system level policy is supported by the components that interface with the outside world (external subjects to the system).

Security Risk

The goal of a security program is to prevent the disclosure of sensitive information to unauthorized sources and to protect the integrity and availability of the systems and the data critical to mission operations. This goal is accomplished through the process of risk management. Risk management attempts to:

o Identify, control, and minimize the occurrence and effect of uncertain events that would compromise the security goals

o Obtain and maintain the authority for approval of operations involving sensitive or critical data and/or functions through a Designated Approval Authority (DAA)

o Facilitate information system management throughout the system's life cycle based on security requirements and protection levels.

Risk Modeling is a method of correctly determining evaluation criteria for specification, design/development, and accreditation purposes. This paper presents an approach which extends Yellow Book and Landwehr-Lubbes methods to complex networks.

Yellow Book Guidelines - The National Computer Security Center developed the Orange Book to identify protection requirements associated with a gradation of risk levels. To assist in the assessment of risk level the NCSC also provided the Yellow Book guidance (CSC-STD-003-85, illustrated in Figure 5). The Yellow Book considers these parameters: the maximum sensitivity of the data to be protected by the system; the user with the minimum clearance level who potentially has access to the system; and whether or not the system was developed in an open or closed environment. A closed environment exists where there is adequately secure design and development with proper configuration control and assurance. Yellow Book risk indices (exposure levels) are summarized in Table 1.

The Yellow Book guidelines also make recommendations on security mode of operation based on the degree of exposure (maximum data sensitivity level minus minimum user clearance level). Data exposure in a dedicated mode or system high environment is by definition zero, and in controlled or multi-level environments the potential exposure is equal to the separation between the high and low levels being protected.

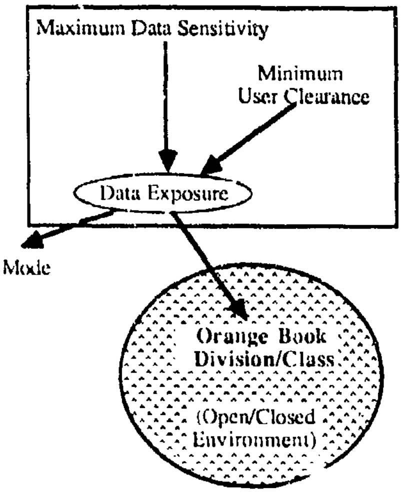


Figure 5. Yellow Book Approach

Table 1. Exposure Levels

Maximum Data Sensitivity	Minimum User Clearance							
	0 U	1 N	2 C	3 S	4 TS/BI	5 TS/SBI	6 IC	7 MC
0 U	0							
1 N	1	0						
2 C or >1	2	1	0					
3 S or C + >1	3	2	1	0				
4 S + >1	4	3	2	1	0			
5 TS or S + 2	5	4	3	2	1	0		
6 TS + 1	6	5	4	3	2	1	0	
7 TS + >1	7	6	5	4	3	2	1	0

Landwehr - Lubbes Approach - In order to try to loosen the strict guidance of the Yellow Book and to consider other variables, the Landwehr-Lubbes approach uses the fact that different users possess different capabilities, thereby potentially reducing the identified risk and criteria levels. In addition to the data exposure parameters of the Yellow Book, this approach considers the user capability, nature of the communications path and local processing capability. Thus, users may be categorized by risk levels. Expanding on the previous figure, Figure 6 shows the addition of the Landwehr - Lubbes criteria which combines system risk with data exposure to determine criteria levels. The matrices for determining process coupling risk and system external risk are shown in Tables 2 and 3. Table 4 shows how to use data exposure and system external risk levels to arrive at an Orange Book evaluation criteria division and class for sensitivity.

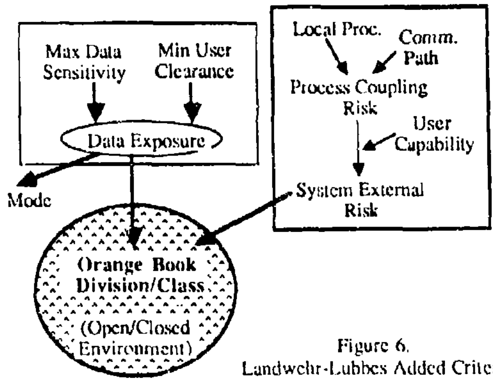


Figure 6.
Landwehr-Lubbes Added Criteria

Table 2. Process Coupling Risk

Local Processing Capability	Communication Path		
	1. S/F Net (one way)	2. S/F Net (two-way)	3. LA Net or Direct Connection (LAN,DDN)
1. Receive-only Terminal	2	3	4
2. Interactive Terminal (fixed function)	2	4	5
3. Programmable Device (Access via PC or programmable host)	4	5	6

Table 3. System External Risk

User Capability	Process Coupling Risk (From Table 2)				
	2	3	4	5	6
1. Output-only (Subscriber)	3	4	5	6	7
2. Transaction Processing (Analyst)	-	5	6	7	8
3. Full Programming	-	6	7	8	9

Table 4. Orange Book Levels

Sensitivity Data Exposure (from Table 1)	System External Risk (from Table 3)						
	3	4	5	6	7	8	9
0	C1	C1	C1	C1/C2	C2	C2	C2
1	C1/C2	C2	C2	C2	C2/B1	B1	B1
2	C2	C2/B1	B1	B1	B1	B1/B2	B2
3	B1	B1	B1/B2	B2	B2/B3	B3	B3/A1
4	B2	B2/B3	B3	B3-A1	A1	A1	A1
5	B3/A1	A1	A1	-	-	-	-
6							
7							

Table 5. Mapping System Risk using Criticality Division

Criticality Data Exposure	System External Risk						
	3	4	5	6	7	8	9
0	C	C	C	C	C	C	C
1	C	C	C	C	C	B	B
2	B	B	B	B	B	B	B
3	B	B	B	B	B	B	A
4	B	B	B	A	A	A	A
5	A	A	A	-	-	-	-
6							
7	-	-	-	-	-	-	-

Security Risk in Networks. Data exposure and the Landwehr-Lubbes criteria appear to be equally applicable to criticality. Table 5 can be used to determine the appropriate division (A, B or C) of criticality criteria. Prototype evaluation criteria for criticality divisions are described in [7]. Factors for applying the risk methodology to criticality as well as to sensitivity are described below. The method presented here, including rules that utilize the Landwehr-Lubbes matrices, accounts for the propagation of risk in networks.

Network Concatenation/Propagation Rules - It was an objective to follow the principles of the Yellow Book and it also seemed that the essence of risk in the distributed system problem was embodied in the capabilities possessed by the remote users. Before these rules could be applied it was first obvious that the cascading effect of both maximum data sensitivity and minimum user clearance would have to be dealt with. Further, the communication of a user with a remote computer system might not be only through a variety of communications links, but also through systems that may or may not be trusted and may or may not take responsibility for the data and its communication.

The approach taken (Figure 7) was to identify concatenation and propagation rules that applied to the maximum data level being protected (e.g., through the cascading effect), to the minimum user clearance level protected against, and finally to the interpretation of multiple (and remote) communications paths. The rules adopted for maximum data sensitivity and minimum user clearance are given in Figure 8. If we are evaluating System A with respect to system B, then system A assumes the maximum data sensitivity level equal to the maximum of A and B if there are no trusted absorbing nodes or if there are no one-way data lines that only carry data in the direction from A to B.

A "trusted absorbing node" is a node that has a trusted system base at the appropriate level, takes and controls information that comes into it via security policy that considers trust levels of the systems with which it interfaces and controls communications with the destination. A node is not trusted absorbing either if it is not trusted or if it is trusted but merely acts as a store and forward switch in the communications system, taking no responsibility for the labels or the policy associated with the communications.

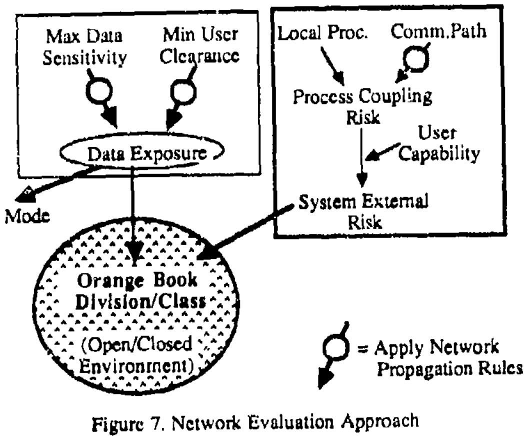


Figure 7. Network Evaluation Approach

Propagation of minimum user clearance is defined similarly; however, note that the direction of the one-way rule is reversed. Both of these rules apply to criticality as well as sensitivity, with the exception that the directions of the one-way rules are reversed in both cases. In criticality we are worried about writing and activating, and less worried about data exposure.

To enhance the Landwehr-Lubbes criteria, we further expand the criteria used to interpret a complex path to determine the matrix value for "communications path" to use in the process coupling risk matrix (previously given in Table 2). These additional criteria are given in Figure 9, where trusted absorbing node and one-way are defined as before. Two-way is defined as in the original Landwehr-Lubbes paper, where there is a two-way store-and-forward capability, but no direct interaction.

Rule 1: Maximum Data Sensitivity

(Evaluate A with respect B)

If trusted absorbing nodes in path, or
one-ways away from A, then
Amax = Amax.
Otherwise Amax = Max(Amax,Bmax)

Rule 2: Minimum User Clearance

(Evaluate A with respect to B)

If trusted absorbing nodes in path, or
one-ways in the direction of A, then
Amin = Amin.
Otherwise Amin = Min(Amin,Bmin)

(For criticality the one-way rules are reversed)

Figure 8. Network Propagation Rules

Rule 3:
To determine comm. path for Table 2

If one-way in direction of A, or trusted absorb. node in path	- No path
If one-way away from A	- 1
If two-way	- 2
Otherwise (e.g., LAN)	- 3

Figure 9. Network Propagation Rules (cont.)

Risk Evaluation Model and Examples - The determination of evaluation criteria in networked systems can now be accomplished by applying the concatenation and propagation rules, and then performing the evaluation implied by the original approaches of the Yellow Book and/or Landwehr-Lubbes criteria.

The problem is not a simple one as can be seen from the simple network example. In theory, every path from each source to each destination element must be considered in this evaluation, or must at least go as far as is required to show that there will be no cascading of security properties. Development of the algorithm into an automated software tool is in progress. This tool facilitates the engineering process, since all but the simplest of analyses become too complex to deal with manually, as will be illustrated.

Any model must have implicit and explicit simplifying assumptions. In our model, the entire threat is through system users who have limited access. It is assumed that computers are physically protected and that communications lines are either physically protected or the data are protected with encryption and integrity encoding. It is also assumed that interface policies have been devised and that these policies can be enforced by trusted systems. As an example, if a trusted system receives data from an untrusted system, it will not trust the labels and will treat those data at system high level of the untrusted system.

The definitions of nodes, systems, and terminals are left to the judgement of the evaluator and ultimately the DAA. Full capability high performance microprocessors might be treated as systems. The definition also might differ depending on whether a sensitivity or a criticality analysis is being made. For example, a network node may be performing routing and other processing based on protocol information, and labels. Other simplifications will be apparent as we go through an example.

The procedure for evaluating risk (i.e., determining protection levels) in heterogenous networks is summarized as follows. Consider the risk evaluation of System A in a network (see Figure 10). For each potential path to system A from each external subject:

- Determine max. data sensitivity (rule 1)
- Determine min. user clearance (rule 2)
- Determine path data exposure (table 1)
- Determine communication path (rule 3)
- Determine process coupling risk (table 2)
- Determine system external risk (table 3)
- Map system risk and data exposure to Orange Book level (table 4)

This yields criteria level for that path. Security requirements and associated protection mechanisms for each path must be analyzed and validated. System A risk level becomes the worst case path. The analysis is repeated for System A criticality threat. Finally, the process is repeated for all systems in the network.

As an example, consider a very small part of the system in Figure 10, consisting only of systems A, B, and C as well as the user terminal connected at B. Here we are performing the evaluation only with respect to A. This evaluation example is shown in Figure 11. We are performing only a sensitivity evaluation; however, a criticality evaluation would follow similarly, but using the slightly altered concatenation/propagation rules and different risk matrices.

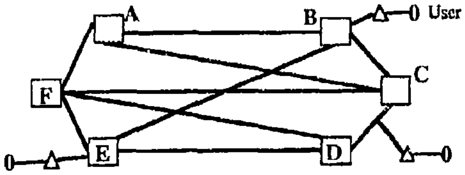


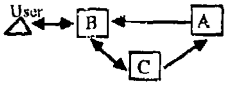
Figure 10. Evaluation of System A in a Network

The elements are given starting states and from these it is determined how risk is propagated into A. The user at B has a Confidential clearance, systems B, A, and C respectively have Confidential, Secret, and Secret minimum user clearance levels. They also possess, respectively, Secret, Secret and Top Secret, maximum data sensitivity. Neither A, B, or C are trusted absorbing nodes. Programming can be accomplished at the terminal and, once a user logs on, programming could be done at any of systems A, B, and C. Two-way (store and forward) links exist between the terminal and system B and between systems B and C. A one-way link connects system A and B where data can travel from A to B, but not in the other direction. Another one-way data link allows flow of data from C to A, but not in the other direction.

Evaluating the results shows a path from the terminal to B to A, but it is one-way and rates a 1 in the Landwehr-Lubbes criteria. The path through B and then C is not considered a path because of the one-way in the wrong direction. (Note that Landwehr-Lubbes is worried about leakage of sensitive information but is not concerned with the user being able to send data into A, which is a criticality problem.) The minimum user clearance in A must be updated to Confidential since data from A is now exposed to that level. Further, the maximum data sensitivity of A must be updated to Top Secret since there is a potential leakage path from C to A.

Now we are able to assess the risk level at System A (only) based on the information given in this simple example, and from that determine the applicable Orange Book level. Using Table 2, the process coupling risk is determined to be a 4. Further, the system external risk is determined to be a 7 from Table 3. The data exposure between Confidential user and Top Secret data from Table 1 is determined to be 3. Based on this exposure, the Yellow Book would recommended a B3 class (as with Landwehr-Lubbes, open environments are assumed). From Table 4, the Landwehr-Lubbes approach (with network propagation effects factored in) recommends a B2/B3 level of criteria.

Evaluate System A with respect to B & C
(Potential paths are BA and BCA).



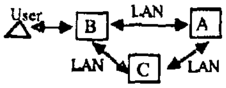
Initial Parameters	User at B	System B	System C	System A
Min User Clearance	C	C	S	S
Max Data Sensitivity		S	TS	S
Trusted Absorption		No	No	No
Local Process Capab.	P			
User Capability		P	P	P

Risk Calculation for Path BA (BCA is not a valid path in this case).

Max Data Sensitivity	(rule 1):	max (A,B,C) = TS
Min User Clearance	(rule 2):	min (A,B) = C
Path Data Exposure	(table 1):	(5,2) = 3
Comm Path	(rule 3):	1
Process Coupl. Risk	(table 2):	(3,1) = 4
System Ext. Risk	(table 3):	(3,4) = 7
Orange Book Level	(table 4):	(3,7) = B2/B3
		(Yellow Book => B3)

Figure 11. Risk Evaluation Example (1)

Evaluate System A with respect to B & C
(Potential paths are BA and BCA).



Initial Parameters	User at B	System B	System C	System A
Min User Clearance	C	C	S	S
Max Data Sensitivity		C	TS	S
Trusted Absorption		No	No	No
Local Process Capab.	P			
User Capability		P	P	P

		Risk Calculation for Path BA	Risk for Path BCA
Max Data Sensitivity	(rule 1):	max (A,B) = S	Max (A,B,C) = TS
Min User Clearance	(rule 2):	min (A,B) = C	Min (A,B,C) = C
Path Data Exposure	(table 1):	(3,2) = 1	(5,2) = 3
Comm Path	(rule 3):	3	3
Process Coupl. Risk	(table 2):	(3,3) = 6	(3,3) = 6
System Ext. Risk	(table 3):	(3,6) = 9	(3,6) = 9
Orange Book Level	(table 4):	(1,9) = B1	(3,9) = B3/A1
		(Yellow Bock => B1)	(Yellow Bool. => B3)

Figure 12. Risk Evaluation Example (2)

We purposely went through this first example step by step, relating it to the appropriate rules and tables. If we were to evaluate the network in Figure 10 just with respect to system A, we would have to consider each potential path from each user and from each of the other systems to system A. A Local Area Network evaluation example is presented in Figure 12. Studying numerous examples and results of the automated evaluation tool provides insight into network security problems. One revelation is that the security design solution with respect to node A may be not changing node A at all. The solution may be to insert a more restrictive communication link on the other side of the network to reduce A's exposure. Although this is intuitively obvious for simple networks, it is less obvious in complex networks.

Conclusions

We have presented a deterministic approach for dealing with the distribution of risk in connected systems. The methodology is more qualitative than quantitative, since many risk factors are difficult to quantify precisely. We used as a starting point the NCSC Yellow Book guidance and the Landwehr-Lubbes approach. The evaluation methodology described here enables consistent determination of network criticality and sensitivity evaluation criteria (i.e., requirements). This approach may also be adapted for other than DoD environments, where a hierarchical set of security requirements exists. The risk evaluation methodology described here has been programmed to simulate many different system environments.

References

[1] Johnson, H.L. and J.D. Layne, "A Mission-Critical Approach to Network Security," Proceedings of the 10th National Computer Security Conference, September 1987, pp. 15-24

[2] Biba, K.J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April 1977

[3] CSC-STD-004-85, Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements, June 1985

[4] DoD 5200.28-STD, Trusted Computer System Evaluation Criteria, December, 1985

[5] Landwehr, C.E. and H.O. Lubbes, "Determining Security Requirements for Complex Systems with the Orange Book," Proceedings of the 8th National Computer Security Conference, September 1985, pp. 156-162

[6] NCSC-TG-005, Trusted Network Interpretation, 31 July 1987

[7] CTA, Draft AFSPACECOM Trusted System Evaluation Criteria, NCCS-86-03-383, March 1987

Automated Audit Trail Analysis and Intrusion Detection: A Survey

Teresa F. Lunt

SRI International, Computer Science Laboratory
333 Ravenswood Ave., Menlo Park, CA 94025

Abstract

Today's computer systems are vulnerable to both abuse by insiders and penetration by outsiders, as evidenced by the growing number of incidents reported in the press. Because closing all security loopholes from today's systems is infeasible, and since no combination of technologies can prevent legitimate users from abusing their authority in a system, auditing is viewed as the last line of defense. What is needed are automated tools to analyze the vast amount of audit data for suspicious user behavior. This paper presents a survey of the automated audit trail analysis techniques and intrusion-detection systems that have emerged in the past several years.

1 Introduction

The last few years have seen a sudden and growing interest in automated security analysis of computer system audit trails and in systems for real-time intrusion detection. There is a growing number of research activities devoted to the subject, and some operational systems and even a few commercial products have appeared.

The earliest work on the subject was a study by Jim Anderson [1], who categorized the threats that could be addressed by audit trail analysis as

- External penetrators (who are not authorized to use the computer)
- Internal penetrators (who are authorized to use the computer but not the data, program, or resource accessed), including
  - Masqueraders (who operate under another user's id and password)
  - Clandestine users (who evade auditing and access controls)
- Misfeasors (who are authorized to use the computer *and* resources accessed but misuse their privileges)

Anderson suggested that external penetrators could be detected by auditing failed login attempts and that some would-be internal penetrators could be detected by observing failed access attempts to files, programs, and other resources. He suggested that masqueraders could be detected by observing departures from established patterns of use for individual users. All of these approaches have been adopted in subsequent studies.

Anderson offered no suggestions for detecting legitimate users who abuse their privileges. To detect such abuse however, *a priori* rules for *acceptable* behavior could be established; this approach has been taken in a few studies. Comparison with the norm established for the class of user to which the user belongs also could detect abuse of privilege; this approach is under consideration by the research group at SRI.

The clandestine user can evade auditing by using system privilege or by operating at a level below which auditing occurs. The former might be detected by auditing all use of functions that turn off or suspend auditing, change the specific users being audited, or change other auditing parameters. The latter might be addressed by low-level auditing, such as auditing system service or kernel calls. Anderson suggested monitoring certain system-wide parameters, such as CPU, memory, and disk activity, and comparing these with what has been historically established as usual or normal for that facility. At least one subsequent study has included this approach [2].

2 The Experiments

Subsequent to Anderson's study, early work focused on developing procedures and algorithms for automating the offline security analysis of audit trails. The aim of such algorithms and procedures was to provide automated tools to help the security administrator in his or her daily assessment of the previous day's computer system activity [3,4]. One such project used existing audit trails and studied possible approaches for building automated tools for audit trail security analysis [3]. Another such project considered building special security audit trails and studied possible approaches for their automated analysis [4]. These projects provided the first experimental evidence that users could be distinguished from one another based on their patterns of use of the computer system [3], and that user behavior characteristics could be found that could be used to discriminate between normal user behavior and a variety of simulated intrusions [4].

2.1 The Sytek Work

A tool that ranked user sessions by their suspiciousness would allow the system security officer to analyze audit trail records that are most likely to represent intrusions without having to wade through volumes of records of mostly normal user activity. The Sytek work sought to provide a feasibility demonstration for such a tool [5].

Sytek's work was guided by concepts from pattern recognition theory. User sessions were recognized as normal or intrusive based on patterns formed by the individual records on the audit trail for that session. The Sytek study defined several audit record *features* as functions of the audit record fields. For each user, expected values for the features were determined through a process called *training* (that is, for each feature, the set or range of values was determined from the audit data). The study then tested the features for their ability to discriminate between normal sessions and sessions containing staged intrusions. A session was flagged as intrusive by a feature if the value of the feature calculated for the session was outside the user's range or set of expected values. Features that successfully detected the staged intrusions were combined to create a for each user *user profile*—the collection of the normal ranges for each feature.

Sytek wrote software to collect audit data from a Unix[1] system that were analogous to data available in general-purpose operating systems. An audit record containing the command name, associated files, process statistics, and file statistics was generated whenever a user issued a command.

The Sytek team collected one week of audit data and generated a set of statistics to identify features of the audit trail that were potentially useful in discriminating among users [6]. Each identified feature was trained on the presumedly normal audit data to establish a range or set of expected values exhibited during each user's sessions. The Sytek team then enacted and audited various intrusion scenarios in such a way that the intrusions were embedded into the audited behavior of legitimate users of the system [7]. The intrusion scenarios included break-ins by outsiders, intruders and legitimate users masquerading as other users, and users deliberately subverting the system in various ways. Sytek then tested the selected features to see whether they were useful in detecting the simulated intrusions [8]. Those features that detected one or more of the simulated intrusions were retained for further study.

Sytek then tested each feature still under consideration against an additional week of (presumedly normal) audit data to determine the percentage of normal sessions the feature flagged as abnormal (i.e., the false-alarm rate). Sytek found that the features *password changed*, *user identity queried*, and *access to system dictionary* performed extremely well. It found the most effective file statistics were *device on which the accessed file resides*, *file size*, *oversized file associated with this command*, *group id of the owner of the accessed file*, and *user id of the owner of the accessed file*. The most effective process statistics were *time of use*, *day of use*, *user program CPU time*, and *maximum total memory use*. These 12 features had low (under 15 percent) false-alarm rates and were selected for use in a pattern classifier that analyzed their composite performance [9,10].

The pattern classifier flagged those sessions that did not fall within the pattern defined by the user profiles. The idea was that the resultant set of flagged sessions should be sufficiently small to enable a security officer to examine the set manually.

The performance of the pattern classifier could be different from that of the individual features taken separately, because (1) if several features individually each flagged a certain session, the composite would flag that session only once, so the composite could flag *fewer* normal sessions and thus have *better* performance than the features taken individually and (2) one feature might not flag the same normal sessions as another feature, so the combination of features could flag *more* normal sessions and thus have *worse* performance than the features taken individually.

Sytek also attempted to compute a *certainty measure* that would indicate the degree of certainty that a flagged session actually represented an intrusion or the degree of suspicion for a user session. They computed a certainty for each feature, namely, the number of audit records within a session that were flagged by the feature. They then computed a certainty for a session as the sum of the certainties for all 12 features.

In tests to analyze for intrusion-detection strength and false-alarm rate, the pattern classifier successfully detected all the simulated intrusions. However, the false-alarm rate was high (between 40 and 70 percent). Much better performance could be expected if a longer training period were used.

Four of the selected features pertained to a user's command usage patterns. These features were very good at detecting the intrusion scenarios but had very high false-alarm rates. Believing that command usage patterns were potentially very useful in discriminating between normal and abnormal behavior, Sytek decided to modify these features to improve their performance. It decided to make these features *fuzzy*; that is, to allow the computed value for a user's session to be a certain distance from the range in the user's profile before the session was flagged as abnormal. The greater the fuzziness, however, the greater the chance of missing intrusions. For each of the four measures, Sytek analyzed the effect of increasing the fuzziness on both the intrusion-detection strength (percentage of scenarios flagged) and on false-alarm rate (percentage of normal sessions flagged). To reach an acceptable false-alarm rate, the intrusion-detection strength was also greatly reduced. Sytek found, however, that for features *proportional real duration of command* (percentage of a session's real elapsed time spent in the command) and *proportional CPU duration of command* (percent of a session's CPU time spent in the command), an acceptable false-alarm rate was achieved with a relatively modest reduction in intrusion-detection strength. Hence, these two features showed considerable promise as discriminators of intrusive behavior.

2.2 The SRI Study

A group at SRI led by Hal Javitz performed an extensive statistical analysis on audit data from IBM systems running MVS and VM. The purpose of the study was to develop analytical statistical techniques for screening computer system accounting data to detect user behavior indicative of intrusions. A high-speed algorithm was developed that could accurately discriminate among users based on their behavior profiles.

Audit data were obtained from normal system accounting records for IBM VM and MVS systems. Because the overwhelming majority of information in the accounting records concerned system usage parameters that either were beyond the control of the user, bore no reasonable relationship to intrusion attempts, or were redundant with information contained on other accounting records, subset of the accounting data was selected that could be used to detect logon activity outside the user's normal range, execution time outside the user's normal range, activity from other than a user's normal set of input or output (I/O) devices, and files for which the user is authorized but which are only infrequently accessed by the user.

From VM accounting records the research group obtained logon time and day of week, logon mode (normal or autolog), terminal used, minidisks accessed, connect time, computer system accessed, CPU time, number of page reads and writes, number of virtual cards read and punched,

[1] Unix is a trademark of AT&T.

and number of virtual lines printed. From MVS accounting records they obtained time and day of week, computer system used, job priority, CPU time, number of job steps, I/O units, card reader class and type, number of card images, number of programs run, number of files accessed, number of changes of index levels, and number of VSAM records inserted, retrieved, updated, or deleted.

They created a database containing 4 weeks of audit data for a set of representative users. They selected a job as the unit of analysis for batch users and a session for interactive users (both are referred to as jobs hereinafter) and characterized each job by variables such as CPU time used, terminal used, and the number of files accessed. They then used these variables to construct a profile, representing the statistical norm for the user's jobs, for each user.

The SRI researchers developed statistical techniques to detect jobs that deviated from the profiles developed for each user's jobs. The statistical algorithms aimed to maximize the *true positive rate* (percentage of intrusions identified as abnormal) and minimize the *false positive rate* (percentage of normal use identified as abnormal). A true positive rate of 100 percent means that all intrusions are detected. The *detection ratio* is the true positive rate divided by the false positive rate. The more variable user behavior is, the higher the false positive rate will be. Although the false positive rate can be reduced by raising the threshold of the statistical test (so that fewer events are considered abnormal), this also lowers the true positive rate.

They compared the results obtained when the following criteria were used to define an abnormal job:

- A job is abnormal if the value of one of its parameters has a probability of occurring of less than 10 percent for that user.
- A user's job is abnormal if a discriminant analysis shows that the probability that the job was submitted by another user is greater than the probability that the job was submitted by the user in question.

They split the 4 weeks of audit data into two sets: the training set (the first 2 weeks of data), used to determine the statistical thresholds for the test, and the verification set (second 2 weeks of data), used to measure the test's performance.

Because all activity represented in the sample database was assumed to be normal, there were no examples of intrusive behavior. Thus, for each measure the researchers calculated a *surrogate* true positive rate (the probability of identifying a user's job as abnormal when measured against another user's job profile) and a *surrogate* false positive rate (the probability of identifying a user's own job as abnormal). These are the true and false positive rates discussed below.

For VM sessions, by far the best indicators were the type of login and the terminal used; both of these had extremely low false positive rates (and low to medium true positive rates, but very high detection ratios). Minidisk id had an extremely high true positive rate, but also a high false positive rate. The computer system used had a fairly high true positive rate and a low false positive rate, with a detection ratio of seven. Most other characteristics had detection ratios of between one and two.

Discriminant analysis was superior to measures of job abnormality based on extreme values of job parameters. In the discriminant analysis, the researchers used a user's training set to estimate the multivariate probability distribution (with respect to parameters such as CPU time, time of day, etc.) of normal jobs for that user. They assumed a certain multivariate probability distribution for intrusive jobs. They then used classical statistical paradigms to determine a rule for classifying a job as normal or abnormal. With this approach, every point in the multivariate space is assigned a value equal to the ratio of the height of the intrusive job probability distribution to the height of the normal job probability distribution. The points with the largest ratios form a *critical region* in which the probability of normal jobs belonging to that region is less than a few percent. Once a user's critical region has been determined, a new job for that user can be considered abnormal if it falls into the critical region, and normal otherwise.

Because audit data containing intrusive jobs were not available, two different approaches were taken to determine a hypothetical multivariate probability distribution for intrusive jobs:

- Nonintrusive profile approach: Assume that intrusive jobs have a uniform distribution
- Surrogate intrusive profile approach: Use other users' jobs to develop a probability distribution for intrusive jobs

For MVS jobs, discriminant analysis produced a true positive rate of over 90 percent and a false positive rate of only 6 or 7 percent. For VM sessions, although the single parameter rules had by far the lowest false positive rates (averaging less than 1 percent), the discriminant analysis method had a much higher true positive rate (over 80 percent). The false positive rate was shown to increase with the number of days since the profile was last updated. The SRI group estimated that with daily profile updating the results would be even better. If additional security-relevant audit data were used in the analysis, they estimated that a discriminant analysis would produce a true positive rate of 90 to 98 percent and a false positive rate of 1 to 3 percent. Thus, these statistical procedures are potentially capable of reducing the audit trail by a factor of 100 while detecting approximately 95 percent of all intrusions [3]. The security officer would still have to determine whether the statistical abnormalities represent actual intrusions.

3 The Intrusion-Detection Systems

The early evidence of the Sytek and Javitz studies was the basis for a real-time intrusion-detection system, that is, a system that can continuously monitor user behavior and detect suspicious behavior as it occurs. This system, known as IDES (Intrusion-Detectio xpert System), is based on the approach that intrusions, whether successful or attempted, can be detected by flagging departures from historically established norms of behavior for individual users [12,13].

Another real-time approach, called *keystroke dynamics*, is based on measurements of certain characteristics, such as typing speed, of a user's keyboard activity. Keystroke dynamics has been found to be a powerful means of continuously verifying the identity of the user doing the typing.

For systems like IDES, different intrusion-detection measures may be appropriate to different classes of user.

For example, for users whose activity is almost always during normal business hours, an appropriate measure might simply track whether activity is during normal hours or off hours. Other users might frequently login in the evenings as well, yet still have a distinctive pattern of use (e.g., logging in between 7 and 9 p.m. but rarely after 9 or between 5 and 9); for such users, an intrusion-detection measure that tracks for each hour whether the user is likely to be logged in during that hour would be more appropriate. For still others for whom "normal" could be any time of day, a time-of-use intrusion-detection measure may not be meaningful at all.

There are obvious difficulties with attempting to detect intrusions solely on the basis of departures from observed norms for individual users. Although some users may have well-established patterns of behavior—logging on and off at close to the same times every day and having a characteristic level and type of activity—others may have erratic work hours, may differ radically from day to day in the amount and type of their activity, and may use the computer in several different locations and even time zones (in the office, at home, and on travel). For the latter type of user, almost anything is normal, and a masquerader might easily go undetected. Thus, the ability to discriminate between a user's normal behavior and suspicious behavior depends on how widely that user's behavior fluctuates and on the user's range of normal behavior. And although this approach might be successful for penetrators and masqueraders, it may not have the same success with legitimate users who abuse their privileges, especially if such abuse is normal for those users. Moreover, the approach is vulnerable to defeat by an insider who knows that his or her behavior is being comparea with his or her previously established behavior pattern and who slowly varies their behavior over time, until they have established a new behavior pattern within which they can safely mount an attack. Trend analysis on user behavior patterns, that is, observing how fast user behavior changes over time, may be useful in detecting such attacks.

Because the task of discriminating between normal and intrusive behavior is so difficult, another study has taken the straightforward approach of automating the security officer's job. Such an approach lends itself to traditional *expert system* technology, in which the special knowledge of the intrusion-detection experts (the system security officers) is codified as rules used to analyze the audit data for suspicious activity. The obvious drawback to this approach is that the security officers, in practice, have only limited expertise. Thus, while automating these rules frees the security officer to perform further analysis, such rules cannot be expected to be comprehensive. This approach would be more aptly called a security officer's assistant.

Several study teams are attempting to comprehensively characterize intrusions (e.g., MIDAS [2]). These systems encode information about known system vulnerabilities and reported attack scenarios, as well as intuition about suspicious behavior, in rule-based systems. The rules are fixed in that they do not depend on past user or system behavior. (An example of such a rule might be that more than three consecutive unsuccessful login attempts for the same user id within 5 minutes is a penetration attempt.) Audit data from the monitored system are matched against these rules to determine whether the behavior is suspicious. A limitation of this approach is that it looks for known vulnerabilities and attacks, and the greatest threat may be unknown vulnerabilities and the attacks that have not yet been tried; one is in a position of playing catch-up with the intruders.

Below is a survey of these various intrusion-detection systems.

3.1 A Priori Rules for Normal Program Behavior

Paul Karger has suggested what he calls a *knowledge-based name checker* to compare the names and types of objects requested (for reading, writing, creation, or destruction) by a program with the names and types of objects expected for the program [11]. He posits as an example a FORTRAN compiler containing a Trojan horse that surreptitiously modifies a user's LOGIN.CMD file while compiling the user's program. The name checker expects the FORTRAN compiler to require read access to a file with a user-supplied name and a suffix of .FOR and to create new files with the same name but suffixes of .OBJ and .LIS. If the compiler attempts to create or to write to a file named LOGIN.CMD, the name checker would recognize that such a file is unexpected for the FORTRAN compiler. Other rules, for a Unix system for example, could check whether a user program asks for **set-uid** privileges.

Although Karger envisions the name checker being used for access control decisions, it could also be used as a rule-based form of real-time intrusion-detection. He suggests obtaining the rules for the behavior expected of commands from information already known to the computer system; for example, in VAX/VMS from the command definition tables. For user programs and batch jobs, the user would encode the rules in what Karger calls a *special directory tree*, which would enumerate the objects on which the program is expected to operate.

3.2 IDES

SRI International is developing a prototype intrusion-detection system called IDES (Intrusion-Detection Expert System) [12,13]. The goal of IDES is to provide a system-independent mechanism for real-time detection of all types of security violations, whether they are initiated by outsiders who attempt to break into a system or by insiders who attempt to misuse the privileges of their accounts. The IDES approach is based on the hypothesis that any exploitation of a computer system's vulnerabilities entails behavior that deviates from previous patterns of use of the system; consequently, intrusions can be detected by observing abnormal patterns of use. The IDES prototype is based on the IDES model developed by Dorothy Denning [14,15]. This model is independent of any particular target system, application environment, system vulnerability, or type of intrusion, thereby providing a framework for a general-purpose intrusion-detection system.

The IDES prototype is an independent system that runs on its own hardware (a Sun Workstation[2]) and processes audit data received in real time from a target system [12,13]. The user activity monitored by the IDES prototype includes login, logout, program execution, directory modification, file access, system call, session location change, and network activity.

[2] Sun Workstation is a trademark of Sun Microsystems, Inc.

IDES is driven by the arrival of audit records, each of which describes behavior relevant to possibly several intrusion-detection measures. (A *measure* is an aspect of user behavior.) There are two kinds of measures: *discrete* and *continuous*. A *discrete* measure is one whose domain of values is a finite, unordered set (e.g., the set of locations). Such measures are generally constant for a particular user session, for example location and time of login. A *continuous* measure is one whose value is a number or count that accumulates over a user session (e.g., connect time, CPU time, and I/O activity).

To determine whether user behavior as reported in the audit data is normal with respect to past or acceptable behavior, IDES includes user behavior profiles for the measures. (A *profile* is a description of the expected behavior of a user with respect to a particular measure.) The profiles are periodically updated based on observed user behavior, and the profile data are aged using a decay factor that gives the data a half-life of 50 days. Thus, the profile reflects a moving time window of behavior for each user. *Anomalous* behavior is user behavior that deviates from the expected behavior for some measure by an amount indicated in the user profile for that measure. Because IDES can be configured to monitor arbitrarily detailed user behavior, it is potentially capable of detecting intrusions (for example, by masqueraders) that cannot be detected by the target system's access controls.

The IDES prototype has demonstrated its ability to adaptively learn users' behavior patterns; as users alter their behavior, the thresholds maintained in the profiles change. This capability makes IDES a flexible system: it does not have to be given rules determined by a human expert in order to learn what constitutes suspicious behavior; rather, IDES derives its own rules. Thus, IDES is potentially sensitive to abnormalities that human experts may not have considered.

The IDES prototype currently monitors a DEC-2065 machine at SRI running a locally customized version of the TOPS-20 operating system[3]. SRI modified the TOPS-20 operating system to collect audit data, transform the data into the IDES format, encrypt the formatted data, and transmit the records to IDES according to the IDES protocol.

IDES's flexible system-independent audit record format and protocol for the transmission of audit records make it adaptable to different host systems without fundamental alteration (although the particular measures and parameters chosen will depend on the system and users being monitored). SRI's plans are to adapt IDES to monitor a network of Sun workstations and to monitor a large IBM mainframe system running MVS.

Now that the framework has been established, adding additional intrusion-detection measures to IDES is straightforward. In ongoing work, SRI is implementing a greater variety of intrusion-detection measures, including some "second order" measures to detect behavior trends, thereby improving the intrusion-detection capability of IDES. In addition, an expert system and rule-base that encodes information about hypothesized intrusion scenarios and suspicious behavior is being added to IDES.

The IDES intrusion-detection processes are implemented on a Sun 3/260, and the IDES security administrator interface is implemented on a Sun 3/60[4]. The security administrator interface maintains a continuous display of various indicators of user behavior on the monitored systems and allows the security administrator to choose from a menu of built-in queries or to build ad hoc queries against the audit data and profiles.

3.3 MIDAS

SRI's IDES prototype detects intrusions by flagging user behavior that deviates from that user's past behavior. Another approach is to develop an intrusion-detection system that encodes *a priori* rules that define an intrusion. This approach is used in the Multics Intrusion Detection and Alerting System (MIDAS), being developed by the National Computer Security Center to monitor a U.S. government Multics system [2].

MIDAS is implemented on a stand-alone Symbolics LISP machine. It uses a home-grown expert system shell, capable of 150 inferences per second, with a forward-chaining inference engine and an explanation facility. Its rules are elaborated in LISP, and statistical user profiles are maintained in LISP structures. The rules are compiled for fast performance. At the time of writing, MIDAS includes about 40 rules.

MIDAS is based on Denning's intrusion-detection model [15]. MIDAS monitors at the user command line level and logs all commands used. MIDAS uses four types of heuristic rules:

- *Immediate*—These are hard-and-fast rules that make no use of information of past or expected user behavior. They are intended to detect those events that, considered in isolation from any other information, are suspicious.

- *Anomaly*—These rules use statistical user profiles to detect when a user's behavior departs from a pattern established by observing past behavior. User profiles are updated at the completion of a user session. The profiles contain a list of the user's usual commands, the usual access times and location for the user, and the expected typing rate for the user. MIDAS also profiles the observed behavior of remote systems.

- *System-wide state*—MIDAS also can maintain a system-wide profile to characterize what is normal for the system globally. For example, an unusually high number of invocations of the copy command might indicate suspicious activity.

- *Sensitive path*—A command sequence can be characterized as abnormal if its probability of occurring is sufficiently low. This type of heuristic rule can also be used to determine whether a user's command sequence is similar to those characterizing a known or postulated attack. Attack scenarios are obtained through interviews with system security officers, hackers, and experts in penetration testing. Use of the sensitive-path heuristic rule could enable MIDAS to detect an attack in progress *before* the damage occurs. The sensitive-path type of heuristic rule is not currently implemented on MIDAS.

[3] DEC-2065 and TOPS-20 are trademarks of Digital Equipment Corporation.

[4] Sun 3/260 and Sun 3/60 are registered trademarks of Sun Microsystems, Inc.

MIDAS combines different intrusion indicators to decide whether an intrusion is occurring. A login time unusual for a given user, for example, is not alone sufficient to raise an alarm; but if combined with other anomalous data, however, MIDAS might decide an intrusion was in progress.

MIDAS's rules attempt to detect attempted break-ins, masqueraders, penetrators, Trojan horses, viruses, and misuse. To detect attempted break-ins, MIDAS uses rules involving password failure on a system account, login failure with an unknown user name, login attempt from outside the continental United States, and login attempt to a locked account. To detect masqueraders, MIDAS uses rules involving unusual login (e.g., time, location, terminal type), unusual commands or command patterns, invalid commands, and user logged in simultaneously from different locations. To detect penetrators, MIDAS uses rules involving attempted use of sensitive commands, attempted use of unauthorized commands, attempted access to sensitive objects, and attempted access to other people's objects. To detect misuse, MIDAS uses rules involving resource overuse, inactive session, and command out of scope for project. To detect Trojan horses and viruses, MIDAS uses rules involving attempted modification to system files and programs and unusual execution of predictable commands (e.g., who taking abnormally long).

A preprocessor on the Multics system formats the audit data for MIDAS. The preprocessor collects audit data from the usual Multics auditing program[5] and from additional audit collection software that was written specifically for MIDAS.

In its current implementation, audit data are accumulated in a Multics file and dumped to tape, and then the tape is fed into MIDAS. A real-time capability is planned for a later implementation phase.

3.4 Ask the Experts

TRW is developing an intrusion-detection system for the U.S. Government using traditional expert system technology [17]. The expert system rules attempt to characterize intrusions, either in general (what TRW calls generic common-sense rules), for the particular organization, or for the particular type of system and installation. The rules are obtained using standard knowledge-engineering techniques such as interviewing and working with system security officers. Known cases of intrusions drive the selection of the rules. The system security officers will be able to add new rules and modify old rules in the rule base.

This expert system is intended to do the work of the system security officer whose job now is to flip through huge printouts of audit traiis looking for problem areas. The benefit from the system is expected to be twofold: first, it will be able to analyze data that are too voluminous for the security officer to thoroughly analyze and to spot long-term trends; and second, it will provide a degree of proficiency that would otherwise be scarce, because experienced security officers are rare.

This system uses the audit trail already produced by the monitored system. Once suspicious activity has been identified, the system is intended to be used to build a case against the intruder. It does not operate in real time, but after the fact (like the security officers it mimics).

In one test, a feasibility system using 50 rules exhibited a false positive rate of about 12 percent, but detected the one intrusion planted in the 500 test cases. In addition, it detected at least one problem that had been thus far undetected. A prototype system is under development.

3.5 NAURS

The Network Auditing Usage Reporting System (NAURS) is used in conjunction with the Terminal Access Controller (TAC) Access Control System for the MILNET and the ARPANET [18,19].

NAURS monitors network activity originating from the TACs and network access controllers (NACs). It collects data about TAC/NAC logins, TAC/NAC login failures, logouts, open and close connections, and TACs coming online, and maintains the data in a database. NAURS provides both background analysis on past activity and real-time analysis of current use. It provides periodic audit trail reports and real-time reports on unusual events, triggered by the events themselves. Interactive query from local terminals is also supported. Incident reports, generated every day from the previous day's audit data, include incidents that satisfy one of three rules about the number of simultaneous logins and duration of sessions.

The threats addressed are twofold: break-in by an outsider (who has discovered a valid TAC id and password) and misuse by a legitimate user (trying to break into various network hosts). (Authorized TAC users are not generally registered users of every host on the network.)

A prototype NAURS exists on a separate machine from the SRI-NIC host. NAURS is not accessible for remote login or file transfer by network users. A planned production-quality NAURS will feature redundancy of equipment, distribution of functionality (five dedicated workstations have been proposed), ability to perform real-time detection of incidents, and redundancy of the audit database. Plans include reports on trends, such as 6-month differential-use trends of port usage (number of logins, length of sessions). Profiles will be maintained for users and devices. Some of these profiles will be established when an individual becomes a registered user, and others will reflect observed user behavior patterns. Proposed additional incident rules are unexpected host connection for a particular user, long idle periods, excessive connect time, simultaneous TAC logins with the same user id but not necessarily from the same TAC, excessive number of simultaneous logins from the same TAC, unusual time of day for a particular user, excessive number of unsuccessful logins from the same TAC and same user id, excessive number of unsuccessful host logins at different hosts, and excessive number of successful host logins at different hosts during the same TAC session.

3.6 Keystroke Dynamics

International Bioaccess Systems Corporation[6] offers a suite of products, collectively called Bioaccess System 2000, that perform intrusion-detection using keystroke dynamics. Among these are two products BioPassword and BioContinuous for biometric access protection for IBM personal computers (PCs).

[5] Because Multics is a B2 system, its auditing facilities satisfy the auditing requirements for B2 trusted computing systems as enumerated in the *Department of Defense Trusted Computer System Evaluation Criteria* [16].

[6] Bioaccess, BioPassword, BioContinuous, and BioNet are trademarks of International Bioaccess Systems Corporation.

Keystroke dynamics technology is based on the "fist of the sender" concept from the days of the telegraph when Morse Code operators could identify a sender by listening to the incoming signals. BioPassword produces an electronic signature based on the unique typing characteristics of each authorized user for keystroke-dynamics verification of user id and password. BioPassword is implemented on a board installed in the CPU socket of the mother board of IBM personal computers; no special keyboard is needed.

A user's electronic signature is stored in nonvolatile memory on the BioPassword board. The first time a person logs on the computer following installation of the board, the signature is developed by having the user type his or her id and password repeatedly (about 12 times). Once a user's electronic signature is installed, BioPassword operates transparently to the user.

BioPassword is automatically activated when the workstation is turned on or reset or when a user logs off, and can be invoked by software for reverification at any time. BioPassword prompts the user for an id and password and verifies both the contents and keystroke dynamics of each against the stored electronic signature, letting the user proceed with normal work only if the verification is positive. Once access is granted, if the workstation is idle for a prespecified period of time, BioPassword times out and requires reverification of the user before continuation of the idled job. All access attempts and logins are audited.

BioContinuous incorporates all of the features of BioPassword and adds continuous real-time verification of users. BioContinuous is a single-board component for the IBM PC. With its own high-speed processor, the BioContinuous board continuously verifies a user's identity in parallel with the user's work on the PC's processor, using over 110 typing characteristics, including intervals, rhythm, an analog of pressure, and error characteristics. After developing an electronic signature for a user id and password, BioContinuous develops an extended electronic signature for each user. This extended electronic signature contains additional biometric signature data that match a user's keystroke characteristics used in normal work. The learning process takes place over a few days, and, once the learn ing process is completed, the user's keystroke dynamics are automatically and continuously verified against his or her extended electronic signature.

BioContinuous includes a *programmable security matrix* containing information that indicates what actions are to be taken when a possible intruder is detected. The action can depend on risk factors, such as which data are being accessed or which function is being performed. Thresholds and alarms can be preselected.

International Bioaccess Systems Corporation is developing a product called BioNet that will add flexibility to PCs connected to a local area network by providing a central storage of electronic signatures. This will allow authorized users to use any workstation on the network without having to store their electronic signatures on each workstation. BioNet also will provide for integrated auditing of an individual user's activity across all the workstations on the local network.

3.7 Discovery

Discovery is an intrusion-detection expert system developed by TRW to address the intrusion threat in an environment in which computer services are sold to outside customers [20]. In such an environment, the customers may not be as concerned with safeguarding the security and integrity of the service provider's system and information assets as would, say, those users (employees of the service company) who create and maintain that information asset. Thus, the customer cannot be relied upon in any program of security instituted at the service company. The threat is that subscribing customers may give away their passwords, or lend their terminal devices, to would-be intruders. Thus, Discovery attempts to detect imposters who have obtained legitimate user ids, access codes, and inquiry formats. The perceived need was for an intrusion-detection mechanism that operates transparently to the subscribers of the service. The goal is that Discovery become a preventive, as well as a detective, control.

Discovery is an expert system that searches for frequently occurring patterns in subscriber inquiries to the data and compares these patterns to daily subscriber inquiry activity to detect variances in normal subscriber behavior. It develops a user profile for each customer by type of service and access method, and updates a user's profile daily if there has been activity for the user access code that day.

Discovery is system-specific in that the intrusion-detection rules are particular to the specific application-dependent data fields, or variables, being monitored. However, Discovery also monitors some variables that are generic to most computer systems, such as date and time of access, type of access, user location, user identifier, password, and port identifier. Discovery allows the security officer to choose the variables to be monitored and the threshold parameters, so that the system can be fine-tuned and the impact of adding new services can be determined. The security officer can also modify, delete, and add variables to be monitored as service offerings change. The thresholds can be set individually for each variable being monitored for each user access code.

Discovery analyzes the daily inquiry activity for each user access code for comparison with the established profile for the customer and also for comparison with a model of illegitimate access. Discovery only analyzes *correct* inquiries submitted by customers; thus Discovery cannot use error patterns as indicators of intrusions. Discovery records all inquiries that fall outside of acceptable thresholds, and provides an explanation for why the inquiry is unacceptable (these are not used in updating the customer's profile). Discovery is not a real-time system, but alerts the security officer to unusual activity at the end of the workday.

While Discovery was under development, a prototype was used to parallel the work of security investigators, in order to ensure that Discovery would make the same decisions as the investigators. TRW found that the use of Discovery resulted in investigative leads being developed more quickly, and the analysis of Discovery's exception data provided more concise leads than did the investigators' conventional methods. Other, unexpected, benefits included the ability to perform marketing analysis on detailed, up-to-the-minute data using Discovery's customer usage patterns. Trends can also be observed by comparing current customer usage data with previous usage data.

3.8 Cly 'e Digital Systems' Audit

Clyde Digital Systems' Audit is a product that audits users of VAX/VMS machines. Audit can create a complete log of all terminal input and output and provides procedures to help analyze the data collected.

Audit can record every byte that passes between a user's terminal and the system, including control and escape sequences, and stores this data in a file, although certain qualifiers can be specified so that particular special characters can later be discarded from the audit log file (for ease of display and formatting, for example). Audit also provides the option to monitor only terminal input (from the user).

Audit also provides a flexible capability for selective auditing. For example, auditing can be activated selectively for terminal sessions satisfying certain criteria, such as for specific users, or specific times of day (producing an audit trail for the user's terminal session), and the use of specific programs can also be selectively audited (producing an audit trail for the specified program).

Auditing can be controlled by VMS-format keyboard commands or from programs.

Audit allows analysis of the audit data by random sampling or through selective analysis based on the system manager's knowledge of external events. Audit's analysis produces three reports: a security summary report, which summarizes the activity of high-risk users (as defined by a predetermined set of 14 risk factor tests and a programmable set of weighting parameters); a security event report, which summarizes the events that caused those users to be considered high-risk; and a supporting data report, which includes data from the audit log to support the conclusions of the first two reports. The risk factors for which Audit tests include sessions outside business hours or on weekends or holidays (the definition of normal business hours and holidays can be selected by the system manager); sessions indicating use of the AUTHORIZE or SYSGEN utilities; sessions indicating browsing; file access alarms; other alarms (alarms can be established for certain activities); repeated unsuccessful login attempts; sessions with dial-up or remote terminals; simultaneous logins for the same user; and attempts to turn off auditing. Some of Audit's 14 tests use data contained in the audit logs, and some use information from the VMS operator log file; no test uses data from both. The operator log file is used to test for file access alarms, other alarms, login failures, and attempts to turn off auditing. The other tests use the audit logs.

Each of the 14 tests has an associated weight and three factors. One factor is for after-hour use; one factor is for activity from a dial-up terminal; and one factor is for activity from a DECnet remote terminal. Whenever an event satisfies one of the tests, its weight is multiplied by its relevant factors and the result is added to the score for that user. Users with sufficiently high scores are considered to be high-risk. The weights and factors can be selected by the system manager. The system manager can also add additional tests.

There has been at least one published report of bypasses of certain Audit tests [21]. Allen Clyde reports that Audit has detected "numerous acts of misconduct, including criminal conduct ... on sensitive computer systems" [22].

4 Conclusions

None of the intrusion-detection approaches described is sufficient alone—each addresses a different threat. A successful intrusion-detection system should incorporate several different approaches. In particular, a statistical user profile approach augmented with a rule-base that characterizes intrusions promises to be an effective combination. Because they use this combination of approaches, two prototype systems—IDES and MIDAS—have the potential to become strong intrusion-detection systems. Of these two, IDES is particularly strong in its statistical approach, whereas MIDAS focuses primarily on enumerating a comprehensive (although site-specific) set of expert system rules.

Although, as Linde notes [23], the more skilled penetrator can disable the auditing mechanisms in order to work undetected, auditing and intrusion-detection mechanisms are still of value in detecting the less skilled penetrator, because they increase the difficulty of penetration.

Acknowledgments

This work was supported by the U.S. Navy, SPAWAR, which funded SRI through subcontract 9-X5H-4074J-1 with the Los Alamos National Laboratory.

References

[1] J. P. Anderson. *Computer Security Threat Monitoring and Surveillance.* James P. Anderson Co., Fort Washington, PA, April 1980.

[2] R. A. Whitehurst. *Expert Systems in Intrusion-Detection: A Case Study.* Computer Science Laboratory, SRI International, Menlo Park, CA, Nov. 1987.

[3] H. S. Javitz, A. Valdes, D. E. Denning, P. G. Neumann. *Analytical Techniques Development for a Statistical Intrusion Detection System (SIDS) based on Accounting Records.* SRI International, Menlo Park, CA, July 1986.

[4] J. van Horne and L. Halme. *Analysis of Computer System Audit Trails — Final Report.* Sytek Technical Report TR-85007, Mountain View, CA, May 1986.

[5] T. F. Lunt, J. van Horne, and L. Halme. Automated Analysis of Computer System Audit Trails. In *Proceedings of the Ninth DOE Computer Security Group Conference*, May 1986.

[6] T. F. Lunt, J. van Horne, and L. Halme. *Analysis of Computer System Audit Trails — Initial Data Analysis.* Sytek Technical Report TR-85009, Mountain View, CA, Sept. 1985.

[7] T. F. Lunt, J. van Horne, and L. Halme. *Analysis of Computer System Audit Trails — Intrusion Characterization.* Sytek Technical Report TR-85012, Mountain View, CA, Oct. 1985.

[8] T. F. Lunt, J. van Horne, and L. Halme. *Analysis of Computer System Audit Trails — Feature Identification and Selection.* Sytek Technical Report TR-85018, Mountain View, CA, Dec. 1985.

[9] T. F. Lunt, J. van Horne, and L. Halme. *Analysis of Computer System Audit Trails — Design and Program Classifier.* Sytek Technical Report TR-85005, Mountain View, CA, March 1986.

[10] J. van Horne and L. Halme. *Analysis of Computer System Audit Trails — Training and Experimentation with Classifier.* Sytek Technical Report TR-85006, Mountain View, CA, March 1986.

[11] P. A. Karger. Limiting the Damage Potential of Discretionary Trojan Horses. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April, 1987.

[12] T. F. Lunt and R. Jagannathan. A Prototype Real-Time Intrusion Detection Expert System. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[13] D. E. Denning, D. L. Edwards, R. Jagannathan, T. F. Lunt, and P. G. Neumann, *A Prototype IDES — a Real-Time Intrusion Detection Expert System.* Computer Science Laboratory, SRI International, 1987.

[14] D. E. Denning and P. G. Neumann. *Requirements and Model for IDES — a Real-Time Intrusion Detection System.* Computer Science Laboratory, SRI International, 1985.

[15] D. E. Denning. An Intrusion-Detection Model. In *IEEE Trans. on Software Eng.* 13-2, p. 222, Feb. 1987.

[16] *Department of Defense Trusted Computer System Evaluation Criteria.* Dept. of Defense, National Computer Security Center, DOD 5200.28-STD, Dec. 1985.

[17] TRW Defense Systems Group. *Intrusion-Detection Expert System Feasiblity Study.* Final Report 46761, 1986.

[18] P. G. Neumann. *Audit Trail Analysis and Usage Data Collection and Processing, Part One.* Computer Science Laboratory, SRI International, Menlo Park, CA, Jan. 1985.

[19] P. G. Neumann and F. Ostapik. *Audit Trail Analysis and Usage Data Collection and Processing, Part Two.* Computer Science Laboratory, SRI international, Menlo Park, CA, May 1987.

[20] W. T. Tener. Discovery: An Expert System in the Commercial Data Security Environment. In *Proceedings of the IFIP Security Conference*, Monte Carlo, 1986.

[21] D. W. Haskin. Keeping Watch on a VAX. In *Digital Review*, Dec. 16, 1987.

[22] A. R. Clyde. Insider Threat Identification Systems. In *Proceedings of the 10th National Computer Security Conference*, Baltimore, Sept. 1987.

[23] R. R. Linde. Operating System Penetration. In *Proceedings of the National Computer Conference*, 1975.

EXPERT SYSTEMS IN INTRUSION DETECTION: A CASE STUDY

Michael M. Sebring, Eric Shellhouse, and Mary E. Hanna
National Computer Security Center
9800 Savage Road
Ft. George G. Meade, Maryland 20755-6000

R. Alan Whitehurst
Computer Science Lab, SRI International
333 Ravenswood Ave.
Menlo Park, California 94025

Abstract - The Multics Intrusion Detection and Alerting System (MIDAS) is an expert system which provides real-time intrusion and misuse detection for the National Computer Security Center's networked mainframe, Dockmaster, a Honeywell DPS-8/70 Multics. The basic design of MIDAS was heavily influenced by the intrusion detection research of Dorothy Denning and Peter Neumann of SRI International. They proposed that statistical analysis of computer system activities could be used to characterize normal system and user behavior. Given such statistical profiles, user or system activity that deviates beyond certain bounds should be detectable. MIDAS has been developed to employ this basic concept in its evaluation of the audited activities of more than 1200 Dockmaster users.

Introduction

The annoying ring of the telephone jarred John out of his contemplation of the Monday morning newspaper. One of his staff answered, then handed him the phone, whispering "It's the Chief".

"Computer Center, John Speaking."

"Hello, John? This is Edward."

"Hi, Ed. What's up?", replied John, trying to sound casual. By the tone of Edward's voice, John could tell he was upset.

"John, I just got a call from Carla in Marketing. She says she got a message when she logged on this morning about having been logged in over the weekend."

"Boy, that's Carla for ya', always so darned dedicated...", John said, frantically trying to think of something to say to distract Edward. He knew what was coming.

"She says she wasn't."

"She says she wasn't what?"

"She says she wasn't on the system over the weekend."

"Oh."

"You people are paid to take care of this system. Why can't you get your act together down there!", said Edward, starting to get worked up. "Don't you ever monitor who's logged on? Carla has access to some of this company's most sensitive information! Why is it that we never know when we've been had until someone steps up and tells us! We look like blithering idiots!"

"Now hold on, Edward! Sure the system monitors who logs on; heck, it even keeps track of the misspelled commands, the access errors, and half-a-dozen other things. But with the number of users on our system, we simply don't have the manpower to pour over those logs day in and day out." But even as he said it, John knew that wasn't true. Not even an army of workers would be able to make sense out of the mountains of log data that poured out of the system every day. More staff wasn't the answer, but John didn't know what was.

"Don't start on me again with that whining for more help. Your department's overstaffed in the first place! And top heavy, too! Why don't you try doing your job for a change!"

The phone went dead in John's hand. "OK troops," he said, turning to his staff with a heavy sigh, "let's get out the logs for this weekend and see what we can find...".

Intrusion Detection

Audit trail analysis seems to be like the proverbial sour grapes; it is so difficult to obtain that it is tempting to dismiss it as unprofitable and abandon any further attempt at it. The factors that make audit trail analysis so difficult may be summarized into three broad categories: the lack of adequate and/or appropriate audit data; the inability of system security officers to utilize available data; and the lack of a precise definition of what to look for.

"Feast or Famine" characterizes the audit trail data of typical systems. Many systems do not provide adequate auditing facilities to be able to detect a penetration or abuse by audit data analysis; these are the famine systems. In other systems, the security officer is inundated with page upon page of audit data until buried under a paper mountain; these are the feast systems. There is a variant of this latter category which allows audit sources to be selectively activated, but even this is not the solution. In such systems the security officer is faced with the unattractive prospect of having to decide which features to activate at the risk of failing to activate the audit facility that would have provided the key bit of data necessary to detect and apprehend a system penetrator.

Even if a system were developed which provided just the right kind and amount of audit data, the security officer still has a formidable task, for only the most blatant of attacks will be discernible through scrutiny of a single day's audit data. The sophisticated penetrator will spread out his activity over a number of days or weeks. They will subtly exploit the dark corners of a system. For the security officers to detect such attacks would require the correlation and recall of an incredible store of data; nothing short of a Herculean feat.

Another problem complicating the task of security officers in their attempts to analyze audit data is the imprecise definition of what characterizes the threat they are attempting to counter. Anderson [12] defines the threat that monitoring system activity is expected to counter as:

> *"The potential possibility of a deliberate unauthorized attempt to:*
> *a) access information*
> *b) manipulate information*
> *c) render a system unreliable or unusable."*

But what does an intrusion look like in terms of the audit data generated? How can we differentiate between authorized use and the unauthorized threat just described? These are certainly not easy questions to answer, but they lie at the heart of any attempt to automate audit analysis aids.

Studying the activity of a successful security officer involved in audit trail analysis may reveal an approach. Consider the process followed by a security officer in tracking down the hacker that has just scribbled all over his company's payroll database. First, he applied a rule of thumb, or heuristic, that most penetration attempts occur in the early hours of the morning when the system is unattended, so he concentrated his search on sessions in that time period. Next, since the only users at that hour were connecting to the system across a network, he looked for individuals whose point of origin fluctuated. His reasoning here was that someone illegally penetrating the system would attempt to cover his actions by varying the network path. These two constraints yielded a set of potential accounts. From these he was able to pinpoint the account that had been compromised because the audited activities of the individual using it simply "didn't feel right" for the particular account. He immediately shut down that account and had a long talk with the account holder who eventually admitted to giving his password to his roommate, a computer "enthusiast". From this example we can see that the reasoning process employed by successful system security officers involves symbolic reasoning, heuristics and uncertainty. This emphasis on knowledge and its application through symbolic reasoning makes intrusion detection an appropriate candidate for expert systems [8].

Expert Systems

The goal of any intrusion detection system must be to aid system security officers in the detection of penetration and abuse. The expert system should provide the knowledge of an "expert" security officer. This is a MINIMUM standard of performance for an intrusion detection system; as already discussed, humans generally don't do a very good job of audit trail analysis. The set of penetrations or abuses detected by a security officer with the aid of the automated system should be a superset of those that would have been detected by the security officer unaided.

Codification and application of knowledge under similar circumstances is the basis of an expert system. This knowledge is encoded in the form of facts (assertions about the state of a problem solution) and heuristics (rules which govern the transformation of the solution state). Expert systems have been developed that have accomplished amazing results in a number of different fields (see [9, 3, 11]). MIDAS is an example of such a system to detect intrusions into a computer system.

MIDAS Design

"Get place and wealth, if possible,
with grace;
If not, by any means get wealth
and place."
- King Midas

The Multics Intrusion Detection and Alerting System (MIDAS) provides real-time intrusion and misuse detection for the National Computer Security Center's networked mainframe, Dockmaster. Dockmaster is a Honeywell Multics computer system employed primarily as an electronic communications mechanism for the national computer security community. MIDAS was developed using the Production-Based Expert System Toolset (P-BEST), an in-house expert system shell that provides the mechanisms for developing, compiling and debugging very powerful rulesets. The P-BEST inference engine controls the assertion of data into the MIDAS knowledge base, and via its forward chaining inference engine, directs rule selection and execution.

The basic design of MIDAS was heavily influenced by the seminal work in this area of J. P. Anderson, the intrusion detection research of Dorothy Denning and Peter Neumann of SRI International, and similar efforts at Sytek [13]. Denning and Neumann proposed that statistical analysis of computer system activities could be used to characterize normal system and user behavior (see [2, 1]). Given such statistical profiles, user or system activity that deviates beyond certain bounds should be detectable. MIDAS has been developed to employ this basic concept in its evaluation of the daily activities of more than 1200 network users.

Architecture

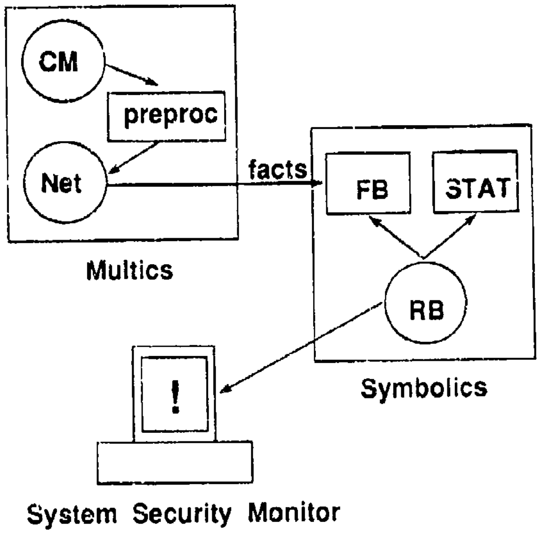


Figure 1 MIDAS Architecture

MIDAS consists of a number of distinct parts, including: a command monitor (CM) that captures command execution data not audited by Multics systems; a preprocessor (preproc) for transforming Dockmaster audit log entries into a canonical form; a network-interface daemon (Net); a statistical database of recorded user and system statistics (STAT); a knowledge base consisting of a representation of current fact base (FB) and rule base (RB); and an extensive end-user interface for communicating with system security officers. The preprocessor, command monitor, and network daemon reside on Dockmaster; the MIDAS knowledge, statistical base, and user interface are installed on a Symbolics Lisp machine.

Each time an audit record or command monitor record is generated, the preprocessor filters out unnecessary data and transforms the remainder into a MIDAS assertion. The assertion is handed to the network interface daemon and passed via local area network to the Symbolics lisp machine hosting the expert system. The fact is placed into the fact base of the expert system. This introduction of a fact into the expert system will cause the creation of rule-fact bindings between the fact and all matching rules in the rule base. Assertion of this fact may satisfy the firing conditions of one or more rules. Any such rules will then fire, potentially transforming the state of the system. Depending upon the nature of the fact, it could cause a chain of rule firings resulting in a number of potential system responses, ranging from warning the operators of suspicious activity to taking direct action to stop a penetrator. The system's reaction is proportional to the extent that the monitored activity deviated from what is considered 'normal' according to the relevant statistical profile.

MIDAS statistics record the aggregation of monitored system activity. Comparing norms derived from past activity aggregation to ongoing actions determines whether the current activity is outside some standard deviation. MIDAS keeps both user and system-wide statistics. User profile statistics, which define normal behavior for a user are maintained (in monthly aggregate form) for each user account throughout the life of the account. These statistics are updated as user behavior changes. MIDAS also keeps current session activity data in a session statistics structure which is maintained for the duration of a user session. User session statistics are initialized at login from the data extracted from user profile statistics. Session statistics include the calculated values that act as thresholds of concern for all activities monitored for that user. For example, if an individual's user statistics indicate that during his 350 sessions he triggered an average 38 system errors, with a standard deviation of 20; a system error concern threshold of 58 (38 + 20) would be stored in his session statistics profile. This value would be the upper limit for normal activity -- if this limit were exceeded suspicion would be aroused, and action might be taken in the form of messages to the operator or by the assertion of a fact noting the suspicion into the knowledge base. When the user logs out, the user statistical profile is updated to incorporate the statistical variance that has been developed from the user's activity during the session. Figure 2 illustrates the cycling of individual user statistics.

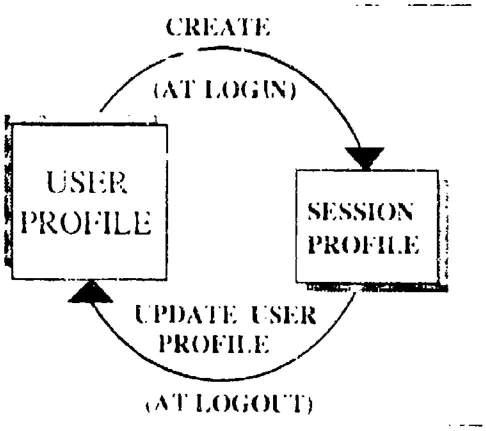


Figure 2 User Statistics Cycle

MIDAS maintains similar statistical structures to determine system wide activity norms.

Heuristics

The logical structure of the MIDAS system revolves around the rules (heuristics) in the rulebase. These rules may be characterized in two ways: according to the type of heuristics they employ, or according to the particular area of surveillance they address.

There are three basic types of heuristics employed to review audit data under MIDAS: immediate attack heuristics, user anomaly heuristics, and system state heuristics.

Immediate Attack - Immediate attack heuristics represent a superficial level of analysis. These rules operate with a very narrow view of the data and are, in some sense, static in their interpretation They are narrow in that they generally involve only a small number of data items in their analysis; and they are static in that they do not make use of any statistical information. In effect, they are intended to detect those audit log entries that are, in isolation of any other information, anomalous enough to raise concern.

Figure 3 gives an example of this type of heuristic encoded in the P-BEST language. This rule concerns attempted system breakin. The rule monitors the knowledge base waiting for the assertion of a bad login attempt in which the account specified by some user was invalid, but commonly used on other systems to denote a privileged account. When the rule finds such an assertion it will warn the MIDAS operator and "remember" another fact; namely, that with high probability, a breakin attempt has occurred.

```
(defrule illegal_privileged_account states
   if there exists a failed_login_item
      such that name is ("root" or "superuser"
         or "maintenance" or "system") and
      time is ?time_stamp and
       channel   is ?channel
   then
      (print "WARNING: ATTEMPTED LOGIN TO
         PRIVILEGED ACCOUNT")
      and remember a breakin_attempt
         with certainty *high*
          such that attack_time is ?time_stamp
         and login_channel is ?channel)
```

Figure 3 Immediate class rule

User Anomaly - User anomaly heuristics make use of the statistical profiles to detect anomalous user behavior. They encode the intuition of the security officer when he says, "it just doesn't feel right." Figure 4 illustrates this sort of rule. In this example, the rule is concerned with user logon analysis.

```
(defrule unusual_login_time states
   if there exists a login_entry
      such that user is ?userid and
       time_stamp is ?login_time
      and (unusual_login_time ?userid ?login_time)
   then
      remember a user_login_anomaly
         such that user is ?userid and
         time_stamp is ?login_time)
```

Figure 4 Anomaly class rule

This example incorporates the notion of a 'usual' login time for a user. If a user accesses the system outside his normal hours, then an anomaly record would be generated. This would, in effect, trigger a heightened level of suspicion about that user.

System State - System state heuristics are analogous to anomaly heuristics, except that they characterize what is normal for the entire system. One example of this type of rule is the detection of an inordinately large number of login failures system-wide. Such an occurrence might be indicative of an attempt to break into the system.

Areas of Concern

In addition to the categories of heuristics, Denning defined eight general areas of concern: breakin, masquerading, penetration, leakage, database inference, Trojan horse, virus, and denial of service [2]. Under MIDAS, Trojan horse and virus attacks are collapsed into a single category because of their similarities. Also, since our concern is mainly with operating system penetration, as opposed to database compromise, inference type attacks are not considered. Finally, denial of service concerns and leakage concerns are combined with misuse concerns. Together with the heuristics described above, these concerns define a matrix which outlines the intended coverage of the MIDAS system.

	IMMEDIATE	ANOMALY	SYSTEM
BREAK-IN	o		o
MASQUERADE		o	
PENETRATION	o	o	o
MISUSE		o	o
TROJAN HORSE		o	o

o = rule coverage

Figure 5 Coveragc of MIDAS Heuristics

Attempted Breakin - This area of concern focuses primarily on login failures. An example of this kind of heuristic was illustrated in figure 3. This rule flags login failures on restricted account names (such as 'superuser', or 'root') as being suspicious.

Another level of analysis involves monitoring parameters which define the state of the entire system at any given time. If the attacker were smart enough to vary the target accounts, the system rules would still detect the abnormal behavior by the rise in system wide login failures.

Other examples from this concern area include: flagging excessive password failures on a system account and noting excessive or abnormal password failure on other accounts.

Masquerade - This area of concern involves the detection of intruders who have obtained access to accounts and valid passwords which do not belong to them. Detecting such occurrences is straightforward: it is based upon the assumption that parameters gleaned from a user's normal interaction with the system may be used to spot activity attributed to that user but which deviates from the user's statistical norms. A number of statistical measures for a variety of factors are collected and stored for each valid user account. Some examples of these factors include: origin of connection (for network users), login time, resource usage, command usage, and command errors. Both the average measure for each factor and the normal deviation is recorded.

During a login session, MIDAS continually monitors the statistics of the current users and compares them against their user profiles. If a user's current activity exceeds acceptable limits suspicion is aroused. An example of this type of rule was given in figure 4

Penetration - Penetration concern involves the detection of any attempted violation of system security mechanisms, and is applicable to valid users as well as masqueraders. This area of concern is addressed by immediate, anomaly, and system wide heuristics targeted toward access or attempted access of system sensitive programs or data.

Misuse - Unusual resource usage may be an indicator of many things. It can certainly increase suspicion that the user is a masquerader. Abnormal resource usage can also indicate that a valid user is engaged in some undesirable activity. For instance, if he directs his printer output to some location other than where he normally sends output, he may be attempting to leak sensitive data [2]. Simple inactivity (logging on and then wandering away from the terminal), while not an attack, represents wasted resource and a potential security compromise. As such it is noted and reported by the system. This area of concern also covers basic detection of covert channel activity. Under Multics release 11.0, all large covert channels (bandwidth 100 BPS) have been eliminated. Moderate (10 - 100 BPS) and small (1 - 10 BPS) channels are captured and audited by Multics [4]. These MIDAS rules act on the occurrence of audited covert channel activity.

Trojan Horse/Virus - This area of concern involves the detection of a Trojan horse or virus which has been introduced into the system. These two areas are not separate because we have not determined a way to differentiate between them given the available audit data. The key factors which are considered when addressing this concern are access violations on system sensitive objects, and execution statistics which violate norms established for given commands. Access violations on sensitive objects may indicate the introduction of a virus into the target system; monitoring execution statistics attempts to detect their presence.

Rule Base Structure

In the discussion to this point, the phrase 'raise suspicion' has been used without reference to exactly what is meant. Most of the rules in the MIDAS system are sensory rules. Sensory rules detect anomalous activity and assert a conclusion into the knowledge base representing the suspected problem. These rules may also issue a warning message. Another category of rules, referred to as secondary rules, operate only on the output of the sensory rules. These rules act like AND gates; firing only when certain kinds of suspicions have been aroused.

Figure 6 represents the structure of the MIDAS rule base. Each area of concern (breakin, masquerading, misuse, etc.) is addressed by a different set of rules. The output of these rules feed the secondary rules, which in turn result in concrete actions being taken.

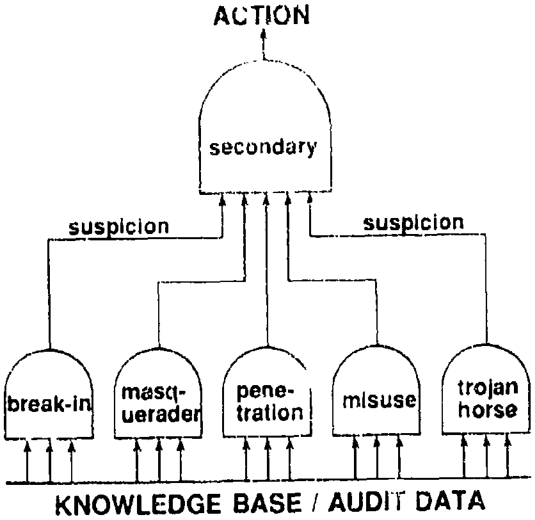


Figure 6 MIDAS Rule Base Structure

MIDAS Operation

MIDAS operates continuously, constantly monitoring user activity and the state of the target system. If it detects anomalies in system operation, relevant messages are displayed on the MIDAS console. The operator may choose to act directly on these warning messages, or to investigate further using the commands available through the user interface. For example, the operator may query the target system status, or a user's status, or trace specific user activity. Based on this analysis, the operator will initiate corrective action.

As discussed previously, MIDAS is composed of a number of distinct parts. First among these is the preprocessor, implemented on Dockmaster, which extracts and reformats relevant audit data. This data is then transferred to the MIDAS workstation, where it is asserted into the expert system shell (P-BEST) and applied to the compiled MIDAS rules and statistical subroutines. Any anomalies detected are reported immediately via the MIDAS user interface.

The logic of the MIDAS rule base is covered in the Architecture section of this paper. More detailed discussions of audit data preprocessing and the user interface are now presented to provide a complete system description.

Audit Data Preprocessing

The MIDAS system acts primarily upon five types of audit data: logins, logouts, commands, detected errors, and I/O requests. This data is extracted from the Dockmaster audit logs and reformatted into a time-sorted series of assertions having the basic structure suggested by Denning [2]:

```
(<subject><object><action><exception>   <time-
stamp>)
```

For most of the different audit assertion types, <subject> is list composed of userid, project, tag, process identifier, terminal type, connection source (local dial, TYMNET, or MILNET indicator), and host id.

Similarly, <time-stamp> is universally formatted as a list containing the elements: absolute time (the number of seconds since midnight), date (YY/MM/DD), and time (HH:MM:SS)

The remaining fields <object>, <action>, and <exception> have varying meanings depending on the audit assertion type. For example, a typical login entry might look something like this:

```
(LOGIN (COLOSSUS FORBIN A 03452 H19 VT1
   456) NIL NIL NIL (120 02/12/88 00:02:00))
```

and a command usage entry might look like this:

```
(CMD (COLOSSUS FORBIN A 03452) NIL
 BOUND_INFO$WHO 0.2 (230 02/12/88 00:03:50))
```

User Interface

The MIDAS User Interface is a comprehensive window-based environment composed of a bit-mapped display which presents four panes arranged within one overall display window, and allows various operator interactions through a mouse menu interface. The four display panes are: the *User Pane*, which displays a list of users currently on the system; the *Command Pane* which provides a mouse/menu driven access to a number of MIDAS commands; the *Warning Pane*, which displays specific warning messages generated by MIDAS; and the *Graphical System Status Pane*, which is used to display the state of MIDAS and the targetted host. The operator can adjust the operation of MIDAS, and trigger some specific report generation through the user interface. (Figure 7 illustrates the MIDAS window display). The MIDAS user interface displays the ongoing analysis of the target system security state.

User Pane - Information provided in the MIDAS *User Pane* consists of the Login Time, Userid, Project, and Tag for all processes in the monitored computer system. (Tag is a one character flag which indicates whether the user is in interactive mode ("a"), or batch mode ("m")).

Two flags may appear prior to the user's name in the *User Pane*:

- The first flag, a question mark (?), indicates that the user is suspected of anomalous activity. This suspicion may be generated as a result of the user's having triggered some combination of MIDAS rules, or by independent observation by the Dockmaster operator.

- The second flag which may appear to the left of a user name is a capital M. This mark indicates that a user's session is now being closely monitored. All audit data pertaining to this user is now also reflected in the MIDAS *Warning Pane*. This is a powerful tool for detailed user monitoring.

Warning Pane - The *Warning Pane* displays the warning messages and MIDAS conclusions generated as a result of MIDAS rule execution. These messages include warnings about breakins, masqueraders, penetrations, misuse, trojan horse/virus detection, and the reasons why these warnings were generated.

Sample Messages

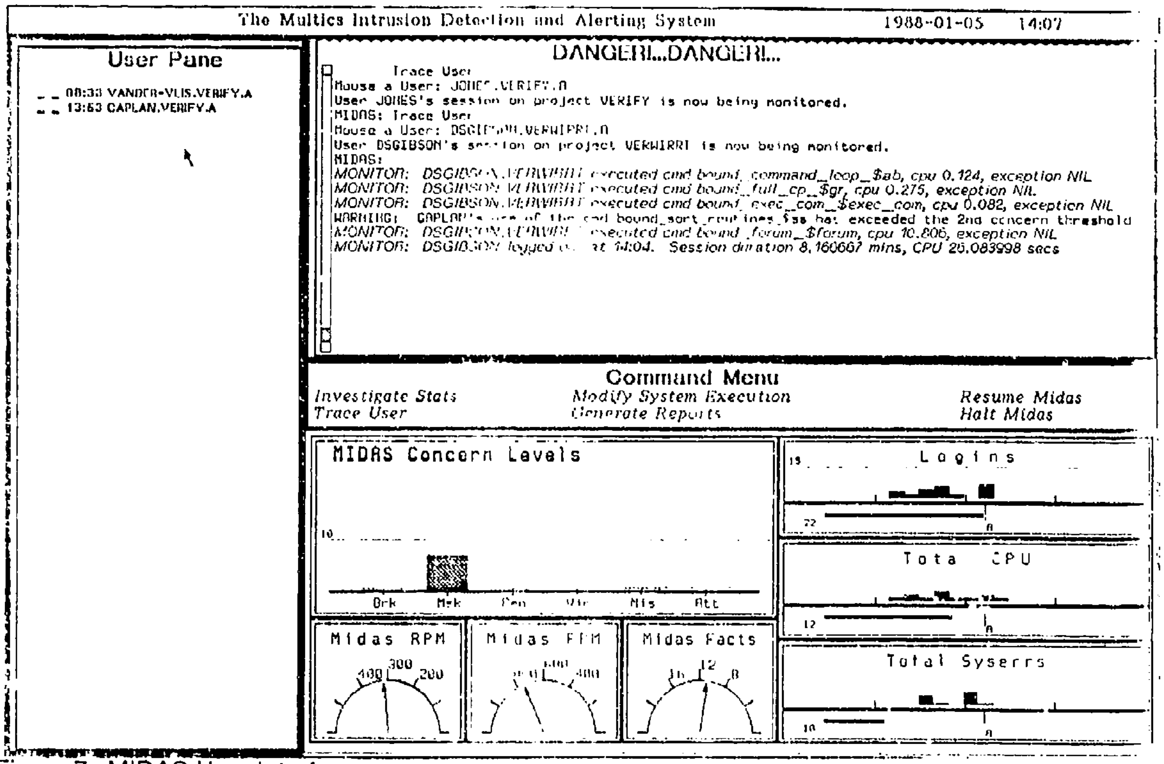
```
  NOTE: FIRST LOGIN  FOR  USER  COLOSSUS
(SRC:  VT1.0438)

  MONITOR: COLOSSUS  EXECUTED  CMD  "LIST",
CPU  .03

  WARNING: FEY  EXCEEDED  1ST  THRESHOLD
FOR  CPU  USE

 ALERT: COLOSSUS IS A MASQUERADER.
     REASONING  IS:
   LOGGED  IN  FROM  AN UNUSUAL  SOURCE
(3106.4452)
    LOGGED  IN  AT  UNUSUAL  TIME  (01:45)
    EXCEEDED  1ST  THRESHOLD  FOR  CMD  ERRORS
(15)
    EXCEEDED  2ND  THRESHOLD  FOR  SYSERRS  78
    EXECUTED  THE  INVALID  COMMANDS  "PRIV",
"SUID"
```

Warning Pane information is generated independent of the MIDAS Window Interface, and thus can be made available on other (non-windowing) versions of MIDAS. *Warning Pane* messages are hierarchically grouped into classes of related messages, from notes, to warnings, to system alerts. Each of these message types has a slightly different format or font in order that



The Multics Intrusion Detection and Alerting System 1988-01-05 14:07

User Pane

```
_ _ 08:33 VANDER-VLIS.VERIFY.A
_ _ 13:53 CAPLAN.VERIFY.A
```

DANGER!...DANGER!...

```
        Trace User
Mouse a User: JONES.VERIFY.A
User JONES's session on project VERIFY is now being monitored.
MIDAS: Trace User
Mouse a User: DSGIBSON.VERWIRRT.A
User DSGIBSON's session on project VERWIRRT is now being monitored.
MIDAS:
MONITOR:  DSGIBSON.VERWIRRT executed cmd bound_command_loop_$ab, cpu 0.124, exception NIL
MONITOR:  DSGIBSON.VERWIRRT executed cmd bound_full_cp_$gr, cpu 0.275, exception NIL
MONITOR:  DSGIBSON.VERWIRRT executed cmd bound_exec_com_$exec_com, cpu 0.082, exception NIL
WARNING:  CAPLAN's use of the cmd bound_sort_routines_$ss has exceeded the 2nd concern threshold
MONITOR:  DSGIBSON.VERWIRRT executed cmd bound_forum_$forum, cpu 10.806, exception NIL
MONITOR:  DSGIBSON logged out at 14:04.  Session duration 8.160667 mins, CPU 26.083998 secs
```

Figure 7 MIDAS User Interface

the message type be easily distinguishable, and in order that the really important messages are easily noted. All messages, from the notes to the system alerts, are written to a daily log along with an analysis of each suspicious user's session (see Figure 8 for a sample session analysis). This log is printed at the end of the day (midnight).

Command Pane - The MIDAS operator has available a number of commands to modify the operating parameters of the system or generate different displays. MIDAS commands are invoked by pointing the mouse at the desired command (in the *Command Pane*), and clicking. Most of the commands listed invoke menus which are in themselves lists of commands. These commands provide the means to display user and system statistics, generate reports, modify MIDAS execution parameters, and execute operator commands. For example information about a selected user (selected by invoking the "Investigate Stats Menu" then selecting the "Analyze User Session" command and mousing on a current user in the *User Pane*) might look something like this:

```
ACTIVITY FOR USER COLOSSUS IS
ANOMALOUS:
   LOGGED IN FROM AN UNUSUAL SOURCE
   LOGGED IN AT AN UNUSUAL TIME
   USE OF COMMAND(S) "PRINT", "WHO" IS HIGH
   USE OF COMMAND(S) "LIST" IS VERY HIGH
   OCCURRENCE OF COMMAND ERRORS IS HIGH
   OCCURRENCE OF SYSERRS IS VERY HIGH
   COLOSSUS IS A SUSPECTED MASQUERADER
```

Figure 8 User Session Analysis

Graphical System Status Pane - This pane (the low right corner of figure 7) displays information about the state of the t.r-get host, and about the state of MIDAS. Currently four histograms and three meters are displayed. The histogram labelled "MIDAS Concern Levels" displays how many facts the expert system has concluded in these areas (attempted breakins, masquerader, etc.). The three histograms labelled "Logins", "Total CPU", and "Total Syserrs" inform the operator about these particular occurences. A quick glance at these histograms can reveal how many of these occurrences have occurred today (or in any particular hour of today) as well as how many occur system wide per day. Finally, across the bottom of this pane, three meters give the operator some indication as to how well the expert system is running. The MIDAS RPM meter informs the operator how many audit records per minute the expert system processes. The MIDAS FPM meter, describes how many facts the expert system processes. Finally the MIDAS Facts meter informs the user how many facts are left in the fact base.

Results

MIDAS tracks all user activity on Dockmaster in real-time. However, in order to implement the system in a modular fashion and test in a controlled environment, MIDAS was initially deployed on only 105 users, approximately 11% of the total user population. Data was transferred daily via magnetic tape. Consequently, the system was tested by flooding it with audit entries in simulated real-time.

MIDAS performed as intended in the test-bed implementation. Under limited testing of crude simulated attacks, almost all anomalous activities were detected. For example, test data developed by changing the userid throughout one session's set of audit data to the userid of another system user resulted in the determination by MIDAS that the test user was a masquerader. In addition, a number of unexpected anomalies were detected in "live" audit data that are now under further review. MIDAS reliably flags a monitored set of 17 different immediate attack activities (For example attempting to use sensitive system commands). Those rules which fire based on overall system state anomalies are also quite reliable.

Despite the limited size of the test-bed, we became confident that MIDAS would be capable of monitoring the complete DOCKMASTER user base. We gained confidence because MIDAS was deliberately designed with speed as a paramount criteria, and because initial test data timings were very favorable.

MIDAS is fast for three reasons. First, MIDAS rules are compiled into lisp object code, not interpreted as is the case using many expert system shells. Second, wherever possible, MIDAS rules have been generalized to handle as many areas of concern as possible. Minimizing the number of rules within the system in turn minimizes the number of rule/fact bindings that occur, thus reducing the number of possibilities the system must check. Third, we have placed a number of analysis functions into the user interface to be triggered at the operator discretion, rather than in the rule base to be triggered nondeterministicly by matching fact patterns. For example, the MIDAS user interface contains a function for checking if any users have been inactive for an excessive period of time. This function detects misuse rather than intrusion, and does not need to be active constantly. Putting its execution at the operator's discretion reduces the load on the rule base.

Actually, MIDAS is faster than we had anticipated. In processing the data for the limited test group of 105 users over a period of approximately 45 days, MIDAS has averaged an evaluation rate of 425 audit entries per minute. The average time it has taken to process an entire day's test-bed activity is 9 minutes. Given that the test population was 11% of the normal target system population, a simple extrapolation indicated that the system could process all audit data for an entire day in less than 2 hours. Even allowing a massive reduction in throughput based on the rule/fact binding complexities that would accumulate during peak periods of Dockmaster usage (30 - 50 users), MIDAS appearred ready to monitor Dockmaster in real-time.

Currently the system runs in real time. The system appears slightly oversensitive. Rules based completely on statistical profiling often trigger too readily because the thresholds of concern are often too low. This problem may be solved by developing better algorithms for concern thresholds, or some basic adjustments to the rules dealing with individual user activity may be requred. As user profiles become normalized, the system will better differentiate between suspicious and normal user activity. MIDAS has been successful in profiling system-wide behavior, by summarizing from the logs such statistics as total login failures, total system errors, etc... As it stands, the system has detected many anomalies, some of a suspicious nature. We will continue investigation of these unusual activities with the help of the system administration personnel. Although the system is still being enhanced and tested, we believe that applying MIDAS to the audit log problem improves detection of computer abuse and misuse.

Future Directions

MIDAS was designed specifically to provide intrusion detection for the National Computer Security Center's Honeywell Multics system. However, MIDAS could easily be generalized to monitor any Multics system. With some effort, it may be operable for a number of different target systems. For example, MIDAS may soon be modified to monitor an IBM system running ACF2. Also, although the MIDAS expert system was developed on a Symbolics workstation, the basic system has been ported to a Sun workstation. Efforts are ongoing to develop a user interface for the Sun version which takes advantage of Sun capabilities for graphics and color display.

A proposed enhancement is to implement Markovian analysis of command input patterns. Under Markovian analysis, each command type is regarded as a state variable, and a state transition matrix is used to characterize the transition frequencies between states. A command input transition would be abnormal if its probability (as determined by the previous system state and the current transition matrix entry) was too low. This mechanism can be used, for example, to determine if the command sequences of a user are similar to those which characterize a penetrator.

Some means for validating the performance of the rule base should be developed. Interim measures include the analysis of MIDAS performance under normal conditions and under 'stress' conditions. These stress conditions will be generated by assembling a tiger team to attempt to compromise the monitored system. However, a more rigorous method for rule base validation and verification is greatly needed. This represents a current area of particular concern in artificial intelligence. Numerous approaches have been proposed for analyzing the structure of rule-based systems to check for consistency and completeness (see [5, 6, 7, 10]). Tools of this nature are presently under consideration as extensions for the Production-Based Expert System Toolset which supports MIDAS.

The rules of the expert system could be improved by interviewing hackers, and those who have caught hackers. Currently the heuristics of the expert system rules are based on the knowledge of system administrators and system programmers. Also, knowledge gained from discovering intrusion, misuse, penetration, etc... will further refine and enhance the rules.

Finally, we would like to enhance MIDAS so that if the system runs unattended, MIDAS can act on its own suspicions. That is, the system could take the least disruptive action to follow up on its conclusions. This could occur as a result of a user's failure to correctly answer a challenge response question issued by MIDAS in reponse to the user's previous anomalous session activity [14]. We want to enhance MIDAS so that it can interact with the target host if it must. This would broaden its scope considerably from that of just an audit reduction tool.

Acknowledgements

A special thanks to Raymond Rankins, John Rutemiller, and Robert Richmond for their coding of parts of the project. We would also like to thank the numerous co-workers who have helped review this paper.

References

[1] Dorothy Denning and Peter G. Neumann. *Requirements and Model for IDES A Real Time Intrusion Detection Expert System.* Final Report 6169, SRI International, Menlo Park, California, August 1985.

[2] Dorothy E. Denning. An intrusion detection model. *IEEE*, 118-131, 1986.

[3] R. O. Duda. *A Computer-Based Consultant for Mineral Exploration.* Final Report, SRI International, Menlo Park, CA, September 1979.

[4] *Final Evaluation Report of Honeywell Multics MR11.0*, National Computer Security Center, 1 June 1986.

[5] James R. Geissman and Roger D. Schultz, Verification and Validation of Expert Systems. *AI Magazine*, 26-33, Feb 1988.

[6] Tin A. Nguyen, Walton A. Perkins, Thomas J. Laffey, and Deanne Pecora. Knowledge base verification. *AI Magazine*, 8(2):69-75, Summer 1987.

[7] Robert M. O'Keefe, Osman Balci, and Eric P. Smith. Validating expert system performance. *IEEE Expert*, 2(4):81-89, Winter 1987.

[8] David S. Prerau. Selection of an appropriate domain for an expert system. *AI Magazine*, 26-30, Summer 1985.

[9] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN*. American Elsevier, New York, New York, 1976.

[10] Scott A. C. M. Suwa and E. H. Shortliffe. An approach to verifying completeness and consistency in a rule-based expert system. *AI Magazine*, 3(4):16-21, 1982.

[11] S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir. A model-based method for computer-aided medical decision making. *Artificial Intelligence*, 11(1):145-172, 1978.

[12] James P. Anderson, *Computer Security Threat*, James P. Anderson Co., April 15, 1980.

[13] Lawrence R. Halme and J. V. Horne, *Analysis of Computer System Audit Trails*. Sytec, Inc., May 1986.

[14] Discussion from Future Directions Session of the Intrusion Detection Workshop hosted by Stanford Research Institute International held March 1988.

[15] Allan R. Clyde, Insider Threat Identification Systems in *Proceedings of the 10th National Computer Security Conference*, 1987, pp. 343-356.

Auditing in a Distributed System: SunOS MLS Audit Trails

W. Olin Sibert
Oxford Systems, Inc.

ABSTRACT

This paper describes the important features of the SunOS MLS auditing mechanism, and how it solves the problems of performing useful audit functions in large distributed systems. The goals and experiences which led to this design are described. The SunOS MLS mechanism is compared with other implementations.

INTRODUCTION

This paper begins with a brief overview of the SunOS MLS system: its hardware, its software interface, and its additional security features. That overview serves merely to introduce the system; some familiarity with UNIX and the Trusted Computer System Evaluation Criteria [DoD85] is expected for complete understanding of this paper.

The system overview is followed by three sections describing the characteristics that distinguish auditing in SunOS MLS from more conventional implementations. The section on audit message life cycle describes how an audit message travels from its point of origin to permanent storage. This mechanism was designed to minimize overhead for message generation while still strictly limiting the amount of audit data lost due to a system failure. That section also describes the methods the administrator can use to manage large volumes of online audit data, and how the data can be migrated offline.

The section on audit analysis describes the audit analysis tool, which is how the "single system image" of SunOS MLS is implemented for auditing. This tool has extensive merging and selection capabilities, and is the primary mechanism for processing audit data before analysis or display.

The section on audit message format summarizes the "Flexible Audit Message Format", which was designed as a system-independent format suitable for use in arbitrary operating systems, not just SunOS MLS. This format is being considered by the IEEE P1003.6 and X/Open standards subcommittees on security.

The paper ends with a section describing the implementation characteristics of SunOS MLS auditing. This includes some comparisons between SunOS MLS and other systems ([Picciotto87], [Gligor86]), as well as a preliminary discussion of SunOS MLS auditing performance.

OVERVIEW: WHAT IS SunOS MLS?

Sun's SunOS MLS product is secure distributed system which is targeted for evaluation at the B1 Criteria level and which is currently undergoing developmental evaluation with the NCSC. It is a variant of Sun's standard SunOS system (release 4.0) with which it has complete application compatibility except in areas where security requirements prohibit. SunOS MLS runs on Sun's Sun-3 and Sun-4 hardware product line, which ranges from 1.5 MIPS desktop workstations through 10 MIPS workstations and file server machines. It has been under development since mid-1986, and is described in more detail by [Sun87].

SunOS is a version of the UNIX operating system which includes compatibility with the AT&T System V, Release 3 definition, numerous enhancements from the Berkeley 4.2/4.3bsd systems, and Sun's own extensions. In addition to the basic UNIX functions, SunOS includes SunView, a window-based user interface, and full support for the TCP/IP and NFS (Network File System) network protocols.

SunOS MLS is an extended version of the basic SunOS system intended to meet the B1 class requirements of the Trusted Computer System Evaluation Criteria (TCSEC) [DoD85]. In addition to auditing, which this paper describes, it includes protection of user passwords, support of mandatory security labels in the file system and in NFS, device labeling, mandatory security for socket-based interprocess communication, and an extension to the window interface, Secure SunView, which places mandatory access control labels on all on-screen windows and allows simultaneous display and manipulation of data at many different labels. A more complete description is found in [Sun87].

UNIX is a registered trademark of AT&T. Ethernet is a registered trademark of Xerox Corporation. Sun Microsystems is a registered trademark of Sun Microsystems, Inc. SunOS, NFS, Sun-3, Sun-4, and SunView are trademarks of Sun Microsystems, Inc. POSIX is a trademark of the Institute of Electrical and Electronic Engineers. X/Open is a registered trademark of the X/Open Company, Ltd.

The work described herein was performed under contract to Sun Microsystems, Inc.

The statistics and mechanisms presented in this paper are taken from a pre-release version of SunOS MLS, and do not represent a commitment to any specific implementation or performance characteristics of the actual SunOS MLS product.

SunOS MLS Configuration

A SunOS MLS system is a distributed system comprising one or more physical machines (such as workstations or file servers) connected to a dedicated Local Area Network (LAN). Because the LAN (which uses Ethernet-based technology) represents the communication path between the CPUs in the distributed system, it is also referred to as the interconnect or "backplane" for the system. Some machines may be referred to as "servers", generally because their primary purpose is to export disk storage to other machines. In a typical configuration, most machines are "diskless" and use NFS to reference their data, which is stored on one or more servers.

Because all the machines, their peripherals (if any) and the LAN interconnect, are part of the same system and equally trusted, physical security is required for all those components to ensure that no compromise occurs due to violation of hardware integrity. In a fully secure configuration, no "foreign" hardware may be attached to the LAN; it is used only for communication among a set of machines all running the same TCB software.

Single System Image

Although each machine in a SunOS MLS system is a partly independent processor running its own instance of the TCB and its own set of users, the entire set operates as a single system. This is possible because a user's (and administrator's) view of the system is independent of the machine being used. All machines share the same file system, and a file name has the same meaning regardless of location.

Similarly, all administrative functions may be performed (by an appropriately authorized user) from any machine. The administrative databases are all maintained at a single point, and distributed throughout the system by Sun's Yellow Pages distributed database mechanism. In particular, this is true of authentication data, so that user identity is unique regardless of location.

As is described below, this single system image is very important for auditing. This concept allows the system administrator to view and analyze the audit trail for the entire system as a single entity, even though the audit data was generated by numerous independent machines and may be stored in multiple locations. Because file names and user identities are unique through the system, it is straightforward to analyze the merged audit data.

Accountability

An important aspect of SunOS MLS for auditing is the *audit user ID*. This is a unique user identity, kept in addition to the standard UNIX *real user ID* and *effective user ID* values, that identifies a process (subject). The audit user ID is assigned to a process only by its initial login through the trusted path, and its value is the same as the initial values of the other user IDs. This identity is inherited by all descendants of the initial process, and, in effect, provides accountability back to the user whose fingers are at the keyboard. Unlike the effective user ID and real user ID, the audit user ID's value is never changed. All activities performed between login and logout, regardless of which window they are performed in, or which machine the process runs on, are accountable to the original logged in user. For example, the audit user ID is maintained when the user issues the *su* command to switch to a privileged role or uses the *rlogin* command to initiate a session on another machine.

AUDIT MESSAGE LIFE CYCLE

The generation side of the audit mechanism is responsible for responding to "audit" calls from TCB programs, generating messages, and writing those messages to permanent storage for analysis. Although this is a conceptually simple path, the requirements for high bandwidth and reliable transmission often make this a complex process. Even in a conventional multi-user timesharing system, the path for an individual audit message may include several buffers, each perhaps slower to access, but less likely to overflow. In a distributed system, where audit message storage may be accessed through a communication interface, the problems are exacerbated.

Audit Message Pre-Selection

The first part of an audit message's life is really the decision of whether to generate the message at all. The TCSEC requires not that all security-relevant events be *audited*, but merely that they be *auditable*. It also specifies a minimum set of characteristics for selecting[1] specific audit messages. This allows the administrator some flexibility in making the tradeoff between what to collect and the volume of information recorded. These administrative control mechanisms will probably be different in different systems, but they always rely on some form of categorization of messages.

Although the most powerful selection mechanisms are available only at analysis time, some limited options are available to control the set of messages recorded. For each user, the system administrator may specify a set of audit event classes for which messages should be recorded. These are further divided into two sets: messages to be recorded when an attempted operation is successful, and messages to be recorded when an attempted operation fails for any reason. These per-user values actually just modify a system-wide default; rather than specifying the exact set of classes for each user, the administrator writes specifications such as "the default, plus successful access changes, plus all failed attempts". Thus, the administrator can establish a set of audit classes for the whole system, and adjust it individually for particularly trustworthy or particularly suspicious users.

The class selection mechanism is based on audit message types (see **AUDIT MESSAGE FORMAT**, below). Every distinct operation generates a message of a different type. One set of message types is defined to describe each of the operations defined in the POSIX specification, and individual systems (such as SunOS MLS) define

[1] The TCSEC allows events to be "selectively audited" either by making the selection at generation time (pre-selection), or by picking specific messages out of the audit trail at analysis time (post-selection). SunOS MLS provides selection by user identity (and message class) at both generation and analysis time, but only provides selection by object security label (and most other attributes) at analysis time.

additional message types for their extensions. Additional message types can also be defined by third-party applications. The number of types may be quite large: in SunOS MLS, it is approximately 300. The message type is a fine-grained selection mechanism, and corresponds directly to the operation performed by an administrator or a user program.

There is a system-wide table that maps each of these individual message types into one of a small number of message classes. The message class indicates the class of operations (such as "administrator action", "file modification", etc.) to which the message belongs. Message classes are used to identify subsets of the complete audit trail which are to be recorded for particular users or programs (thus reducing the volume of data). Since message classes are intended for administrative control of the audit mechanism, there should be only a fairly small number defined. It is expected that the set of classes may be different in different system implementations; in SunOS MLS, 13 classes are currently defined.

Audit Messages in the Kernel

A SunOS MLS audit message starts life in the kernel (the hardware privileged part of the TCB software). It may have been generated either by an auditing call internal to the kernel, or by a system call made from some trusted process. In either case, the information for the audit message is gathered up and formatted into an audit message data structure, which is then stored in one of a small set of buffers in kernel memory. If a failure occurs in the local machine, no more than those buffers worth of audit data can be lost (up to 10 audit messages). Once a message is placed in a buffer, the "audit daemon" is notified.

The Audit Daemon

The audit daemon is an independent process which runs on each machine. Unlike ordinary processes, it runs almost entirely in kernel mode. Therefore, except when handling errors, all the data it manipulates is in kernel memory, and not subject to swapping or paging. This allows the audit daemon to respond very quickly to arriving audit messages and ensures that it is not a bottleneck.

The audit daemon's job is to take the audit messages from their kernel buffers and write them to the destination file. In normal operation, the audit daemon is awakened whenever a message is placed in a kernel buffer. It runs promptly and performs a normal file system *write* operation to write out the message. This process is repeated until all the kernel buffers are again empty, at which point the audit daemon goes back to sleep and awaits another message. The audit daemon runs in kernel mode to avoid an extra buffering step and to improve context switch efficiency. Its processing loop is invoked by a special system call which never returns from the kernel unless an error occurs.

In addition to this normal mode of operation, the audit daemon is also responsible for creating audit files, for handling any errors which occur while writing to an audit file, and for monitoring the amount of space still available for writing more messages. Whenever an I/O error or a file system full condition occurs, the audit daemon returns to user mode, selects a new location for audit data, and creates a new file into which messages will be written. It then again invokes its special system call to write messages to this new file. No messages are lost on the local machine when this occurs, since the kernel buffers remain full while waiting for the audit daemon to find a new home for them.

Each audit daemon has a list of directories (known as "audit file systems") from which it can choose a location for audit files. It consults this list whenever a new audit file must be created. Typically, this list is different for each machine or small group of machines, in order to spread the audit traffic evenly. Normally, the directory is chosen based on a fixed algorithm, but the audit daemon also has a control interface that allows an administrator to direct its attention to a particular audit file system, or simply to close out the current audit file and open a new one.

When the audit daemon is unable to find a destination for the audit messages, or if the audit daemon itself suffers a failure, the kernel buffers continue to fill up. As soon as all 10 kernel buffers are in use, the machine ceases to perform any auditable operations until the condition is remedied or until it is rebooted. The audit daemon's operations while trying to create new audit files are not audited until after a new audit file is available, to avoid an infinite loop. Because the audit daemon keeps trying to create new audit files, as soon as the error condition is remedied, it will succeed, drain the kernel buffers, and the processes on the machine which are being audited (and therefore were hanging, awaiting kernel buffers) will resume normal operation.

This recovery mechanism is important because of the distributed nature of the SunOS MLS system. Because audit files are usually physically resident on disks attached to remote machines, the audit daemon references them using the NFS protocol over the LAN interconnect. The failure or temporary unavailability of one of these remote machines should not halt the entire system.

Audit File Systems

Audit files are typically kept in dedicated file systems[2] reserved for audit data alone. This is done to keep the audit data from interfering with other user and system files: if an audit file system becomes full, the effect is only to direct audit messages to another location, rather than the more serious effects of exhausting disk storage used for other purposes.

Audit file systems are also typically kept on a small set of machines acting as "audit servers", and referenced through NFS. This allows for efficient and reliable storage because the audit servers can be chosen to have large amounts of disk storage and high reliability. Storing audit files for many machines on a single server also speeds analysis, since those files can all be accessed directly on that machine, rather than through NFS. Although, for instance, audit data could be stored on the local disks attached to individual desktop machines, this would be inefficient for access, and would also mean that some audit data would be unavailable for analysis simply

[2] A SunOS file system is a fixed-size region of disk, or an entire disk, which contains a portion of the system's directory hierarchy.

because the user of some machine turned its power off. Finally, use of audit servers may improve the physical security of audit data. Although the TCB prevents users from accessing any data, even that on local disks, except as allowed by the security policy, transmission of audit data to a remote machine in a physically more protected environment may still be desirable.

Recovery From Unusable File Systems

An audit file system may become unusable either because it is inaccessible (its server machine has crashed, or its network connection is broken), or because it is full. In the first case, after some brief attempts at error recovery, an audit daemon simply selects the next audit file system from its list, and attempts to create a new audit file.

In the second case, the file system has reached one of two limits: soft or hard. The soft limit is a variable threshold set by the administrator which causes the audit daemon to run the *audit_warn* command script. After encountering the soft limit, the audit daemon attempts to switch to using another audit file system which has not yet reached its soft limit. Encountering the hard limit simply means that no space remains, and that either a new location must be found or the machine will hang awaiting space somewhere.

In all these limit and error cases, the *audit_warn* script is run. A default version of this script is shipped with the product. An installation may modify it to take more complex recovery actions. The default action on these conditions is simply to warn the administrator about whatever condition has arisen, by sending mail, and by printing a message on the console for the more serious conditions. However, the script can be tailored to perform arbitrarily complex actions as well, such as automatically deleting or moving old audit files from a full file system, terminating user processes to prevent additional activity, or changing the audit flags for existing processes to reduce the set of events being audited.

Archiving Audit Files

In addition to the live audit files that are being written by the audit daemons, the administrator must also manage old audit data. The audit reduction tool provides numerous ways of doing this

The first thing to do with audit data is generally to combine it (a day's worth at a time, perhaps) into a single file and move it to another file system. The destination need not be a dedicated audit file system, since the combined file will not grow unpredictably after it is created. Often, this combination process involves several steps and intermediate destinations, but as long as the directories are appropriately organized, the rearrangements will be transparent to the audit analysis tools.

Another form of archiving is tape. Although no software is provided specifically for managing tapes of audit archives, the ability to combine and trim audit files makes tape management much simpler.

Two other forms of management for online audit files are available: compression and trimming. Compression uses the standard SunOS *compress* program to reduce the size of audit files; the reduction tool automatically handles compressed audit data, uncompressing when reading it, and generating compressed data on request.

Finally, audit files can be trimmed of unwanted messages. This allows, for example, an administrator to keep a full year's worth of login and logout messages online, while having the other messages readily available in complete audit files on tape. The trimming capability is also implemented in the audit reduction tool.

AUDIT DATA ANALYSIS

Audit analysis in SunOS MLS is performed with the aid of the *auditreduce* program. This is used to perform a logical merge of all the audit files in the system, select some messages for processing, and output them as a stream of messages for processing. Of course, *auditreduce* does not *physically* merge all the audit files every time it is run — that could represent gigabytes of data. Rather, it selects messages (by time and machine identity) only from appropriate files, and merges those, trimming out unwanted messages as early as possible in the merge. In this way, it provides the most efficient possible presentation of any desired subset of the system-wide audit trail.

No actual analysis is performed by *auditreduce*; rather, its purpose is to write (to *stdout*) a stream of messages for processing by another program. The simplest example is the *praudit* program, which simply displays the messages in human-readable form. This can be combined with *grep* and other SunOS utilities to make more specific selections.

The dynamic read mode of *auditreduce*, rather than reading messages already present in audit files, watches all the audit files and file systems for new messages and files, and writes them to its output as soon as they appear. This output can then be piped into a program or even a simple shell script to perform real-time analysis and alarms. It can even be piped into a real-time alarm shell script; a program has been developed independently of the mainstream SunOS MLS development effort to take advantage of this capability: it dynamically displays the most common recent audit messages in a graphical form.

Merging Audit Files

The merge of audit files relies on the fact that all the audit messages in a file are recorded in time-sorted order. Because each audit file is written initially by exactly one process, some machine's audit daemon, the daemon can easily ensure this. Furthermore, the filenames of all audit files contain a pair of timestamps and machine name[3], so that the origin and times of audit messages within a file can be determined by efficient examination of the file's

[3] This convention is implemented by the audit daemon and by *auditreduce* (when it writes files), and is relied upon by *auditreduce* when reading files. Audit files with other names are inconvenient to manipulate, and *auditreduce* provides a function to regenerate the timestamps. This is important for fixing up files which were not closed normally (because of a crash or file system), and whose ending timestamp still indicates "I/O in progress".

name, rather than its contents.

Whenever audit data is generated by independent processes, and more so when generated on independent machines, synchronization of time stamps in audit messages can be a problem. In SunOS MLS, this is not a significant problem, because, in general, all of a particular subject's (process's) auditable activities are recorded on the local machine where the subject is running. Some activities, such as remote login, may additionally be recorded on another machine, and then followed by a series of messages on that other machine (recorded as the activities of a different subject, with the same identity), but there is always an easy way to track the origin of the activity. Therefore, as long as normal administrative procedures are used to keep the clocks in different machines approximately synchronized, the time stamps in audit messages will be in proper sequence.

Because a single process is limited in the number of files it can have open at one time, the merge is performed in multiple processes. This allows *auditreduce* to process files from an arbitrarily large number[4] of machines. Considerable effort has been made to ensure that *auditreduce* performs efficiently even for very large configurations.

Selecting Audit Messages

The other half of *auditreduce* is message selection: choosing which messages will be passed through to the display or analysis programs. Selection options are provided to select on any criterion which can be assessed from a single message: time, type of message, selection class, originating user, security label, etc. The selections are all performed at the earliest possible point in the merge, in the subprocesses. This reduces the amount of data which travels among the family of processes creates by an *auditreduce* invocation. Selection by time is the most important heavily optimized criterion: as described above, at a coarse granularity, messages can be selected by time based only on the timestamps in filenames, and without opening a file unless it is known to contain messages from the interval of interest.

Audit Migration Facilities

Some miscellaneous facilities are provided by *auditreduce*, primarily in support of the audit file migration strategies described above. Messages are combined from multiple files into one using the options to write an output file, delete input files, and read all messages from any input files processed, even if the messages are outside the specific time intervals specified. Input files can be in compressed format, and output files may be requested to be written in compressed format.

Compression is performed using the standard SunOS *compress* program, which uses adaptive Lempel-Ziv coding. On English text, the typical compression ratio is only 50 to 60%, but on SunOS MLS audit data, it generally achieves 75% to 90% compression. These ratios can be achieved with audit files containing 1 megabyte of data. The compression is most efficient when large amounts of audit data are being collected, since when a single process generates many messages in rapid succession, the messages will usually have significant redundant content which can then be removed by *compress*.

AUDIT MESSAGE FORMAT

The final important aspect of auditing in SunOS MLS is the format for audit messages. Because this format offers significant benefits for third-party software developers, it is being proposed as an extension to the IEEE POSIX[5] standard, and is being considered by both the IEEE P1003.6 committee and the X/Open[6] security subcommittee.

Goals

The basic problem which makes audit messages difficult to interpret and analyze is that they come from a wide variety of sources and contain many different types of information. For example, SunOS MLS can generate approximately 300 distinct audit messages. Despite all this variety, however, the messages contain only a relatively few distinct types of data which are interesting for analysis: times, labels, file pathnames, subject (process) identities, etc. The multiplicity of formats is caused by the need to report different sets of these datatypes for different operations.

The goals of the audit message format are fourfold:

1) Easy selection of audit messages on a variety of criteria;

2) Easy addition of new audit messages as functions are added to the system (without changes to audit analysis tools);

3) Allowing third-party software developers to create additional audit analysis tools which are independent of a particular version of SunOS MLS; and

4) Allowing third-party software to generate its own audit messages which can be meaningfully analyzed with existing analysis tools.

The initial implementation of auditing in SunOS MLS did not meet these goals. It used an inflexible, fixed-format message, in which additional data was simply tacked on following the header in a message-dependent way. As a consequence, both *auditreduce* (for message selection) and *praudit* had to understand the format of every single audit message. Whenever a new type of audit message was added to the system, *praudit* always (and *auditreduce* often)

[4] Limited only by configurable table size limits in the kernel. The implementation has worked well with over 1000 files. The number of files which must be open at a time is equal to the number of machines which generated them: exactly one file at a time from each machine is needed because the files, as well as the messages in them, are kept in strict time sequence.

[5] POSIX is the IEEE's Portable Operating System Interface specification, which is based on common UNIX system interfaces. The P1003.6 committee is developing security extensions for the basic POSIX functions suitable for use at all TCSEC levels.

[6] X/Open is an international organization of UNIX system vendors which develops portability standards based on its members' systems. The security subcommittee is developing security extensions intended primarily for commercial applications and the C2 TCSEC level.

had to be modified to understand the specific message format associated with the new message type. This was clearly undesirable even within the scope of SunOS MLS development, to say nothing of its consequences for third-party developers. As a message format, it satisfied none of the above goals.

The remainder of this section discusses how these goals are met by the current design.

Audit File Format

In the scheme described here, an audit file is treated as a sequentially accessed stream of bytes. The stream is broken into variable-length records. Each audit file contains an identifying header, followed by an arbitrary number of records, as shown below:

Header	Message #1	Message #2	....
0	256	327	361

> NOTE: The numbers along the bottom of all the diagrams indicate byte *offsets* from the beginning. In this diagram, they are only for illustrative purposes, and do not represent any required values.
>
> In the diagrams showing individual tokens, the number at the beginning of each token is its token type, which is a one-byte value appearing at the beginning of all tokens to identify their contents.

Although the size is arbitrary, it is useful, though not required, to keep the audit files to a manageable size by periodically instructing the audit daemon to switch to a new file.

Although this format allows only sequential access[7], and does not support backward reading or random access, its simplicity is important, because it allows audit data to be passed between programs easily, or moved between systems without regard to internal file formats.

Flexible Audit Message Format

The message format treats each audit message as a string of "audit tokens". Each of the tokens is a self-identifying piece of data, representing a file pathname, a subject, a label, etc. The token starts with an identifying byte, which is followed by a string of bytes representing the rest of the data in a token type dependent format. A message looks something like this:

12	Header Token	24	Subject Token	33	Label Token	23	Path Token
0	1	13	14	24	25	56	57

To a large extent, each audit message, and even each token within the message, can be considered independently of all others, which simplifies interpretation and message selection.

There are three classes of tokens: Control, Data, and Modifier (identified as C, D, and M in the table below). Each of these classes contains several distinct token types, identified by the one-byte identifier at the beginning of each token. There are currently 17 defined token types.

Control tokens are essentially part of the audit system's overhead: they identify the beginning (and end) of messages. Data tokens provide the primary identification of a subject or object: a data token should provide enough information to know what the message is referring. A data token may be followed by one or more modifier tokens. Modifier tokens provide additional information about a subject or object. This information is not included with the data tokens for two reasons, both having to do with the applicability of this message format to arbitrary systems, not just SunOS MLS. First, an implementation could choose to save space by not recording information that its customers don't care about (for example, file attributes or the supplementary group list). Second, an implementation can always save space by not recording information that doesn't make sense for that system (such as labels in a C2 system). These variations represent an implementation's "auditing style", and may be built in to the system or available to an administrator as configuration options. Because the individual audit tokens are largely self-defining, an analysis program can work regardless of the auditing style of the system generating the messages.

The average size of SunOS MLS audit messages is between 120 and 180 bytes, with 6 to 10 tokens per record. The compression typically reduces the message size to between 20 and 30 bytes of compressed data per message.

Example of Audit Message

As an example, the audit message for an *unlink*[8] system call might contain the following tokens, laid out in the message as shown in the previous diagrams:

Header Token

12	Message Length	Message Type	Message Time	
0	1	3	5	12

Subject Token

24	Audit User ID	Real User ID	Effective User ID	Real Group ID	Process ID	
0	1	3	5	7	9	10

[7] This restriction applies only to the simplest implementations. The TRAILER token type allows backward reading and binary searching.

[8] The operation *unlink (Path)* removes a link to the file named *Path*, deleting the file's contents if that was the last link to the file.

Label Modifier Token

33	32-byte binary label (Sun specific format)

0 1 32

Pathname Token

23	Root Directory	Working Directory	Pathname Argument

0 1 (variable) (variable)

The first token, present in all audit messages, is the header, which gives the type, time, class, result, and length of the entire message. The second token, present in most messages, identifies the subject performing the operation. The third is a label, the label of the subject. This is an independent (modifier) token to allow the format to be used on systems (such as class C2) which do not implement labels and therefore would not want to reserve space for labels in all their audit messages. The fourth token is the file pathname for the target of the link.

As this is an example only, it is somewhat simplified: the actual audit message for *unlink*() also includes the label of the object being unlinked and the return value from the system call (to indicate success or failure).

Audit Token Types

The message header token is present in all audit messages, and contains three pieces of information in a fixed format: the message type, the time the message was generated, and the total length of the message. The total length of the message is used to allow sequential processing of the variable-length messages. The message type is used to identify a specific operation, such as a system call or administrative operation (see **Audit Message Pre-Selection**, above).

The subject token identifies a subject (process). It contains the process's process ID, audit user ID, real user ID, and effective user ID. For a system with mandatory access control, this token is always followed by a label token identifying the subject's label. The subject's audit user ID is an identity which is assigned at login time and cannot be changed even by the *setuid* system call (unlike the "real" and effective user IDs). In a system with mandatory access controls (such as SunOS MLS), a subject token is always followed by a label modifier token.

The file path token type contains the complete pathname needed to identify an object, including the process's current root directory and working directory, as well as the name which was supplied for the object itself. All three are always included, even though the pathname supplied as the argument to a system call might be an absolute pathname, making the working directory irrelevant. Similarly to subject tokens, in SunOS MLS, a path token are always followed by a label modifier token unless the designated object does not exist.

In addition to these token types, there are others which identify Two additional token types allow the inclusion of arbitrary text of binary data in a message. These are used when the data does not correspond to any of the defined token types, and where additional data about an operation is required. Text and data tokens are distinct types to allow the analysis tools to select on the contents of text. An opaque data token is generally intended for interpretation by a special-purpose analysis tool, whereas the text token and miscellaneous/arbitrary data tokens are intended for reading by a human auditor.

The table below lists all the defined token types, their class (C for control tokens, D for data tokens, M for modifier tokens), and a brief description.

Name	Class	Description
HEADER	C	Beginning of a message (length, type, time)
TRAILER	C	End of a message; contains the length for backward reading
SUBJECT	D	Subject attempting the audited operation
SERVER	D	Identity of server process acting for subject
DATA	D	Miscellaneous binary data; includes information about datatype (character, integer, etc.) and instructions for printing (decimal, hexadecimal, string, etc.)
PATH	D	Complete pathname(s) identifying a file system object (root directory, current directory, and supplied name)
IPC	D	System V IPC object (Shared Memory, Semaphore Set, Message Queue)
PROCESS	D	Process that is target of operation
TEXT	D	Text message; distinct from DATA in that length is implicit, reducing the token's size
RETURN	D	Return value and error code from system call
OPAQUE	D	Application-specific structured binary data; generated only by non-TCB programs
PACKET	D	Header and identifying information from an IP packet
ATTR	M	Attributes (type, owner, permissions, etc.) of file system object
IPC_ATTR	M	Attributes of System V IPC object
LABEL	M	Label for subjects and objects
GROUPS	M	Group list (supplementary group IDs) for a subject
NET_ADDR	M	Address (4-byte IP format) identifying location of a subject or object

Writing Audit Messages

To further insulate programs generating audit messages from their format in storage, a function is provided which accepts as arguments the message type, class, and pointers to data to be inserted as additional tokens in the message. Because a file token is generated from a name and inode pointer (or perhaps just a name), this allows a generating program to supply these pointers without worrying about whether the system has mandatory access control so that labels have to be included in the audit message.

This interface is available both within the kernel, for internal use by the SunOS MLS TCB, and as a system call for use by the trusted processes in the SunOS MLS TCB and by third-party trusted software.

Application-Generated Audit Messages

The system allows programs other than the supplied TCB software to generate audit messages. This allows an installation to write programs that generate audit messages describing their activities. Because these messages use the same token-based format as TCB-generated audit messages, they can be analyzed with the same tools.

If these messages could mimic the messages generated by TCB software, or in some way overwhelm the audit system's capacity, the integrity of the audit trail would be lost. The system protects against this in two ways. First, all application-generated messages are identified by a specific message type, set by the TCB when the message is written. This precludes programs from imitating genuine TCB audit messages since the message types will always differ. Second, application-generated messages belong to a special class of audit messages, and are only recorded if that class of messages is being audited. Thus, the system administrator can control, on a per-user basis, which users are permitted to generate non-TCB audit messages.

IMPLEMENTATION CHARACTERISTICS

The SunOS MLS audit mechanism is quite similar to other UNIX-based audit implementations (such as [Gligor86] and [Picciotto87]). The principal differences are the system-independent nature of message and file formats and the need for a "single-system view" assembled dynamically (by *auditreduce*) from a potentially widely-distributed collection of audit data files. This section explores those differences and summarizes the performance characteristics of the SunOS MLS implementation. The comparisons are not made with any other specific systems, but rather with general characteristics that appear in many systems.

Comparison With Other Implementations

A daemon process for writing audit data was chosen, despite the small additional overhead it entails, to de-couple the writing of audit data from its generation in the kernel. This simplifies use of the audit trail by non-kernel software, but mostly is important because it allows the target location (file or otherwise) of audit messages to be chosen with great flexibility.

The additional levels of buffering bring a cost in reliability, by increasing the amount of data lost in the event of failure, but this seems more than compensated for by the automatic file switching capability provided by the daemon. In any case, the maximum amount of data loss is limited and predictable, and the daemon structure is such that a more reliable transport mechanism (or perhaps one using non-erasable optical storage) could easily be integrated, whereas such a change might be very difficult in a system where the kernel does all message processing directly.

Audit messages in SunOS MLS are larger than in many other systems, because of the additional information they include for identifying objects. This resulted from a tradeoff between simplicity of analysis tools and size of messages: the less context the analysis tool has to remember (such as each process's current working directory), the easier its job is. In practice, this seems largely compensated for by the degree of compression provided by the *compress* program. When the audit data is particularly bulky and contains mostly redundant information, compression ratios of nearly 8 to 1 are possible. Thus, although the data is temporarily bulkier, in permanent storage (after the automatic daily consolidation), the bulk is comparable to other implementations. The additional CPU overhead for decompression at analysis time appears minimal.

Similarly, the machine-independent format carries a significant space penalty relative to other implementations, and again, this results from the tradeoff between audit trail size and flexibility of analysis tools. This tradeoff, too, is largely masked by the efficiency of compression.

The *auditreduce* program, in combination with self-identifying data in messages, provides essentially all the types of selection and analysis that can be provided when examining messages sequentially. The audit class mechanism provides some capability for pre-selection, but is not nearly as powerful as *auditreduce*.

The SunOS MLS audit mechanism is intended to meet or exceed the TCSEC B1 requirements specifying which events are to be audited and what forms of selective auditing may be performed. However, it is also intended to meet practical needs, both for human auditors and automated analysis systems, such as the the Intrusion Detection Expert System (IDES) [Lunt88], which analyzes patterns in audit data to detect unauthorized use of a system. The capabilities of *auditreduce* are particularly important for manual interpretation of audit data.

Performance Characteristics

As SunOS MLS had not been distributed to the field when this paper was written, these numbers are necessarily tentative. However, they indicate that the size of data collected and the overhead for collection is quite comparable to that for other systems. Most of the numbers below describe size of the raw binary audit data; compressed data is treated at the end.

With a minimal set of audit classes selected (logins, logouts, and administrative and privileged activity), a system of 10 SunOS MLS machines (workstations and servers) generates about 100K bytes of uncompressed audit data per day for the entire system. If auditing of failed operations is added, this increases to 1–2 megabytes per day. If auditing of all event classes for success and failure is enabled, this increases to 10-30 megabytes per day (again, for the whole 10-machine sy: tem). It must be emphasized that these numbers are generated by the "normal" activity of a software development group, which consists primarily of text editing and compilation. Any heavy file system activity increases the bulk considerably.

The maximum capacity of the audit system seems to be about 20 megabytes of raw data per hour on a typical machine. If all audit classes are turned on, and the machine is set to running a test suite which primarily exercises the file system, it can generate about that much data in an hour. The machine is still usable in this state; although performance is certainly slowed, normal interactive work can still take place in much the same way as on a slower (previous generation) machine.

When audit data is compressed (by the automatic daily consolidation), typical compression ratios range from 3.5–to–1 to 5–to–1. When the audit data is heavily redundant (such as when all audit classes are selected), the compression ratio can reach 8–to–1. This reduces a daily 30 megabytes to 7 or 8, or the flat-out 20 megabytes per hour per machine to a more manageable 2.5.

Performance of any audit system is so dependent on the nature of the workload as to essentially defy characterization. With the minimal set of audit classes described earlier, the performance impact is negligible. Performance impact on machines used as file servers is also essentially negligible, since auditing and access control is performed on the client machines. This is less true for machines used as servers for file systems receiving audit data, although even there, buffering in the client machines reduces the impact. Since a SunOS MLS file server can support an aggregate throughput (for all its clients) exceed 200K bytes per second, even an additional 20 megabytes per hour represents a small fraction of that capacity.

Finally, auditing of security-relevant events does not affect the performance of CPU-bound programs. Because a SunOS MLS system is typically not resource limited except for CPU-bound jobs or relatively brief periods of heavy I/O activity, the most important measure of auditing performance may be perceived impact on response time, which is minimal because of the high performance of the individual workstations.

CONCLUSIONS

Distributed systems pose significant difficulties in storing audit messages. Use of multiple buffers and failure recovery algorithms makes auditing practical and efficient in a distributed system.

The *auditreduce* tool gives the administrator of a distributed system the all-important big picture. It also provides the management capabilities for maintaining and archiving the enormous volumes of audit data which are created in a large SunOS MLS configuration.

Pre-selection of "interesting" audit messages is important for reducing the volume of messages generated. As yet, the capabilities for doing so in SunOS MLS are rather primitive, but further work is planned to investigate selection by label, by object identity, and other potentially interesting criteria. Even so, the current implementation allows the volume of audit data to be adjusted over nearly two orders of magnitude.

Because of the general message format, it is straightforward to use auditing in third-party trusted software, and to create third-party analysis tools. This has already happened within Sun: several audit display tools have been created outside the product development effort, and it is hoped that similar efforts will take place at field sites once the product is delivered.

As of this writing, there is too little experience with SunOS MLS to quantify the performance impact of auditing, and even the storage requirements are not entirely clear.

REFERENCES

[Denning86] Dorothy E. Denning, *An Intrusion-Detection Model*, Proceedings 1986 IEEE Symposium on Security and Privacy, 7–9 April 1986, Oakland, California

[DoD85] Department of Defense – Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD-5200.28-STD, December 1985

[Gligor86] V. D. Gligor, et al., *On the Design and Implementation of Secure Xenix Workstations*, Proceedings 1986 IEEE Symposium on Security and Privacy, 7–9 April 1986, Oakland, California

[Lunt88] Teresa F. Lunt and R. Jagganathan, *A Prototype Real-Time Intrusion-Detection Expert System*, Proceedings 1988 IEEE Symposium on Security and Privacy, 18–21 April 1988, Oakland, California

[Picciolo87] J. Picciolo, *The Design of an Effective Auditing Subsystem* Proceedings 1987 IEEE Symposium on Security and Privacy, 27–29 April 1987, Oakland, California

[Sun87] Sun Microsystems, *Computer Security at Sun Microsystems, Inc.*, Proceedings 10th National Computer Security Conference, 21–24 September 1987, Baltimore, Maryland

Integrating Security in a Large Distributed System

M. Satyanarayanan
Department of Computer Science
Carnegie Mellon University

Abstract

Andrew is a distributed computing environment that is a synthesis of the personal computing and timesharing paradigms. When mature, it is expected to encompass over 5000 workstations spanning the Carnegie Mellon University campus. This paper examines the security issues that arise in such an environment and describes the mechanisms that have been developed to address them. These mechanisms include the logical and physical separation of servers and clients, support for secure communication at the remote procedure call level, a distributed authentication service, a file-protection scheme that combines access lists with Unix mode bits, and the use of encryption as a basic building block. The paper also discusses the assumptions underlying security in Andrew and analyses the vulnerability of the system. Usage experience reveals that resource control, particularly of workstation CPU cycles, is more important than originally anticipated and that the mechanisms available to address this issue are rudimentary.

1. Introduction

Andrew is a distributed computing environment that has been under development at Carnegie Mellon University since 1983. An early paper [18] describes the origin of the system and presents an overview of its components. Other papers [24, 10] focus on the distributed file system that is the information sharing mechanism of Andrew.

The characteristic of Andrew that has influenced almost every aspect of its design is its scale. The belief that there will eventually be a workstation for each person at CMU suggests that Andrew will grow into a distributed system of 5000 to 10000 nodes. A consequence of large scale is that the laissez-faire attitude towards security typical of closely-knit distributed environments is no longer viable. The relative anonymity of users in a large system requires security to be maintained by enforcement rather than by the goodwill of the user community.

A sizable body of literature exists on algorithms for security in distributed environments. The survey by Voydock and Kent [28] describes many of these algorithms and discusses the basic security problems they address. In contrast, this paper focuses on the design and implementation aspects of building a secure distributed environment. It puts forth the fundamental assumptions on which security in Andrew is based, examines their effect on system structure, describes associated mechanisms, and reports on usage experience.

Although Andrew is no longer an experimental system it is far enough from maturity that many of its details are still evolving. Rather than trying to describe a moving target, this paper presents a snapshot of Andrew at one point in time. The point of reference is the date of the official inauguration of Andrew, on November 11 1986. At that point in time there were over 400 Andrew workstations serving about 1200 active users. The file system stored 15 gigabytes of data, spread over 15 servers. The system was then mature and robust enough to be in regular use in undergraduate courses at CMU and in demonstrations of Andrew at the EDUCOM conference on educational computing. In the rest of this paper the present tense refers to the state of the system at this reference point. Exceptions to this are explicitly stated.

The paper begins with an overview of the entire system and an identification of its major components. Section 3 then discusses the underlying assumptions and the conditions that must be met for Andrew to be secure. Sections 4 to 7 describe the protection domain, authentication, and enforcement of protection in the distributed file system. Section 8 discusses the problem of resource control. Section 9 underlines the fundamental role of encryption and proposes that encryption hardware be made an integral part of all workstations in distributed environments. Section 10 deals with various other security concerns, while Section 11 examines the ways in which the security of Andrew could be compromised and suggests solutions to some of the possible modes of attack. Finally, Section 12 ends the paper with an outline of changes that are in progress or have occurred since the snapshot presented here.

2. System Structure

Andrew combines the user interface advantages of personal computing with the data sharing simplicity of timesharing. This synthesis is achieved by close cooperation between two kinds of components, *Vice* and *Virtue*, shown in Figure 1. A Virtue workstation provides the power and capability of a dedicated personal computer, while Vice provides support for the timesharing abstraction. Although Vice is shown as a single logical entity in Figure 1, it is actually composed of a collection of servers and a complex local area network. This network spans the entire CMU campus and is composed of Ethernet and IBM Token Ring segments interconnected by optic fibre links and active elements called *Routers*. Figure 2 shows the details of this network.

Andrew is a joint project of Carnegie Mellon University and the IBM Corporation. The author was supported in the writing of this paper by the National Science Foundation (Contract No. CCR-8657907). The views and conclusions in this document are those of the author and should not be interpreted as representing the official policies of the National Science Foundation, the IBM Corporation or Carnegie Mellon University.

Each Virtue workstation runs the Unix 4.2BSD operating system[1] and is thus an autonomous timesharing node. Multiple users can concurrently access a workstation via the console keyboard, via the network or via lines that are hardwired to the workstation. But the most common use of a workstation, and the usage mode most consistent with the Andrew paradigm, is by a single user at the console.

A distributed file system that spans all workstations is the primary data-sharing mechanism in Andrew. In Virtue, this file system appears as a single large subtree of the local file system. Files critical to the initialisation of Virtue are present on the local disk of the workstation and are accessed directly. All other files are in the shared name space and are accessed through an intermediary process called *Venus* that runs on each workstation. Venus finds files on individual servers in Vice, caches them locally and performs emulation of Unix file system semantics. Both Vice and Venus are invisible to processes in Virtue. All they see is a Unix file system, one subtree of which happens to be identical on all workstations. Processes on two different workstations can read and write files in this subtree just as if they were running on a single timesharing system.

A mainframe computer that runs a Venus can also share Vice files. It is more likely to have multiple concurrent users and make greater use of its local file system than a Virtue workstation. It will probably enforce local resource usage controls too. From the point of view of security in Andrew, however, such a mainframe is no different from a Virtue workstation.

3. Assumptions

Saltzer [22] makes an important distinction between a *securable system* and specific *secure instances* of that system. Our purpose in this section is to describe the level of security offered by Andrew and to state the assumptions under which this is achieved. The degree to which a specific Andrew site is secure depends critically on the effort taken to meet these assumptions.

It is easiest to characterise Andrew using the taxonomy introduced by Voydock and Kent. Their survey [28] classifies security violations into unauthorised *release* of information, *modification* of information, and *denial* of resource usage. The security mechanisms in Andrew primarily ensure that information is released and modified only in authorised ways. The difficult issue of resource denial is not fully addressed. The complexity of this problem is apparent if one considers a situation where a defective piece of network hardware floods the network with packets. The resulting denial of network bandwidth to legitimate users is clearly a security violation in the strict sense of the term. However, it is not clear what Andrew could possibly do in such situations except to bring the problem to the attention of system administrators. This issue of resource control is discussed at length in Section 8.

Alternative taxonomies of security also exist. Wulf [30], for instance, considers the security of the Hydra operating system in the light of the problems of *mutual suspicion, modification, conservation, confinement,* and *initialization*. It is more difficult to characterise Andrew within this framework. Since Vice and Virtue do not trust each other until a user successfully executes the authentication procedure described in Section 5, there is indeed mutual suspicion. But users do depend on Vice to provide safe, long- term storage of their files and to enforce their protection policies. Andrew can protect against modification of files by other users, but there is no safeguard against incorrect modifications by Vice itself. Since Andrew supports revocation it does address the problem of conservation. But the problem of confinement, extensively discussed by Lampson [15], is one that Andrew makes no attempt to solve. It is not clear how the initialization problem in Wulf's model applies to Andrew.

The Department of Defense taxonomy of computer systems described by Schell [26] classifies computer systems into four major categories with numerous subcategories. Security ranges in strength from class D (*minimal protection*) to class A2 (*verified implementation*). In this classification scheme, Andrew appears to fit best into class C2 (*controlled access*) or, possibly, B1 (*labelled security*).

For simplicity, we shall restrict our attention in the rest of this paper to the model put forth by Voydock and Kent. We do recognise, however, that a complete analysis of Andrew security in terms of a variety of taxonomies would be a valuable exercise in itself.

A fundamental assumption pertains to the question of who enforces security in Andrew. Rather than trusting thousands of workstations, security in Andrew is predicated on the integrity of the much smaller number of Vice servers. These servers are located in physically secure rooms, are accessible only to trusted operators, and run software that is above suspicion. No user software is ever run on servers. For operational reasons, it is necessary to provide utilities that can be run on servers to directly manipulate Andrew file system data. These utilities can be run only by superusers on servers.[2] Both access to servers and the ability to become superuser on them must be closely guarded privileges.

Workstations may be owned privately or located in public areas. We assume that owners may modify both the hardware and software on their workstations in arbitrary ways. It is therefore the responsibility of the user to ensure that he is not being compromised by software on a private workstation. Such a piece of software, referred to as a *Trojan horse* [9], is trivially installed by a superuser. Consequently the user has to trust every individual who has the ability to become superuser on the workstation. A user who is seriously concerned about security would ensure the physical integrity of his workstation and would deny all remote access to it via the network.

In the case of a public workstation, it is assumed that there is constant surveillance by administrative personnel to ensure the integrity of hardware and software. It is relatively simple to visually monitor and detect hardware tampering in a public area. But it is much harder to detect a miscreant becoming superuser and installing a Trojan horse. Keeping the superuser password on a workstation secret is not adequate because workstations can be easily booted up standalone, with the person at the console acquiring superuser privileges. An organisation that is serious about security would have to physically modify workstations so that only authorized personnel can boot up public workstations standalone. At the present time public workstations at CMU do not have such physical safeguards.

It is common for a pool of private workstations to be used by a small collection of users. Workstations located in shared offices or laboratories are examples of such situations. From the point of view of security, such workstations are effectively co-owned by all users who can physically access them. It is their joint responsibility to ensure the integrity of the hardware and software on the workstations.

It should be emphasised that the preceding discussion of software integrity on workstations pertains to local files. There are usually only a few such files, typically system programs for initialising the workstation and for authenticating users to Vice. All other user files are stored in Vice and are subject to the safeguards discussed in Section 6.

The network underlying Andrew has segments in every building at CMU, including student dormitories. It is impossible to guarantee the physical integrity of this network. It can be tapped at any point, and private workstations with modified operating systems can eavesdrop on network traffic. A consequence of these observations is that end-to-end mechanisms based on encryption are the only way to ensure secure communication between Vice and Virtue. These mechanisms are described in Section 5.

The routers shown in Figure 2 are dedicated computers that run specialised software. The integrity of these routers is not critical to Andrew security. Because Andrew uses end-to-end encryption, a compromised router cannot expose or modify information that is transmitted through it. At worst, it can cause packets to be misrouted or

[1] Unix is a trademark of AT&T.

[2] The servers also run Unix 4.2BSD. A "superuser" is a privileged Unix user free of normal access restrictions.

modified in ways that cause the receiver to reject them. These are essentially cases of resource denial, which Andrew does not attempt to address completely. Physical damage to a network segment has similar consequences.

Finally, the design of the Andrew file system postulates the use of an independent, secure communication channel connecting all the Vice servers. This is used for administrative functions such as tape backups and distribution of the protection database described in Section 4. This secure channel has to be realised either by a separate, physically secure network or by the use of end-to-end encryption as in the case of Vice-Virtue communication. At the present time, neither of these of measures is used at CMU. The secure communication channel is the same as the public network, and communication on it is unencrypted.

4. The Protection Domain

The fundamental protection question is "Can agent *X* perform operation *Y* on object *Z*?" We refer to the set of agents about whom such a question can be asked as the *Protection Domain* [23]. In Andrew, the protection domain is composed of *Users* and *Groups*. A user is an entity, usually a human, that can authenticate itself to Vice, be held responsible for its actions, and be charged for resource consumption. A group is a set of other groups and users, associated with a user called its *Owner*. The name of the owner is a prefix of the name of the group. It is possible to impose meaningful structure in the names of groups, although Andrew ignores such structure. For example, "Bovik:Friends", "Bovik:Friends.CatLovers", and "Bovik:Friends.CatHaters" could mnemonically indicate the purpose of three groups owned by user "Bovik".

Vice internally identifies users and groups by unique 32-bit integer identifiers. An id cannot be reassigned after creation. Such reassignment would require elimination of all existing instances of the id from long-term Vice data structures, an operational nightmare in a large distributed system. User and group names, on the other hand, can easily be changed.

A distinguished user named "System" is omnipotent; Vice applies no protection checks to it. Our original intent was that "System" would play the same role that a superuser plays in Unix systems. In practice we have found it more convenient to define a special group named "System:Administrators." It is membership in this group rather than authentication as "System" that now endows special privileges. An advantage of this approach is that the actual identity of the user exercising the privileges is available for use in audit trails. We consider this particularly important in view of the scale of Andrew. Another advantage is that revocation of special privileges can be done by modifying group membership rather than by changing a password and communicating it securely to the users who are administrators.

The protection domain includes two other special entities: the group "System:AnyUser", which has all authenticated users of Vice as its implicit members, and the user "Anonymous" corresponding to an unauthenticated Vice user. Neither of these special entities can be made a member of any group. Although the current implementation blurs the distinction between these two entities[3], we forsee situations where the distinction would be valuable. For example, when the support for independent administrative domains discussed in Section 10.3 is operational it would be convenient to be able to recognize and grant specific privileges to all authenticated members of a particular administrative domain.

Membership in a group can be inherited. The *IsAMemberOf* relation holds between a user or group *X* and a group *G*, if and only if *X* is a member of *G*. The reflexive, transitive closure of this relation for *X* defines a subset of the protection domain called its *Current Protection Subdomain (CPS)*. Informally, the CPS is the set of all groups that *X* is a member of, either directly or indirectly, including *X* itself. This hierarchical structuring of the protection domain is similar to the schemes in the CMU-CFS file system [1] and Grapevine [3].

The CPS is important because the privileges that a user has at any time are the cumulative privileges of all the elements of his CPS. For example, suppose "System:CMU", "System:CMU.Faculty" and "System:CMU.Students" are three groups with the obvious interpretations. If the second and third groups are members of the first, new additions to those groups will automatically acquire privileges granted to "System:CMU." Conversely, when a student or faculty member leaves, it is only necessary to remove him from those groups in which he is explicitly named as a member. Inheritance of membership thus conceptually simplifies the maintenance and administration of the protection domain. The scale of Andrew makes this an important advantage.

A common practice in timesharing systems is to create a single entry in the protection domain to stand for a collection of users. Such a collective entry, often referred to as a "group account" or a "project account," may be used for a number of reasons. First, obtaining an individual entry for each human user may involve excessive administrative overheads. Second, the identities of all collaborating users may not be known a priori. Third, the protection mechanisms of the system may make it simpler to specify protection policies in terms of a single pseudo-user than for a number of users.

We believe that this practice should be strongly discouraged in an environment like Andrew. Collective entries will exacerbate the already-difficult problem of accountability in a large distributed system.[4] The hierarchical organisation of the protection domain, in conjunction with the access list mechanism described in Section 6, make the specification of protection policies simple in Andrew. In spite of this we are disappointed to observe that there are some collective entries at CMU. We conjecture that this is primarily because the addition of a new user is cumbersome at present. In addition, groups can only be created and modified by system administrators. As discussed in Section 12, these problems are being addressed and we hope that collective entries will soon become unnecessary.

5. Authentication and Secure Communication

Authentication is the indisputable establishment of identities between two mutually suspicious parties in the face of adversaries with malicious intent. In Andrew, the two parties are a user at a Virtue workstation and a Vice server, while the adversaries are eavesdroppers on the network or modified network hardware that alters the data being transmitted.

From a user's point of view, using Virtue seems no different from using a standalone workstation. In response to a standard Unix login prompt, the user provides his name and password. While logged in, he may access local files as well as Vice files located on many servers. Venus establishes secure, authenticated connections to these servers as they are needed. The establishment of a connection is completely transparent to the user. In particular, he does not have to supply his password each time a new connection is made.

The authentication mechanism we use is a derivative of Needham and Schroeder's original scheme [19] using private encryption keys. The overall function is decomposed into three major components:

- a *Remote Procedure Call* mechanism that provides support for security.
- a scheme for obtaining and using *Authentication Tokens*.
- an *Authentication Server* that is a repository of password information.

[3] Files stored in Vice by an unauthenticated user appear as if they were stored by "System:AnyUser" rather than by "Anonymous."

[4] Our use of the group "System:Administrators" rather than the pseudo-user "System" is motivated in part by this concern.

5.1. Secure RPC

Early in our implementation, it became clear that the remote procedure call package used between Vice and Virtue was a natural level of abstraction at which to provide support for secure communication. Birrell's report on security in the Cedar RPC package [4] independently confirmed the validity of our decision.

The interface of the RPC package is described in detail in the user manual [25]. When a client wishes to communicate with a server, it executes a BIND operation that sets up a logical *Connection*. Connections are relatively cheap to establish and require only about a hundred bytes of storage overhead at each end. A connection may be set up to be at one of four levels of security:

OpenKimono neither authenticated nor encrypted.

AuthOnly authenticated, but RPC packets not encrypted.

HeadersOnly authenticated and RPC packet headers, but not bodies, encrypted.

Secure authenticated, and RPC packets fully encrypted.

Only the last of these four levels provides true end-to-end security; the second and third levels are provided as a compromise between security and efficiency, and the first can be used when secure communication is not required.

A client can specify the kind of encryption to be used when establishing a connection. The server provides a bit mask indicating the kinds of encryption it can handle, and will reject attempts by a client to use any other kind. This flexibility makes it feasible to equip servers with encryption hardware as well as a suite of software encryption algorithms of differing strength and cost. A workstation owner can make a tradeoff between economy, performance and degree of security in determining the kind of encryption to use. The preferred approach is, of course, to equip all workstations with encryption hardware. Section 9 discusses encryption in greater detail.

For all the authenticated security levels, the BIND operation involves a 3-phase handshake between client and server. The client side of the application provides a variable-length byte sequence called *ClientIdent*, and an 8-byte encryption key for the handshake. The server side of the application supplies a procedure, *GetKeys*, to perform key lookup and a procedure, *AuthFail*, to be invoked on authentication failure. The latter allows the server to record and possibly notify an administrator of suspicious authentication failures.

At the end of a successful BIND, the server is assured that the client possesses the correct handshake key for ClientIdent. The client, in turn, is assured that the server is capable of deducing the handshake key from ClientIdent. The possesion of the handshake key is assumed to be prima facie evidence of authenticity.

The steps performed by the RPC package during BIND are as follows:

1. The client chooses a random number, X_r and encrypts it with its handshake key, *HKC*. It sends the result, $(X_r)^{HKC}$, and ClientIdent (in the clear) to the server.
2. When the BIND request arrives at the server, the RPC package invokes GetKeys with ClientIdent as a parameter.
3. GetKeys does a key lookup and returns two keys. One of these keys is a handshake key, *HKS*, and the other is a session key, *SK*, to be used after the connection is established. If the return code from GetKeys indicates that the key lookup was unsuccessful, the BIND request is rejected immediately and AuthFail is invoked with ClientIdent and the network address of the client as parameters.
4. Otherwise the server decrypts $(X_r)^{HKC}$ with its handshake key, yielding $((X_r)^{HKC})^{HKS}$.
5. The server adds one to the result of its decryption, then encrypts this and a new random number, Y_r, with its handshake key. It sends the result, $(((X_r)^{HKC})^{HKS}+1), Y_r)^{HKS}$, to the client.
6. The client uses its handshake key to decrypt this message. If *HKC* and *HKS* match, the first number of the decrypted pair will be (X_r+1). If this is the case, the client concludes that the server is genuine. Otherwise the server is a fake and BIND terminates.
7. The client adds one to the second number of the decrypted pair and encrypts it with its handshake key. It sends the result, $(((Y_r)^{HKS})^{HKC} + 1)^{HKC}$, to the server.
8. The server decrypts this message with its handshake key. If *HKC* and *HKS* match, the decrypted number will be (Y_r+1). In that case the server concludes that the client is genuine. Otherwise the client is a fake and the BIND terminates after AuthFail is invoked.
9. The server then encrypts the session key, *SK*, and a randomly chosen initial RPC sequence number, *x0*, with its handshake key. It completes BIND by sending the result, $(SK, x0)^{HKS}$, to the client. All future encryption on this connection uses *SK*. The sequence numbers of RPC requests and replies will increase monotonically from *x0*.

The correctness of this authentication procedure hinges on the fact that possession of the handshake key by both parties is essential for all steps of the handshake to succeed. Without the correct key, it is extremely unlikely that an adversary will be able to generate outgoing messages that correspond to appropriate transformations of the incoming messages. Mutual authentication is achieved because both the client and the server are required to demonstrate that they possess the handshake key. The use of new random numbers for each BIND prevents an adversary from eavesdropping on a successful BIND and replaying packets from that sequence.

Figure 3 summarises the steps involved in the BIND authentication procedure. It is important to note that the RPC package makes no assumptions about the format of ClientIdent or the manner in which GetKeys derives the handshake key from ClientIdent. The next section describes how this generality is used in Andrew in two different ways: at login, to communicate with an authentication server, and each time Venus contacts a file server. A connection is terminated by an UNBIND call which destroys all state associated with that connection.

Security in Andrew is not critically dependent on the details of the authentication handshake. The code pertaining to it is small and self-contained. The handshake can therefore be treated as a black box and an alternative mutual authentication technique substituted with relative ease.

5.2. Authentication Tokens

Andrew uses a two-step authentication scheme based on *Tokens* for reasons of transparency as well as robustness. This approach provides a number of advantages over a single-step authentication scheme:

1. It allows Venus to establish secure connections as it needs them, without users having to supply their password each time.
2. It avoids having to store passwords in the clear on workstations.
3. It limits the time duration during which lost tokens can cause damage.
4. It allows system programs other than Venus to perform Vice authentication without user intervention.

Authentication tokens are pairs of objects whose possession is indirect proof of authenticity. Such a pair is like a *Capability* [14] in that no consultation with an external agency is required when using them, but is different from a capability in that it estabiishes identity rather than granting rights. Tokens are conceptually similar to *Authenticators* described by Birrell [4].

One of the components of the pair, the *Secret Token*, is encrypted at creation and can be sent in the clear. The other component, the *Clear Token*, has fields that are sensitive and should be sent only on secure connections. Both tokens contain essentially the same information: the Vice id of the user, a handshake key, a unique handle for identifying the token, a timestamp that indicates when the token becomes valid, and another timestamp that indicates when it expires. The secret token contains, in addition, a fixed string for self-identification. The appearance of this string when decrypting a secret token confirms that the right key has been used. The secret token also contains noise fields that are filled with new random values each time a token is created. This is done to thwart attempts to break the key used for encrypting tokens.

The Unix program for logging in on workstations has been extensively modified, although its user interface is unaltered. LOGIN now contacts an authentication server using the RPC mechanism described in Section 5.1. The name and password typed in by the user are used as the ClientIdent and handshake key respectively. The GetKeys routine in the authentication server obtains this password from an internal table. When the RPC handshake completes, a secure, authenticated connection has been established between LOGIN and the authentication server. LOGIN uses this connection to obtain a pair of tokens for the user. The authentication server generates a new handshake key for each pair of tokens it creates. It encrypts the secret token with a key known only to itself and the Vice file servers. LOGIN now passes the clear and secret tokens to Venus, which retains them in an internal data structure. At this point LOGIN terminates, and the user can use the workstation.

Whenever Venus needs to establish an RPC connection to a Vice file server on behalf of a user, it invokes BIND using the secret token for that user as ClientIdent and the key in the clear token as the handshake key. In the first phase of the BIND, the GetKeys routine on the server is invoked with ClientIdent as the input parameter. The server obtains the handshake key from the secret token after decrypting it. The authentication procedure is critically dependent on the assumption that only legitimate servers possess the key to decrypt secret tokens. At this point Venus and the server each have a key that they believe to be the correct handshake key. The remaining steps of the BIND proceed as described in Section 5.1, leading to mutual authentication. If the BIND is successful, the server uses the id in the secret token as the identity of the client on this RPC connection and sets up appropriate internal state.

Since tokens have a finite lifetime, a user will need to periodically reauthenticate himself. At present, tokens are valid for 24 hours at CMU. The program LOG, which is functionally identical to LOGIN, can be used for reauthentication without explicitly logging out. This allows [illegible] retain logged-in context.

When multiple users are logged into a workstation, Venus maintains a separate secure RPC connection for each of them for each of the Vice file servers they have accessed. When a user logs out of a workstation, Venus deletes his tokens. In the future Vice may support other services besides a distributed file system. The components of such services which execute in Virtue will be able to use tokens for authentication, just as Venus does at present.

5.3. Authentication Server

The authentication server, which runs on a trusted Vice machine, is responsible for restricting Vice access and for determining whether an authentication attempt by a user is valid. To perform these functions it maintains a database of password information about users. An excerpt of this database is shown in Figure 4. The passwords stored in the database are effectively in the clear, but are encrypted with a key known to the server so that non-malicious system personnel are prevented from accidentally reading the passwords. This database is used for password lookup whenever a user logs in to a Virtue workstation. It is updated whenever a user is created, deleted or has his name or password changed. Users can change their own password; other operations can only be performed by system administrators.

Server performance is considerably improved by exploiting the fact that queries are far more frequent than updates. This makes it appropriate for the server to maintain a write-through cache copy of the entire database in its virtual memory. A modification to the database immediately overwrites cached information. The copy on disk is not, however, overwriten. Rather, an audit trail of changes is maintained in the database by appending a timestamped entry indicating the change and the identity of the user making the modification. On startup the authentication server initialises its cache by reading the database sequentially. Later changes thus override earlier ones. An offline program has to be run periodically to compact the database.

The key used by the authentication server for encrypting secret tokens has to be known to all the Vice file servers. This key should be changed periodically if an Andrew site is serious about security. The Vice file servers remember the two most recent such keys and try them one after the other when decrypting a secret token. This allows unexpired tokens to be used even if the authentication server has changed keys. At present key distribution is manual; this should be automated in the future.

For robustness, there is an instance of the authentication server running on each Vice file server. These are slaves and respond only to queries. Only one server, the master, accepts updates. Changes are propagated to slaves over the secure communication channel referred to in Section 3. For this specific application, nonuniform propagation speed and the temporary inconsistencies that may result do not pose a serious problem. For further robustness, each instance of the authentication server has an associated watchdog Unix process that restarts it in the event of a crash.

Each server instance has a log file in which authentication failures and unsuccessful attempts to update the password database are recorded. Figure 5 shows an excerpt from such a log. It would not be difficult to provide a more sophisticated and timely warning mechanism for system personnel if suspicious events are observed by authentication servers.

6. Protection in Vice

As the custodian of shared information in Andrew, Vice enforces the protection policies specifed by users. The scale, character and periodic change in the composition of the user community in a university necessitate a protection mechanism that is simple to use yet allows complex policies to be expressed. A further consequence of these factors is that revocation of access privileges is an important and common operation. In the light of these considerations we opted to use an *Access List* mechanism in Andrew. The next three sections describe how access lists are implemented, how they are used for file protection, and how Vice represents and maintains information on the protection domain.

6.1. Access Lists

The access list mechanism is implemented as a package available to any service in Vice, though only the distributed file system currently uses it. An entry in an access list maps a member of the protection domain into a set of *Rights*, which are merely bit positions in a 32-bit integer mask. The interpretation of rights is specific to each Vice service. The total rights possessed by a user on an object is the union of all the rights possessed by the members of his CPS. In other words, he possesses the maximal rights collectively possessed by himself and all the groups of which he is a direct or indirect member.

An access list is actually composed of two sublists: a list of *Positive Rights* and a list of *Negative Rights*. An entry in a positive rights list indicates *possession* of a set of rights. In a negative rights list, it indicates *denial* of those rights. In case of conflict, denial overrides possession.

Negative rights are primarily a means of rapidly and selectively revoking access to sensitive objects. Although such revocation is more properly done by changes to the protection domain, the changes may take time to propagate in a large distributed system. Negative rights can reduce the window of vulnerability, since changes to access lists are effective immediately. As an example, if it is discovered that a member of a large group is misusing his privileges, he can be immediately given negative rights on objects used by the group. He can also be deleted from all groups that may directly or indirectly give him rights on those objects. After the membership changes are effective at all Vice servers, he can be removed from the negative rights lists. Negative rights thus decouple the problems of rapid revocation and propagation of information in a large distributed system. They can also be used to specify protection policies of the form "Grant rights R to all members of group G, except user U." Rabin and Tygar, in their recent work on ITOSS [21], independently confirm the advantages of providing negative privileges.

The algorithm executed during an access list check is quite efficient. Suppose A is an arbitrary access list and C is the CPS of U. The entries in A and C are maintained in sorted order. The rights possessed by U are determined as follows:

1. Let M and N be rights masks, initially empty.
2. For each element of C, if there is an entry in the positive rights list of A, inclusive-OR M with the rights portion of the entry.
3. For each element of C, if there is an entry in the negative rights list of A, inclusive-OR N with the rights portion of the entry.
4. Bitwise subtract N from M.
5. M now specifies the rights that U possesses.

Profiling of the Vice servers in actual use confirms that the overheads due to access list checks are negligible.

6.2. File Protection

Vice associates an access list with each directory. The access list applies to all files in the directory, thus giving them uniform protection status. The primary reason for this design decision is conceptual simplicity. Users have, at all times, a rough mental picture of the protection state of the files they access. In a large system, the reduction in state obtained by associating protection with directories rather than files is considerable. A secondary benefit is the reduced storage overhead on servers. Usage experience in Andrew has proved that this is an excellent compromise between providing protection at fine granularity and retaining conceptual simplicity. In the rare instances where a file needs to have a different protection status from other files in its directory, we place that file in a separate directory with appropriate protection and put a symbolic link to it in the original directory.

Seven kinds of rights are associated with a directory:

read (r)	read any file
write (w)	write any file
lookup (l)	lookup status of any file
insert (i)	insert a new file in this directory (only if it does not already exist). This is particularly useful in implementing mailboxes.
delete (d)	delete any existing file
administer (a)	modify the access list of this directory
lock (k)	lock any file. This has turned out not to be a particularly useful right, but continues to be supported for historical reasons.

The three most commonly used combinations of rights are **rl**, for read access, **rwlidk** for write access, and **rwlidka** for complete access. Figure 6 shows an example of the access list on a Vice directory. Modifications to access lists take effect immediately.

Certain privileges commonly found in timesharing systems do not make sense in the context of Andrew. Execute-only privilege, for example, is not a right that Vice can enforce since program execution is done by Virtue. Revocation of read rights is another area where Vice can do little since Virtue caches files. At best it can ensure that new versions of a file are not readable by the user whose access is revoked.

6.3. Protection Domain Representation

Protection domain information is maintained in a database that is replicated at each Vice file server. The database consists of a data file on disk and an index file that is cached in its entirety in virtual memory. The index file enables id-to-name translations in constant time, and name-to-id translations in logarithmic time. For each entry, the index also contains the offset in the data file where the first byte of information about the corresponding user or group is stored. A typical lookup of the database by user or group name involves a search to find the id, followed by a seek operation and a read operation on the data file.

Each entry in the database corresponds to a single user or group. It consists of a name and an id followed by three lists specifying membership information. The first list specifies the groups to which that user or group directly belongs, while the second list is the precomputed CPS. For a user, the third list enumerates the groups owned by the user; for a group, it is the list of users or groups who are its direct members. Each entry also has an associated access list, that is unused at the present time. We intend to allow users to directly manipulate the database via a protection server. The access lists will then control the examination and modification of group membership. Figure 7 shows an excerpt of the database.

When Venus makes a secure RPC connection on behalf of a user, the file server caches the CPS of the user in virtual memory and uses it on access list checks. At present, changes to the protection domain do not affect the cached copy until the RPC connection is terminated. It would be relatively simple to modify the server to invalidate cached CPS copies whenever the protection domain changes.

At present, changes to the protection domain are manually performed at a central site in Vice. Utilities are available to simplify the creation or deletion of a user or to modify the membership of a group. These utilities also precompute the CPS by transitive closure and construct the index file. Modifications performed at the central site are asynchronously propagated to all other Vice sites via the secure communication channel mentioned in Section 3. In our experience, the minor temporary inconsistencies that occasionally arise due to varying propagation speeds have not significantly affected the usability of the system.

7. Protection in Virtue

As a multi-user Unix system, Virtue enforces the usual firewalls between multiple users concurrently using a workstation. In addition, its role in Andrew places other responsibilities related to security on it:

- It emulates Unix semantics for Vice files.
- It ensures that caching is consistent with protection in Vice.
- It allows owners full control over their workstations, without compromising Vice security.
- It provides user and program interfaces for explicitly using the security mechanisms of Vice.

The next four sections describe these functions in detail.

7.1. Unix Emulation

Virtue provides strict Unix protection semantics for local files and a close approximation for Vice files. Each Unix file has 9 *Mode* bits associated with it. These mode bits are, in effect, a 3-entry access list specifying whether or not the owner of the file, a single specific group of users, and everyone else can read, write or execute the file.

Venus does the emulation of Unix protection for Vice files. In a prototype implementation of Andrew, the mode bits in a file were derived from the access list of its directory and could not be changed by applications. Unfortunately a few applications, such as version control software, encode state in the mode bits. In addition, our users expressed a desire to be able to prevent themselves from accidentally deleting critical files in a directory. We have therefore evolved a scheme in which the Vice access list check described in Section 6.1 performs the real enforcement of protection and, in addition, the three owner bits of the file mode indicate readability, writability or executability. These bits, which now indicate what can be done to the file rather than who can do it, are set and examined by Venus but ignored by Vice. For directories, the mode bits are completely ignored. The directory listing program, LS, has been modified in Andrew to omit mode bits for directories and show only the owner bits for files. Figure 8 shows an example of a directory listing in Vice.

Since the group mechanisms of Vice and standard Unix are incompatible Venus does not emulate Unix group protection semantics. Our experience indicates that no real applications have been affected by this. From the point of view of an application all Vice files belong to a single Unix group.

7.2. Caching Protection Information

Although ignorant of the Vice group mechanism, Venus caches protection information. When a directory is cached on behalf of a user, Vice includes rights information for the user and System:AnyUser. Future requests are checked by Venus without contacting Vice. If a different user on that workstation wishes to access the same directory, and the rights for System:AnyUser are inadequate, Venus explicitly obtains his rights from Vice. Protection information can be cached for a small number of distinct users on each directory. If there are more users on a workstation the protection checks will be functionally accurate, but will take longer because of ineffective caching. Vice notifies Venus whenever the protection on a cached directory changes.

Caching interacts with Unix semantics in a counter-intuitive manner. In Unix, protection failures can only occur when opening a file. In Andrew, a protection failure can occur when closing a file if the protection on one of the directories in its path was changed while the file was open. There is no simple solution to this problem because Vice cannot delegate the responsibility of checking access on store operations. It cannot trust the access check that Venus performs when opening a cached file.

This difference from Unix semantics affects a number of common Unix applications that do not expect the close operation to fail, and hence do not check return codes from it. In rare instances, the user of such an application may be unaware that one or more files were not stored in Vice because of a protection violation. We do try to inform users of the problem by printing a message on the workstation console. However, using the console as an out-of-band notification mechanism does not help in situations where there is no user to act upon the message. The only robust solution to this insidious failure mode is to modify the applications to check return codes.

7.3. Superuser Privileges

Certain sensitive operational procedures in Unix can only be performed by the pseudo-user "root". Workstation owners need to become root on occasion to perform these procedures. As a result, root is logically equivalent to a group account as discussed in Section 4. An RPC connection on behalf of root provides no knowledge about which actual user it corresponds to.

A further complication is that the initialisation of a workstation causes a number of standard processes belonging to root to come into existence automatically. Since there may be no users logged in, Venus may not have tokens with which to make authenticated connections for these processes.[5] We address these problems by treating root specially and granting it the same default access privileges in Vice as System:AnyUser. RPC connections made on behalf of root are unauthenticated and insecure. Usage experience indicates that this provides a good compromise between security and usability.

The *Setuid* mechanism in Unix effectively provides amplification of rights [13]. When a file marked setuid is executed, it acquires the access privileges of the owner of the file rather than the user executing the file. The interpretation and enforcement of the setuid property is done by Virtue, but Vice requires authentication tokens for the owner of the program being run setuid. Since the tokens will not be available except in the unlikely case of the owner of the file being logged in to the workstation, Andrew cannot support the setuid mechanism in its general form. However, many useful system utilities on workstations are owned by root and run setuid. Since root has only System:AnyUser privileges on Vice files, and since RPC connections for root do not require tokens, we are able to support setuid in this limited form.

If naively implemented, setuid programs owned by root would make Trojan horses trivial. A user could become root on his workstation, store a Trojan horse program in Vice and mark it setuid. If this program were run by any other user, it would be able to compromise his workstation. To guard against this, we define a special Vice user "stem." No one can be authenticated as stem, but a system administrator can make stem the owner of a file. When Venus caches a setuid file owned by stem, it translates the owner to root and honours the setuid property. If the file is not owned by stem, the setuid property is ignored.

7.4. Vice Interface

Virtue provides a number of programs to allow users to use the security mechanisms of Vice. FS is a program to allow users to set and examine Vice access lists. LOGIN, LOG, and SU are modified versions of standard Unix programs. They prompt for a password, contact the authentication server, obtain tokens and pass them to Venus. A modified version of the Unix PASSWD program allows users to change their passwords by contacting the authentication server.

For other applications, Virtue provides a library of routines to get, set and delete tokens stored by Venus. An important user of these routines is the Andrew version of the standard Unix program RSH that allows a user to execute a program on a remote workstation. Another important user is REM, a program that makes idle workstations available for remote use [20]. Both these programs extract tokens from the workstation a user is at, and passes them to the remote Venus so that it can access Vice files on behalf of the user. Since the clear and secret tokens are sent in the clear by these programs, they violate the security assumptions of Section 3. Nevertheless, these programs are popular in our user community.

There are occasions when a user may wish to voluntarily restrict his rights. For example, he may wish to run a program being debugged in an environment that will not allow it to modify critical files. Virtue allows a user to temporarily disable his membership in one or more groups. Such a group may be reenabled at a later time. We also intend to allow groups to be disabled by default, but this is not implemented at the present time except for the special group System:Administrators.

To implement this temporary disabling of membership, Virtue associates a unique integer called a *Process Access Group (PAG)* with each process. When a process forks, its child inherits the PAG. Venus associates secure RPC connections to a server with (user, PAG) pairs. Usually all the processes of a user have a single PAG. If a user disables his membership of a group, the process in which the disabling command was issued acquires a new PAG. Each time another server is contacted on behalf of the new (user, PAG) pair, Venus makes a secure RPC connection and requests the server to disable membership in the specified groups. The server constructs a reduced CPS for that connection and uses it on access list checks. PAGs also change when a LOG or SU command is executed.

8. Resource Usage

The absence of a focal point for allocation of resources makes resource control difficult in a distributed system. Processes in a typical timesharing system are constrained in the rate at which they can consume resources by the CPU scheduling algorithm. No such throttling agent exists in a typical distributed system. Another significant difference is that a process in a timesharing system has to be authenticated before it can consume appreciable amounts of resources. In contrast, each Andrew workstation can be modified to anonymously consume network bandwidth and server CPU cycles.

As discussed in Section 3, Andrew is not designed to be immune to security violations by denial of resources. However, it does provide control over some of the resources. The major resources in Andrew are:

- network bandwidth,
- server disk storage and CPU cycles,
- workstation disk storage and CPU cycles.

In the next three sections we examine how Andrew treats these resources.

8.1. Network Bandwidth

Since Andrew does not provide mechanisms to control use of network bandwidth, responsible use of the network is primarily achieved by peer

[5] Automatic logging in of root would require the password to be stored in the clear on workstations, a security risk we were unwilling to assume.

pressure and social mores of the user community. Blatant misuse, such as by flooding with packets, is relatively easy to detect. But it is hard to detect subtle misuse. For example, a malicious user can generate a level of traffic that degrades performance but does not bring useful network activity to a standstill. Or he can use multiple widely-separated public workstations to generate high volumes of traffic. Identifying the user can be particularly difficult because he can modify workstations to generate packets with arbitrary source addresses.

In our experience, network-related problems have not been due to malicious activity. Occasionally we observe high network utilisation and poor file transfer rates on segments of the network that support non-Andrew diskless workstations. The problem has not proved serious enough yet to warrant special attention. In one memorable instance, a bug in the low-level network code on workstations was triggered by a malformed broadcast packet generated by a non-malicious user during debugging. The bug affected every workstation in the environment and effectively halted all of them.

8.2. Server Usage

Because of the long-term, shared nature of the resource, we felt it important to be able to control disk usage on servers. An Andrew system administrator can specify a storage quota for the Vice files of a user. The quota is actually placed on a *Volume*, an encapsulation of a small subtree of the Vice file space [27], and can be changed with ease.

When storing a file on behalf of a user, a server will abort a store operation if his quota is exceeded. This can cause a problem similar to the one described in Section 7.1; an application program that does not check the return codes from a close operation will not report a failure caused by the quota being exceeded. But our users and system personnel consider server disk storage an important enough resource that they have tolerated this problem.

A minor exposure arises from the manner in which electronic mail is implemented in Andrew. Each user has a mailbox directory on which System:AnyUser has insert rights. Mail is delivered by storing a file in this directory. A malicious user could exhaust the quota of another user by sending him large quantities of junk mail. In practice, this has not proved to be a problem.

Although a user cannot execute a program on a server, his Venus can consume server CPU cycles in file system operations. Excessive demands on a server are a form of resource denial to other users. At present, Vice does not constrain the amount of server CPU cycles a user can utilise. It could do so, if necessary, since user requests come in on distinct RPC connections.

8.3. Workstation Usage

Andrew does not restrict the amount of space used by local files on workstations. For cached Vice files, Venus employs an LRU algorithm to limit disk usage below a value specified at initialisation. The algorithm is not infallible because read and write operations are not intercepted by Venus. It is possible for a program to open a short file and then append a large amount of data thereby exceeding the cache limit. In practice this has rarely been a problem.

Since a workstation can be privately owned, it would seem inappropriate for Andrew to constrain the use of its CPU cycles. However, the problem has proved more complex than we anticipated. The primary source of difficulty is the fact that each workstation is a full-fledged Unix system. Hence it is possible to remotely access one workstation from another via standard Unix programs such as TELNET and RSH. Since the Vice file space is identical at all workstations, it is particularly easy for a user to use any workstation as his own. Such convenience was, of course, a fundamental motivation for the distributed file system.

Unfortunately, an individual at a workstation perceives the attempt to use its cycles by another user as a security violation. This perception is particularly strong if the first user is at the console of the workstation. Totally disabling the network daemons that allow remote access is not a viable solution for two reasons. First, system personnel sometimes need to remotely access workstations for troubleshooting. Second, an owner may wish to access his workstation from home. Our modem access facilities require the network daemons to be present.

We have evolved a mechanism whereby TELNET access to a workstation can be restricted to a list of users stored in the local file system of that workstation. This restriction is, however, stronger than what most users desire. When he is not using his workstation, a user is usually amenable to others using it. It is also unacceptable for public workstations, because every Andrew user should be able to use them. At the present time we do not have a completely satisfactory solution to this resource problem. The REM system, mentioned in Section 7.4, allows a user to specify the conditions that must be satisfied for his workstation to become available for remote use. Although satisfactory to a logged-in user, this approach is harsh on the REM user who is in constant danger of having his computation aborted at the remote site. A full-fledged *Butler* mechanism [7], that migrates remote users rather than aborting them would be a more acceptable alternative.

The problem of controlling workstation CPU usage will become acute as Andrew grows. The large pool of idle workstations available for parallel computation, and the development of applications that exploit such parallelism will make remote use even more attractive in future.

9. Encryption

Security in Andrew is predicated on the ability of clients and servers to perform encryption for authentication and secure communication. The design and implementation of the encryption algorithm has to satisfy certain properties:

- It must be difficult to break, given the computational resources available to a malicious individual in a typical Andrew environment.
- It must be fast enough that neither the latency perceived by clients nor the throughput of servers be noticeably degraded.
- It must be cheap enough that it does not appreciably increase the cost of a workstation owned by an individual.

Based on considerations of strength and standardization, we have chosen the *Data Encryption Standard (DES)* [17] published by the National Bureau of Standards as the preferred encryption algorithm in Andrew. Since the encryption algorithm is a parameter to our RPC mechanism, it is possible to use other algorithms. We believe, however, that standardising on DES is appropriate in our environment. This algorithm has been publicly scrutinized for many years and although concerns have been expressed about its strength [8], we feel that DES is adequate for the level of security we require.

At the present time the latency for a simple interaction between a client and server is about 20 to 25 milliseconds, and the file transfer rate is about 50 to 70 kbytes per second. We expect these numbers to improve over time, as Venus, Vice and the routers in the network are improved. The fastest software implementation of DES that we are aware of runs at less than 5 kbytes per second on a typical workstation. Software encryption would therefore be an intolerable performance bottleneck in our system; hardware is essential.

Encryption devices embedded in low-level communication hardware have been available for some time in mainframes. In many cases such devices provide secure machine to machine communication over an insecure link, but are not accessible to higher level software. Andrew depends on end-to-end encryption where the ends are user level processes on workstations and Vice servers. Since every connection has a distinct key, only RPC software can determine the key to use in encrypting a packet. Transparent embedding of encryption capability at a low level is therefore not useful in Andrew.

Although a number of VLSI chips for DES are available [2, 29], integration of such chips into workstation peripherals is not common. A commercially available device for the IBM PC-AT [11, 12] could be used in our IBM RT-PC workstations, but its performance of 50 kbytes per second is barely adequate. We have therefore built a prototype device [6] for the IBM RT-PCs using the AMD 9568 chip. Based on our parts cost and labour, we estimate that a commercial version of this device, produced in quantity, would cost an end user between $500 and $800. As perceived by a user-level process, the time to encrypt N bytes using the device is $N * k + C$, where k is 4 microseconds per byte, and C is 470 microseconds. The overhead of the device is thus under a millisecond for a small packet and the asymptotic encryption rate is about 200 kbytes per second. We are currently redesigning the device to reduce k in the above expression to about 0.6 microseconds per byte, yielding an asymptotic encryption rate of over 1 Mbyte per second. At the present time, we do not have encryption devices for the Sun and Microvax workstations in our environment.

A difficult nontechnical problem is justifying the cost of encryption hardware to management and users. Unlike extra memory, processor speed, or graphics capability, encryption devices do not provide tangible benefits to users. The importance of security is often perceived only after it is too late. At present, encryption hardware is viewed as an expensive frill. We believe, however, that the awareness that encryption is indispensable for security in Andrew will eventually make it possible for every client and server to incorporate a hardware encryption device.

In the interim, while the logistic and economic aspects of obtaining encryption hardware are being addressed, Andrew uses exclusive-or encryption in software. Although it is trivially broken, we felt it worth our while to use it for two reasons. First, it exercises all paths in our code pertaining to security, and allows us to validate our implementation. Second, although a weak algorithm, it does require a user to perform an explicit action to violate security by decrypting data. Merely observing a sensitive packet on the network by accident will not divulge its contents.

10. Other Security Issues

We now consider three diverse questions from the viewpoint of security in Andrew:

- How do low-power personal computers access Vice files?
- Can diskless workstations be made secure?
- Is decentralised administration of Andrew possible?

Sections 10.1 to 10.3 examine these questions. In focusing only on security our discussion ignores many broader issues and implementation details.

10.1. PC Server

Personal computers (*PCs*) such as the IBM PC and Apple Macintosh differ from Andrew workstations in that they do not run Unix and often do not possess a local disk. They are thus not capable of being full-fledged clients of the Andrew File System. Since a significant number of Andrew users also use PCs, we have developed a mechanism that enables PCs to access Vice files.

Vice access from a PC is mediated by a server called *PCServer*, that makes a Unix file system transparently accessible from a PC. Since Vice files are part of the Unix file name space of an Andrew workstation, PCServer automatically makes them accessible from PCs. The primary advantage of this decoupling is that it allows the Andrew File System to exploit techniques essential to scalability, without distorting its design to accommodate machines of inadequate hardware capability.

Communication between a PC and PCServer uses a protocol distinct from that used in the Andrew file system. The protocol supports encryption using a key that is randomly generated and sent in the clear when a client-server connection is established. It does not incorporate the 3-way BIND handshake described in Section 5.1, but does support a weaker form of authentication. The workstation running PCServer also runs an authenticator process called *Guardian*. When a PC user needs to access Vice files, he supplies his Andrew user id and password. These are transmitted to Guardian, which contacts the Andrew authentication server and obtains authentication tokens in a manner identical to LOGIN, as described in Section 5.2. The password and tokens are logically sent in the clear, but are encrypted with a fixed key known to Guardian and PCServer. Although this is not secure, it does provide a modicum of privacy. Guardian hands these tokens to Venus and then forks a dedicated Unix PCServer process on behalf of the user. This process acts as the surrogate of the PC user and services file requests from his PC.

From the point of Venus, it appears as if the PC user had actually logged in at the workstation running PCServer. Enforcement of protection for Vice files is performed exactly as described in Section 6.2. The main security exposure in using PCServer is the information sent in the clear between the PC and Guardian during the establishment of a session.

10.2. Diskless Workstations

Operating workstations without local disks has been shown to be a viable and cost-effective mode of operation [16]. However, the impact of diskless operation on security has been ignored in the literature. To be secure when operating diskless, two factors have to be considered. Page traffic has to be encrypted, and workstations have to be confident of the identity of their disk servers so that Trojan horses are avoided.

How fast will encryption have to be done to avoid significant performance penalty when running diskless? Cheriton et al [5] present data from the V kernel on a Sun workstation indicating that it takes about 5 milliseconds plus disk access time to remotely read or write a random 512-byte block of data. These numbers are for file access, but to a first approximation we assume that they also hold for page access. Assuming that the server does write-behind, a page fault with replacement would involve a remote page write, a disk access at the server, and a remote page read. This yields a page fault service time of 30 milliseconds, assuming a typical disk latency of 20 milliseconds. If encryption is to degrade paging performance by no more than 5%, it has to be possible to encrypt 2 512-byte pages in no more than 1.5 milliseconds. This implies an average encryption rate of about 700 kbytes per second. For the more typical Unix page size of 4K bytes, an encryption rate in the range of 0.5 to 1 Mbyte per second still seems necessary. As described in Section 9, encryption hardware whose performance meets these demands seems feasible, though not readily available.

Mutual authentication is a more difficult problem. To perform a 3-phase authentication handshake, the client and server need to share a secret key. Where can this key be stored at the client? Embedding it in the ROM containing the boot sequence seems the only realistic solution. However, this does violate the goal, mentioned in Section 5.2, of not storing long-term authentication information in the clear on workstations. Authentication based on public keys might avoid this problem, but this has to be investigated.

Although these problems are not insurmountable, we know of no implementations of diskless workstations that address them. Concerns regarding security played a small but nontrivial part in our decision to avoid diskless operation in Andrew.

10.3. Decentralised Administration

Our discussion has assumed that there is a single protection domain for all of Andrew, and that the Vice id and Virtue id of a user are identical. While this is true at present, the growth of Andrew makes it increasingly attractive to allow multiple protection domains. The motivation for this comes from two distinct scenarios.

First, an established non-Andrew timesharing system or collection of workstations may join the Andrew environment. An existing user of both environments may have different user names and ids in the two environments. In the merged environment, Vice and Virtue will view the individual as two distinct users. Changing the id in either environment is difficult, because ids are embedded in long-term data structures in both Unix and Vice file systems.

Second, individual organisations may wish to administer a collection of Vice file servers, control their resources, and restrict access to a set of Andrew workstations. Such decentralised operation is likely to provide greater flexibility and responsiveness to users. It would also allow each organization to have its own set of privileged groups, such as System:Administrators.

A mechanism that addresses these issues by supporting independent Andrew *Cells* [31] is being implemented. A cell corresponds to a complete autonomous Andrew system, with its own protection domain, authentication and file servers, and system administrators. The name spaces of two or more such cells can be merged to form a unified Andrew environment. Users see a uniform, seamless, file name space and are not hindered by the multiplicity of protection domains.

It is Venus that makes cells transparent during file access. Each user and group id in the composite environment has a cell id as its prefix. The LOG program, mentioned in Section 5.2, allows a user to direct his authentication request to a specific cell. The authentication procedure is identical to that described in Section 5.2, except that tokens are stamped with the cell id of the authentication server who created them. Venus maintains a collection of tokens, one secret-clear pair for each cell to which the user has authenticated himself. When establishing a secure connection to a Vice server, it uses the tokens appropriate to the cell in which the server is located. If the user has not authenticated himself to that cell, he gets System:AnyUser privileges in it.

The name, id and password of an individual may be different in each cell. Application programs that translate ids to user names, such as LS, have to be modified to take this into account. However, long-term data structures on disk do not have to be modified to allow access to multiple cells. Since all Vice files stored on a server belong to the same cell, their access lists specify only users and groups who are in that cell. Thus a server does not need the cell prefix when performing an access list check. The prefix is used only when a secure RPC connection is being established.

11. Risk Analysis

In this section we briefly consider how security could be subverted in Andrew. Our analysis is not intended to be exhaustive nor is it a proof of security. Its primary purpose is to summarise the discussions of the preceding sections of this paper. A secondary goal is to illustrate the complexity of applying relatively simple security algorithms to a real distributed environment of substantial scale and diversity.

A fundamental assumption in Andrew is that encryption of sufficient strength and speed is available to Vice and Virtue. Otherwise it is trivial to violate security. For the purposes of this section, we assume that all servers and workstations have DES hardware. We also assume that all RPC connections on behalf of users are authenticated and fully encrypted.

Low-level network attacks can, at worst, result in denial of service to users. Since RPC packets are encrypted end-to-end, eavesdropping will not reveal useful information. Mutilating RPC packets will not violate security either. Such packets will be rejected by the recipient because RPC sequence numbering information is encrypted and it is extremely unlikely that a mutilated RPC packet will have the correct sequence number when decrypted.

With patience and considerable computational resources, a malicious individual could eavesdrop on client-server traffic and break the key under which the traffic is encrypted. Since a new random session key is generated when an RPC connection is established, breaking that key will only give access to one server. To masquerade as the user, the eavesdropper would have to carefully intersperse fake RPC requests encrypted under the session key. The session key is not adequate to establish connections with other servers.

Greater damage can be done by breaking the key in secret and clear tokens. One way to do this is to break the key used by the authentication server for encrypting secret tokens. Periodic changing of this key is therefore essential. An alternate way to break the key in a token pair is to observe a number of BIND requests that involve the same pair of tokens. This is unlikely, because tokens expire after 24 hours, and the number of BIND requests made by a user in that period is not likely to be sufficient to mount a serious key-breaking effort. A compromised token pair allows the miscreant to establish secure RPC connections with the privileges of the victim on any Vice file server. It is not adequate, however, to establish a secure connection to the authentication server.

The most damage is caused when the password of a user is broken, particularly if he is a system administrator. However, the password is typically used only once a day when the user is contacting an authentication server for tokens. The standard practice of changing passwords periodically will reduce the total amount of information available for key-breaking.

A well-known mode of attack is via a Trojan horse. Public workstations are particularly susceptible to this. A Trojan LOGIN program on a workstation could compromise the password of every individual who uses that workstation. As mentioned in Section 3, a concerned site should ensure that rebooting a workstation standalone is impossible for normal users. This would defeat the simplest way to install a Trojan horse.

A more subtle way to introduce a Trojan horse is by masquerading as a server that is temporarily down, and handing out fraudulent binaries. During their reboot sequence, workstations fetch new copies of a few local binaries from Vice over insecure connections. To avoid this problem, automatic updating on reboot should be disabled. Instead, the owner of the workstation should explicitly update these files, using binaries fetched on his secure RPC connection.

Workstations with multiple logged-in users make a number of other security threats possible. A malicious user with superuser privileges could cause Venus to dump core, examine the dump and extract the tokens of other logged-in users. Andrew does not provide any special mechanisms to protect against such threats. As mentioned in Section 3, users of a shared workstation have to trust all individuals who could become superuser on that workstation. A superuser can also read and modify all cache copies of files on the workstation.

Vice is critically dependent on the physical security of its servers and on carefully restricted superuser access on them. For maximum security, servers should disallow TELNET access. Physically secure machine rooms and trustworthy operators are, of course, also essential. A malicious individual with superuser access on a server could read or modify all Vice file data.

Membership in the group System:Administrators has to be carefully guarded. A system administrator can modify any access list in the system, and can therefore read or write any file. He can also change storage quotas and modify the ownership of files. For increased security, it would be relatively simple to modify Vice to grant System:Administrator privileges only to individuals who are logged in at one of a specific set of physically secure workstations, in addition to being authenticated.

To keep things in perspective, it should be noted that this section is deliberately negative in tone. Most of the scenarios described here are highly unlikely, and typically involve the violation of the assumptions discussed in Section 3. A site which adheres to those assumptions will find Andrew more secure than any existing distributed system of comparable functionality. Further, in spite of the attention it pays to security, Andrew remains a highly usable system.

12. Conclusion

As mentioned at the beginning of this paper, Andrew is an evolving system. A number of changes have been made since the date of the snapshot on which this paper is based. Many of these changes have been improvements to existing functionality. The protection database now stores its index as part of itself rather than in a separate file. This eliminates the occasional inconsistences between index and data that used to occur when propagating protection domain information. The remote procedure call mechanism described in Section 5.1 has been replaced by one that is more stringent in its use of memory. The primary reason for this change was the desire to run Venus on workstations with severly limited physical memory. The details of the authentication handshake are different from that described in Section 5.1, but the same effect is achieved. We are in the process of designing a faster encryption device for the IBM RT-PCs, as discussed in Section 9. Finally, the cell mechanism referred to in Section 10.3 is a significant change that will provide new functionality when complete.

We have long appreciated the need for users to be able to create and manipulate groups themselves, rather than submitting requests to the administrators of Andrew. A *Protection Server* that implements this functionality has been planned but not implemented yet. This server could be an extension of the existing authentication server and would allow us to merge security information that is current'y distributed over a number of different data structures: the "/etc/passwd" file on workstations, the password database on authentication servers, the protection domain database on file servers, and the files containing direct group membership information from which system personnel generate the protection domain database. This change would considerably simplify administration and operation of Andrew.

In conclusion, we believe that security issues will assume greater significance as distributed systems of increasing size and complexity are built. A substantial amount of theoretical research has already been done on security algorithms for distributed environments. Applying those principles to the design of real systems is complicated by the many levels of abstraction spanned, by the need for compatibility and by the many detailed aspects of the systems that are affected. Andrew is an attempt to seriously address these issues. It offers substantially greater security than existing distributed systems without significant loss of usability or performance.

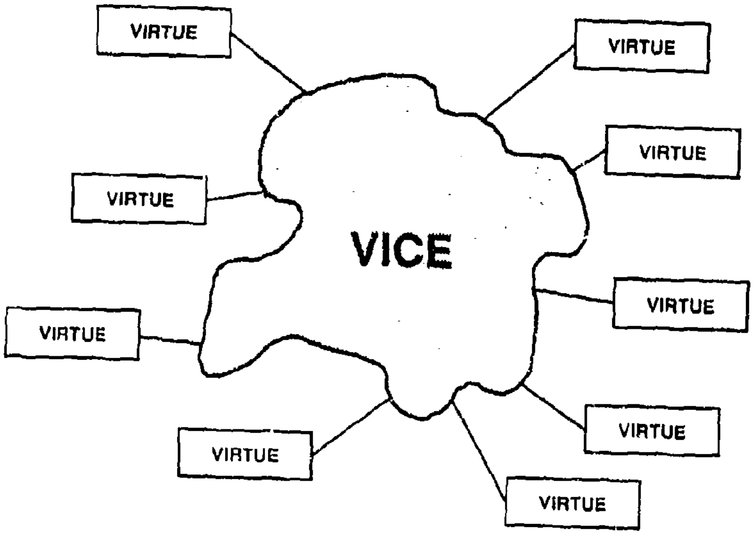
Credits

The security mechanisms described here were developed by the Andrew File System group, whose membership over time has included John Howard, Michael Kazar, David King, Sherri Menees, David Nichols, M. Satyanarayanan, Robert Sidebotham, Michael West, and Edward Zayas. Preliminary design work on security in Andrew was done by David Gifford, M. Satyanarayanan and Alfred Spector.

Many of the mechanisms described in this paper were implemented by M. Satyanarayanan. David Nichols and Michael Kazar contributed to the mechanisms in Virtue, while Michael West contributed to those in Vice. The security mechanisms that support PC Server were built by Larry Raper and Jonathan Rosenberg. Paul Crumley designed and implemented the encryption hardware for RT PCs.

James Kistler, James Morris, Jonathan Rosenberg, Robert Sansom, Ellen Siegel and Doug Tygar provided valuable comments on this paper.

I. Figures



The amoeba-like structure in the centre is a collection of insecure networks and secure servers that constitute Vice. Virtue is typically a workstation, but can also be a mainframe.

Figure 1: Vice and Virtue

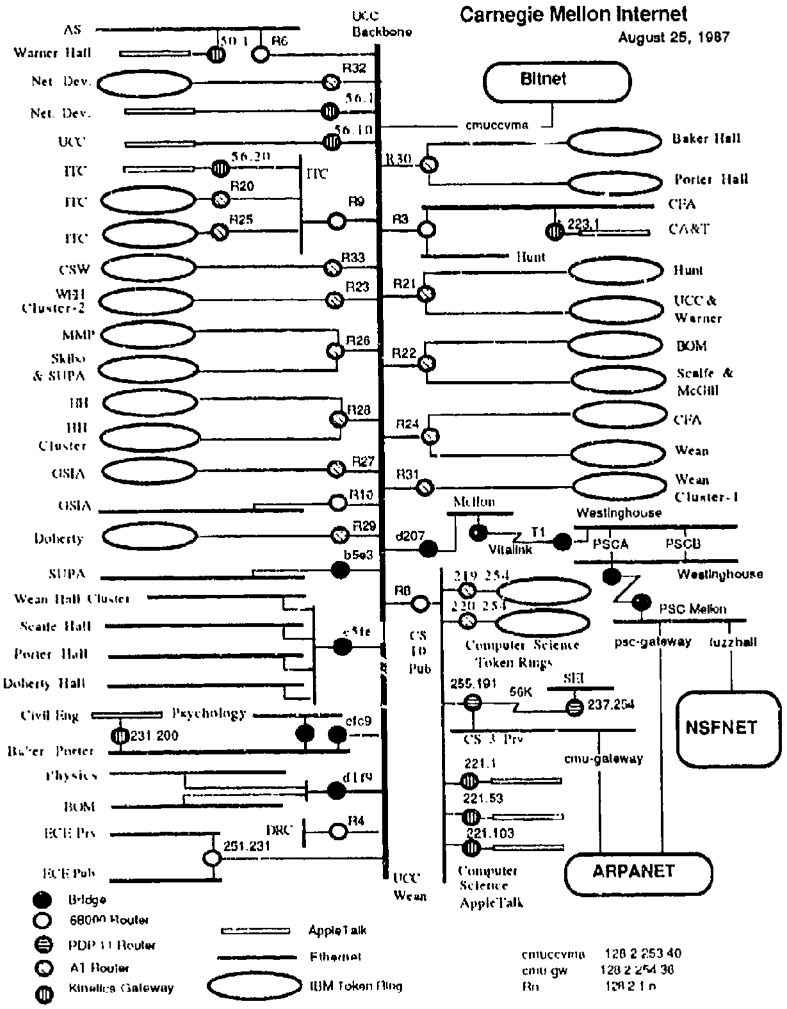
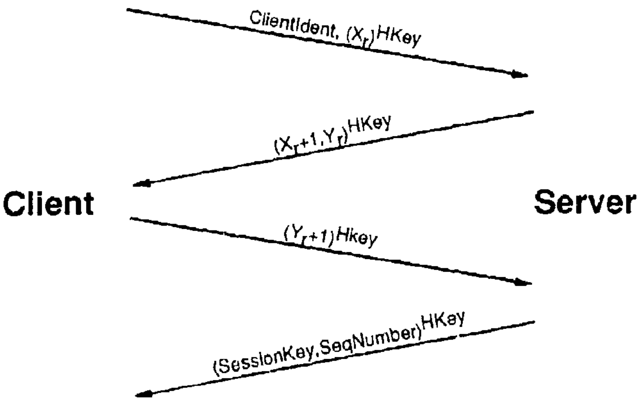


Figure 2: CMU Network



This figure shows the sequence of events in the BIND handshake. Each arrow represents a packet. The notation $(a)^b$ means that a is encrypted with key b. The last packet of the exchange conveys a randomly selected session key and a randomly selected initial sequence number to the client.

Figure 3: RPC BIND Authentication Handshake

```
277     545c5058595a5156        aad     Anthony Datri
265     575c585f5b5b575a        ab0q    Alfred Blumstein
672     13020a030619091f        ab2g    A. Leonard Brown
969     5f55595c595a555e        ab6q    Ahmadou Barry
131     565956595f5a545e        abrahams        Julia Abrahams
913     565857585d5a5459        ac2d    Arjun Bijoy Chatterjee
.........................................................................
.........................................................................
.................................................................. ........
283     13020a030619091f        zubrow  David Zubrow
18      0503135c5a6e676f        # By 18 at Wed Mar 19 13:09:23 1986
18      0503135c5a6e676f        # By 18 at Wed Mar 19 16:36:55 1986
1022    0b03170407096 76f       rk27    # By 18 at Wed Mar 19 16:37:37 1986
1023    1500081d190b156f        bd0p    # By 18 at Wed Mar 19 16:37:37 1986
1024    150018030c1d146f        cc37    # By 18 at Wed Mar 19 16:37:38 1986
1025    0b0315021b1d676f        cc38    # By 18 at Wed Mar 19 16:37:38 1986
1026    150f13020502146f        jo15    # By 18 at Wed Mar 19 16:37:38 1986
.........................................................................
.........................................................................
```

Each entry corresponds to information about one user. The first field is the Vice id of the user; the second is his encrypted password; the third field is the name of the user. Other fields are ignored by the authentication server. The first few lines correspond to entries that were present when the database was initialised. The entries at the bottom represent modifications. Each modification is tagged with the identity of the user making the change and the time the change was made.

Figure 4: Excerpt from Authentication Database

```
Date:   Mon Sep 29 09:51:13 1986

09:51:13        Server successfully started
11:03:49        Authentication failed for "fs0t" from 128.2.14.11
11:05:22        Authentication failed for "fs0t" from 128.2.14.11
11:05:54        Authentication failed for "an09" from 128.2.14.8
11:09:50        Authentication failed for "wh0s" from 128.2.14.4
11:10:25        Authentication failed for "wh0s" from 128.2.14.4
11:12:28        Authentication failed for "ao07" from 128.2.14.14
11:12:58        Authentication failed for "wh0s" from 128.2.14.4
11:20:43        Authentication failed for "ao07" from 128.2.14.14
12:00:26        Authentication failed for "ks2n" from 128.2.13.3
13:58:46        Authentication failed for "dans" from 128.2.243.3
15:22:26        Authentication failed for "dt1a" from 128.2.17.17
16:16:17        AuthChangePasswd() attempt on dh2u by ja8c denied
16:19:17        AuthChangePasswd() attempt on dh2u by ja8c denied
16:24:57        Authentication failed for "ak11" from 128.2.14.14
16:36:53        Authentication failed for "ja8c" from 128.2.17.4
20:46:03        Authentication failed for "jc55" from 128.2.14.11
21:47:13        Authentication failed for "cm2m" from 128.2.14.20
22:20:17        Authentication failed for "jr45" from 128.2.17.20
23:30:16        Authentication failed for "l116" from 128.2.14.20
23:30:56        Authentication failed for "l116" from 128.2.14.20
23:44:58        Authentication failed for "ef0u" from 128.2.11.62
23:53:59        Authentication failed for "gw0v" from 128.2.36.6

Date:   Tue Sep 30 09:51:50 1986

09:51:50        Authentication failed for "bk0u" from 128.2.14.12
09:56:23        Authentication failed for "bk0u" from 128.2.14.12
09:57:51        Authentication failed for "bk0u" from 128.2.14.12
10:16:48        Authentication failed for "je0x" from 128.2.14.3
11:22:16        Authentication failed for "ls24" from 128.2.14.10
11:32:02        Authentication failed for "mb3h" from 128.2.14.16
11:35:55        Authentication failed for "ls24" from 128.2.14.10
12:38:45        Authentication failed for "km35" from 128.2.14.9
```

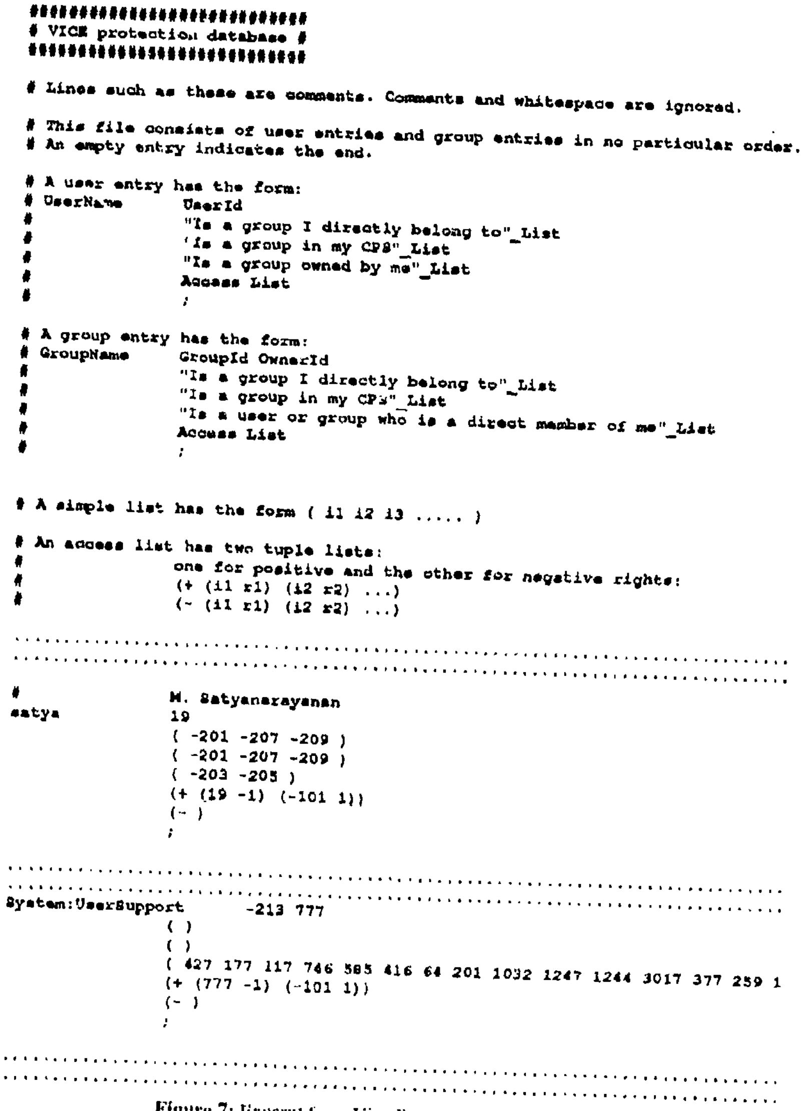
This figure shows typical entries from the authentication log. Most of the entries are invalid authentication attempts, probably caused by a user typing in his password incorrectly. Each entry identifies the user and the workstation from which the operation was attempted. Two of the entries are failed attempts by one user to change the password of another user.

Figure 5: Excerpt from Authentication Log

```
mozart> fs la /cmu/itc/satya/s11
Normal rights:
  System:ITC.FileSystemGroup rlidwk
  System:AnyUser rl
  satya rlidwka
Negative rights:
  System:ITC.UserInterfaceGroup rlidwka
mozart>
```

This figure shows how an access list is displayed in Andrew. The string ''mozart>'' is the prompt by the workstation. The command ''fs la'' lists the specified directory. Note the use of negative rights: a member of System:ITC.UserInterfaceGroup would have no rights on this directory, even though System:AnyUser has read and lookup rights.

Figure 6: Access List on a Vice Directory



```
##############################
# VICE protection database #
##############################

# Lines such as these are comments. Comments and whitespace are ignored.

# This file consists of user entries and group entries in no particular order.
# An empty entry indicates the end.

# A user entry has the form:
# UserName        UserId
#                 "Is a group I directly belong to"_List
#                 'Is a group in my CPS"_List
#                 "Is a group owned by me"_List
#                 Access List
#                 ;

# A group entry has the form:
# GroupName       GroupId OwnerId
#                 "Is a group I directly belong to"_List
#                 "Is a group in my CPS"_List
#                 "Is a user or group who is a direct member of me"_List
#                 Access List
#                 ;


# A simple list has the form ( i1 i2 i3 ..... )

# An access list has two tuple lists:
#                 one for positive and the other for negative rights:
#                 (+ (i1 r1) (i2 r2) ...)
#                 (- (i1 r1) (i2 r2) ...)

..............................................................................
..............................................................................

#                 M. Satyanarayanan
satya             19
                  ( -201 -207 -209 )
                  ( -201 -207 -209 )
                  ( -203 -205 )
                  (+ (19 -1) (-101 1))
                  (- )
                  ;

..............................................................................
..............................................................................
System:UserSupport      -213 777
                  ( )
                  ( )
                  ( 427 177 117 746 585 416 64 201 1032 1247 1244 3017 377 259 1
                  (+ (777 -1) (-101 1))
                  (- )
                  ;

..............................................................................
..............................................................................
```

Figure 7: Excerpt from Vice Protection Domain Database

```
mozart> ls -l /cmu/itc/satya
total 120
-rwx        1 satya        1385 Jul 24 14:23 3270.keys
d           4 satya        2048 Nov 17  1986 411
-rw-        1 satya        1979 Oct  3  1986 Buildfile
d           2 satya        8192 Sep 17  1986 Mailbox
d           2 satya       12288 Feb  8 10:39 Maillib
d           2 spooler     18432 Aug  2  1986 PrintDir
d           2 satya        2048 Mar  3  1986 Templates
-rw-        1 satya        5219 Jul 20 16:42 articles.frm
-rw-        1 satya        6620 Jul 20 16:36 articles.per
d           8 satya        2048 Jul  9 11:36 bench
d           2 satya        2048 Jan  5  1987 days
d          10 satya        2048 Jan  9  1986 dtape
d           2 satya        2048 Jun  9 17:25 infp
d           2 satya        2048 Nov 25  1986 lib
d           2 satya        2048 Oct 28  1986 mailib
d          18 satya        2048 Jun 12 11:26 misc
d           3 satya        4096 Jul  2 13:47 mrp
d           3 satya        2048 Sep 18  1986 net
d          25 vasilis      2048 Mar 11 16:50 oldrcs
l           1 satya          38 Jul  2  1986 personal -> /cmu/itc/satya/privat
d          19 satya        2048 Jun 29 09:32 pgms
-r--        1 satya         728 Apr  7 13:14 preferences
d           2 satya        2048 Mar 23 14:49 private
d           5 satya        4096 Jul  2 16:53 public
d           2 satya        2048 Apr 15 20:38 rp2p
d           5 satya        6144 Jul 14 14:48 rpc2
d           2 satya        2048 May 24 14:50 rpc2eg
d           3 satya        6144 Jul 23 16:15 s11
d           3 satya        2048 Jan 19  1987 scribelib
d           2 satya        2048 Jul 24 10:24 sec
d           3 satya        2048 Oct 31  1985 unc
```

This is an example of a directory listing in Andrew. For files, the status of the owner mode bits are shown as "r", "w" and "x". These bits are not shown for directories, since mode bits do not apply. Note the use of a symbolic link to obtain a protection status for the file named "personal" that is different from other files in this directory. The file is physically located in the directory "private" which has more restrictive access than the directory shown here.

Figure 8: Listing of a Vice Directory

References

1. Accetta, M.J., Robertson, G.G., Satyanarayanan, M. and Thompson, M. The Design of a Network-Based Central File System. Tech. Rept. CMU-CS-80-134, Department of Computer Science, Carnegie Mellon University, August, 1980.

2. *MOS Microprocessors and Peripherals.* Advanced Micro Devices, 1985.

3. Birrell, A.D., Levin, R., Needham, R.M. and Schroeder, M.D. "Grapevine: an exercise in distributed computing". *Communications of the ACM 25*, 4 (April 1982).

4. Birrell, A.D. "Secure Communication Using Remote Procedure Calls". *ACM Transactions on Computer Systems 3*, 1 (February 1985), 1-14.

5. Cheriton, D.R. and Zwaenepoel, W. The Distributed V Kernel and its Performance for Diskless Workstations. Proceedings of the 9th ACM Symposium on Operating System Principles, October, 1983.

6. Crumley, P. TRADMYBD: Data Encryption Adapter Technical Reference Manual and Programmers' Guide, Version 0.20. Tech. Rept. CMU-ITC-059, Information Technology Center, Carnegie Mellon University, December, 1986.

7. Dannenberg, R.B. *Resource Sharing in a Network of Personal Computers.* Ph.D. Th., Department of Computer Science, Carnegie Mellon University, 1982.

8. Diffie, W. and Hellman, M.E. "Privacy and Authentication: An Introduction to Cryptography". *Proceedings of the IEEE 67*, 3 (March 1979), 397-427.

9. Grampp, F.T. and Morris, R.H. "Unix Operating System Security". *Bell Laboratories Technical Journal 63*, 8 (October 1984), 1649-1672.

10. Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and West, M.J. Scale and Performance in a Distributed File System. Proceedings of the 11th ACM Symposium on Operating System Principles, November, 1987.

11. *IBM 4700 Personal Computer Financial Security Adapter: Guide to Operations.* IBM Corporation, 1985. Number 6024361.

12. *IBM 4700 Personal Computer Financial Security Adapter: Microcode Users Guide.* IBM Corporation, 1985. Number 6024362.

13. Jones, A.K. *Protection in Programmed Systems.* Ph.D. Th., Department of Computer Science, Carnegie Mellon University, 1973.

14. Jones, A.K. and Wulf, W.A. "Towards the Design of Secure Systems". *Software Practice and Experience 5* (1975), 321-336.

15. Lampson, B.W. "A Note on the Confinement Problem". *Communications of the ACM 16*, 10 (October 1973).

16. Lazowska, E.D., Zahorjan, J., Cheriton, D.R. and Zwaenepoel, W. "File Access Performance of Diskless Workstations". *ACM Transactions on Computer Systems 4*, 3 (August 1986).

17. Meyer, C.H. and Matyas, S.M. *Cryptography: A New Dimension in Computer Data Security.* John Wiley & Sons, 1982.

18. Morris, J. H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S. and Smith, F.D. "Andrew: A Distributed Personal Computing Environment". *Communications of the ACM 29*, 3 (March 1986)

19. Needham, R.M and Schroeder, M.D. "Using Encryption for Authentication in Large Networks of Computers". *Communications of the ACM 21*, 12 (December 1978).

20. Nichols, D.A. Using Idle Workstations in a Shared Computing Environment. Proceedings of the 11th ACM Symposium on Operating System Principles, November, 1987.

21. Rabin, M.O. and Tygar, J.D. An Integrated Toolkit for Operating System Security. Tech. Rept. TR-05-87, Aiken Computation Laboratory, Harvard University, May, 1987.

22. Saltzer, J.H. "Protection and the Control of Information Sharing in Multics". *Communications of the ACM 17*, 7 (July 1974), 388-402.

23. Satyanarayanan, M. Users, Groups and Access Lists: An Implementor's Guide. Tech. Rept. CMU-ITC-84-005, Information Technology Center, Carnegie Mellon University, August, 1984.

24. Satyanarayanan, M., Howard, J.H., Nichols, D.N., Sidebotham, R.N., Spector, A.Z. and West, M.J. The ITC Distributed File System: Principles and Design. Proceedings of the 10th ACM Symposium on Operating System Principles, December, 1985.

25. Satyanarayanan, M. RPC2 User Manual. Tech. Rept. CMU-ITC-84-038, Information Technology Center, Carnegie Mellon University, 1986 (revised).

26. Schell, R.R. Evaluating Security Properties of Computer Systems. Proceedings of the 1983 IEEE Symposium on Security and Privacy, April, 1983, pp. 89-95.

27. Sidebotham, R.N. Volumes: The Andrew File System Data Structuring Primitive. European Unix User Group Conference Proceedings, August, 1986. Also available as Technical Report CMU-ITC-053, Information Technology Center, Carnegie Mellon University.

28. Voydock, V.L. and Kent, S.T. "Security Mechanisms in High-Level Network Protocols". *Computing Surveys 15*, 2 (June 1983).

29. *Data Communication Products Handbook.* Western Digital Corporation, 1985.

30. Wulf, W.A., Levin, R. and Harbison, S.P.. *HYDRA/C.mmp: An Experimental Computer System.* McGraw-Hill Book Company, 1981.

ISSUES IN PROCESS MODELS AND INTEGRATED ENVIRONMENTS FOR TRUSTED SYSTEM DEVELOPMENT

Ann B. Marmor-Squires
TRW Federal Systems Group
2751 Prosperity Avenue
Fairfax, VA 22031
(703) 876-4100

Patricia A. Rougeau
TRW Federal Systems Group
2751 Prosperity Avenue
Fairfax, VA 22031
(703) 876-4100

ABSTRACT

This paper explores various research issues that need to be addressed in process models and automated support environments to better build trusted systems for Defense needs. A broader notion of trustworthiness of systems is discussed; the shortcomings of the typical security engineering life cycle and the nature of the software development life cycle as well as the related trust issues are explored. The paper then describes several contributing technologies that, when merged, offer a next generation integrated approach to trusted system development to better support Defense needs. The paper concludes with a set of recommendations for research directions to be pursued.

BACKGROUND

A significant long-range problem that needs to be addressed is the critical need for systems to meet major current and future Defense system requirements for security, integrity and high reliability. The current generation software development paradigms for producing systems is woefully inadequate for developing and demonstrating mission critical systems in which we have a high degree of trust in their correct and secure operation. The existing formal security modeling and specification and verification technologies that have been developed and applied primarily in the trusted systems arena are labor-intensive and inadequately supported with cost-effective analysis techniques and tools. Various computer security projects in which formal methods have been applied in the past have produced systems with inadequate performance to meet Defense system needs and with system implementations that bear little resemblance to the abstract proofs of security or "trustworthiness" about them. To date, these formal methods have not been coupled and effectively integrated with other testing, analysis and configuration management techniques into a sound engineering development paradigm for large complex systems meeting future DoD requirements. The trusted systems that have been produced to meet the higher levels of the Trusted Computer System Evaluation Criteria (TCSEC) are not widely applicable and, due to their limitations, are not replacing less secure and less trusted systems that are in widespread use in the Defense community.

This paper explores some of the issues to be addressed in development paradigms and automated support environments to meet the challenge of current and future Defense system trust requirements.

THE PROBLEM

Although good progress has been made in the last several years in producing secure components, extending this progress to building secure multi-component operational systems has been slow, expensive and complex. Some major aspects of the problem include:

- Expansion or variation of the definition of security beyond confidentiality as it applies to large systems that are mission critical.
- The nature of the security engineering life cycle, which often is separate from and parallel to the system development life cycle.
- The nature of the software development life cycle, which no longer fits a "waterfall" paradigm and for which new process models and supporting development environments must be explored.
- The tendency of contributing technologies like formal verification and Ada tool development to be isolated and not integrated with mainstream system development environments.

These aspects will be discussed in the following sections.

VARIATIONS ON THE DEFINITION OF SECURITY

Traditionally, discussions on security have concentrated on a definition of security to mean confidentiality, that is, the protection against unauthorized disclosure or modification of data. Confidentiality was considered more important than other aspects of the system requiring more expertise and attention. The activity of separating the system into Trusted Computing Base (TCB) and non-TCB, or trusted and untrusted components, was formulated so that more attention (verification, testing, design scrutiny) could be placed on the trusted part. In this regard, such things as denial of service were not strictly security concerns.

Although denial of service is not strictly security related per the TCSEC, for programs which have a high degree of mission criticality, the line between security, denial of service and mission criticality becomes blurred. In fact, SDI is one of these mission critical programs. In the SDI security approach, confidentiality is only one leg of three aspects simultaneously addressed: confidentiality, assured service and integrity. In this case, confidentiality is concerned with identification and authentication, least privilege, object reuse, audit, data protection, key management, TEMPEST, personnel security, OPSEC, traffic flow security, trusted facility management and trusted distribution. Integrity is concerned with error detection and correction, data authentication, source/recipient authentication, consistency, concurrency controls, and N-man control. Assured service is concerned with both assured communication service and assured computing service. Assured communication service includes the link level, the network level, and the application level. Assured computing service includes hardware and software assurance, trusted recovery, deadlock and algorithm design, as well as component fault tolerance and secure failure. This is an excellent example of where a process model and development environment are needed in which several aspects are equally important in the competition for design scrutiny and dollars.

THE SECURITY ENGINEERING LIFE CYCLE

Figure 1 shows the Security Engineering Life Cycle as a parallel activity to the system development process. Throughout the life cycle of the project, depending on the project size, a security engineer or a group of security engineers performs the analysis and produces documentation to support the security engineering of the system. This begins with the Security Requirements Analysis phase, which leads to the formulation of security policy and the security model.

Next is the first of series of risk and vulnerability assessments to be performed. The second one should be performed between preliminary design and critical design, as important design tradeoffs are being made. At this point, covert channel analysis can also be performed, so that any identified covert channels can be addressed in the critical design. Another risk assessment/vulnerability analysis/covert channel analysis should be done when the system is being realized in code, to ensure that no new vulnerabilities or covert channels have been introduced.

After the security model is done, the Security Architecture is produced, which includes all aspects relevant to the security posture at that point in the design. It should include assumptions about the system operation, assertions about the security enforcement techniques, an initial partitioning of the system into trusted and untrusted segments, expected operational scenarios, and an initial security design of the Trusted Computing Base. Next comes the Descriptive Top Level Specification and the Formal Top Level Specification as specified by the TCSEC. Finally comes Security Testing Activities and Documentation, and then Certification and Accreditation activities.

There are several important aspects that are inadequate in this life cycle to meet current and future Defense needs:

- It is too expensive. It costs to maintain a staff of security engineers and to produce extensive documentation which must be carefully configuration managed, and maintained.
- The parallel nature of the process focuses exclusively on security rather than on trust and assurance of the system as a whole. This lack of integration of security into the mainstream life cycle undermines broader trust and integrity concerns of a system with a dedicated mission.
- The separation of security engineers from normal design and development engineering often sets project groups against one another as important design decisions are made and tough tradeoffs must be made on requirements allocation, performance, etc.
- It is labor intensive because there is little automation which supports this parallel life cycle. Tool use is sporadic, at the discretion of individual subsystem managers, and is not well integrated with other project software development tools.
- It is fragile. The life cycle does not take advantage of the true nature of building systems today, which involves feedback, iteration and interaction as shown in Figure 2. The interactive nature involves prototypes, analyzing the effects of one requirement or design alternative on the rest of the system, modeling and sensitivity analysis. The parallel life cycle is much more rigid. If any assumptions are changed along the way, the model and design can be invalidated and a "startover" can result.

TRUST ISSUES IN SOFTWARE DEVELOPMENT LIFE CYCLE

As a consequence of the problems discussed above, the current typical software development life cycle is inadequate in meeting the needs of developing complex systems having a high degree of trust and reliability requirements. Current and future system requirements raise trust-related issues for both the process model and the software development environment that are not being addressed today. These include the need for:

- Effective development paradigm for trusted systems.
- Access control, trustworthiness and integrity of the software development environment.
- Support for the trusted system evaluation process.

Most software systems today are developed with a "waterfall" method of software development which was appropriate when it was introduced but no longer meets the needs of developing complex systems. The waterfall paradigm treats the software development cycle as a series of sequential steps, each of which is completed before the next step is begun. First the requirements are completely specified, then the preliminary design is completely done, and so on. The waterfall model does not adequately address concerns of developing program families and reusable components, nor does it adequately organize software to accommodate change. It assumes a relatively uniform progression of elaboration steps and does

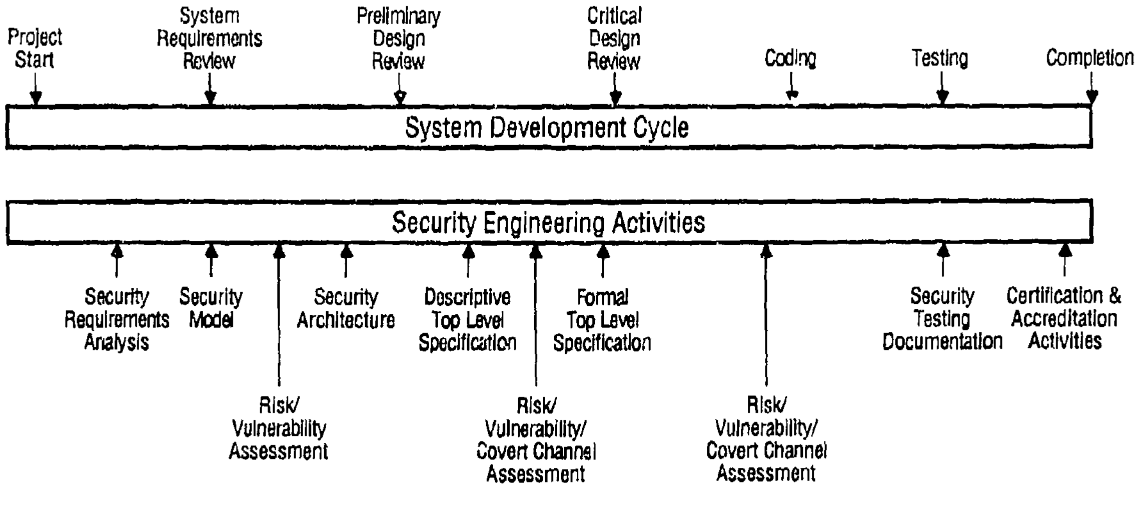

Project Start
System Requirements Review
Preliminary Design Review
Critical Design Review
Coding
Testing
Completion
System Development Cycle
Security Engineering Activities
Security Requirements Analysis
Security Model
Security Architecture
Descriptive Top Level Specification
Formal Top Level Specification
Security Testing Documentation
Certification & Accreditation Activities
Risk/ Vulnerability Assessment
Risk/ Vulnerability/ Covert Channel Assessment
Risk/ Vulnerability/ Covert Channel Assessment

Figure 1. Security Engineering Activities Parallel System Development Cycle

not accommodate the realistic evolutionary development needed to develop complex systems and which is made possible by current rapid prototyping capabilities. The waterfall model also does not address the possible future modes of software development such as those incorporating program transformation capabilities or knowledge-based software capabilities.

The current development paradigm used for trusted systems is security engineering as a parallel activity to the waterfall development process as illustrated in Figure 1 and described in the previous section. This is an inadequate and ineffective approach for building complex systems having requirements for a high degree of reliability, integrity and trustworthiness. What is needed is a next generation development paradigm for trusted systems that effectively couples and integrates a new process model with formal methods and security engineering methodologies and puts these critical development components in the context of sound software engineering practices and automated support environment necessary for developing large complex Defense systems. This new process model should be a risk-driven process model; that is, it should be a process model which can take into account the particular vulnerabilities and risks and environment of the target system at each stage of development. This enables the development process to react to feedback and changes with appropriate elaboration steps to reduce the identified risks, for example, additional system prototyping or formal specification.

A second trust-related issue to be addressed is the impact on the software engineering environment to be used in support of the trusted target system development. Current software engineering environments do not adequately address these needs. The Trusted Computer System Evaluation Criteria were developed for operating systems and do not adequately address the trustworthiness needs of the software engineering environment. Issues identified are the appropriate level of the Evaluation Criteria as a goal for software engineering environments to be used for trusted system development, architectural impacts on the environment and need for tool and project database integrity in the environment. These issues are only now starting to be considered in efforts to standardize the interface specifications for software engineering environments for security applications. More detailed analysis of these issues is needed before standardization efforts get cast in concrete.

Currently, there is no effective integration of th tools to support formal methods in trusted system development with other development and analysis tools needed to support the software engineering process. There is no support in environments for the appropriate notations, descriptions and representations needed for trusted system development. In place of a consistent design/development notation throughout the development, tools are applied to different notations with gaps between the notations requiring manual translation. For example, at the security model and high level specification stages, the notation of a formal specification language, such as Gypsy or Special might be used. Later, in the preliminary design stage, a pseudo code language such as PSL might be employed. Since the two notations are different, automated traceability between the two is very difficult. The implementation is often carried out in yet another notation, for example,

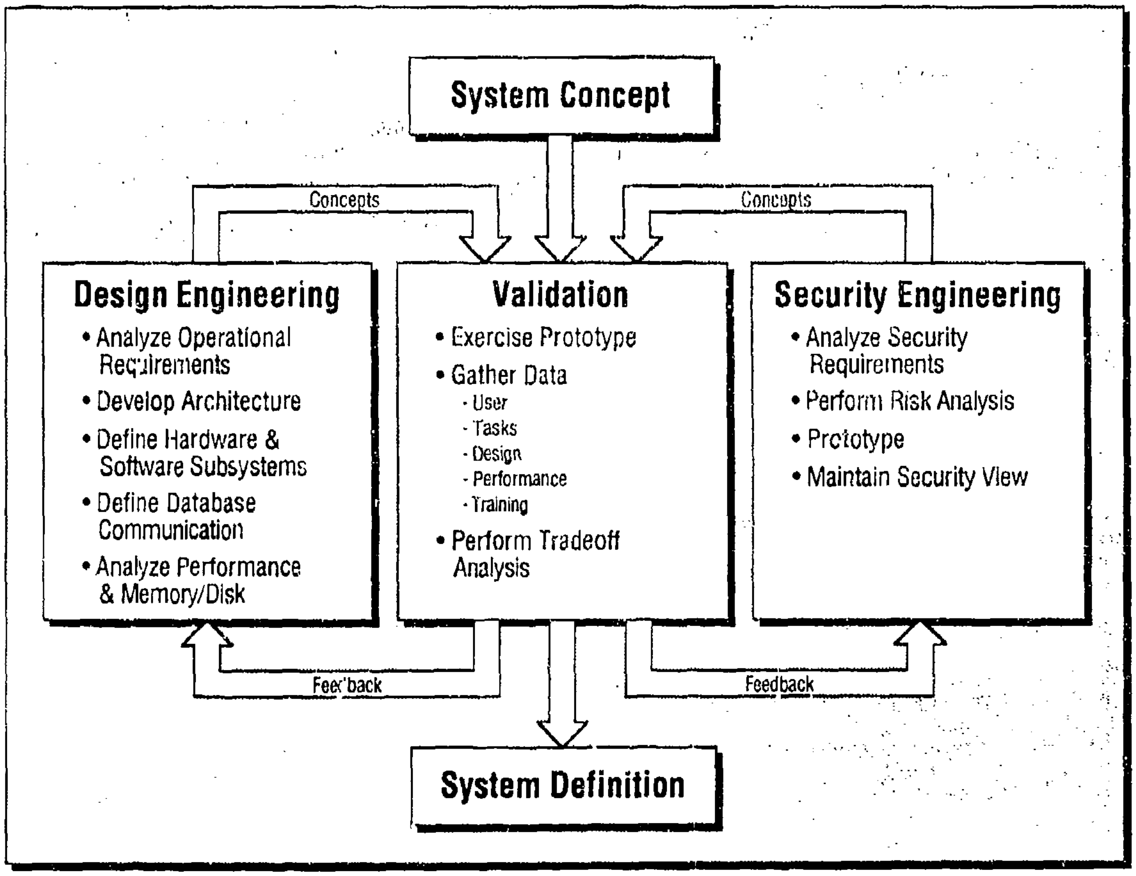


Figure 2. Security Engineering/Design Engineering Interaction

in C, Pascal or Ada. Added to that are the variety of rotation which are employed for testing and configuration management.

Without a consistent notation, traceability throughout the development is very difficult, if not impossible, for a complex system. Formal specification and verification technology is not integrated with and not on a par today with the other tools and techniques used in system development, is not in widespread use today and is not used in a cost-effective manner for critical system components.

Another trust issue of software engineering environments for trusted system development is the need for a trustworthy and controlled environment for the development life cycle. This includes concern for trustworthiness of the tools in the environment and the integrity of the project database, access control of the developers to portions of the developing target system, configuration management and control of the environment as well as access control to the project database and the tools. These concerns are currently not being addressed.

Current software engineering environments also do not provide support for the trusted system evaluation process, i.e., for evaluation of the target mission-critical trusted system against the appropriate Evaluation Criteria. Needed support would include evaluation tools in a controlled environment for the initial evaluation as well as any required periodic re-evaluation of a system to retain its evaluation level. Such automated tools might include tools for documentation tracing, covert channel analysis, and configuration management. Some of the analysis tools useful for evaluation are part of a normal suite of development tools within an environment while others are for evaluation purposes only. There must be access control to those tools as well as a high degree of trust in their proper use.

The various trust-related issues in development paradigm, software development environment and support for the evaluation process need to be addressed in order to meet the needs of future complex systems requiring a high degree of trustworthiness of operation and integrity of the system.

CONTRIBUTING TECHNOLOGIES FOR TRUSTED SYSTEM DEVELOPMENT

The broad computer security and software engineering communities have a challenge and an opportunity to effectively address the issues raised in the previous sections for developing trusted systems by integrating several fundamental contributing technologies to achieve a better result than any of the components alone could produce. These technologies include a new risk-driven process model coupled with formal methods and security engineering methodologies and integrated automated support environments incorporating tools needed for developing large complex Defense systems. In order to have widereaching impact, we propose that this technology integration be done in the context of Ada trusted system development for DoD.

While each of the contributing technologies exist in some fashion today, most are state-of-the-art rather than state-of-the-practice. Nor are they used in an integrated manner in a unifying framework for trusted system development. In addition, the communities of interest (computer security, formal methods/verification, software engineering, software environments) have not communicated very much or very effectively with one another. Each community of interest is narrowly focused on only one aspect of what is necessary for building next generation DoD systems. Positive impact on future DoD trusted system requirements will be based on coupling these efforts.

The computer security community has focused on security aspects in the narrow sense (as defined in TCSEC) rather than on more broadly defined trust. The needs of computer security have driven the development of security policy models, security engineering methods and formal specification and verification methods, languages and tools. This driving force has helped focus the efforts to solve problems in computer security but have not adequately addressed trust issues such as those to be found in SDI or other mission critical systems. It is not clear whether the techniques developed will scale up effectively or whether significant enhancement and redevelopment will be necessary. Current SDI-sponsored security architecture studies are beginning to address some of the broader trust issues.

The formal methods/verification community over the last decade has largely been driven by computer security community needs. The positive impact has been the development of a number of prototype and operational systems with formal specifications of their security related properties and mechanical proofs of consistency of specifications with respect to the security properties. The computer security community has also funded the development of several verification environments that are largely still experimental. This current generation of verification technology cannot in its current form be applied to large scale systems nor is it considered state-of-the-practice by software developers. This is in large part because the verification tools and techniques have not been integrated into the software development process. Although it would seem that this integration should occur naturally, it has not yet occurred. The U.S. verification community has also, until recently, ignored Ada verification issues.

The software engineering community has focused on the software development life cycle (methods and tools) and has largely ignored the needs of software development for trusted systems. Research efforts have addressed new process models and alternative development paradigms such as those based on risk analysis, automatic programming, rapid prototyping, program transformations and knowledge-based software assistant capabilities. Important software engineering issues such as accommodating families of systems, prototyping of key system capabilities, reuse of previous software and scaling up to very large systems are equally important for trusted systems yet they have not been addressed in that context. Issues relating to Ada (e.g. Ada process model, software reuse in Ada, ...) are also a current focus of attention. Integration of the computer security and formal methods technology in the context of a sound software engineering paradigm is a critical research need to address the problem of trusted system development.

The software environment community has focused on the development of integrated software development environments but has not involved the integration of formal specification and verification tools into such environments. Since environments are to provide automated support to the development and analysis of complex Defense systems, if the target system has trust requirements, the ability to analyze a system's behavior through formal and informal means is critical. This means that an environment needs to provide a spectrum of analysis tools which include the appropriate, integrated role for verification tools. This integration has not been an area of concentration for the software environment community. With the advent of Ada for DoD mission-critical systems, we have an opportunity to focus attention on a next generation Ada environment that would support trusted system development.

RECOMMENDATIONS

Since the trust issues raised here have the potential for having unwieldy or complex solutions, the challenge for the research community is to be able to aim for long term goals while simultaneously seeking near-term and intermediate results. Results need to be realized in worked examples which drive and focus the research as well as paper studies, since theoıy is often changed in the execution of the ideas.

Progress toward goals could take place in two environments. The first is to have research in a laboratory environment where the integration of the technologies can be attempted in worked prototype examples which would be scaled up later in operational systems if they prove fruitful. The second is to attempt limited integration in actual operational system development. A valuable initial step would be to hold a workshop or summer study on future trusted systems development. This would bring together key contributing technologists from the computer security, formal verification, software engineering and software environments communities and focus on integrating appropriate aspects of their work and identifying gaps in the research to be funded.

Research areas and directions for studies and prototype developments that we have identified include the following:

- Establishment of a framework for modeling and development of trusted systems including development/interpretation of evaluation criteria for software environments for trusted systems.
- Investigation of trust issues (security, integrity, assurance) with respect to Ada trusted system development including impact on development paradigm and support environment architecture and tools.
- Research to develop a next generation process model for trusted systems that integrates the appropriate contributing technologies.
- Prototype development of "trustworthy" automated support environment for the next generation process model for trusted systems.

Demonstration of applicability of the new process model and environment on a set of high impact worked examples.

Several of these research directions are long-range projects spanning five years or more but near-term and interim results can be achieved and be made known in a broad community. We can have a positive impact on DoD trusted system requirements by leveraging and combining several contributing technologies today.

REFERENCES

1. B.W. Boehm. *A Spiral Model of Software Development and Enhancement*, TRW Defense Systems Group, Redondo Beach, California, 1984.
2. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December, 1985.
3. A.B. Marmor-Squires. *Multilevel Security STARS SEE Architecture Study*, IDA Paper P-1834, Institute for Defense Analyses, Alexandria, Virginia, June, 1985.
4. W.T. Mayfield and S.R. Welke. *Proceedings of the Second IDA Workshop on Formal Specification and Verification of Ada*, 23-25 July 1985, IDA Memorandum Report M-135, Institute for Defense Analyses, Alexandria, Virginia, November 1985.
5. C.G. Roby. *Proceedings of the first IDA Workshop on Formal Specification and Verification of Ada*, 18-20 March 1985, IDA Memorandum Report M-146, Institute for Defense Analyses, Alexandria, Virginia, December 1985.
6. P.A. Rougeau. *Integrating Security into a Total Systems Architecture*, Proceedings of AIAA Third Aerospace Computer Security Conference: Applying Technology to Systems, Orlando, Florida, December, 1987.

A Technique for Removing an Important Class of Trojan Horses from High Order Languages

John McDermott

Center for Secure Information Technology
Naval Research Laboratory, Code 5540
4555 Overlook Ave.
Washington, DC 20375
(202) 767-3770
mcdermott@nrl-css.arpa

1. Introduction

In his 1984 Turing Award Lecture[1], Ken Thompson described a sophisticated Trojan horse attack on a compiler, one that is undetectable by any search of the compiler source code. The object of the compiler Trojan horse is to modify the semantics of the high order language in a way that breaks the security of a trusted system generated by the compiler.

The Trojan horse Thompson described is a form of virus, inasmuch as it is self-reproducing, but it has other characteristics that differentiate it from viruses that exploit the implementation details of a computer system. First of all, the self-reproduction is symbiotic, that is, the Trojan horse depends on the source text of the legitimate compiler for its continued existence. The virus only reproduces itself in the output stream of the compiler, when the compiler is compiling itself (thus destroying the original virus). A second difference is the relative portability of the virus to different systems. The compiler Trojan horse Thompson described is less dependent on the design details of a particular machine because it exploits the portability of high order languages. A final difference is the location of the virus in the executable file. The compiler Trojan horse is inserted in a place that is hard to search, that is in mid-file. While this is possible for any form of virus, it is more difficult for viruses that do not have the compiler's internal functions at their disposal.

In his lecture, Thompson asserted that "no amount of source-level verification or scrutiny will protect you from using untrusted code." When no other means are used to provide assurance, this is true. However, this paper describes a technique that will remove such Trojan horses when used in conjunction with high-order language source code analysis.

This paper does not address what is referred to here as a *general virus*, that is, one that infects programs by direct modification of their images on disk (or other secondary storage). The general virus is a much larger problem; for example, detection of arbitrary general viruses is undecidable[2].

Section 2 of this paper explains why this class of Trojan horse and virus is important for trusted systems. Section 3 describes the basic compiler Trojan horse, and Section 4 describes the defense in detail. Section 5 gives a brief sketch of some possible measures and countermeasures. The paper concludes with some possible applications of this technique to building trusted systems.

2. The Importance of Compiler Trojan Horse Viruses

Compiler Trojan horses that are based on a symbiotic relationship between the source code and the binary code of the same compiler are important for several reasons besides the difficulty of detecting such a Trojan horse. Such a Trojan horse can compromise the security of many systems without propagating itself to those systems, it can compromise several classes of systems without redesign, and it is difficult to locate in an executable file.

The compiler Trojan horse can introduce unauthorized bypasses of security mechanisms into trusted systems, yet never exist on those systems. If such a Trojan horse is successfully installed on a development system, it can infect every system ever developed there. It can do this because the compiler will generate security flaws in every trusted system it generates code for, whether it is compiling for its own host or some other system.

Unlike the general virus, the compiler Trojan horse virus does not depend on low level machine or operating system dependent details of the implementation it infects. One of the subtleties of Thompson's design is the use of the compiler's code generator to install the binary version of the viruses. If the Trojan horse virus is created in the same intermediate language that is passed to the code generator, it will be appropriately translated, linked, and loaded for whatever machine the compiler is targeted for. This applies not only to the self-reproducing or viral component, it also applies to the other malicious code that may be installed by the Trojan horse.

The compiler Trojan horse virus can easily install itself in the middle of the disk image of program, because the compiler inserts it as it generates code. A general virus can be inserted into the middle of the disk image of a program too, but the problem is more difficult for the general virus. In the latter case, the virus must be designed to obtain privileges that permit it to modify the disk images of programs. It must also be designed to perform link editing correctly, a non-trivial task when the program disk image is in link editor output format.

An assertion that general viruses cannot insert themselves in mid-program would be untrue. Nevertheless, it is significantly more difficult to create such a general virus and each copy of the general virus must contain all the code necessary for correct mid-program insertion.

3. The Trojan Horse

The compiler Trojan horse is introduced into the system by a procedure that resembles a compiler compiling itself. The authorized Trojan-horse-free version of the compiler is used to compile a Trojan horse version of itself that always reproduces the Trojan horse in the compiler. For this to occur, there must be at least one read access to the legitimate compiler source code, one or more execute accesses to the compiler (even viruses must be debugged), and at least one write access to the legitimate compiler binary. None of these accesses must necessarily occur on the same system.

The compiler Trojan horse is created as follows:

1) a source code version of the compiler is written that includes two Trojan horse capabilities, a security mechanism bypass and a self-reproducing feature,

2) the legitimate Trojan horse free version of the compiler is used to produce an object code version of the Trojan horse compiler, and

3) the Trojan horse object code version is installed over the legitimate object code version of the compiler.

At the end of this process the source code of the compiler is free of Trojan horses but the executable file has two Trojan horse features. The first Trojan horse feature adds unauthorized security mechanism bypasses whenever it compiles the appropriate source code. The second Trojan horse feature reinserts the object code of the Trojan horse into the compiler whenever it compiles the legitimate compiler source code. The Trojan horse object code will remain in the system undetected by any analysis of the high order language source code of the compiler. Additionally, targeted security mechanisms generated by this compiler will have unauthorized and undetected bypasses installed.

This attack destroys the semantics of high order languages and undermines tr st in assurance measures that depend on them. This threat is a serious problem because many valuable assurance techniques depend on high order language semantics. The availability of a technique for ensuring that the semantics of a high order language are free of such problems is essential for trust in high orde language assurance techniques.

4. The Defensive Technique

The defense against this attack is based on the same concept of a compiler compiling itself. The defense exploits the symbiotic relationship between the source text of the legitimate compiler and the self-reproducing feature of the Trojan horse object code[3].

The Trojan horse reproduces itself whenever it compiles the compiler. To do this, it must first recognize some key portion(s) of the source text of the legitimate compiler. If the Trojan horse compiler compiles a compiler that it cannot identify as such, it will not reproduce in that program's object code. If the suspect compiler is fed a disguised version of itself, or a simple temporary compiler of a different design, it will not reproduce any such Trojan horses.

If the legitimate compiler can be hidden from its Trojan horse symbiont, so to speak, the Trojan horse will be erased by itself. Since it will not reproduce and the object code generated by the disguised compiler will be used in all future compilations, the threat is removed.

Hiding the legitimate compiler is not necessarily simple. First, recognition of an arbitrary compiler from an analysis (by the Trojan horse) of the function of the program can be ruled out as beyond the state of the art. However, the Trojan horse has many clues at its disposal within the superficial features of the legitimate source text. If, for example, every identifier was scrambled, the Trojan horse still might detect a pattern in the keywords of the compiler source. If these were scrambled too (this is not difficult on many compilers), the Trojan horse could analyze comments, constant strings (such as "C Compiler, Version 3.7", requests for library units, or error messages (e.g., the C compiler is the only one with 198 error messages).

With careful analysis and design (by the defenders), a reduced function compiler could be created, one that even incorporated large amounts of code that had nothing to do with compilation. This temporary compiler would be compiled first, and then its Trojan horse free object code version used to compile the undisguised legitimate compiler. An integrated editor, compiler, and link editor that can scramble and unscramble program text as appropriate would be a very useful application of this technique.

5. Measures and Countermeasures

There are compiler Trojan horse measures beyond the text string searches mentioned above. Three measures that are of practical significance are, in increasing order of effectiveness for the Trojan horse:

1) attempt to recognize the compiler by its function.

2) make the Trojan horse part of the compiler's legitimate features.

3) extend the symbiosis.

The following discussion explains each measure and proposes a countermeasure to meet it.

5.1. Recognizing the Function of the Legitimate Compiler

While the problem of recognizing the function of an arbitrary program is not decidable, it is possible for the Trojan horse to recognize the function or a key part of the function of a restricted class of compilers. By making simplifying assumptions, the Trojan horse can still identify the legitimate compiler. An example of such an assumption is to assume the compiler will be generated by compiler construction tools. The output of a lexical analyzer generator or a compiler compiler contains regular patterns of instructions and data that could be associated with the tool and its target program.

The countermeasure to this approach is to construct a very simple but extremely modular compiler by hand. This compiler could employ techniques not used in compiler construction tools to complicate the detection problem. If this compiler is also divided into very small modules in separately compiled source files, the Trojan horse will never have enough source text to identify the compiler function.

5.2. Installing the Trojan Horse in a Compiler Feature

One measure mentioned in[3] is to incorporate part of the compiler function into the Trojan horse. When the Trojan horse is removed by recompilation, no compiler on the system will work. A good concrete example of this is the run time support for the language. If the Trojan horse is placed in the object code of the run time support mechanism of the language in such a way that it performs some of the run time support, it cannot be removed without breaking the language.

Any practical plan for disguising a compiler must take a system view of the entire language. Many languages assume the existence of support mechanisms that are outside the definition of the language or are not part of the compiler program. These support mechanisms can be suitable targets for the compiler Trojan horse, so this defensive technique should be applied to them also.

5.3. Extending the Symbiosis of the Trojan Horse

The final and most effective measure for the Trojan horse is to extend the symbiosis. As originally defined, the Trojan horse is a symbiotic system where the legitimate compiler source text is necessary for the continued existence of the object code virus. If a general virus is added as an additional symbiotic feature, the total Trojan horse system becomes a general virus with limited complier Trojan horse capabilities, since the security mechanism bypass feature is still portable.

In the original case, if the object code of the compiler was modified without including the symbiotic self-reproducing feature, the compiler Trojan horse would be destroyed when the compiler was recompiled. However, in the original case, the compiler Trojan horse with the self-reproducing feature is protected by its relationship to the compiler's legitimate source; whenever the Trojan horse detects the compiler in its own input, it reproduces.

In the case of the third measure, extending the symbiosis, the compiler Trojan horse system is expanded to include a general virus that resides at the start of some other program. That general virus component can then protect the original compiler Trojan horse. To do this, the general virus modifies the source of the compiler to include the compiler Trojan horse, recompiles it, and deletes the modified source file. All of this can be done in background mode, in such a way that the original compiler source is not modified, only a copy of it. The Trojan horse in the compiler can check selected executable files to see if its symbiont general virus is there, and if not, restore it. Thus both virus and compiler Trojan horse would exist in mutually reinforcing positions on an infected system.

The third measure results in a general virus, that is, the resulting Trojan horse is no longer strictly in the class addressed by this technique. The offending program gives up portability and relative simplicity for an increased likelihood of survival on a single hardware base. It increases its chances of survival by establishing a symbiont that will not be overwritten when the compiler is recompiled.

Nevertheless, the defensive technique of disguising the legitimate compiler will present the symbiotic general virus with the same problem. The general virus must also identify the source text by the same means as its partner within the compiler object code. If the compiler has been successfully disguised, it will be equally protected from both the original Trojan horse and any symbiotic extensions that must identify the compiler. However, the disguise techniques must now be in effect at all times, and furthermore, all source files must be disguised in addition to the compiler.

6. Conclusions

In general, this technique increases the complexity of the problem facing a compiler Trojan horse. A more complex compiler Trojan horse is more likely to have errors, it will take longer to design, develop, and test, and it will probably be easier to detect because it is larger. Use of this technique makes the outcome of such an attack depend more on personal skill and less on system features.

The technique can be applied with varying amounts of resources and achieve reasonably proportionate assurance results. A low resource approach would merely scramble identifiers and constants before recompiling the compiler, while a high resource approach would be a programming system that completely hid all of its data. In the latter case, vendors could have such systems independently verified.

This technique is appropriate for very high assurance systems (e.g. beyond A1) where every available defensive measure is desired. A high assurance version of this technique should be supported by a trusted development environment and additional measures to intended to cope with general viruses[4,5].

The technique is also appropriate for some systems of moderate assurance, depending on their design. Moderate assurance systems may contain components that are very vulnerable to Trojan horse attacks. Consideration should be given to using a low resource version of this technique to provide some assurance that the development system has not introduced a Trojan horse.

A final practical issue is the examination of the high order language source for the compiler. The technique described in this paper assumes that there is no Trojan horse in the source text of the compiler. This assumption may be difficult to meet for two reasons. First, the source code for a compiler is usually very closely held by the compiler vendor. This raises the question of certification and of publicly available source text for temporary filter compilers; both are beyond the scope of this paper. Second, contrary to the statement in [1], the task of insuring that the compiler source code is trustworthy is non-trivial. Examination of compiler source code is probably the first use that should be made of practical automated high order language analysis techniques. Since the compiler source code is the most sensitive source code associated with the creation of a trusted system, it should be the most profitable place to look for high order language security flaws.

References

1. Ken Thompson, "Reflections on Trusting Trust," *CACM*, vol. 27, no. 8, pp. 761-763, August 1984.
2. Fred Cohen, "Computer Viruses: Theory and Experiments," *Proc. 7th National Computer Security Conference*, pp. 240-263, Sep 1984.
3. Steve Draper, "Trojan Horses and Trusty Hackers," *CACM*, vol. 27, no. 11, p. 1085, November 1984.
4. Fred Cohen, "A Cryptographic Checksum for Integrity Protection," *Computers & Security*, vol. 6, no. 6, pp. 505-510, North-Holland, Dec. 1987.
5. Catherine L. Young, "Taxonomy of Computer Virus Defense Mechanisms," *Proc. 10th National Computer Security Conference*, pp. 220-225, Sep 1987.

COMPUTER SECURITY CONSIDERATIONS
FOR
A TACTICAL ARMY SYSTEM

William Neugent
The MITRE Corporation
HQ USAREUR, ODCSOPS, Ops. Div.
APO New York 09403

Abstract: Operational experience with a tactical Army system illustrates the need for computer security safeguards. Tactical environments can require group userids and distributed control over security management and operation. Careful security management planning is critical. Overrun protection might be needed. Penetrators can exist within military networks.

INTRODUCTION*

The new generation of Department of Defense (DoD) computer security (COMPUSEC) policy does not explicitly address tactical systems[1,2,3]. This is sometimes understood as implying that the policies do not apply to tactical systems or that they need to be "interpreted" for tactical systems, in the same way that the Trusted Network Interpretation interprets the Orange Book for application to networks [2,3]. Furthermore, in the field, there is often a belief that COMPUSEC safeguards are unnecessary for tactical systems, and that physical and communications security provide sufficient protection. Some tactical systems thus have little or no COMPUSEC protection.

This paper examines COMPUSEC requirements for one tactical Army system, based upon analysis and experiences in acquiring and operating the system. The paper reaffirms that COMPUSEC safeguards are indeed required. In fact, the incidents reported in this paper (e.g., a DoD penetrator) made system users into a community of COMPUSEC believers, and might be of similar benefit to users of other systems. A further conclusion is that, for the system examined, the Orange Book and its companion documents are valid, but that supplementary guidance is desirable on COMPUSEC management and operation in tactical situations.

The overall purposes of this paper are to focus more attention on field COMPUSEC needs and to provide motivation for improving field practices.

SYSTEM DESCRIPTION

The Army system from which the observations in this paper were drawn is a Command and Control (C^2) system being fielded in USAREUR. Initial operation was in February 1987. The system has since grown rapidly into a network.

By Spring 1988, the system included 63 user computers operating at about 25 sites in five European countries. At all sites the computers are interconnected via fiber optic Local Area Networks (LANs). A Wide Area Network (WAN) backbone is provided by four transportable Defense Data Network-compatible packet switching nodes. While the entire interconnected network is operational only during exercises, an increasing percentage is used for normal day-to-day operation.

The most important function provided is electronic mail, which has been adapted into a message service employing preformatted messages. The system also provides word processing, a spreadsheet, and a Data Base Management System (DBMS). The DBMS is being used in a distributed fashion, with numerous applications built upon it.

The system processes classified data. Operation began in the dedicated security mode of operation and is evolving to the system high security mode of operation[4].

Security requirements for the system were thoroughly defined, and the system was acquired with the necessary security foundation. The security safeguards were phased into use gradually, in order to reduce system management complexity and avoid denials of service, while learning how best to employ the safeguards. This approach has provided insights on operation with and without safeguards and on phasing the safeguards into use.

TACTICAL REQUIREMENTS

From the above description, the system might not appear to be tactical. In fact, it is one of the increasing number of DoD systems that must support both non-tactical operation (for routine peacetime work) and tactical operation (for exercises and wartime). This section identifies system requirements that characterize the tactical environment and that have influenced COMPUSEC requirements for the system. The tactical environment has many distinguishing requirements (e.g., pertaining to communications, power, weight, durability, and ease of use) that greatly influence system definition and use; the focus of this paper is on COMPUSEC requirements. The system, then, must meet the following requirements:

- o Be transportable; support multiple moves on short notice. (Many sites move during exercises; some sites move several times during an exercise.)
- o Support roles and functions that exist only in exercises and wartime, as well as roles and functions that exist during peacetime only or during both peacetime and wartime.

*This paper is derived from work performed under contract F19628-86-C-0001 for the United States Army, Europe (USAREUR), Office of the Deputy Chief of Staff for Operations.

- Support a hastily assembled and rapidly changing user community.

- Be survivable; operate despite a high rate of system and communication failures.

- Operate using only 300-3400 Hertz unconditioned voice channels for remote connections.

- Defend against violations resulting from site overrun.

The remainder of this paper examines COMPUSEC requirements in light of these tactical requirements.

AUTHENTICATION

Many tactical systems do not use passwords. The USAREUR system also was initially fielded without passwords, but passwords were found to be necessary. Furthermore, a number of users requested a password capability. Passwords have since been added to the system.

Army Regulation 380-380 stresses the desirability of using individual userids[4]. Although the system has phased in passwords, individual userids are not used, since individual userids can be difficult to use in tactical environments. There are several reasons:

- People often work in teams, sharing the use of workstations, such that it would be unacceptably cumbersome and time consuming for individual users to log in and out, especially if it is necessary each time to return the system to its pre-login operating state.

- The user community often is hastily assembled and changes rapidly, making userid and password management difficult, especially in a crisis environment.

For these reasons, group userids are used in exercises, and would be in wartime as well. For peacetime operation, however, consideration is being given to evolving to individual userids, because of needs for improved control and for individual accountability. Systems such as this one that support both peacetime (non-tactical) and exercise/wartime operation thus might have to support both individual and group userids. This does not pose technical difficulties, since userids can be used to support either individuals or groups. The issue, rather, is a management issue, regarding how userids are to be used. Where both individual and group userids are used, care is needed in integrating the userids with the system addressing approach (e.g., for mail).

Even when group userids are used, it can be difficult to prepare and distribute userid (and password) tables in situations where the user community is being rapidly assembled. Two types of distribution are needed: the tables must be installed (and tailored) on all appropriate systems and users must be informed of their userids and passwords. This process is made more difficult by the use of classified passwords. In one USAREUR exercise, users at several sites experienced substantial denial of service due to delays in distributing and installing userids and passwords.

To avoid such denial of service delays, it should be possible to create and manage userids and passwords in a distributed manner, without reliance on a remote central site. This distributed management also creates a need for a network-wide userid naming convention, to ensure that userids are unique. Of course, in avoiding denial of service delays, there is also an operational requirement to carefully plan security initialization of a network.

In tactical situations, distributed operation must be supported to the greatest extent feasible, since connectivity might not be available. For example, even if userid creation normally is done via a security server, it should be possible for standalone or isolated workstations to perform this function when the server is unavailable. Workstations should not be totally dependent on a server for their security or operation. A workstation forced to operate without a network or server should sacrifice neither security nor local operating capability.

Important authentication requirements for distributed tactical environments are that users must be capable of invoking the password change procedure on demand, and that security officer involvement must not be necessary. The reason for these requirements is that users might quickly have to change their passwords to react to changing tactical situations, and that a security officer might not be available to issue passwords. Both requirements are recommended in the National Computer Security Center's (NCSC) DoD Password Management Guideline[5].

A final requirement related to authentication is the need to authenticate the originator of transactions, such as messages. Users must not be able to originate transactions that appear to be from other users, because this capability could be used to originate false transactions, which might cause serious disruptions. The USAREUR system has had one such false transaction, and its defenses have been bolstered to prevent reoccurrences. Other Theater systems also have had false transactions. On one system, a user, pretending to be another user, sent an insulting E-mail message to a third user. The third user responded by returning an insulting note to the listed originator, who was unaware of the initial message. Fortunately, the problem was identified, and audit trails enabled identification of the culprit. The point of this discussion, however, is that the threat is real and must be countered. On a related topic, it is probably the case that transaction integrity is more important in tactical than strategic systems, since tactical systems often require quicker response times, and since tactical integrity violations might prevent weapons strikes or cause strikes against friendly forces.

ACCESS CONTROL

It is sometimes said that there is no need for discretionary access controls in tactical systems. That is not true for the USAREUR system. Although the vast majority of system users sees no need to control "read" access to data, a small minority of users does require such protection. Furthermore, there is a strong need for "write" protection. During exercises and wartime, many users are highly stressed and might not be adequately trained. The system must prevent these users from accidentally corrupting or destroying data. Discretionary access controls help to provide this capability by reducing the number of people who can change data.

As was true for userid and password tables, the most difficult aspect of managing discretionary access controls in a tactical environment is in ensuring that permissions are properly set when a large amount of equipment is initially deployed or when a user community is hastily assembled. Complicating the assignment of permissions are the facts that organization structures change in wartime and that people switch organizations. It is important that this aspect of system operation be anticipated, taught, and practiced.

During exercises and wartime, all sites in the USAREUR network operate at the same security level, i.e., the maximum allowable classification is the same for all systems. This simplifies interoperability among network users. Where communication outside the network is required (with systems that operate at higher or lower classification levels), users employ a floppy disk downgrade process similar to that described in the National Telecommunications and Information Systems Security (NTISS) Advisory Memorandum on Office Automation Security Guideline[6].

During peacetime, many sites operate at a different security level than they do in exercises and wartime, e.g., many operate at the unclassified level. These sites require the capability to alternate between operation at different security levels, in a "periods processing" fashion. This is one of the most important security requirements and is especially critical for exercises.

Vulnerability to Internal Penetrators

With the increased use and size of networks, more thought must be given to network security and to the vulnerability of the network as a whole to every one of its users. DoD system managers often assume that, because all users have security clearances, there is no penetrator threat. That is not true.

During one USAREUR exercise, a person caused significant disruption. This person was cleared and authorized to access portions of the network, but his actions exceeded his authorizations. The first indication of a problem came when a site received an alert and found that its password table had been destroyed, necessitating a lengthy reboot, and resulting in the loss of some data. This was the first of three such attacks, all of which required a lengthy system reboot.

With the first alert, the site called the network operations center, which began to trace the source of the activity. Meanwhile, the person had broken into the operating system of one of the two central database servers (supporting the overall network), and twice attempted to reinitialize the large hard disk. Fortunately, both attempts failed, or substantial amounts of exercise data would have been lost.

Network operations personnel were able to locate the person, and the activities promptly ceased. But significant disruptions had been caused, and greater difficulties had been only narrowly avoided. The point to this incident is that the threat exists.

In this case, because the network was being initially fielded, some of the internal controls were not yet in place. This was intentional, to avoid denial of service due to improperly initialized security controls. The particular approaches used by the intruder would not have worked had the full security defenses been in place.

Even with the full defenses, however, the system is vulnerable to knowledgeable users. Unlike some tactical systems, the USAREUR system is based on a commercial operating system that is familiar to many users and that provides rich functionality to users who can penetrate the user interface. Even were the system trusted, vulnerability would remain. This was vividly illustrated by the penetration of the National Aeronautics and Space Administration (NASA) Space Physics Applications Network (SPAN), which uses an operating system that has received a class C2 rating from the NCSC[7]. Another example was the penetration during UNIX Expo 1987 of Gould's C2-certified UTX/32S, in which the system administrator was duped into allowing an intrusion[8]. (Indeed, the system administrator might prove to be the Achilles' heel for many trusted systems.)

In the USAREUR case, as increased security is phased in, users are informed of what is allowable and not allowable and of the risks they assume by being a member of the network. Beyond this, there are limits to what can be achieved. It is unlikely that the risks will constrain or limit network expansion. Like some other advanced technologies, networks are volatile tools that increase both system capability and system vulnerability.

Overrun Protection

A final area of access control in which tactical systems warrant special consideration is overrun protection. The threat is that an enemy might overrun a site and access the data stored there.

The typical defense against overrun is to physically destroy the system and media. This is simplified by having fewer media to destroy or by selectively destroying only the most sensitive media (e.g., those with current data pertaining to many sites, rather than those that contain only old audit data). In extreme cases, DoD policy is to use Anticompromise Emergency Destruct (ACED) devices to accomplish the destruction[9]. ACEDs are dangerous, however, and can cause substantial damage if accidentally activated.

An alternative solution to destroy data is to encrypt stored data, so that the data can be rendered unavailable (for an estimated period of time) by destruction of the encryption key. This approach has been considered for use by the Department of State[10]. This approach could result in substantial data loss if the key is accidentally destroyed, but a copy of the key can be stored at a second site to reduce this risk.

While most tactical systems probably do not have a sufficient overrun risk to warrant special protection, those sites that are at high risk should consider file encryption. The likelihood of accidental destruction must be anticipated and countered, however, so that the safeguard does not cause greater losses than the threat itself.

A second threat associated with overrun is that an enemy might use the site's people and equipment to access other systems. One technique that can be useful in defending against this threat is a "duress code," such as is available on the Access Control Encryption (ACE) cards developed by Security Dynamics. A duress code is a special Personal Identification Number (PIN) that allows access but sets off an alarm at the operator console.

AUDITING

Some tactical users have said that auditing might be desirable during peacetime or exercises, but is undesirable during wartime. Such users have stated that (1) there would be no time to analyze an audit trail during wartime, and (2) removal of auditing could increase system performance. Neither reason is valid. Audit safeguards serve four purposes: surveillance, deterrence, damage assessment, and problem analysis and resolution. These purposes are at least as important in wartime as they are in peacetime. In addition, auditing should be sufficiently unobtrusive that it not impede operation, whether in peacetime or wartime. There might be cases in which auditing should be changed or even reduced in wartime, but it cannot be discarded. Of course, if a choice must be made between operation and security in tactical wartime situations, the choice normally should be for operation. For example, take the case of workstations on a LAN, where the workstations regularly forward their audit data to a central server. If the server is not available and audit storage space is exhausted, the old audit data should be overwritten rather than suspending operation for want of audit storage.

Security auditing is needed in tactical systems, with attention focused on how to audit most effectively and efficiently. For example, unless prodigious amounts of data can be stored and analyzed, the audit trail should not normally compile a list of all authorized user activities (e.g., file open, program initiation), but must record attempts at unauthorized activity and all changes to security parameters. The audit concept cannot assume that there is a security officer who sits at a console, monitoring the system, or who has time to search archival audit files looking for suspicious activity (although software might be used to conduct such searches). Instead, the concept should focus on alerts and reports. Alerts would be sent to the overall system operator or system administrator (if there is one), who would notify a security officer if warranted. Alerts might include such actions as the addition of a new userid or the receipt of a login from a user already logged in to another workstation. (In most cases these actions will be normal and authorized, but perhaps not in all cases.) Reports would be compiled from many events and distributed to the security officer. Reports are useful in identifying groups of users with a high error rate (who might therefore need guidance or training). The most requested report probably will be a simple daily listing of who logged in and out, when, and what external resources were accessed.

CONCLUSIONS

Experience with one tactical system reaffirms that computer security safeguards are needed, not only to prevent loss of sensitive data, but also to ensure the accuracy of data and transactions. The required COMPUSEC features for this tactical system are almost identical to those for most strategic systems. The main differences are in how the features are used.

To help ensure that COMPUSEC features are properly used in tactical systems, the DoD should incorporate into overall COMPUSEC policy brief guidance on COMPUSEC management and operation in tactical environments. AR 380-380 already includes some tactical COMPUSEC guidance, but could benefit from expansion to address points made in this paper[4]. Furthermore, requirements and procedures that pertain to tactical systems should be incorporated into Security Features User's Guides, Trusted Facility Manuals, Standard Operating Procedures, and Operating Concept documents.

Recent improvements in DoD COMPUSEC policy and technology should result in improved DoD security, but care must be taken that the new policies and technologies are properly applied. This paper examines the application of COMPUSEC in a tactical system. Similar field reportage should be encouraged both for other tactical systems and in other application areas, so that empirical data is used to validate and improve upon our policies and technologies.

ACKNOWLEDGMENTS

The author is grateful for the review and comments provided by Mr. Shel Dick of Headquarters, USAREUR, and by Dr. John Vasak of The MITRE Corporation.

REFERENCES

[1] DoD Directive 5200.28 (March 1988), "Security Requirements for Automated Information Systems (AISs)," Deputy Secretary of Defense.

[2] DoD 5200.28-STD (December 1985), Department of Defense Trusted Computer System Evaluation Criteria, Deputy Secretary of Defense.

[3] NCSC-TG-005 (July 1987), Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, National Computer Security Center.

[4] Army Regulation 380-380 (March 1987), Automation Security, Headquarters, Department of the Army.

[5] CSC-STD-002-85 (April 1985), Department of Defense Password Management Guideline, National Computer Security Center.

[6] NTISSAM COMPUSEC/1-87 (January 1987), Advisory Memorandum on Office Automation Security Guideline, National Telecommunications and Information Systems Security Committee, National Security Agency.

[7] Marbach, W. D., A. Nagorski, and R. Sandza (28 September 1987), "Hacking Through NASA," NEWSWEEK.

[8] Smith, K. (February 1988), "Tales of the Damned," UNIX Review, Vol. 6, No. 2.

[9] DoD 5200.1-R (June 1986), Information Security Program Regulation, Deputy Secretary of Defense.

[10] McNulty, L. (1987), private communication, Department of State.

COMSEC INTEGRATION ALTERNATIVES

John Linn
BBN Communications Corporation
150 CambridgePark Drive
Cambridge, Massachusetts 02140

Abstract

Encryption-based communications security (COMSEC) functions can be integrated into a network system in several ways. The "traditional" approach places COMSEC modules on a communications path, essentially transparent to the components communicating across that path. An alternative approach integrates COMSEC functions within a front end processor, interposed on a communications path yet participating in an explicit host-FE protocol with the host computer it serves. A third approach positions COMSEC functions within a peripheral operating under host computer control. This paper explores the implications of each approach, with regard to protocol layer placement, comprehensiveness of security services offered, applicability to environments, TCB boundaries and evaluation concerns, transparency, several dimensions of cost, and the ability of an approach to provide enhanced functions in support of an associated host computer.

Introduction

In order to achieve a total information security (INFOSEC) solution in a network environment, communications security (COMSEC) and computer security (COMPUSEC) techniques must be combined. These techniques can be combined in many ways. Each COMSEC approach raises a different set of COMPUSEC issues, not only with regard to a host computer's internal processing but also with regard to COMSEC component control. Many COMSEC components incorporate their own embedded control processors, raising internal COMPUSEC issues distinct from those of the host computers they serve. Some approaches rely on host computers to perform COMSEC control functions along with the user applications they support. This paper examines the implications which different approaches impose, both on COMSEC component design and on the designs of the hosts the COMSEC components serve.

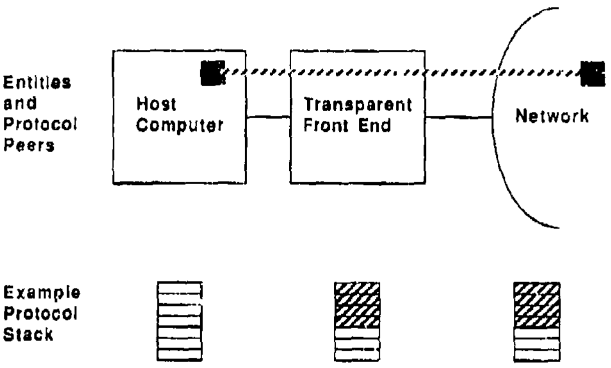
Definition of Alternatives

This section defines three categories of COMSEC integration strategies, informally termed the transparent front end (FE) approach, the non-transparent FE approach, and the peripheral approach. The FE approaches are distinguished from the peripheral approach by being interposed on a host's communications path. This placement assures that no network communication can occur except under security component control; all communications are mediated through the FE. The peripheral approach, on the other hand, is invoked under host control. Its hardware structure does not guarantee that security component processing is invoked on all communications. The FE approaches are distinguished from each other based on whether the host explicitly requests services and/or receives results from the COMSEC component, or the component acts as a transparent and silent communications partner.

Outboard Transparent Integration

In the outboard transparent integration approach, an encryption component is interposed on a computer's communications path in a manner which is transparent to the computer. This allows encryption to provide a variety of security services without impact on existing host protocol implementations, but precludes the encryption component from providing functions which require explicit interaction with the host. As a result, the host computer cannot select or influence the security functions which the component provides. For example, the set of OSI security services provided by the component may be fixed when the component is manufactured, or may be loaded into the component as configuration data (with different choices for different destinations, if appropriate), but cannot be selected dynamically by the host for individual instances of communication.

Figure 1 illustrates an example of COMSEC integration within a transparent front end system. The host computer is a protocol peer with another host computer (not shown), accessed through the network. The example protocol stack shows OSI layers 1-3 (physical through network layers) passed through the front end to the network without encryption, with layers 4 through 7 (transport through application) encrypted before transmission. This is a typical protocol layer placement for a transparent FE.



Outboard Transparent Integration (Transparent FE)

Figure 1

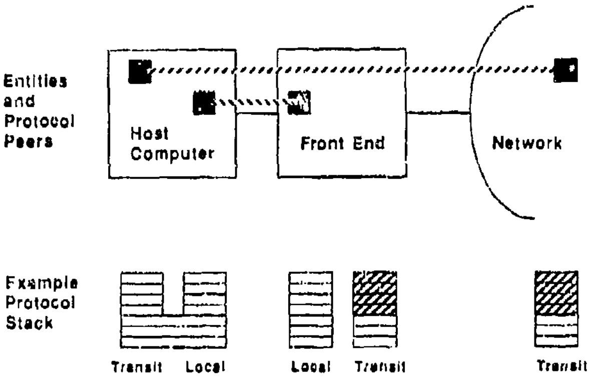
Outboard transparent integration is the "traditional" approach to COMSEC placement, reflecting the development of COMSEC and COMPUSEC as distinct disciplines and consideration of COMSEC as a function within the communications realm rather than the subscriber realm. The choice of the term "outboard" to define this approach is not meant to imply that the encryption component must be in a separate stand-alone physical package from its associated host. It may, for example, be a circuit board which plugs into a computer's bus. The salient characteristic is that all network communications accesses must traverse the FE.

In the past, most COMSEC components have been transparent link encryptors. These components operate at the OSI Reference Model's layer 1 or the lower part of layer 2, and treat any protocol control information of layers above as uninterpreted data. The internal processing requirements for an encryptor operating in this range of the protocol hierarchy are modest. Generally, the level of internal processing complexity in a COMSEC component increases when a component operates at a higher point in the protocol hierarchy. In particular, a transparent COMSEC component operating at the upper part of layer 2 or the layers above can be a significantly complex embedded computer system. Part of the complexity comes from the fact that a transparent COMSEC FE must duplicate protocol layers already present within an associated host. The layers below the point where encryption is integrated must be terminated from the host's viewpoint and regenerated for the network's benefit, making for a relatively complex "two-headed" implementation.

Outboard Integration with Host-FE Protocol

When encryption is incorporated into an outboard component which participates in an explicit protocol with the host computer it serves, different implications arise. The host computer must act as a peer in the explicit protocol, meaning that the host computer's software must operate differently than it would operate if no encryption component were present. If this can be accomplished, then the encryption component can provide valuable services to its host which a transparently integrated outboard component cannot offer. As a special case, a non-transparent FE could be designed in such a manner that it could operate with reduced functionality even if no peer for the host-FE protocol were available. Such a hybrid would operate as a transparent FE if its host did not support the host-FE protocol.

Figure 2 illustrates an example of COMSEC integration within a non-transparent front end system. As in Figure 1, the host computer is a protocol peer with another host computer (not shown), accessed through the network. Layers 4 through 7 of this traffic are encrypted. In contrast to Figure 1, this host computer also acts as a peer with the local host-FE protocol module in the front end system; two protocol substacks from layers 4 through 7 correspond to the transit (host-host) and local (host-FE) paths. In the example, therefore, demultiplexing between transit and local traffic streams is carried out using network layer mechanisms.



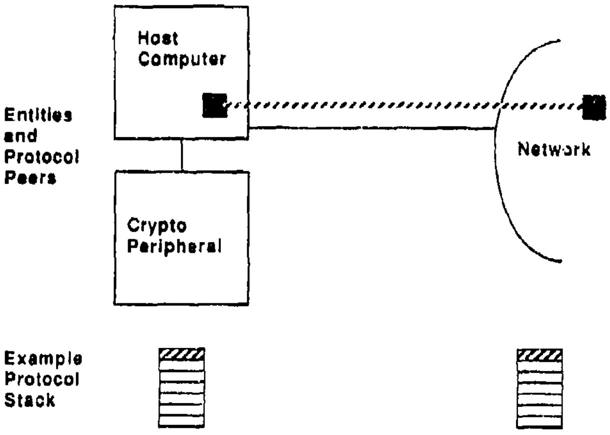
Outboard Integration with Host-FE Protocol (Non-Transparent FE)

Figure 2

There are many variations of the outboard non-transparent approach, distinguished by the host-FE protocol's characteristics. As with the outboard transparent integration approach, the choice of the term "outboard" to describe the approach is not meant to preclude the encryption module from sharing common physical packaging with the host it serves. One variation resembles the transparent approach, in that the protocol layering between host and FE is the same as that between FE and network (except for any layers added by the FE in support of COMSEC functions). In this variation, the host-FE protocol is used to carry information such as connection requests and authentication data between the host and FE. In another variation, implementation of protocols below a given layer is delegated to the FE. Independent of security concerns, many communications front end processors have been designed in this manner in order to offload low layer protocol processing from a host; COMSEC integration within such a front end is a natural extension.

Inboard Integration as Peripheral

When encryption is embedded in a peripheral device operating under host software control, many new and qualitatively different functions become possible. Encryption can be applied in a fashion specific to upper-layer protocols and can distinguish among individual users. The price for this flexibility is a trust requirement imposed on the host directing the cryptographic peripheral's operations. Invocation of cryptographic processing is controlled by the host computer's TCB.



Integration as Peripheral

Figure 3

Figure 3 illustrates an example of COMSEC integration within a cryptographic peripheral rather than a front end system. As in Figures 1 and 2, the host computer is a protocol peer with another host computer (not shown), accessed through the network. Since the interface between the host and the peripheral is a local matter, no explicit communications protocol applies between these two components. The host invokes the peripheral's functions in order to provide cryptographic protection for an application layer protocol, such as X.400 messaging or FTAM. While peripheral-based COMSEC integration is not confined to application layer uses, it offers a better approach to upper-layer security requirements than front end approaches can provide.

Cryptographic peripherals can be designed in various ways, offering significantly different service interfaces to their associated host computers. Typically, a processor-less peripheral presents its host with an interface which allows the host great flexibility in terms of choice of operations but requires the host to interact with the peripheral at a low level. A peripheral with an onboard processor can offer a more constrained interface, and can group low-level primitive cryptographic functions into atomic operations. In all cases, any unencrypted encryption keys should be protected within the peripheral's physical boundaries and should not be accessible to the host processor. These techniques can help to protect the internal integrity of COMSEC functions. While they reduce important aspects of the trust requirements placed on the host processor's TCB, they do not relieve the host of responsibility for ensuring that COMSEC functions are invoked when appropriate. The host's TCB must still assure that data which should be encrypted is in fact encrypted.

In a multi-level system, the input to an encryption function often has a higher security level than the function's output; appropriate labeling must be applied and enforced at the function's interfaces. In certain cases, it may be feasible and desirable to enforce labeling conventions within the peripheral's boundary.

Even when all the functions associated with a cryptographic peripheral's outboard processor are considered, they are likely to be significantly smaller than those associated with an FE's processor. No local operating system or communications protocol support is ordinarily required. Often, processing requirements can be satisfied with a microcontroller rather than a full-fledged processor.

Areas for Comparison

This section introduces several criteria which are important in comparing alternative approaches, and considers the relationship between those criteria and the implementation options.

Protocol Layer Placement

This criterion measures an approach's ability to provide protection in a fashion specific to a broad range of protocol layers. While protection applied at any layer provides a measure of protection for layers above, it is not tailored to the special capabilities and needs of specific protocols at higher layers and hence can't satisfy all security requirements of a layered protocol architecture. For example, network layer (layer 3) encryption can protect the stream of traffic between a pair of hosts, but can't distinguish between different individual users on those hosts as they send interpersonal messages at the application layer (layer 7).

The peripheral approach is most successful with regard to this criterion. It can be applied at any layer up to and including the application layer. The transparent FE approach is least successful, as it is generally difficult to apply above the network layer. The non-transparent FE can be applied above the network layer, especially if implementation of lower layers is delegated to the FE, but cannot be easily extended all the way to the application layer

Security Service Comprehensiveness

This criterion measures an approach's ability to provide a comprehensive range of security services. While the set of appropriate functions varies depending on the choice of protocol into which COMSEC is integrated, some generalizations can be made about different approaches' attributes. To succeed in this criterion, an approach should be able to provide the full range of OSI security services which are appropriate to its protocol placement.

A distinction can be made between "value-added" services and services which restrict a host computer's actions. Administratively-directed access control is a good example of the latter category. The FE approaches are superior to the peripheral approach in their ability to enforce controls restricting a host.

On the other hand, certain services are best provided within a host. For example, the non-repudiation service provides message accountability. A host's users should not be expected to accept individual responsibility for an incoming message merely because it was acknowledged by an FE-based encryptor. User-level accountability should be based on a path extending all the way to a user.

No single approach is optimal for providing all types of services. Certain services are best provided by extending the endpoints of COMSEC protection into subscriber hosts, most closely approaching the hosts' users. Other services are best provided by a device operating independently from a subscriber host and immune from its control. In order to evaluate an approach with regard to service comprehensiveness, one must first establish the set of services which are imporant in a given market sector.

Applicability to Environments

In some environments, a COMSEC component's primary role is to provide value-added security services when such services are requested by a subscriber host or process. Typically, this paradigm is associated with unclassified environments, although it may also be applicable to certain classified contexts, particularly when trust levels and/or access class ranges are uniform within a network. Such services are best provided by a peripheral or non-transparent FE COMSEC module; a transparent FE, by its nature, lacks a control interface through which an associated host computer can select optional services.

In other environments, a COMSEC component assumes an enforcement filter function, restricting the actions of an associated host computer in order to enforce a network security policy. Such a policy may be rule-based and/or identity-based. Administratively-directed access control and restriction of covert channel bandwidth from a host into a network offer good examples of filtering functions which an associated security component may impose on a host. These types of functions are most conveniently provided by a FE module, either non-transparent or transparent. Integration of enforcement functions within the host at which the enforcement is directed imposes severe trust requirements on the host's architecture.

TCB Boundaries and Evaluation Concerns

This criterion measures the size of the trusted computing base concerned with COMSEC-related functions. It is an indication of the system-level evaluation task's scope and difficulty. In the transparent FE approach, COMSEC-related TCB functions are confined to the outboard subsystem. In general, the complexity of an embedded system such as a cryptographic FE (either transparent or non-transparent) is less than that of a general-purpose host computer, simplifying evaluation, but should not be dismissed as insignificant. A protocol-oriented cryptographic component may easily contain thousands or tens of thousands of lines of code. When the non-transparent FE approach is used, most COMSEC-related TCB functions remain in the outboard subsystem. Depending on the capabilities built into the host-FE protocol, employment of such a protocol may or may not imply the need for host-based trusted functions. Integration of COMSEC within a peripheral places a larger trust burden on the peripheral's associated host, although appropriate peripheral design strategies can act to bound this concern.

Transparency

This criterion measures the extent to which an existing host computer (hardware and/or software) must be modified in order to coexist with COMSEC. Success in this area allows a COMSEC integration approach to be applied in order to protect existing unmodified hosts. Clearly, the transparent FE approach is most successful in satisfying this criterion. It is followed by the non-transparent FE approach, typically requiring only software modification. The peripheral approach, typically requiring software and hardware modifications, is least transparent.

Costs

The incremental costs of adding security to a network system can be evaluated in several dimensions. Unit purchase cost for the security components is the most obvious parameter, but operational support costs often assume dominant importance over a system's life cycle. Another dimension of cost deals with the security components' impact on overall system performance.

The peripheral integration approach offers the potential for significant advantages with regard to purchase cost. When COMSEC is integrated within a host computer's hardware base, less duplication of hardware components, packaging, and software is required in order to incorporate security. Both FE approaches incorporate their own separately-packaged autonomous processors, and are therefore likely to be more expensive to produce than a peripheral implementation. On the other hand, an FE approach can be applied to a diverse range of host computer types, which may allow improved economies of scale for FE approaches.

The peripheral approach also offers benefits with regard to maintenance and operational support. When COMSEC is provided by a computer vendor, the vendor can maintain the COMSEC components along with their associated host computer. This integrated support improves overall system availability by reducing "finger pointing" among multiple vendors of different network system components.

Different integration alternatives have different impacts on system performance. The peripheral approach's impact is direct; processor cycles used for COMSEC component control are not available to perform other tasks. While tangible, this burden will often be fairly minor. If applied indiscriminately, FE approaches can perturb the operation of communications protocols in ways which can impact performance severely. For example, introduction of added control information into transmitted messages can cause packets to be fragmented into multiple packets, increasing overhead and communications costs. Careful system engineering is required in order to avoid such inefficiencies.

Enhanced Host-COMSEC Interaction

This criterion considers the richness of a COMSEC component's service interface, in terms of the types of control which can be invoked and data which can be passed across that interface. For example, authentication data carried in a peer-peer protocol between COMSEC components can be reflected to an associated host and used by the host as an input to its internal access control and authentication mechanisms. In the other direction, host-resident data can be provided as an input to access authorization decisions made within a COMSEC component. These examples illustrate the composition of COMSEC and COMPUSEC into an overall INFOSEC architecture. The peripheral approach allows the most powerful service interface, followed by the non-transparent FE. The transparent FE presents no explicit service interface, and therefore is least attractive with regard to this criterion.

Conclusions

Each approach has good and bad points with respect to different criteria. This reflects the fact that different approaches are best suited to different parts of the overall COMSEC market space.

The peripheral approach is quite flexible, offers a powerful service interface, and can be applied throughout the protocol hierarchy. The peripheral approach is poorly suited, however, to performing enforcement functions to restrict its associated host. Because of this characteristic, its cost and functionality benefits will probably first be fully appreciated by customers with unclassified processing requirements. Until highly trusted hosts become more widespread, it appears that FE-based protection will be required for many classified processing needs. It is possible, however, for FE-based approaches at lower protocol layers to be used in conjunction with host-based COMSEC providing fine granularity protection at higher protocol layers.

The FE approaches are more costly, less flexible, and their protocol layer applicability is more limited. On the other hand, their encapsulation of functions within a separate security perimeter simplifies system-level evaluation. The transparent FE offers a unique benefit; it can provide a "security overlay" at network interfaces, protecting the traffic of existing, unmodified host computers. The non-transparent FE's explicit service interface allows it to provide better support to host-based functions than a transparent FE can provide.

Acknowledgment

I would like to thank the anonymous reviewer of a draft version of this paper for comments which suggested useful areas towards which to focus additional discussion.

Architectural Model of the SDNS Key Management Protocol

Paul A. Lambert
Motorola GEG

Abstract

The Secure Data Network System (SDNS) project has developed a security architecture within the Organization of International Standardization's (ISO) Open System Interconnection (OSI) computer network model. The foundation of the SDNS security architecture is based on a distributed key management technique that is embodied in an application layer Key Management Protocol (KMP). In the SDNS architecture the KMP provides a uniform mechanism for the establishment of secure communications. This paper describes the security services furnished by the KMP and examines the relationship of the KMP to the OSI reference model.

Introduction

Key management is defined by ISO as "the generation, distribution/issuance, storage, updating, destruction, and archiving of keys." The SDNS model for key management distributes this functionality into every security device in a communication system. The generation or distribution of a key is then the responsibility of each pair of communicating security devices. Before any instance of secure communication between SDNS systems the peer devices must use the Key Management Protocol to establish their identities and then determine a key to use for subsequent communications.

Correct identification of remote systems is assured by mutual authentication. The authentication of SDNS security devices is based on the exchange of credentials. The credentials provide identity information and information that may be used for access control decisions. A drivers license or a credit card are good analogies to the security devices credentials. The credentials are issued by a central authority and are subsequently used without any interaction with the central authority. SDNS will provide a Key Management Center (KMC) that will be the responsible authority for the creation and distribution of credential material.

The SDNS Key Management Protocol is the mechanism for the exchange of credentials. The basis of the KMP is a simple four-way handshake. Two communicating systems must first exchange the credentials that describe themselves. Each of the peer systems validates the credentials and then exchange messages that determine how they will communicate. This second pair of messages is encrypted by a key that each system has determined from the exchanged credentials. The ability of each peer to decrypt the second exchange tests the correctness of the key that was selected or formed from the credential exchange. In this manner the successful validation of the second pair of messages completes the mutual authentication of the peer devices.

The SDNS project has defined specific cryptographic algorithms and data formats. While these algorithms ensure the interoperability of government certified security systems, the definition of the architectural model and the protocols are independent of these algorithms. The protocols are capable of supporting multiple algorithms and define only the mechanisms for transferring information used in the key management operations.

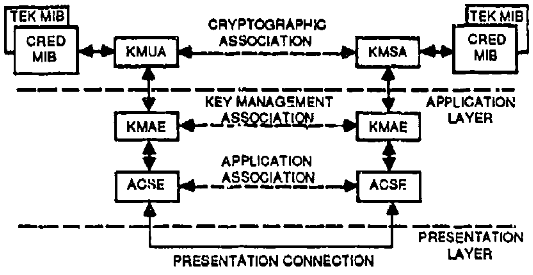


Figure 1. SDNS key management model.

Application Layer Model

Key management is defined within the SDNS architecture as an application service element. A model of SDNS key management in the application layer of the OSI reference model is illustrated by Figure 1. In this model, key management is divided into agents that support the user and a Key Management Application Entity (KMAE) that supplies the communication services. The communication services consist of the ISO standard Application Control Service Element (ACSE) and the SDNS defined Key Management Application Element (KMAE). In this model the Key Management Protocol (KMP) provides the services defined for the KMASE.

The Key Management User Agent (KMUA) and the Key Management Server Agent (KMSA) are the local and remote entities that act on behalf of a user to manage the TEKs and credentials. These agents provide all of the authentication and access control services based on information provided by the KMAE. The KMUA and KMSA are also responsible for the generation of TEKs based on the credentials. The management of the credentials and keys is indicated by the management information bases (MIBs) in Figure 1. The establishment of a cryptographic association is used in this model to indicate the existence of a TEK shared between the agents. This association includes attributes that describe the intended usage of the TEK and access control limitations.

It is important to note that the duration of the cryptographic association is normally longer than the duration of the key management association. The key management association and the lower layer protocols are used to establish the TEK and it's attributes. After a cryptographic association is formed the key management and application associations may be released. This allows the use of the TEK to be independent of the instance of communication used to establish the TEK.

The KMP requires the services of the ISO application layer protocol ACSE and the presentation protocol. These protocols provide the means for the communication of key management information. SDNS has defined requirements on the protocols used for the application, presentation, session, and transport layers to ensure the interoperability of SDNS key management implementations.

Communication Protocol Requirements

The communication protocols required by the KMP are standardized for the application, p,esentation, session, and transport layers of the OSI reference model. OSI conformant protocols are specified by SDNS for these communication layers. The ACSE protocol is required in the application layer for SDNS key management. ACSE provides services for the establishment and termination of application associations.

A minimal Presentation Protocol is required by SDNS key management. The presentation layer functions include the negotiation of transfer syntaxes and may provide transformations between the transfer and abstract syntaxes. SDNS key management uses a single standard transfer syntax.

The Session Protocol provides services for the organization and synchronization of a dialog between presentation layer entities. The KMP requires only the kernel and duplex function units of the standard Session Protocol. The transport layer provides for the reliable transfer of data. Transport protocol class 4 (TP4) is required to support SDNS key management. Lower classes of service may be negotiated. For example, while TP4 is the default for interoperability, TP0 may be used in reliable network environments.

Dual Stack Model

The communication architecture of SDNS key management allows the key management communication services to be separate from the user communication services. This separation of the upper protocol layers is shown in the dual stack model of Figure 2. Each stack represents the profile of the communication services required by the applications. The upper layers of this model are logically distinct. The lower layer protocols may or may not be shared by key management and user traffic.

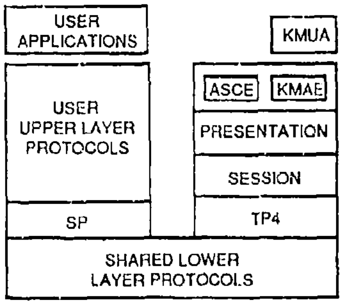


Figure 2. SDNS Dual Stack Model

This model illustrates a security protocol (SP) in the path of user communications. In the currently defined SDNS framework this security may be implemented at the link layer, network layer, or transport layer. When the security protocol needs a traffic key for its security services it makes a request to the KMUA. This request typically occurs when a new instance of communication is to be established through a remote security system. The KMUA then uses the KMP to establish a cryptographic association with the remote KMSA. After all authentication and access control rules are validated, the KMP releases the key management association and leaves the cryptographic association in place. The new cryptographic association includes the TEK, access control privileges, and the security protocol that will use the TEK. The KMUA then returns this information to the security protocol which subsequently uses this information for its security services.

The dual stack model allows SDNS key management to be used for ISO or non-ISO user traffic. It also allows for the support of security at intermediate systems. Intermediate system security may be provided by the SDNS security protocols SP2 (link layer) or SP3 (network layer). An example of the key management architecture for an SP3 intermediate system security device is shown in Figure 3. In this figure the communication service requirements for the KMP are identical to the previous illustration. The user upper layer protocols are removed to a separate host end system. Traffic from the protected host is carried on subnetwork one. The ISO network layer model defines the roles of the Subnetwork Independent Convergent Protocol (SNICP), Subnetwork Dependent Convergence Protocol (SNDCP), and the Subnetwork Access Protocols.

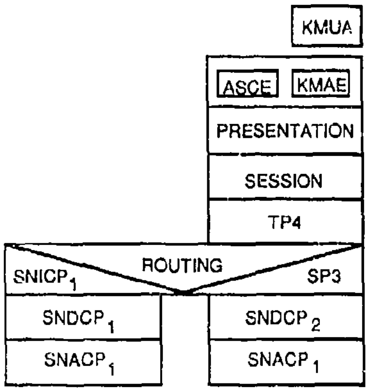


Figure 3. Intermediate System Key Management Model

Key management is provided at intermediate systems in the same manner as end systems. An application association is formed between security devices whenever a new instance of secure communication is required. The KMP then uses this association to form the cryptographic association. The association is used to support security services between the intermediate systems.

KMP Services

The KMP provides mechanisms for the following basic services:

- Authentication of peer devices.
- Access control based on authenticated credential information.
- Traffic key establishment and maintenance with other security devices.
- Rekey with the KMS to obtain new credentials.
- Establishment and maintenance of cryptographic associations with other security devices.
- Distribution of Compromise Key Lists.

The above services are used together for three basic functions - the establishment of cryptographic associations, rekey to obtain new credentials, and the distribution of Compromise Key Lists. Each of these basic functions always provide capabilities for authentication and access control.

Cryptographic Association

A principle function of the KMP is the establishment of cryptographic associations with other security devices. A cryptographic association consists of a Traffic Encryption Key (TEK) and associated attributes that determine the usage of the TEK. The TEK established by the KMP may be used by any other security service that uses cryptography. SDNS has currently defined protocols that provide security services at the Transport and Network layers of the OSI reference model. Extensions are planned for protocols that will support Link layer security. SDNS has also defined security extensions to X.400 for electronic mail security. The SDNS electronic mail does not require a cryptographic association, but may use the KMP for the rekey of credentials.

The TEK usage is determined by an option negotiation performed during the key establishment. The categories of options include algorithm options, labeling requirements, keying granularity, security protocols, and protocol options. These selections can be placed in an order to express the preferences of the initiating device. The responding device then selects an appropriate intersection if possible. This negotiation allows SDNS devices to support several security protocols or service options. Defaults have been defined to ensure interoperability. The negotiation enables the device to interoperate and where applicable provide additional services beyond the defaults.

Access control attributes are an important feature of the cryptographic association. The cryptographic association is only formed if mandatory and discretionary access control rules based on the credentials pass appropriate checks. During the traffic key establishment Peer Access Authorization (PAA) is enforced. Within the SDNS project the specification of consistent access policies and access control label formats have been defined by the SDNS Access Control working group.

Maintenance of the cryptographic association is provided for by a procedure to update a TEK, and by the capability to assign a new TEK to an existing cryptographic association.

Credential Rekey

The KMP provides support for the electronic distribution of credential material. This capability to rekey credentials is intended only to be used on an infrequent basis. The rekey of credentials is similar to the periodic issuance of new credit cards or drivers licenses. The rekey provides a mechanism for the re-certification of a device by the Key Management Center (KMC). The rekey in effect changes the expiration date of the credentials. It is not intended that the rekey be used to change the identity or access control attributes contained within the credentials.

Valid credentials are required to obtain credential material from the KMC. To enroll a device into the SDNS system manual or out of band techniques are required to install the first set of credential material. After an SDNS device is enrolled all future interactions with the KMC can be performed through a communication network.

In addition to the interactive rekey, the KMP supports a staged rekey that allows credential material to be distributed to security devices that may not have direct connectivity to the KMC. The staged rekey uses rekey agents that act as surrogate KMCs. The rekey agents accept requests from devices for new credentials and store the request for later delivery to the KMC. Multiple requests may then be grouped together into a single batch to be interactively rekeyed with the KMC. After an agent obtains the new credential material from the KMC, the KMP provides services for the delivery of the material from the agent to the security device that originally requested the rekey. Multiple agents are allowed between the requesting device and the KMC. The KMP, however, does not define an agent to agent rekey protocol. It is assumed in the staged rekey model that any mechanism that meets a systems security policy requirements can be used to move the rekey between agents. This allows for a variety of rekey material delivery techniques that include manual delivery on disks or data keys, or the use of electronic mail.

Compromise Key List

The KMP provides limited support for the management of access control information. In particular the KMP can be used to transfer lists of credential identifiers that are no longer valid. This Compromise Key List (CKL) can be compared to lists of credit card numbers at retail stores. In the same manner that a store clerk might check the list of credit card numbers to ensure the validity of a monetary transaction, an SDNS device can compare a credential identifier to the CKL to ensure the validity of a credential.

To guarantee that SDNS security devices have the latest CKL, CKL version numbers are exchanged during traffic key establishment. This allows devices to learn the latest CKL version number from their peers. To obtain a new CKL, devices may request CKL from either peer devices or from the KMC. Since the KMC is the ultimate authority for the creation and distribution of CKL, security policy may require some security devices within a community to poll the KMC for the latest CKL. CKL is also delivered with credential material during a rekey operation.

Summary

The KMP provides a uniform mechanism for the establishment of secure communications in the SDNS architecture. The KMP establishes a cryptographic association that can then be used at any layer of the reference model to provide security services. The TEK bound to the cryptographic association is installed only after the identities of peer devices are authenticated and access control checks based on the authenticated information pass. Option negotiation performed during the key establishment allows security services and protocols at any layer of the OSI reference model to be flexibly supported. The option negotiation includes the ability to bind additional access control information to the cryptographic association. Additional capabilities are provided by the KMP to support the maintenance of cryptographic associations, obtain new credentials from the Key Management Center, and obtain the latest Compromise Key List from a peer device or from the KMC.

The key management architecture is supported by communication requirements that define an interoperable stack of protocols for use by the KMP. This architecture supports security installed in end systems or intermediate systems. Although the protocols specified for use by the KMP are ISO conformant, user traffic does not have to be carried on ISO protocols.

Acknowledgements

The author of this paper gratefully acknowledges the contributions and insights provided by all the members of the SDNS Protocol and Signaling Working Group. Participants in the SDNS working groups included representatives from the following organizations - Analytics, AT&T, Bolt Beranek and Newman Inc. (BBN), Defense Communication Agency, Digital Equipment Corporation, GTE, Honeywell, Hughes, IBM, Motorola, National Bureau of Standards, National Security Agency, UNISYS, Wang, and Xerox.

References

[1] *SC 21 N1386 Proposed Draft for the Management Information Services Definition, Part 7: Security Management Service Definition*

[2] *ISO 7498 Information Processing Systems - Open Systems Interconnection - Basic Reference Model*

[3] *ISO/DIS 8648 Information Processing Systems - Data Communications - Internal Organization of the Network Layer*

Investigating Formal Specification and Verification Techniques for COMSEC Software Security

Andrew Moore

Code 5540
Naval Research Laboratory
Washington, D.C. 20375

ABSTRACT

The replacement of hard-wired crypto processors by software- and firmware-controlled processors requires techniques to assure that COMSEC software meets its security requirements. Toward the goal of gaining this assurance, we explore the application of formal specification and verification techniques to COMSEC software. This paper identifies a class of COMSEC systems and a nontrivial member of the class, A Secure Voice Terminal (ASVT). The ASVT provides a research vehicle from which to study and compare verification systems to gain insight into their suitability for verifying COMSEC software. A formal specification of the ASVT in Hoare's Communicating Sequential Processes (CSP) demonstrates the feasibility of applying formal techniques to a class of COMSEC systems. The successful application of these techniques to COMSEC software can increase assurance that the software meets its security requirements.

1. INTRODUCTION

As computers come to dominate communication systems and processing the reliability of software and hardware that control them becomes crucial to the security and performance of these systems. The replacement of hard-wired crypto processors by software- and firmware-controlled processors requires techniques to assure that COMSEC software meets its security requirements.* Toward the goal of gaining this assurance, we explore the application of formal specification and verification techniques to COMSEC software. This paper identifies a class of COMSEC systems and formalizes a nontrivial member of the class, A Secure Voice Terminal (ASVT). The language selected for this formalization, Hoare's Communicating Sequential Processes (CSP) was chosen primarily for its capacity to describe the asynchronous operation of separate processors within the COMSEC system.

The Verification Assessment Study (VAS) [8] concludes that example problems do not perform adequately as benchmarks to determine the "quality" of verification systems. The VAS does however promote their use to compare and contrast those systems. The ASVT provides a research vehicle from which to study and compare verification systems to gain insight into their suitability for verifying a class of COMSEC systems. Section 2 of this paper identifies the class of COMSEC systems studied and a class of properties to be proved of those systems. Section 3 informally describes the security policy and functionality of the ASVT. In sections 4 and 5 a subset of Hoare's CSP is presented and used to derive a formal specification of the ASVT. Finally, section 6 summarizes some lessons learned from this effort and the direction for future work.

2. A CLASS OF COMSEC SYSTEMS

The class of COMSEC systems of interest in our study are those systems composed of a set of devices connected by potentially unreliable media that allow users stationed at different devices to communicate. Each device allows the processing of information but does not allow its permanent storage. No storage is provided beyond that of temporary finite buffers. The COMSEC system has no control over the integrity or privacy of data transmitted on the communication medium between devices. COMSEC security requires securing communication between users of a COMSEC system so that only those individuals stationed at a device properly connected to the system may enter information to or extract information from the system. The attackers of COMSEC security are those individuals who attempt to affect communications over the system in any other way. Future references to COMSEC systems refer specifically to the class of systems described above.

The threats to security for information processing systems fall in three categories: unauthorized disclosure of information, unauthorized modification of information, and denial of use of system resources. Conventional computer security measures require control on the flow of information between the device and the user. The Orange Book [2] requires the definition of a trusted computing base and a security policy identifying subjects, objects and a set of rules to determine whether a subject can be permitted access to an object. COMSEC system security measures, on the other hand, require control on the flow of information between devices. Any attacks directed at the potentially unreliable communication media may result in a release of message contents to malicious individuals or miscommunication between authorized communicants.

The basic solution to the COMSEC security problem is a probabilistic one, encryption. The encryption of information flowing across a communication medium allows the attacker to gain data, but not useful information. Strategic cipher methods exist to prevent the attacker from being able to decipher the information intercepted in a reasonable amount of time. Unfortunately, encryption alone does not solve all COMSEC problems. According to Victor Voydock [10] major goals in communication security are to (1) prevent the release of message contents, prevent message traffic analysis, detect message-stream modification, detect denial of message service, and detect spurious association initiation. Achieving these goals may require the use of complicated communication protocols between users of the COMSEC system.

Rather than chose a communication protocol arbitrarily as the basis of our investigation, we have decided to concentrate on a class of properties called Red/Black separation that apply to a broad range of COMSEC systems. Red/Black separation primarily deals with preventing the release of message contents. COMSEC systems are usually partitioned into distinct sets of Red functions and Black functions called the Red processing partition and the Black processing partition, respectively. All information that is plaintext is considered to be Red data and all information that is encrypted is considered to be Black data. It is the responsibility of the COMSEC boundary to mediate all information flow between the Red and Black processing partitions to ensure that no Red data is conducted into the Black processing partition of the system without first being encrypted. This property is referred to as Red/Black separation.

3. AN EXAMPLE OF COMSEC SOFTWARE

The example chosen as the basis of our investigation is a simplification of an existing system, the Advanced Narrowband Digital Voice Terminal (ANDVT), which provides secure voice

* For our purposes, COMSEC software is software that controls the cryptographic process, interfaces with the cryptographic algorithm or implements cryptographic algorithms.

and data processing facilities to users. [3] Although the ANDVT is too complex to investigate with each verification system in full detail, its basic functionality has been abstracted into an example that can be studied with candidate verification systems in a reasonable amount of time. This example, called ASVT for A Secure Voice Terminal, enforces a security policy based on Red, Black separation.

ASVT Structure and Operation

The ASVT is split into three major blocks of operation : the Voice Processing Block, the Modem Processing Block, and the COMSEC Module. A block diagram of the system given in Figure 3-1 illustrates the four external interfaces through which voice transmissions may flow, Red Audio, Red Digital, Black Analog, and Black Digital. There are also three internal interfaces, Voice COMSEC, Voice Modem, and Modem COMSEC, for voice traffic.

The ASVT has a control panel which allows its user to perform various operations on the terminal. The control panel includes a power switch, a Push To Talk (PTT) button, and a mode selector dial. The power switch allows the user to start up and shutdown the ASVT. The user may transmit information by depressing PTT or receive information by releasing PTT. The mode selector dial allows the user to choose between the ciphertext or the plaintext mode of operation. The ASVT also has a key port which allows the user to install new keys for the encryption/decryption of information. The terminal must be clear of all voice transmissions before a user may change the status of the control panel or install a new key.

Five applications of the ASVT are possible for both transmission and reception of information. The application in effect at any given time is dependent on the current ASVT configuration. The current configuration is defined by the two external interfaces hooked up, and the positions of the power switch, the PTT button, and the mode selector dial on the control panel. The five ASVT applications are illustrated in Figure 3-2. The Analog User, Digital User, Digital Line 1, and Digital Line 2 applications use the COMSEC module and may only be used while in ciphertext mode. There is also one Plaintext Application which allows the plaintext transmission and reception of information by the ASVT while in plaintext mode. Turning the power switch on brings the ASVT up in the Analog User Application and ready to receive data - that is, with the

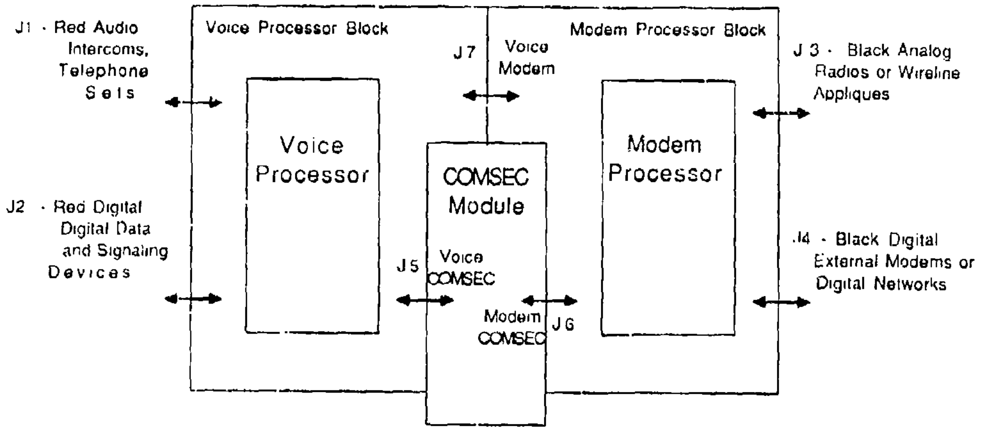
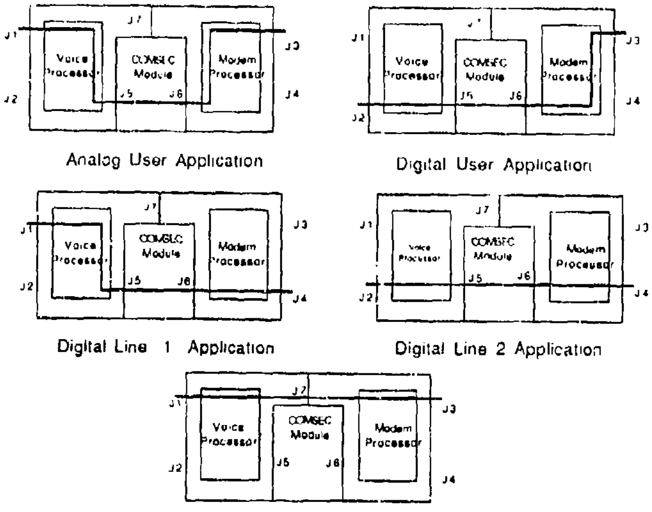


Figure 3-1. ASVT Block Structure



Analog User Application

Digital User Application

Digital Line 1 Application

Digital Line 2 Application

Plaintext Application

Figure 3-2. ASVT Applications

RED_AUDIO and BLACK_ANALOG external interfaces hooked up, the PTT button released, and the mode selector dial set in the ciphertext position.

During transmission, the Voice Processing Block performs Red voice processing functions to analyze voice transmissions, the Modem Processing Block performs Black modem processing functions to code important bits and modulate the resulting bit stream, and the COMSEC Module Block performs the encryption of voice. During reception, these functions are inverted. The position of the PTT button determines the direction of the flow of data for the ASVT in each application. In the ciphertext mode, depressing the PTT button allows Red unencrypted data to enter the Voice Processing Block from either the Red Audio or Red Digital interface. Releasing the PTT button allows Black encrypted data to enter the Modem Processing Block from either the Black Analog or Black Digital interface. The Plaintext Application does not perform any transformation of information input. The details of the transformation each application performs on a message have little or no bearing on the statement of the security policy. For ease of exposition these details are omitted.

Those applications that use the COMSEC module require a key for the encryption and decryption of information. Keys may be inserted into the COMSEC module by the ASVT operator through the key port. Users at two different ASVT terminals may communicate through applications other than plaintext only if they have the same key installed. All key management and distribution for the ASVTs is done manually by ASVT operators: no keys are transmitted automatically between ASVTs. The ASVT is initially configured with a predefined default key.

An operator wanting to transmit data, either voice or digital, on the ASVT for a given application will turn the power switch on, hook up the interface channels for that application, select the ciphertext or plaintext mode according to the application, depress the PTT button and send the ASVT data. If the operator releases PTT during transmission, information may then be received. Likewise, an operator wanting to receive data, either voice or digital, on the ASVT for a given application will turn the power switch on, hook up the interface channels, select the ciphertext or plaintext mode according to the application, and release the PTT button. If the operator depresses PTT during reception, information may then be transmitted. If the terminal is not configured properly for one of the five applications, information may be neither received nor transmitted.

ASVT Security Policy

The security policy for the ASVT is based on the concept of Red/Black separation. It is split into two policies, an external security policy and an internal security policy. The external security policy treats the ASVT as a black box and specifies Red/Black separation on its input and output. More specifically, all information transmitted by the ASVT when in the ciphertext mode of operation must be Black, encrypted, data. Thus, the only way for the ASVT to transmit Red, unencrypted, data is when the mode selector dial is set in the plaintext mode.

The internal security policy enforces Red/Black separation on the components of the ASVT. The Voice Processor Block is the Red processing partition and the Modem Processor Block is the Black processing partition. During transmission, the COMSEC module takes Red data and encrypts it creating Black data. During reception, the COMSEC module takes Black data and decrypts it creating Red data. Furthermore, the only way data may change from Red to Black or Black to Red is by going through the COMSEC module. The internal policy requires that, when in the ciphertext mode, all information transmitted to the modem processor, the Black partition, must be Black, encrypted, data. Thus, the only way for Red data to enter the Black processing partition is when the ASVT is in the plaintext mode of operation.

4. THE CSP LANGUAGE

Process isolation reduces the amount of software that interfaces with critical COMSEC functions. This, in turn, reduces the amount of software within the COMSEC boundary, a primary objective of the assurance task. Process isolation may be accomplished through physical means, by separating processes on distinct processors, or through logical means by separating processes using proper programming techniques. The highest degree of COMSEC assurance is gained by separating processes on distinct processors. Highly reliable COMSEC software, therefore, requires the asynchronous operation of separate processors within the COMSEC system. The CSP process description and specification language [6] was chosen primarily for its capacity to describe the concurrent operation of processes and properties about them in a formal manner. Although a thorough knowledge of CSP is not necessary to understand the specification of the ASVT, a general knowledge is helpful. The rest of this section reviews the terminology and notation that is used in the CSP specification of the ASVT.

A CSP process is the behavior pattern of some object. Each process has an alphabet associated with it defining the set of events relevant to its description. Processes are permitted to communicate through those channels included in their alphabet. The notation used in CSP aids in the description of a process in terms of the events in its alphabet. Unless otherwise specified, the following conventions are used:

1. Symbols in upper case italics letters denote distinct processes, e.g. *A_SECURE_VOICE_TERMINAL*, *MODEM_PROCESSOR*, *SKIP*.
2. Symbols in upper case bold denote specifications, e.g., **CM_RED_BLACK_SEPARATION**, **INCOMING_MK**.
3. Symbols in upper case roman denote distinct channels through which processes may communicate, e.g., RED_AUDIO.
4. Symbols in lower case roman are variables denoting, for example, channels, values of messages sent through channels, function names mapping values to new values, etc. The symbol s, however, is reserved to denote arbitrary sequences of events in which a process may engage, called traces.
5. Symbols in lower case italics denote event names, e.g., *ptt_off*.
6. The alphabet of a process P is denoted αP.

Processes may communicate through channels. "CHANNEL1 ? m" denotes the event in which value m is input on CHANNEL1 and "CHANNEL1 ! m" denotes the event in which value m is output on CHANNEL1. The choice operator " | " allows the behavior of a process to be influenced by outside events. If P and Q are processes and $e1$ and $e2$ are events, the process $(e1 \rightarrow P \mid e2 \rightarrow Q)$ behaves like process P if $e1$ is the first event to occur and behaves like process Q if $e2$ is the first event to occur. The process $P <| \mathrm{b} |> Q$ is defined as P if b else Q. This definition is generalized to allow any arbitrary nesting with binding from left to right if not otherwise specified by parentheses. Thus, $P1 <| \mathrm{b1} |> P2 <| \mathrm{b2} |> P3$ is defined as the process if b1 then $P1$ else if b2 then $P2$ else $P3$.

A trace of a process is a finite sequence of events in which a process has engaged at some moment in time. A trace may be denoted by the letter s or a sequence of events enclosed in angle brackets, e.g., $<event1, event2, event3>$. The first event of a trace s is denoted by s_0 and the result of removing the first symbol is s', e.g., $<a,b,c>_0 = a$ and $<a,b,c>' = <b,c>$.

$P;Q$ signifies a process that acts like the successful termination of P followed by Q. If P does not terminate successfully then Q does not start. The trace of this process is s;t where s is the trace of P an [illegible] is the trace of Q. A process successfully terminates if and only if the process ends with *SKIP*. A process unsuccessfully terminates if and only if the process ends with *STOP*.

Processes may be defined to run concurrently through the " ‖ " operator, e.g., $P \| Q$ is a process that runs process P in parallel with process Q. Such concurrent processes require simultaneous participation of those events that occur in both αP and αQ. Events occurring in P's alphabet but not Q's may be engaged in by P independently of Q. This generalizes to the situation in which many processes run in parallel in the obvious way. A communication can occur between two processes running in parallel if and only if both processes have that communication event in their alphabets and both processes simultaneously engage in that event, i.e., whenever one process outputs a value oato the channel, the other process simultaneously inputs the same value from the channel. Thus,

$$(\mathrm{CH}\,!\,v \rightarrow P) \,\|\, (\mathrm{CH}\,?\,m \rightarrow Q(m)) \;=\; \mathrm{CH}\,.\,v \rightarrow (P \,\|\, Q(v))$$

If only one process in a concurrent combination of processes has the communication event in its alphabet, then that process may engage in the communication event independently of the other processes.

The above notation is used for the procedural description of the behavior of processes. A specification of a process or system is a predicate description of the way it is intended to behave. The specification of processes uses the six conventions listed previously and a notation that is self-explanatory. Notation that may be used in the ASVT specification and procedural description is summarized in Table 4-1 for easy reference.

Notation	Meaning
P & Q	P and Q (both true)
P V Q	P or Q (one or both true)
not P	[illegible]s not true
$P \rightarrow Q$	if P then Q
some x.P	there exists an x such that P
all x.P	for all x. P
P sat S	process P satisfies specification S
{a,b,c}	the set with members a,b,c
$\langle a,b \rangle$	the trace containing the events a, b
$\in$	is a member of
s_0	the head of s
s'	the tail of s
tr	an arbitrary trace of the process
$C\,?\,m$	from channel C input m
$C\,!\,m$	on channel C output value of m
$C.m$	a communication event of m on C
αP	the alphabet of P
$a \rightarrow P$	event a then process P
$(a \rightarrow P \mid b \rightarrow Q)$	a then P choice b then Q
$P \lhd b \rhd Q$	P if b, else Q
$P;Q$	P successfully followed by Q
$P \| Q$	P in parallel with Q
SKIP	a process that does nothing but terminate successfully

Table 4-1. CSP Language Subset Summary

5. CSP SPECIFICATION OF THE ASVT

ASVT security requires enforcing both the external security policy and the internal security policy. This section focuses on the CSP specification of the internal security of the system as described in section 3. The complete CSP specification is presented in [9].

This section specifies an abstract model of security for secure voice terminals containing three components - a voice processor, a modem processor, and a COMSEC module. An interpretation instantiates the abstract model to an ASVT-specific model. A functional specification of the ASVT describes the operation of the ASVT. Finally, an informal argument is given that the COMSEC module conforms to the security policy. In the following exposition, values of functions may be written as sets of ordered pairs, e.g., if f is a function then (a,b) ∈ f if and only if f(a) = b. Specific fields of an n-tuple may be specified by the n-tuple name followed by a period followed by a field name for that n-tuple, e.g., if c = (f1, f2, f3, f4), the f4 field of c is specified c.f4. Table 4-1 summarizes other notation used in the specification.

Internal Abstract Security Model

The ASVT internal security policy requires that the only way for Red data to enter the Modem Processor Block, the Black processing partition, is when the system is in the plaintext mode of operation. Our goal is to state an abstract model of this policy in CSP. Toward this goal, let *VOICE_PROCESSOR*, *COMSEC_MODULE*, and *MODEM_PROCESSOR* be the CSP processes of the voice terminal's voice processor component, COMSEC module component, and modem processor component, respectively. We assume the existence of the following constant sets, functions, and channels :

M is a set of messages.

MS is a set of operating modes such that plaintext ∈ MS.

K is a set of keys. Let initial_key ∈ K be the default initial key.

KCH is the communication channel over which the keys used for encryption and decryption are passed.

VCHS/CCHS/MCHS are sets of communication channels over which messages pass for the voice processor/COMSEC module/modem processor.

CHC is a function from VCHS ⋃ MCHS ⋃ CCHS to {red,black} which describes the type of messages that may be transmitted over CH when not in plaintext mode, e.g., only Black data may be transmitted over channel CH if CHC(CH) = black.

VCS/CCS/MCS are sets of 4-tuples describing the possible configurations for each component. For in_chan, out_chan ∈ VCHS/CCHS/MCHS, mode ∈ MS, push_to_talk ∈ (true,false), (in_chan,out_chan,mode,push_to_talk) signifies that for a particular value of push_to_talk, messages may come in through in_chan, go out through out_chan, and may bypass encryption/decryption if mode = plaintext.

VTS/MTS are sets of functions from M to M describing the set of voice processor/modem processor message transformations. For all T ∈ VTS ⋃ MTS, m ∈ M, T(m) is the message output when the message m is input.

CTS is a set of functions from M x K to M describing the set of COMSEC module transformations. For all T ∈ CTS, (m,k) ∈ M x K, T((m,k)) ∈ M is the message output by the transformation when the message m is input with key k installed.

VCT/CCT/MCT are functions from VCS to VTS/CCS to CTS/MCS to MTS describing the transformation each component configuration performs on messages input. Thus, for all c ∈ VCS/CCS/MCS c performs transformation VCT(c)/CCT(c)/MCT(c) on messages input.

CDP is a function from VTS ⋃ CTS ⋃ MTS to {cipher, decipher, plain} returning the type of transformation performed. Thus, for all T ∈ VTS ⋃ CTS ⋃ MTS, T performs an encryption if CDP(T) is cipher, performs a decryption if CDP(T) is decipher, and performs no encryption or decryption if CDP(T) is plain.

Since the abstract voice terminal and its components can not, in general, decide whether data received is Red or Black, we must rely on the red or black mark given to each channel by the CHC function. The security model must ensure that only encrypted data flows through channels marked black and that only unecrypted data flows through channels marked red. Besides internal communication among the three processes, the voice terminal can communicate with its external environment through an external interface. Since we have no control on the external environment, we assume that, through the external interface, channels marked red receive only Red data and channels marked black receive only Black data . It is the job of the security policy model to describe the necessary data flow restrictions internal to voice terminal given this assumption.

Given the above structure for our abstract voice terminal, a number of requirements for Red/Black separation exist. Assume that the terminal is not in plaintext mode and tl at *VOICE_PROCESSOR* and *MODEM_PROCESSOR* have no way of encrypting or decrypting data. Red/Black separation requires that *MODEM_PROCESSOR* input and output data only through channels marked black. Although *VOICE_PROCESSOR* may receive Red data through its external interface, the security model must ensure that *COMSEC_MODULE* encrypt all data transmitted from *VOICE_PROCESSOR* to *MODEM_PROCESSOR*. Since *COMSEC_MODULE* requires Red data for encryption, *VOICE_PROCESSOR* may input and output data only through channels marked red. But *MODEM_PROCESSOR* may only output through channels marked black. *COMSEC_MODULE* must, therefore, decrypt all data transmitted from *MODEM_PROCESSOR* to *VOICE_PROCESSOR*. Finally, *COMSEC_MODULE* must neither encrypt nor decrypt data coming from and going to the same processor. Although the ASVT components may perform a transformation of messages received, they must not be able to construct new messages.

A trace of a voice terminal component includes all the communication events specifying the messages that have been processed by that component. The specification functions **INCOMING_MSGS** and **OUTGOING_MSGS** return, for trace s, the set of messages input and, respectively, output to channel chan.

```
INCOMING_MSGS (s, chan) =
        if s = ·    ·
        then {}
        else if $s_0$ = (chan ? m)
                then {m} $\bigcup$ INCOMING_MSGS(s',chan)
                else INCOMING_MSGS(s',chan)
```

```
OUTGOING_MSGS (s, chan) =
        if s = ·    ·
        then {}
        else if $s_0$ = (chan ! m)
                then {m} $\bigcup$ OUTGOING_MSGS(s',chan)
                else OUTGOING_MSGS(s',chan)
```

The specification for *VOICE_PROCESSOR*, **VP_RED_BLACK_SEPARATION**, states that all messages that exit the voice processor when not in plaintext mode must be a proper plain transformation of a message that entered the component through a channel marked red and must exit the component through a channel marked red. In addition, all messages must enter the component only through channels marked red.

VOICE_PROCESSOR sat **VP_RED_BLACK_SEPARATION**

```
VP_RED_BLACK_SEPARATION =
  all c.
   (c ∈ VCS
    & not (c.mode = plaintext)
    →
    ((all out_message.
       out_message ∈ OUTGOING_MSGS(tr,c.out_chan))
       →
       some in_message, t.
       (t ∈ VTS
        & VCT(c) = t
        & in_message ∈ INCOMING_MSGS(tr,c.in_chan)
        & t(in_message) = out_message
        & CDP(t) = plain
        & CHC(c.out_chan) = red)
     & (all in_message.
        in_message ∈ INCOMING_MSGS(tr,c.in_chan)
        →
        CHC(c.in_chan) = red))
```

The specification for *MODEM_PROCESSOR*, **MP_RED_BLACK_SEPARATION**, states that all messages that exit the modem processor when not in plaintext mode must be a proper plain transformation of a message which entered the component and must exit through a channel marked black. In addition, all messages must enter the component only through channels marked black.

MODEM_PROCESSOR sat **MP_RED.BLACK_SEPARATION**

```
MP_RED_BLACK_SEPARATION =
  all c.
  (c ∈ MCS
   & not (c.mode = plaintext)
   →
   ((all out_message.
       out_message ∈ OUTGOING_MSGS(tr,c.out_chan)
       →
       some t, in_message.
        (t ∈ MTS
         & MCT(c) = t
         & in_message ∈ INCOMING_MSGS(tr,c.in_chan)
         & t(in_message) = out_message
         & CDP(t) = plain
         & CHC(c.out_chan) = black))
    & (all in_message.
       in_message ∈ INCOMING_MSGS(tr,c.in_chan)
       →
       CHC(c.in_chan) = black))
```

The specification for *COMSEC_MODULE*, **CM_RED_BLACK_SEPARATION**, states that all messages that exit the module when not in plaintext mode must be equivalent to the proper transformation of a message that entered the terminal. Furthermore, if the message entered through a channel marked red and exited through a channel marked black, the transformation must perform an encryption of the message. If the message entered through a channel marked black and exited through a channel marked red, the transformation must perform a decryption of the message. Finally, if the input and output channels are marked the same, the transformation must perform neither an encryption nor a decryption of the message. The specification functions **INCOMING_MK** and **OUTGOING_MK** specify, for trace s, the set of messages input and, respectively, output through channel chan and the key that was installed when each message was input or output.

```
INCOMING_MK (s, chan, key) =
      if s = < >
      then {}
      else if $s_0$ = (KCH ? k)
          then INCOMING_MK(s', chan, k)
          else if $s_0$ = (chan ? m)
              then {(m,key)} ⋃ INCOMING_MK(s',chan,key)
              else INCOMING_MK(s',chan,key)
```

```
OUTGOING_MK (s, chan, key) =
      if s = < >
      then {}
      else if $s_0$ = (KCH ? k)
          ᵗʰen OUTGOING_MK(s', chan, k)
          else if $s_0$ = (chan ! m)
              then {(m,key)} ⋃ OUTGOING_MK(s',chan,key)
              else OUTGOING_MK(s',chan,key)
```

COMSEC_MODULE sat **CM_RED_BLACK_SEPARATION**

```
CM_RED_BLACK_SEPARATION =
  all c, out_message, key.
   (c ∈ CCS
    & not c.mode = plaintext
    & (out_message,key) ∈ OUTGOING_MK(tr,c.out_chan,initial_key))
    →
    some t, in_message.
     (t ∈ TS
      & CCT(c) = t
      & t((in_message,key)) = out_message
      & (in_message,key) ∈ INCOMING_MK(tr,c.in_chan,initial_key)
      & ((CHC(c.in_chan) = red
         & CHC (c.out_chan) = black)
        → CDP(t) = cipher)
      & ((CHC(c.in_chan) = black
         & CHC (c.out_chan) = red)
        → CDP(t) = decipher)
      & (CHC(c.in_chan) = CHC (c.out_chan)
        → CDP(t) = plain)
```

A voice terminal that is described as a concurrent combination of *VOICE_PROCESSOR*, *MODEM_PROCESSOR*, and *COMSEC_MODULE* and is an interpretation of the model described in this section guarantees conformance with the Red/Black separation property.

Internal Abstract Model Interpretation

An ASVT interpretation of the abstract model defined above requires defining the operations, sets and functions of the model in terms appropriate to the the ASVT application as described in section 3. The operations appropriate to the process describing the ASVT, *A_SECURE_VOICE_TERMINAL*, and each of its components are defined by each process's alphabet.

α*A_SECURE_VOICE_TERMINAL*
= {*hook_up*$_{ch1,ch2}$, *select_mode*$_m$, *ptt_on*, *ptt_off*, *get_key*,
got_key, *power_on*, *power_off*, KCH.k, RED_AUDIO.m,
RED_DIGITAL.m, VOICE_MODEM.m, VOICE_COMSEC.m,
MODEM_COMSEC.m, BLACK_ANALOG.m,
BLACK_DIGITAL.m}

α*VOICE_PROCESSOR*
= {*hook_up*$_{ch1,ch2}$, *select_mode*$_m$, *ptt_on*, *ptt_off*, *get_key*,
got_key, *power_off*, RED_AUDIO.m, RED_DIGITAL.m,
VOICE_MODEM.m, VOICE_COMSEC.m}

α*COMSEC_MODULE*
= {*hook_up*$_{ch1,ch2}$, *select_mode*$_m$, *ptt_on*, *ptt_off*, *get_key*,
got_key, *power_off*, KCH.k, VOICE_COMSEC.m,
MODEM_COMSEC.m}

α*MODEM_PROCESSOR*
= {*hook_up*$_{ch1,ch2}$, *select_mode*$_m$, *ptt_on*, *ptt_off*, *get_key*,
got_key, *power_off*, BLACK_ANALOG.m,
BLACK_DIGITAL.m, VOICE_MODEM.m,
MODEM_COMSEC.m}

where

hook_up$_{ch1,ch2}$ connects channel ch1 ∈ {RED_AUDIO, RED_DIGITAL} to channel ch2 ∈ {BLACK_ANALOG, BLACK_DIGITAL} for input and output, provided that such a connection results in a proper configuration for some application of the voice terminal. Otherwise it has no affect.

select_mode$_m$ configures the system to operate in mode m ∈ MS so that only if m = plaintext does the system bypass the encryption/decryption of messages.

ptt_on configures the system so that messages may be transmitted by the user.

ptt_off configures the system so that messages may be received by the user.

get_key notifies the system that a new key is about to be installed.

got_key notifies the system that a new key has been installed.

power_on powers up the system.

power_off shuts down the system.

KCH.k is a communication event of key k over channel KCH.

RED_AUDIO.m, RED_DIGITAL.m, BLACK_ANALOG.m, BLACK_DIGITAL.m, VOICE_MODEM.m, VOICE_COMSEC.m, MODEM_COMSEC.m are communication events of message m over the associated channel.

The sets and functions defined in the abstract model are refined in Figure 5-1.

Functional Specification

A_SECURE_VOICE_TERMINAL is described as three internal communicating processes that execute concurrently - *VOICE_PROCESSOR*, *COMSEC_MODULE*, and *MODEM_PROCESSOR*. The initial configuration of the system is the Analog User Application with the PTT button released. Thus the ASVT is set in ciphertext mode and is ready to receive information. This requires the initial voice processor configuration to be vp_receive_audio, the initial COMSEC module configuration to be cm_receive, and the initial modem processor configuration to be mp_receive_analog. Once the system receives the *power_on* signal, these processes start executing in parallel. This concurrent activity ceases when *power_off* is signaled. The system then shuts down and waits for the *power_on* event to occur.

```
A_SECURE_VOICE_TERMINAL =
      power_on →
         (VOICE_PROCESSOR$_{vp\_receive\_audio}$
            ‖ COMSEC_MODULE$_{cm\_receive,initial\_key}$
            ‖ MODEM_PROCESSOR$_{mp\_receive\_analog}$);
         A_SECURE_VOICE_TERMINAL
```

The component functional specification, given in Figure 5-2, provides a CSP operational description of each component. The occurrence of each event of the component's alphabet causes some action to occur within the component, whether it be the processing of a message, the modification of its current configuration, or synchronization with the other components of the terminal. The effect each event has on the system as a whole is as specified previously. This formal description of ASVT functionality corresponds to the informal description of its behavior presented in section 3. Key to understanding this functional specification is the fact that processes executing in parallel

```
MS = {ciphertext, plaintext}

CHC = {(RED_AUDIO,red), (RED_DIGITAL,red), (BLACK_ANALOG,black), (BLACK_DIGITAL, black),
          (VOICE_MODEM, black), (VOICE_COMSEC, red), (MODEM_COMSEC, black}

VCHS = {RED_AUDIO, RED_DIGITAL, VOICE_MODEM, VOICE_COMSEC}
VCS = {vp_transmit_audio          (* = (RED_AUDIO,VOICE_COMSEC,ciphertext,true) *),
          vp_transmit_digital       (* = (RED_DIGITAL,VOICE_COMSEC,ciphertext,true) *),
          vp_transmit_plaintext     (* = (RED_AUDIO,VOICE_MODEM,plaintext,true) *),
          vp_receive_audio          (* = (VOICE_COMSEC,RED_AUDIO,ciphertext,false) *),
          vp_receive_digital        (* = (VOICE_COMSEC,RED_DIGITAL,ciphertext,false) *),
          vp_receive_plaintext      (* = (VOICE_MODEM,RED_AUDIO,plaintext,false) *)}
VTS = {synthesize, analyze, null}
VCT = {(vp_transmit_audio, analyze), (vp_transmit_digital, null),
          (vp_transmit_plaintext, null), (vp_receive_audio, synthesize),
          (vp_receive_digital, null), (vp_receive_plaintext, null)}

CCHS = {RED_AUDIO, RED_DIGITAL, VOICE_COMSEC, MODEM_COMSEC}
CCS = {cm_transmit        (* = (VOICE_COMSEC,MODEM_COMSEC,ciphertext,true) *).
          cm_receive        (* = (MODEM_COMSEC,VOICE_COMSEC,ciphertext,false) *)}
CTS = {encrypt, decrypt}
CCT = {(cm_transmit, encrypt), (cm_receive, decrypt)}

MCHS = {RED_AUDIO, RED_DIGITAL, VOICE_MODEM, MODEM_COMSEC}
MCS = {mp_transmit_analog   (* = (MODEM_COMSEC,BLACK_ANALOG,ciphertext,true) *),
         mp_transmit_digital       (* = (MODEM_COMSEC,BLACK_DIGITAL,ciphertext,true) *),
         mp_transmit_plaintext  (* = (VOICE_MODEM,BLACK_ANALOG,plaintext,true) *),
         mp_receive_analog       (* = (BLACK_ANALOG,MODEM_COMSEC,ciphertext,false) *).
         mp_receive_digital       (* = (BLACK_DIGITAL,MODEM_COMSEC,ciphertext,false) *),
         mp_receive_plaintext    (* = (BLACK_ANALOG,VOICE_MODEM,plaintext,false) *)}
MTS = {encode_modulate, demodulate_decode, encode, decode, null}
MCT = {(mp_transmit_analog, encode_modulate), (mp_transmit_digital, encode),
          (mp_transmit_plaintext, null), (mp_receive_analog, demodulate_decode),
          (mp_receive_digital, decode), (mp_receive_plaintext, null)}

CDP = {(analyze, plain), (synthesize, plain), (encrypt, cipher),
      (decrypt, decipher), (encode_modulate, plain), (encode, plain),
      (demodulate_decode, plain), (decode, plain), (null, plain)}
```

Figure 5-1. Internal Abstract Model Interpretation

require simultaneous participation of those events shared by the process' alphabets.

ASVT Verification

The internal ASVT security model is the instantiation of the abstract internal security model with the internal model interpretation. Likewise, the external ASVT security model is the instantiation of an abstract external security model with an ASVT abstract model interpretation [14]. Verification of the ASVT requires proving that the ASVT functional specification conforms to both the ASVT external and internal security models. A manual formal verification of the ASVT using the CSP proof rules introduced in [6] is too long and detailed for presentation here. CSP does, however, simplify the task of informally arguing that the ASVT conforms to the security policy described by the models.

An argument that the ASVT conforms to the internal security policy proceeds by induction on the length of the trace of each ASVT component. Rather than prove that all of the components meet their specification, we prove only the most difficult, *COMSEC_MODULE*. The proof of the other two components proceed similarly. Let s be the trace for *COMSEC_MODULE*.

Base Case : Let s = ⟨ ⟩.
  Then all c.
      INCOMING_MK (s, c.out_chan, initial_key) = {}
      and **OUTGOING_MK** (s, c.out_chan, initial_key) = {}
  which implies
      CM_RED_BLACK_SEPARATION = true
  so trivially,
      COMSEC_MODULE sat
            CM_RED_BLACK_SEPARATION. .

Induction Step : (Let @ be the trace append operation.)
  Assume that
      COMSEC_MODULE sat
            CM_RED_BLACK_SEPARATION,
  Prove that
    all cmo ∈ α*COMSEC_MODULE*.
        COMSEC_MODULE sat
            CM_RED_BLACK_SEPARATION$_{s@\langle cmo \rangle}$.

Proof : **CM_RED_BLACK_SEPARATION** specifies that, when not in plaintext mode, for all messages m (1) if m is output from the *COMSEC_MODULE* via some configuration then it is a proper transformation of some message input for that configuration, (2) if m is input through a channel marked black and output through a channel marked red it must be encrypted, (3) if m is input through a channel marked red and output through a channel marked black it must be decrypted, and (4) if m is input and output through channels with the same mark it must be neither encrypted nor decrypted. The only events in α*COMSEC_MODULE* that affect **INCOMING_MK** (or **OUTGOING_MK**) are those that input a message (or output a message) or those that input a new key. Noting that events that input a new key only affect future messages it is trivial to prove that

```
$VOICE\_PROCESSOR_c =$
  $select\_mode_m \rightarrow VOICE\_PROCESSOR_c$
      $\lhd|$ not m $\in$ MS $|\rhd$
          $VOICE\_PROCESSOR_{(c.in\_chan,VOICE\_MODEM,plaintext,c.push\_to\_talk)}$
      $\lhd|$ m = plaintext & c.push_to_talk $|\rhd$
          $VOICE\_PROCESSOR_{(VOICE\_MODEM,c.out\_chan,plaintext,c.push\_to\_talk)}$
      $\lhd|$ m = plaintext & not c.push_to_talk $|\rhd$
          $VOICE\_PROCESSOR_{(c.in\_chan,c.out\_chan,ciphertext,c.push\_to\_talk)}$
      $\lhd|$ c.push_to_talk $|\rhd$
          $VOICE\_PROCESSOR_{(VOICE\_COMSEC,c.out\_chan,ciphertext,c.push\_to\_talk)}$
  | $pll\_on \rightarrow VOICE\_PROCESSOR_c \lhd|$ c.push_to_talk $|\rhd VOICE\_PROCESSOR_{(c.out\_chan,c.in\_chan,c.mode,true)}$
  | $pll\_off \rightarrow VOICE\_PROCESSOR_{(c.out\_chan,c.in\_chan,c.mode,false)} \lhd|$ c.push_to_talk $|\rhd VOICE\_PROCESSOR_c$
  | $hook\_up_{ch1,ch2} \rightarrow VOICE\_PROCESSOR_c$
          $\lhd|$ not (ch1 $\in$ {RED_AUDIO, RED_DIGITAL} & ch2 $\in$ {BLACK_ANALOG, BLACK_DIGITAL})
              $\vee$ (ch1 = RED_DIGITAL & c.mode = plaintext) $\vee$ (ch2 = BLACK_DIGITAL & c.mode = plaintext $|\rhd$
                  $VOICE\_PROCESSOR_{(ch1,VOICE\_COMSEC,c.mode,c.push\_to\_talk)}$
             $\lhd|$ c.push_to_talk & (ch1 = RED_DIGITAL $\vee$ (ch1 = RED_AUDIO & c.mode <> plaintext)) $|\rhd$
                  $VOICE\_PROCESSOR_{(ch1,VOICE\_MODEM,c.mode,c.push\_to\_talk)}$
          $\lhd|$ c.push_to_talk & ch1 = RED_AUDIO & c.mode = plaintext $|\rhd$
                  $VOICE\_PROCESSOR_{(VOICE\_COMSEC,ch1,c.mode,c.push\_to\_talk)}$
          $\lhd|$ not c.push_to_talk & (ch1 = RED_DIGITAL $\vee$ (ch1 = RED_AUDIO & c.mode <> plaintext)) $|\rhd$
                  $VOICE\_PROCESSOR_{(VOICE\_MODEM,ch1,c.mode,c.push\_to\_talk)}$
  | (c.in_chan ? msg $\rightarrow$ c.out_chan ! VCT(c)(msg) $\rightarrow VOICE\_PROCESSOR_c) \lhd|$ c $\in$ VCS $|\rhd VOICE\_PROCESSOR_c$
  | $get\_key \rightarrow got\_key \rightarrow VOICE\_PROCESSOR_c$
  | $power\_off \rightarrow SKIP$

$COMSEC\_MODULE_{c,k} =$
  $select\_mode_m \rightarrow COMSEC\_MODULE_c$
      $\lhd|$ not m $\in$ MS $|\rhd$
          $COMSEC\_MODULE_{(c.in\_chan,c.out\_chan,plaintext,c.push\_to\_talk),k}$
      $\lhd|$ m = plaintext $|\rhd$
          $COMSEC\_MODULE_{(c.in\_chan,c.out\_chan,ciphertext,c.push\_to\_talk),k}$
  | $pll\_on \rightarrow COMSEC\_MODULE_{c,k} \lhd|$ c.push_to_talk $|\rhd COMSEC\_MODULE_{(c.out\_chan,c.in\_chan,c.mode,true),k}$
  | $pll\_off \rightarrow COMSEC\_MODULE_{(c.out\_chan,c.in\_chan,c.mode,false),k} \lhd|$ c.push_to_talk $|\rhd COMSEC\_MODULE_{c,k}$
  | $hook\_up_{ch1,ch2} \rightarrow COMSEC\_MODULE_{c,k}$
  | (c.in_chan ? msg $\rightarrow$ c.out_chan ! CCT(c)(msg,k) $\rightarrow COMSEC\_MODULE_{c,k}) \lhd|$ c $\in$ CCS $|\rhd COMSEC\_MODULE_{c,k}$
  | $get\_key \rightarrow$ KCH ? new_key $\rightarrow got\_key \rightarrow COMSEC\_MODULE_{c,new\_key}$
  | $power\_off \rightarrow SKIP$

$MODEM\_PROCESSOR_c =$
  $select\_mode_m \rightarrow MODEM\_PROCESSOR_c$
          $\lhd|$ not m $\in$ MS $|\rhd$
              $MODEM\_PROCESSOR_{(VOICE\_MODEM,c.out\_chan,plaintext,c.push\_to\_talk)}$
          $\lhd|$ m = plaintext & c.push_to_talk $|\rhd$
              $MODEM\_PROCESSOR_{(c.in\_chan,VOICE\_MODEM,plaintext,c.push\_to\_talk)}$
          $\lhd|$ m = plaintext & not c.push_to_talk $|\rhd$
              $MODEM\_PROCESSOR_{(MODEM\_COMSEC,c.out\_chan,cipher\_text,c.push\_to\_talk)}$
          $\lhd|$ c.push_to_talk $|\rhd$
              $MODEM\_PROCESSOR_{(c.in\_chan,MODEM\_COMSEC,cipher\_text,c.push\_to\_talk)}$
  | $pll\_on \rightarrow MODEM\_PROCESSOR_c \lhd|$ c.push_to_talk $|\rhd MODEM\_PROCESSOR_{(c.out\_chan,c.in\_chan,c.mode,true)}$
  | $pll\_off \rightarrow MODEM\_PROCESSOR_{(c.out\_chan,c.in\_chan,c.mode,false)} \lhd|$ c.push_to_talk $|\rhd MODEM\_PROCESSOR_c$
  | $hook\_up_{ch1,ch2} \rightarrow MODEM\_PROCESSOR_c$
          $\lhd|$ not (ch1 $\in$ {RED_AUDIO, RED_DIGITAL} & ch2 $\in$ {BLACK_ANALOG, BLACK_DIGITAL})
              $\vee$ (ch1 = RED_DIGITAL & c.mode = plaintext) $\vee$ (ch2 = BLACK_DIGITAL & c.mode = plaintext $|\rhd$
                  $MODEM\_PROCESSOR_{(ch2,MODEM\_COMSEC,c.mode,c.push\_to\_talk)}$
          $\lhd|$ not c.push_to_talk & (ch2 = BLACK_DIGITAL $\vee$ (ch2 = BLACK_ANALOG & c.mode <> plaintext)) $|\rhd$
                  $MODEM\_PROCESSOR_{(ch2,VOICE\_MODEM,c.mode,c.push\_to\_talk)}$
          $\lhd|$ not c.push_to_talk & ch2 = BLACK_ANALOG & c.mode = plaintext $|\rhd$
                  $MODEM\_PROCESSOR_{(MODEM\_COMSEC,ch2,c.mode,c.push\_to\_talk)}$
          $\lhd|$ c.push_to_talk & (ch2 = BLACK_DIGITAL $\vee$ (ch2 = BLACK_ANALOG & c.mode <> plaintext)) $|\rhd$
                  $MODEM\_PROCESSOR_{(VOICE\_MODEM,ch2,c.mode,c.push\_to\_talk)}$
  | (c.in_chan ? msg $\rightarrow$ c.out_chan ! MCT(c)(msg) $\rightarrow MODEM\_PROCESSOR_c) \lhd|$ c $\in$ MCS $|\rhd MODEM\_PROCESSOR_c$
  | $get\_key \rightarrow got\_key \rightarrow MODEM\_PROCESSOR_c$
  | $power\_off \rightarrow SKIP$
```

Figure 5-2. Component Functional Specification

```
INCOMING_MK (s@emo, chan, key) =
    if emo = chan?m
    then {m,key} ⋃ INCOMING_MK (s, chan, key)
    else INCOMING_MK (s, chan, key)

OUTGOING_MK (s@emo, chan, key) =
    if emo = chan!m
    then {m,key} ⋃ OUTGOING_MK (s, chan, key)
    else OUTGOING_MK (s, chan, key)
```

By (a3), we need to ensure only that if emo is a communication event then (1), (2), (3), and (4) are true for all possible configurations of *COMSEC_MODULE* in which the mode selector is in the ciphertext position. (1) is true since, in the functional specification of $COMSEC\_MODULE_{e,k}$, before CCT(e)(msg,k) may be output over e.out_chan, e must be an element of CCS and msg must be input over e.in_chan. (2), (3), and (4) are true by inspection of the set CCS and the functions CHC, CCT, and CDP in Figure 5-1. *End of Proof*

An informal argument very similar to this one can be used to convince oneself that the functional specification conforms to the external security policy. These arguments increase our confidence that the ASVT functional specification conforms to its security policy. Although we have not formally verified the ASVT using CSP, initial results using the Gypsy verification system [5] suggest that it is, in fact, formally verifiable.

6. CONCLUSIONS

The increased use of software in the development of COMSEC equipment requires techniques to assure that COMSEC software meets its security requirements. Formal specification and verification provides a promising approach to meet the security assurance needs of COMSEC software. The practical application of this approach requires an analysis of existing formal specification and verification techniques to determine their suitability for verifying that COMSEC software meets its security requirements. Rather than evaluating verification systems in general, we have taken the approach of comparing verification systems for a specific class of applications and class of properties to be proved. A nontrivial COMSEC system, representing the class of applications, and a security policy for the system, representing the class of properties, has been formally specified in CSP. The CSP specification of the ASVT demonstrates the feasibility of applying formal specification techniques to COMSEC system designs and provides a vehicle from which to study and compare systems for verifying COMSEC software.

Many computer security and verification experts agree that a system's security policy need be stated only in terms of the interface between the system and its external environment [4,8]. No reference to the internal structure of the system should be necessary. While computer security policies typically state requirements on the flow of information between the device and the user, COMSEC security policies typically state requirements on the flow of information between devices or processes. Specification of COMSEC security, therefore, often does require reference to the internal structure of the system. For instance, the ASVT security policy requires control on the information flow from the voice processor to the modem processor. Specification of this property requires knowledge of the internal structure of the ASVT. COMSEC systems, in general, must enforce both an external security policy, stating restrictions on information flows between the device and the outside world, and an internal security policy, stating restrictions on information flows between the processes internal to the device.

Toward the goal of assuring COMSEC software security, it appears that formally verifying that a nontrivial CSP process description meets its specification is too cumbersome to carry through without automated assistance. A formal proof that the ASVT conforms to its security policy explodes into a great number of cases that could be handled with speed and accuracy using a mechanical verification system. Since our goal is to survey existing verification techniques, rather than engineer new ones, we leave formal verification up to existing automated techniques. Nevertheless, there are a number of significant advantages that have arisen from using CSP in this project. During the formulation of the ASVT example, CSP provided insight into aspects of the asynchronous communication of processes within the ASVT that were not immediately obvious from its informal description. The language also provided a concise medium for review of ASVT functionality and security policy. The CSP specification allowed us to informally convince ourselves that the ASVT process description conforms to its security policy.

An important lesson learned while trying to formalize the ASVT in CSP is that the strength of CSP lies in its fundamental concepts and operations. Hoare's book presents the basic concepts of the CSP theory in an intuitive manner and then builds on the theory by defining new operations from previously defined operations. We quickly became overwhelmed by the wealth of notation contained in his book. After floundering for awhile, we discovered that a combination of set theory, arithmetic, and the most fundamental concepts and operations of CSP is sufficient to formally specify a broad range of systems. With this realization progress came more readily.

Future objectives of this project are to investigate existing automated formal verification techniques and to demonstrate and document the use of these techniques by verifying that the ASVT satisfies its security policy. The CSP specification of the ASVT serves as a concise description from which different teams may pursue the formal verification of a system in parallel, with a high degree of confidence that they are all working from a common understanding of the problem. We have already begun specifying the ASVT using Gypsy [5], mEVES [1], and FDM [7]. These verification systems will be compared for their ability to promote COMSEC software security.

ACKNOWLEDGEMENTS

The author wishes to thank the Space and Naval Warfare Systems Command for their support of the work reported here. Charles Payne of NRL helped refine the ASVT description and gave many suggestions which improved the presentation of this paper. Carl Landwehr provided the inspiration for writing this paper as well as many helpful comments throughout its development. I would also like to thank all the participants of the NRL CSP seminar, especially John McDermott, Mark Cornwell, and William Spears, as well as the reviewers of this paper.

REFERENCES

[1] Cragen, D., et. al., "m-EVES: A Tool for Verifying Software," I.P. Sharp Associates Limited Technical Report CP-87-5402-26, October 8, 1987.

[2] *Department of Defense Trusted Computer system Evaluation Criteria*, December, 1985.

[3] Garner, Jack, Tim McChesney, and Michael Glancy, "Advanced Narrowband Digital Voice Terminal," *SIGNAL Magazine*, November 1982.

[4] Good, Donald I., "Structuring a System for A1 Certification," Internal Note #145, Institute for Computing Science, University of Texas at Austin, Texas, January 1984.

[5] Good, Donald I., Robert L. Akers, Lawrence M. Smith, "Report on Gypsy 2.05," Revision of Technical Report #48, Computational Logic, Inc., October 2, 1986.

[6] Hoare, C.A.R, **Communicating Sequential Processes**, Prentice-Hall International, Englewood Cliffs, New Jersey, 1985.

[7] Kemmerer, R.A., "FDM - A Specification and Verification Methodology," System Development Corporation, Santa Monica, Calif., SP-4088, November 1980.

[8] Kemmerer, R.A., "Verification Assessment Study Final Report," S-228, 204, National Computer Security Center, March 1986.

[9] Moore, Andrew P., "Specification of an Example of COMSEC Software: The ASVT," NRL Technical Memorandum 5590-121.APM:apm.

[10] Voydock, Victor L. and Kent, Stephen T., "Security Mechanisms in High-Level Network Protocols," Advances in Computer System Security, Volume 2, edited by Rein Turn, 1981.

Adding Ada[1] Program Verification Capability to the State Delta Verification System (SDVS)[2]

David F. Martin and Jeffrey V. Cook

Computer Science Laboratory
The Aerospace Corporation
P. O. Box 92957
Los Angeles, CA 90009-2957

Abstract

An approach to, and progress toward, the addition of Ada program verification capability to the State Delta Verification System (SDVS) are presented. In the past, SDVS has been used extensively to verify the microprogrammed implementation of computer instruction sets written in the computer description language ISPS. The generality and modularity of SDVS permit its convenient extension to other programming languages and verification applications. A plan for the incremental adaptation of SDVS to a series of Ada subsets of increasing semantic complexity is outlined. The simplest of these Ada subsets is called Core Ada. Interfacing a new language to SDVS requires constructing a translator from that language into state deltas, which are the semantic basis of SDVS. A general strategy for constructing such translators via their denotational semantic specification and a straightforward manual translation of the specification into a Common Lisp program are presented, and the successful application of this strategy to Core Ada is reported. Future work on Core Ada and more complex Ada subsets is discussed.

1 Introduction

This paper describes an approach to adding Ada program verification capability to the State Delta Verification System (SDVS). SDVS was developed in the Computer Science Laboratory of The Aerospace Corporation and a production-quality version for use in microprogram verification has recently been completed. That version of SDVS is specialized for the verification of the microprogrammed implementation, written in the computer description language ISPS, of computer instruction sets. SDVS is general and modular enough to be adapted, with a reasonable amount of effort, to other verification applications and programming languages. Our plan for, and progress toward, adapting SDVS to Ada are presented.

SDVS, state deltas, and the translation of Ada statements into state deltas are briefly described in Section 2. Next, our plan for adapting SDVS to Ada verification is presented in Section 3. Central to this plan is the incremental adaptation of SDVS to a series of Ada subsets of increasing semantic complexity. An overview of the simplest of these subsets, Core Ada, is given in Section 4. Section 5 presents our approach to the design, specification, and rapid implementation of the SDVS Ada translators, a discussion of our experience with the Core Ada translator, and required additions to the SDVS inference engine. Finally, in Section 6 we discuss the research and implementation issues relevant to adapting SDVS beyond Core Ada, future prospects, and conclusions.

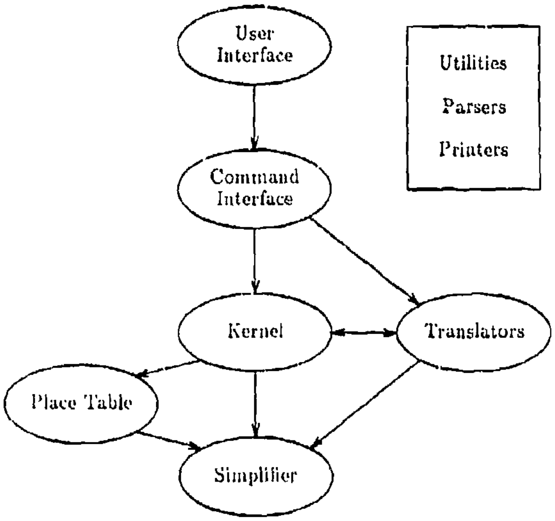


Figure 1: Structure of the State Delta Verification System.

2 The State Delta Verification System (SDVS)

A good general introduction to SDVS is given in [1], even though some information specific to an older version of SDVS is found there. Some of the following description of SDVS is taken from [2], where much more detail about SDVS can be found. The structure of SDVS is shown in Figure 1. It is highly modular; the components of most interest to us are the *simplifier* and the *translators*, as they are the components most affected by the enhancement of SDVS's verification capability as a result of the addition of new programming languages.

SDVS is a system for checking proofs about the course of a computation. SDVS is based on a specialized form of temporal logic whose formulas are called *state deltas*. Technically, SDVS checks proofs of state deltas, which provide an operational semantic representation of computation. SDVS can handle proofs of claims of the form "if P is true now then Q will become true in the future". If P is a program (perhaps with some initial assertions) and Q is an output assertion, then the above claim is an input-output assertion about P. SDVS can also handle claims of the form "if P is true now then Q is true now". In this case if P is a program and Q is a specification then the

[1] Ada is a registered trademark of the U. S. Government — Ada Joint Program Office.

[2] This research was supported by the National Computer Security Center under contract FO 4701-85-C-0086.

claim asserts the total correctness of P with respect to Q. SDVS is also capable of handling proofs that one program correctly implements another, i.e., *multilevel* correctness proofs.

Central to SDVS's behavioral model are *places* (which can be viewed as abstract memory locations or even program variables) that contain (abstract) *values*, the places' "contents". In its simplest form, a *state* (of a computation or a machine) is an association of places with their contents. SDVS's *place table* records a state.

A *computation* is specified by a state delta or set of state deltas as a sequence of state changes. The application of state deltas to effect state changes is a form of *symbolic execution*. One can facilitate compact descriptions of computations by including in state deltas a specification of which places potentially receive new contents. Suppose that a state delta induces a change from a state about which P is true to another about which Q is true. Then assertions true in the P-state about the contents of places whose contents do *not* change are still true in the Q-state. If it is specified that *no* places can change their contents as the state changes from P to Q, then Q must be true in the state satisfying P; this is simply the *static* assertion that P implies Q.

The user communicates with SDVS through several languages. The *user interface language* is used for interactive proof construction. The *proof language* is used to write a proof for the system to check. The *state delta language* is used to express claims to be proven and to describe the relevant programs and specifications. Finally, the *programming languages* (currently ISPS and Core Ada) are used to express the computational objects to be verified. The *translators* function as SDVS's interface to these languages by translating them into the state delta language.

The proof language consists of two parts: *static* and *dynamic*. The static part deals with proving that certain assumptions imply certain conclusions about a given state. For simple theories the SDVS *simplifier* can automatically prove these claims without any user assistance. The dynamic part controls the state transitions made by the system; it contains constructs for proof by symbolic execution for straight-line code, proof by cases for conditional branches, and proof by induction for loops.

2.1 State Deltas

Let $\mathcal{L}$ be a first-order logic that includes constant symbols called *places*. $\mathcal{L}$ is two-sorted (with sorts **place** and **domain**); the first sort denotes the places and the other denotes the domain of values contained by the places. Let $\mathcal{A}$ be a two-sorted model for $\mathcal{L}$ in which the places are interpreted as themselves. $\mathcal{A}$ is called a *base model* for $\mathcal{L}$. Further, assume that $\mathcal{L}$ is augmented with names (interpreted as themselves) for all **domain**-elements of $\mathcal{A}$. This provides convenient support for using $\mathcal{L}$ in applications, such as SDVS, involving symbolic execution. Strictly speaking, the augmented language ought to be called $\mathcal{L}_{\mathcal{A}}$, but explicit mention of $\mathcal{A}$ will often be omitted in the sequel.

Let there be two function symbols, . ("dot") and # ("pound") of type **place** → **domain**, not in $\mathcal{L}$. Let $\mathcal{L}(.)$ and $\mathcal{L}(.,\#)$ denote, respectively, $\mathcal{L}$ augmented with dot, and $\mathcal{L}$ augmented with dot and pound. Dot and pound are interpreted in $\mathcal{L}(.)$- and $\mathcal{L}(.,\#)$-structures as functions that yield the contents of a place.

A *state delta* (SD) is a special formula of the form `[SD` P, C, M, Q`]` where

- P, the SD's *precondition*, is a boolean combination of $\mathcal{L}(.)$-formulas and state deltas;
- C and M, the SD's *comodification* and *modification lists*, respectively, are lists (denoting sets) of places;
- Q, the SD's *postcondition*, is a boolean combination of $\mathcal{L}(.,\#)$-formulas and state deltas.

The modification and comodification lists are often called mod and comod lists for short. Note that this syntactic specification of SDs is recursive; the basis for the construction of SDs consists of those SDs whose pre- and postconditions contain no SDs. SDs are also displayed as `[SD pre:` P `comod:` C `mod:` M `post:` Q`]`. Pre- and postconditions are often represented by lists of formulas; the list represents the logical conjunction of its elements. The *state delta language* $\mathcal{L}_{SD}$ is the set of all boolean combinations of $\mathcal{L}(.)$-formulas and state deltas.

State deltas are the formulas of a variant of temporal logic, interpreted as follows. A *computational model* for $\mathcal{L}_{\mathcal{A},SD}$ is a pair $\mathcal{M} = \langle T, \{\sigma_t\}_{t \in T}\rangle$ where

- T is a totally ordered set (representing time) with a least element; and
- for each $t \in T$, $\sigma_t : \mathcal{A}_{\text{place}} \rightarrow \mathcal{A}_{\text{domain}}$ is a function that gives the contents of each place at time t.

T is called the *time line* of $\mathcal{M}$. For any $t \in T$, σ_t is called a *state* of $\mathcal{M}$, and for any place p, $\sigma_t(p^{\mathcal{A}}) = \sigma_t(p)$ denotes the contents of p at time t. Note that we can write $p^{\mathcal{A}}$ simply as p, since places are interpreted in $\mathcal{A}$ as themselves.

The truth value of $\mathcal{L}_{\mathcal{A},SD}$-formulas is defined as follows. Let $\mathcal{M} = \langle T, \{\sigma_t\}_{t \in T}\rangle$ be a computational model for $\mathcal{L}_{\mathcal{A},SD}$. A base model $\mathcal{A}$ can be extended in two fundamental ways:

- for each $t \in T$, $\mathcal{A}_t$ denotes an $\mathcal{L}(.)$-model extending $\mathcal{A}$ such that $.^{\mathcal{A}_t} = \sigma_t$;
- for each $t_1 \leq t_2 \in T$, $\mathcal{A}_{t_1,t_2}$ denotes an $\mathcal{L}(.,\#)$-model extending $\mathcal{A}$ such that $.^{\mathcal{A}_{t_1,t_2}} = \sigma_{t_1}$ and $\#^{\mathcal{A}_{t_1,t_2}} = \sigma_{t_2}$.

Let $t_1 \leq t_2 \in T$ and let $[t_1, t_2]$ denote the corresponding closed (time) interval. A place p is *T-preserved over* $[t_1, t_2]$ if for all $t_1 \leq t, t' \leq t_2, \sigma_t(p) = \sigma_{t'}(p)$.

Let ϕ be an $\mathcal{L}_{SD}$-formula and $\mathcal{M}$ a computational model. Let $t_0, t_1, t_2 \in T$, where $t_0 \leq t_1 \leq t_2$. We now define what it means for $\mathcal{M}$ to *satisfy* ϕ *at* $t_0 \in T$, denoted $\models_{\mathcal{M},t_0} \phi$. (To do this, an auxiliary notion of satisfaction, denoted $\models_{\mathcal{M},t_1,t_2} \phi$, must be mutually defined.) Thus,

1. if ϕ is an $\mathcal{L}(.)$-formula, then $\models_{\mathcal{M},t_0} \phi$ iff $\models_{\mathcal{A}_{t_0}} \phi$;
2. if ϕ is an $\mathcal{L}(.,\#)$-formula, then $\models_{\mathcal{M},t_1,t_2} \phi$ iff $\models_{\mathcal{A}_{t_1,t_2}} \phi$;
3. if ϕ is a state delta `[SD` P, C, M, Q`]`, then $\models_{\mathcal{M},t_0} \phi$ iff for every $t_1 \geq t_0$ such that $\models_{\mathcal{M},t_1} P$ and all places $p \in C$ are T-preserved over $[t_0, t_1]$, there exists $t_2 \geq t_1$ such that $\models_{\mathcal{M},t_1,t_2} Q$ and all places $p \notin M$ are T-preserved over $[t_1, t_2]$;
4. $\models_{\mathcal{M},t_1,t_2}$ `[SD` P, C, M, Q`]` iff $\models_{\mathcal{M},t_2}$ `[SD` P, C, M, Q`]`;
5. for an arbitrary $\mathcal{L}_{SD}$-formula, $\models_{\mathcal{M},t_0}$ and $\models_{\mathcal{M},t_1,t_2}$ are extended over the boolean connectives in the standard way.

An SD is a description of a transition from one computation state to another. Its precondition describes a state from which the transition can be made, and its postcondition describes the state resulting from the transition. The times t_1 and t_2 above are called the SD's *precondition* and *postcondition time*, respectively. An SD's mod list M specifies those places whose contents are allowed to change between precondition and postcondition time as a result of the transition. The truth value of any assertion about these places *cannot be assumed to be preserved* during the transition. The contents of places not listed in the mod list *must remain unchanged* during the state transition. The quantification "there exists $t_2 \geq t_1$" in clause 3 above asserts that the entity described by the SD performs its described state transition in a *finite* amount of time. Thus SDs assert the *total correctness* (in the Floyd-Hoare sense) of programs whose transitional behavior they characterize with respect to the SD pre- and postconditions (together with the implicit assertions that the places in SD mod lists preserve their contents across the associated state transitions).

The role of an SD's comod list C is more subtle and is best explained within the context of the SDVS system. SDVS assumes the existence of a computational model, and maintains a *current time t* and an associated *current state σ_t*. A true SD is *applicable* in the current state if its precondition is true in the current state. If several SDs are applicable at a given time, then the prover has the option of selecting which of these SDs to apply. When an SD is applied, the state transition that it describes occurs and its postcondition becomes true. The current time is advanced to that SD's postcondition time and the state of the computation is updated. It is important to note the following facts:

- As long as the contents of all places in its comod list remain unchanged, a true SD will remain true and thus applicable any time its precondition is true.
- If the contents of any place in a true SD's comod list have possibly been changed, then the truth of that SD is not necessarily preserved, i.e., its truth becomes *indeterminate*. Since such an SD is not guaranteed to be true, it is no longer applicable even if its precondition is true.

The foregoing definition of state deltas is essentially that of [3]. However, the SDs actually implemented in the SDVS system are more general. In particular, SDs and their pre- and postconditions can contain quantifications of individual variables of both sorts **place** and **domain**. Quantifications of **domain**-variables are used in program specifications. The existential quantification of **place**-variables is used in SDs produced by the translation of Ada block statements to represent the introduction and subsequent deletion of program variables upon block entry and exit, respectively. Also, the operators *dot* and *pound* map places to **place**-values as well as to **domain**-values. The precise definition and investigation of the properties of general SDs are subjects of ongoing research.

In the SDVS system, SDs known to be true are called *usable*; otherwise (e.g., when their truth is indeterminate), they are not usable and SDVS deletes them as candidates for application. The operation of applying of SDs can be used to execute a computation. If a true SD is applied and has other SDs in its postcondition, then these latter SDs become true and can themselves be applied to further advance the computation. To execute purely sequential computations, it is sufficient to require that (1) when an SD is applied its truth *always* becomes indeterminate, rendering it inapplicable, and (2) the truth of any other SDs that were true at the same time that the applied SD was true also becomes indeterminate. Let **ALL** be a special place denoting (the list of) *all* places. Thu any usable SD whose comod list contains **ALL** and whose mod list is nonempty becomes unusable when it or any other usable SD is applied. To this end, the comod list of *every* generated SD is (**ALL**) and the mod list of *every* generated SD is made nonempty by the inclusion in it of a unique place pc that denotes an (implicit) *program counter*. Thus the truth of any usable SD will become indeterminate when it or any other SD is applied.

2.1.1 Translating Core Ada Statements Into State Deltas

Several examples of the translation of Core Ada statements into state deltas will be given in this section. Provided that the value of A is greater than zero beforehand, the conditional execution of the assignment statement `A := A + 1` is translated into the following SD:

```
[SD pre:   (.A GT 0)
    comod: (ALL)
    mod:   (pc, A)
    post:  (#A = .A + 1)]
```

Because the assignment changes A's contents, this SD would be *inconsistent* if A were not in its mod list. Once this SD is applied, it becomes unusable.

Subject to the condition `B[I] < 10`, the assignment `B[I] := B[I] + 2` is translated into

```
[SD pre:   (.B[.I] LT 10, .I GE .B\FIRST, .I LE .B\LAST)
    comod: (ALL)
    mod:   (pc, B[.I])
    post:  (#B[.I] = .B[.I] + 2)]
```

This example indicates that places can be *structured*, with components that are themselves places. In this case the place B has array structure and is indexable. The places B\FIRST and B\LAST contain the current value of B's lower and upper index bounds. The formulas `.I GE .B\FIRST` and `.I LE .B\LAST` in the precondition of this SD act as *guards* against the raising of a CONSTRAINT_ERROR exception, by making this SD inapplicable if either is false. Note that only the I-th element of B, rather than the entire array B, is in the SD's mod list, because only that element of B is modified in this state transition.

The above examples illustrate translation of statements "out of context" in the sense that the larger program in which they are embedded is not considered. The following examples will take context into account, thereby more clearly underscoring the *incremental* nature of the translation. The conditional statement **if** `B[I] > 0` **then** `A := 0` **else** `A := 1` **end if** is translated into *two* SDs:

```
[SD pre:   (.B[.I] GT 0, .I GE .B\FIRST, .I LE .B\LAST)
    comod: (ALL)
    mod:   (pc)
    post:  ( cont1 )]

[SD pre:   (~(.B[.I] GT 0), .I GE .B\FIRST, .I LE .B\LAST)
    comod: (ALL)
    mod:   (pc)
    post:  ( cont2 )]
```

where "~" denotes logical negation, and cont1 and cont2 are *continuations* that *generate* the translations of the assignment statements

A := 0 and A := 1, respectively. It is important to emphasize that **cont1** and **cont2**, despite their appearance in the postconditions of SDs, are *not* formulas themselves, but rather are *continuations* that *generate* formulas (containing SDs that thereby become usable) that those continuations *represent*. This is the sense in which the SDVS translators are *incremental*; additional translation of a portion of a program occurs only when that portion is "reached" by the application of an SD and the additional translation is initiated by the corresponding continuation contained in the postcondition of the applied SD. If both of the above SDs are applicable and the first of them is applied, its postcondition, which contains **cont1**, becomes true. Thereafter, *both* of the above SDs become unusable because of the intersection of (ALL) and (pc). A similar statement can be made when the roles of the above SDs are reversed. The treatment of the above two SDs requires a *proof by cases* in an associated SDVS proof.

The statements while A > 0 loop *loop-body* end loop ; *next-statement* are translated into the following SDs:

```
sd1 = [SD   pre:    (.A GT 0)
            comod:  (ALL)
            mod:    (pc)
            post:   ( cont1 )]

sd2 = [SD   pre:    (~(.A GT 0))
            comod:  (ALL)
            mod:    (pc)
            post:   ( cont2 )]
```

These SDs represent the two possible outcomes of evaluating the loop test A > 0. sd2 allows the loop to be normally terminated; the continuation cont2 initiates the the translation of *next-statement*. sd1 represents the case where *loop-body* is executed followed by another evaluation of the loop test to determine whether further loop iterations are required. The continuation cont1 represents the translation of *loop-body*, which may be a complex statement. The postconditions of the ultimate SDs in the translation of *loop-body* will contain continuations that regenerate **sd1** and **sd2**; in this sense **sd1** is *recursively defined*. The appearance of such recursively defined SDs requires the use of *induction* in associated SDVS proofs.

The block statement **declare** x: integer; **begin** *body* **end**; is translated into the existentially quantified SD

```
sd1 = (∃x)[SD (TRUE)
              (ALL)
              (pc)
              (ALLDISJOINT(pname, .pname, x), sd2)]
```

where

```
sd2 = [SD (TRUE)
          (ALL)
          (pc, pname, x)
          (COVERING(#pname, .pname, x), cont1)]
```

Application of the state delta **sd2** elaborates the declaration x: integer; . The continuation **cont1** represents the translation of *body*; the postconditions of the ultimate SDs of this translation contain continuations that represent

```
sd3 = [SD (TRUE)
          (ALL)
          (pc, pname, x)
          (COVERING(.pname, #pname, x), cont2)]
```

cont2 continues incremental translation and symbolic execution to the block's successor. The block statement introduces new variables into an Ada program, and the corresponding SD introduces new associated places. When this block is entered during execution, the declaration of the program variable *x* is elaborated. **sd1** asserts the existence of a new place **x** by the suitable quantification of **sd1**; this new place **x** is added to the program's universe of places, represented by the unique place **pname**. **sd1**'s postcondition then asserts that the universe of places **pname**, the places that it contains (denoted **.pname**), and the new place **x** are all mutually disjoint. **sd2** then indicates in its mod list that the contents of **pname** and **x** have changed, the former because of its annexation of **x**, and the latter through implicit initialization. **sd2**'s postcondition asserts that the new contents (**#pname**) of the universal place consists of the disjoint union of its old contents (**.pname**) and the new place **x**. (The reader is reminded at this point that all newly introduced place names are fully qualified by the path in the tree-structured environment that leads to the program unit in which they are introduced.) This prevents any troublesome equality of place names. The application of sd3 "reverses" the introduction of the new place **x**. This reversal is an event that occurs upon block exit. To accomplish this, **sd3**'s mod list indicates that the contents of **pname** and **x** have (again) changed, and its postcondition withdraws **x** from the program's universe of places by asserting that the "old" universe **.pname** (containing **x**) is the disjoint union of the "new" universe **#pname** (with **x** withdrawn) and the withdrawn place **x**.

The above descriptions are not a complete portrayal of the translation of loop and block statements. The potential occurrence of exit statements in the bodies of loops and blocks must be accommodated. This is done by building an *execution stack* which upon loop and block entry receives a corresponding element that can be invoked to achieve the proper completion of the execution of the statement. Loop statements are of course the only statements that can be explicitly completed by an exit statement, but loops can contain blocks that must also be completed prior to the exiting of the loops that contain them. The execution stack elements corresponding to blocks are invoked to withdraw, via the process described above, places introduced at block entry. Such elements are invoked if their blocks are left via an exit statement, in order to effect the necessary adjustment of the program's universe of places. Upon loop and block completion, either via an exit statement or otherwise, the corresponding element is popped from the execution stack. If an exit statement leaves several intervening loops and blocks, then each is left in turn, innermost outward, and the execution stack is appropriately popped as this process proceeds. The precise details of this process are too complex to be presented here; see [4].

3 Adding Ada to SDVS

SDVS has been successfully used to verify that ISPS programs are correct with respect to state delta specifications of their behavior and to demonstrate an implementation relation between two ISPS programs. Given these successes, it is reasonable to investigate next the applicability of SDVS to other programming languages, in particular, to Ada [5].

The verification of Ada programs will be a new application area for state deltas and SDVS. Prior work on Ada verification has been done elsewhere [6,7]. Our additional contribution to this body of research will be to apply SDVS's unique features to (a) prove the *total* correctness of Ada programs with respect to specifications, and (b) do multilevel verification of Ada programs. Two such levels of verification could be (1) proving the correctness of an Ada program with respect to a high-level specification, and (2) proving that the microprogrammed implementation of the compiled Ada program is a correct implementation of the Ada source program.

3.1 Ada Verification in SDVS

In order to apply SDVS to the verification of Ada programs, there are several research and implementation issues that must be addressed.

3.1.1 Research Issues

The principal research issues that must be addressed in adapting SDVS to the verification of Ada programs are (1) defining the semantics of Ada (more precisely, the subsets of Ada in which verifiable Ada programs will be written), and (2) augmenting the SDVS simplifier and data-type theory repertoire with components necessary to support the Ada language.

Our approach is to specify and implement translators from verification-oriented subsets of Ada into state deltas. Moreover, the translator specification should be precise and straightforwardly transformable into a corresponding implementation. This will provide not only an implementation-oriented formal specification of the Ada translator, but also a formal semantics of the Ada subsets in terms of state deltas. This specification will also provide the precise and accessible documentation necessary for the construction of a reliable Ada verification system. An initial experiment toward these goals was our formal specification of the translation of ISPS into state deltas [8]. The associated research issues are of course writing the translator specification and, more fundamentally, selecting verification-oriented subsets of Ada to use. These subsets are discussed in more detail below.

Proving the correctness of a program written in a high level language must deal not only with the program's flow of control, but also with its data types. The above-mentioned translator deals with control flow, but SDVS's inference mechanism (its simplifier and data type theories) must deal with Ada data types. The research issue here is how to determine which existing components of SDVS can be directly adapted to deal with Ada data types, and which enhancements to SDVS's inference machinery must be made.

3.1.2 Implementation Issues

The two principal implementation issues that must be addressed are (1) the construction of a translator from our verification-oriented subsets of Ada to state deltas and (2) the incorporation of Ada-specific enhancements into the SDVS simplifier and inference machinery. We will now discuss these issues in more detail.

The translator that currently exists in the SDVS system transforms ISPS programs into their state delta equivalents, by means of two principal components. First a *parser* parses ISPS programs according to a concrete syntax and constructs corresponding abstract syntax trees. Subsequently, these abstract syntax trees are processed by a *translation program* that generates state deltas. The generation of state deltas can be done in two ways: *incrementally*, in concert with the step by step symbolic execution of the translated ISPS program, or from *markpoint to markpoint* (where a "markpoint" is a program point marked by a label), in which the aforementioned incremental translation steps that occur between markpoints are composed into a single equivalent state delta.

The current ISPS translator evolved in an *ad hoc* manner over a period of several years into its present form. In order to document better the existing translator, as well as to establish a systematic methodology for defining and implementing translators from other languages to state deltas, a formal description of the current ISPS translator was written [8]. This formal description was intended to be both a reasonably precise specification and one that would be a guide to the implementer. The former goal was certainly met by the specification, but whether the latter goal was met has yet to be ascertained by the actual construction of a translator under the guidance of the specification. We are convinced that such translator specifications lead to the production of well-documented, correct, and easily maintained translators.

3.2 Incremental Development

The process of adapting SDVS to the verification of Ada programs will be done *incrementally*. The stages of development will be keyed to the identification of Ada language subsets of increasing semantic complexity; each stage will require the implementation of corresponding capabilities in the Ada-to-state-delta translator and the SDVS simplifier and inference machinery.

3.2.1 Ada Language Subsets

Rather than trying to deal all at once with a full verification-oriented subset of Ada, it is more appropriate to proceed by degrees by selecting increasingly complex Ada language subsets to incorporate into SDVS. This incremental approach will lead to a more orderly adaptation of SDVS to a new programming language. Our preliminary hierarchy of Ada subsets consists of the following:

Core Ada: Core Ada is a subset of Ada intended to facilitate the initial adaptation of SDVS to Ada. This core language will consist of assignment statements and simple expression evaluation; straight-line program flow; branching (**if**, **case**), iteration (**loop**), and escape (**exit**) statements; simple input and output (via the GET and PUT procedures); block structure, scoping, and variable declarations; simple packages containing only variable declarations and other simple packages; **use** clauses; and basic data types (integer, boolean, array).

Stage 1 Ada: This next layer adds subprogram declarations, subprogram calls, and record and enumeration data types to Core Ada.

Stage 2 Ada: Next added are user-defined data types, and some higher-level data types from the set {characters, strings, fixed-point numbers, floating-point numbers, access types (pointers)}

Stage 3 Ada: This final layer added to the Ada subsets includes advanced features that will require considerable research. Included are exception handling, overloading, generics, real-time features, and tasking.

This identification of Ada subsets is preliminary. Core Ada and Stage 1 Ada pose no serious technical obstacles that would prevent them from being interfaced with SDVS; the required technology is mature and well known. In Stage 2 Ada, access types (typed pointers) pose the greatest technical challenge. Pointers create problems with *aliasing* (apparently distinct names that nevertheless identify the same object), but some research has been done on how to verify programs that contain Pascal pointers [9]. Finally, Stage 3 Ada constitutes a set of research topics, as it is not presently clear how to interface these Ada language features to SDVS. As our research progresses, the difficulty inherent in adding certain features will become more apparent, particularly in Stage 3 Ada. Determining when enough Ada language features have been incorporated into SDVS depends upon how difficult it is to include them and also upon how difficult it is to prove the correctness of programs that use those features.

3.2.2 SDVS Inference Machinery

Only modest modifications of and additions to the SDVS inference machinery are required to accommodate Core Ada. Most of these affect the simplifier. Core Ada data types are *integer*, *boolean*, and one-dimensional arrays of these types. Explicit declarations of these types must be introduced; currently only *bit string* declarations (ISPS's only data type) exist in the system. The SDVS simplifier already contains most of the logical axiomatization relevant to these data types. For booleans, knowledge must be added about the **xor** (exclusive or) operator and about Ada's ordering on the booleans (in which false < true). For integers, knowledge must be added about the operators /, **mod**, **rem**, and **abs**. The SDVS simplifier already knows about one-dimensional arrays having a lower bound of zero; arrays having arbitrary lower bounds must also be accommodated. Finally, a mechanism for dealing with the scoping inherent in block structure must be added to SDVS. The unique qualification of place names (mentioned later) is a part of the solution. The technique of using existential quantification of SDs to control the set of active places, recently suggested by our colleague Tim Redmond, completes the solution of this problem.

4 Overview of Core Ada

Core Ada is a simple, but nontrivial, subset of Ada. Core Ada is intended to be the basis of a rapid initial adaptation of SDVS to Ada, providing early confirmation of two technically sound but untested techniques: formal (Ada) translator specification and specification-directed translator impiementation.

4.1 Core Ada Language Features

A more detailed description of Core Ada language features now follows. These features are partitioned into four groups: statements, expressions, declarations, and data types.

Statements

Core Ada statements constitute a "structured flowchart" programming language. The kinds of statements included are **null**, assignment, conditional (**if**), **case**, **loop** (**while** loops with and without a condition), block, **exit**, and simple input and output (GET and PUT).

Core Ada assignment statements can perform only *scalar* assignments; the left part of the assignment must represent a *scalar* reference (it cannot be the name of an array). The **exit** statement provides a mechanism to escape from the body of a **loop** statement [especially from an infinite loop (one without a condition)]. An **exit** statement can optionally name the loop from which it escapes, and can also optionally have a condition that must hold in order for the escape to occur. Simple input and output, via the text I/O procedures GET and PUT, are included in Core Ada to provide a means of formally specifying data input to, and output from, Core Ada programs. Standard input and output files are predeclared for this purpose. GET and PUT only input and output *scalar* values; inputting and outputting arrays require loops.

Expressions

A representative class of Ada expressions is included in Core Ada. These expressions contain numeric and boolean constants; simple names (identifiers); compound names (e.g., pkg1.pkg2.obj); array references [e.g., name(expr), where name is a simple or compound name and expr is an expression]; short-circuit boolean operators (**and then**; **or else**); relational operators (=, /=, <, <=, >, >=); binary boolean and arithmetic operators (**and**, **or**, **xor**, +, −, *, /, **mod**, **rem**, **); and unary arithmetic and boolean operators (+, −, **abs**, **not**). Compound names permit in Core Ada the concept of accessing items that are not directly visible.

Declarations

Core Ada includes declarations of objects that can be scalar and one-dimensional array constants and variables. Also included are package specifications and **use** clauses (the package specifications themselves can contain object declarations, **use** clauses, and other package specifications). These package specifications and **use** clauses are included in Core Ada to represent a simple Ada encapsulation mechanism.

Data Types

Core Ada includes only very basic data types: integers, booleans, one-dimensional arrays of integers, and one-dimensional arrays of booleans.

5 Constructing the SDVS Ada Translators

The construction of translators from the Ada subsets into state deltas will start from scratch (i.e., the existing ISPS translator will not be modified and used), and will include both formal specification and implementation. Ideally, a complete formal specification would first be written to guide subsequent implementation. Realistically, most of the formal specification would be written first, implementation would proceed, and then specification and implementation would interact until they *seem* to correspond. The formal specification, accompanied by textual explanation, serves as an indispensable part of the translator documentation.

The Ada subset translator specifications will be written by means of the technique used to write the formal specification of the ISPS translator [8]. The translators will be organized in the same way:

Parsing: A concrete syntax will be written for each subset, and an abstract syntax tree for each program will be produced during the concrete parsing of that program. We use our own powerful tools to specify and implement this process.

Static Checking: This first phase of semantic analysis, called **Phase 1**, detects "static" errors such as items undeclared before their use, inappropriate types, and semantically ill-formed constructs. Provided that no errors occur in Phase 1, an environment for the final phase of the translator will be produced.

Translation: This final phase of semantic analysis, called **Phase 2**, incrementally generates state deltas that carry out the symbolic execution of the subject Ada program.

Other techniques must be carried over from the ISPS experience, such as qualifying the same identifier used in different scopes in order to make their instances unique when they appear in state deltas (which have no scopes).

The presence of recursive subprograms in Ada presents a new challenge. Recursive functions were not allowed in ISPS; ISPS programs were, in effect, block-structured flowchart programs. The SDVS system maintained a map of the association of places with their values, and this map might have been updated each time a state delta was applied. With the introduction of recursion, new instances of places local to a subprogram must be created, or the association maintained by the place-to-value map must be made more complex.

5.1 Translator Specification

The formal specification of both phases of the SDVS Ada translators is written in a continuation-style *denotational semantics* [10]. Because of space limitations, the following descriptions are necessarily simplified and sketchy; full details will appear in a forthcoming technical report [4]. It is assumed that the reader has some familiarity with the principal concepts of denotational semantics as presented in, say, [10].

Phase 1 collects the environment of an *entire* program for subsequent use by Phase 2. Since Ada has scope and visibility rules similar to those of Algol and Pascal, only part of this environment is visible at a given point in a program. Consequently, the environment must be *tree-structured*, and access to its components must be appropriately controlled during Phase 2. The local environments of nonoverlapping blocks are made to lie on different paths of the tree-structured environment. If the same identifier is declared in two blocks, then these two instances of the same identifier can be distinguished by the paths leading to their respective local environments. Moreover, these identifiers can be *uniquely qualified* by being prefixed with a textual representation of the paths leading to their local environments. This technique is indeed necessary when the identifiers appear in state deltas in Phase 2, as state deltas have no mechanism to control the visibility of names.

Continuations are used in Phase 1 of the Core Ada translator in a very straightforward way. They simply sequence the static checking functions from one program element to the next, until the entire program has been checked. If a static checking error occurs, then an error message is produced and Phase 1 terminates. In Phase 2, however, continuations are used in a more explicit and sophisticated way. Continuations must appear in the postconditions of state deltas so that when a state delta is applied, more state deltas are generated in order that their subsequent application can advance symbolic program execution. Continuations are also stacked to control explicitly the orderly completion of nested systems of loops and blocks. The initiation of Phase 2 of the Core Ada translation assumes that a program has first successfully passed Phase 1. In fact, Phase 2 can be used as the continuation for Phase 1.

5.2 Translator Implementation

The SDVS Ada translators are implemented in Common Lisp. The translator specifications are written in a style intended to facilitate a straightforward manual translation into a corresponding Common Lisp program. We have adhered to this strategy quite faithfully in our implementation of the Core Ada translator, which we now briefly describe. Full details will appear in a forthcoming technical report [11].

The Common Lisp implementation of denotational translator specifications involves the implementation of only two objects, semantic functions and data structures. For Core Ada, two main Common Lisp functions are defined, one for each of the two phases. The name of each such function is used to prefix the names of the remainder of the functions defined in that phase. Common Lisp data structures are defined for each of the specification's data structures, and additional data structures are defined for subtrees of Core Ada abstract syntax trees.

To effect a transparent implementation of the Core Ada translator, the names of the Common Lisp functions are chosen to correspond with those of the denotational semantic functions, with a prefix to indicate whether the function is from Phase 1 or 2. For each semantic function operating on a single syntactic class, a Common Lisp function is defined to handle arbitrary syntactic objects in that class, with subordinate functions (whose names are appropriately suffixed) defined to handle the semantic equations for specific syntactic cases. If the objects of the syntactic class can appear in sequences, an additional Common Lisp function is defined to implement the semantic function for sequences. The bodies of the Common Lisp functions are for the most part obtained via direct transformations of the corresponding semantic equations. These transformations from the denotational to the Common Lisp style are detailed in [11]. Some transformations that are not so direct, such as the representation of continuations by functions and the treatment of the ellipsis notation, are justified in [11].

The translator implementor has more freedom in the choice of Common Lisp data structures than in the choice of Common Lisp function definitions; the Common Lisp functions must correspond more or less directly to their counterparts in the specification, whereas the data structures may be chosen for maximum efficiency. Operators defined on the translator specification's data structures, however, are implemented by Common Lisp functions that are functionally equivalent to the corresponding operators, and whose names are derived from the operator names.

5.3 Core Ada Achievements

Phases 1 and 2 of the Core Ada translator have been completely specified, fully implemented, and successfully tested. The implementation of several Core Ada language features was tested by symbolically executing the operation of programs containing those features on concrete input data. Our approach to translator construction using denotational specifications that are *rapidly* translated into a Common Lisp implementation, has been very successful. The Core Ada translator was produced quickly and its checkout required a minimum of debugging effort because of the close correspondence between specification and implementation. When errors occurred, their cause (usually a minor bug in the specification, the corresponding part of the implementation, or both) was quickly determined and corrected. As a result, the Core Ada translator is well documented and very reliable.

Additions to the SDVS inference machinery required by Core Ada are complete and have been successfully tested. Using these additions to SDVS, several Core Ada programs have been proved totally correct with respect to input and output specifications. Additional experience will be gained by performing more proofs. It was our colleague Tim Redmond who suggested that the semantics of *loop* statements be expressed as *recursive* formulas; this suggestion has been incorporated into the Core Ada translator. Corresponding proof strategies based on fixed-point induction are being investigated [12].

6 Conclusions

Through the language Core Ada, a modest amount of Ada program verification capability has been successfully added to the State Delta Verification System (SDVS). Research into the application of this enhanced capability of SDVS is ongoing. Our next goal is to specify and implement Stage 1 Ada, which will add Ada subprograms and some new data types to the verification capabilities of SDVS.

In our Core Ada experience, our strategy of first formally specifying a language translator in the style of denotational semantics and then translating the specification into a Common Lisp program has been proven successful. The resulting Core Ada translator is well documented, easily maintained, and very reliable. We expect continued success in the construction of translators for more complex Ada subsets.

The generality and modularity of SDVS have shown it to be quite adaptable to new languages and verification applications.

Acknowledgments

The authors thank their colleagues Ivan Filippenko, for his contribution to the definition of state delta semantics given in Section 2; Tim Redmond, for generously sharing his considerable knowledge of state deltas; and Mike Meyer, for his expert editorial assistance.

References

[1] L. Marcus, S. D. Crocker, and J. R. Landauer, "SDVS: A system for verifying microcode correctness," in *17th Microprogramming Workshop*, pp. 246–255, IEEE, Oct. 1984.

[2] L. Marcus, "SDVS 6 Users' Manual," Tech. Rep. ATR-86A(2778)-4, The Aerospace Corporation, 1987.

[3] L. Marcus, T. Redmond, and S. Shelah, "Completeness of state deltas," Tech. Rep. ATR-85(8354)-5, The Aerospace Corporation, 1985.

[4] D. F. Martin, "A formal description of the incremental translation of Core Ada into state deltas in the State Delta Verification System," Tech. Rep. ATR-88(3778)-1, The Aerospace Corporation, 1988.

[5] U. S. Department of Defense, *Reference Manual for the Ada Programming Language (ANSI/MIL-STD-1815A)*, 22 January 1983.

[6] D. Guaspari, C. Harper, and N. Ramsey, "An Ada verification environment," in *Proceedings 10th NBS/NCSC National Computer Security Conference*, pp. 366–371, NBS/NCSC, Sept. 1987.

[7] D. C. Luckham, F. W. von Henke, B. Krieg-Bruckner, and O. Owe, *ANNA – A Language for Annotating Ada Programs.* Berlin: Springer-Verlag, 1987. Lecture Notes in Computer Science, Volume 260.

[8] D. F. Martin, "A preliminary formal description of the incremental translation of ISPS into state deltas in the State Delta Verification System," Tech. Rep. ATR-86A(2778)-7, The Aerospace Corporation, 1987.

[9] D. C. Luckham and N. Suzuki, "Verification of array, record, and pointer operations in Pascal," *ACM Trans. Programming Languages and Systems*, vol. 1, pp. 226–244, 1979.

[10] M. J. C. Gordon, *The Denotational Description of Programming Languages: An Introduction.* New York: Springer-Verlag, 1979.

[11] J. V. Cook, "Implementing the formal description of the incremental translation of Core Ada into state deltas," Tech. Rep. ATR-88(3778)-2, The Aerospace Corporation, 1988.

[12] T. Redmond, "Fixed point methods in temporal logics," Tech. Rep. ATR-88(8354)-1, The Aerospace Corporation, 1988.

EHDM VERIFICATION ENVIRONMENT: AN OVERVIEW

F. W. von Henke, J.S. Crow, R. Lee, J.M. Rushby, R.A. Whitehurst
SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025

Abstract

This article reports on the status of the EHDM specification and verification system, a state-of-the art environment designed specifically to meet the needs of security verification.

1 Introduction

The EHDM system is an integrated environment for the development and analysis of formal specifications and abstract programs. It has been under development at SRI's Computer Science Laboratory (CSL) since 1983, with sponsorship from the National Computer Security Center (NCSC). As its full name (Enhanced Hierarchical Development Methodology) suggests, the development has built on SRI's experience with, and ideas from, previous system-building efforts, including the original Hierarchical Development Methodology (HDM) [16] and STP [18]. The language of EHDM and the EHDM implementation are quite different from those earlier efforts, however; EHDM incorporates many modern ideas and techniques concerning language design, specifications, and development environments in order to provide a state-of-the-art verification system.

1.1 State-of-the-Art Verification Systems

The capabilities expected of a state-of-the-art specification and verification system have been summarized in the final report of the Verification Assessment Study [10], sponsored by the National Computer Security Center in 1985. That report describes the kind of system that could be built with the technology that existed in 1985 and concludes that a state-of-the-art verification system should include:

1. A specification language based on a first-order, typed predicate calculus.
2. An imperative language derived from the Pascal/Algol 60 family of programming languages.
3. A formal semantic characterization of the specification and programming languages.
4. A verification condition generator (VCG).
5. A mechanical proof checker with some automated proof-generation capabilities.
6. A (small) supporting library of (reusable) theorems.
7. A glass (as opposed to hard copy) user interface, possibly using bit-mapped displays.
8. A single system dedicated to one user at a time (e.g., a workstation dedicated to the verification process).
9. The embedding of these components in a modest programming environment with utilities such as version control.

The current EHDM environment matches these expectations rather well and goes beyond them in some important respects.

1. While the specification language is based on a typed first-order predicate calculus, it also includes elements of richer logics, such as higher-order logic, lambda calculus, and Hoare logic, for greater expressiveness.
2. Although the EHDM environment does not provide an imperative programming language, it does include a variety of constructs for expressing imperative behavior; these are sufficient for modeling simple imperative programs at a level of detail similar to Ada or Pascal (see Section 2.3). A concrete link to Ada is provided by a tool within the EHDM environment that translates imperative code-level specifications from EHDM into executable Ada code (see Section 4.3).
3. We are developing the formal semantics of the EHDM specification language, as well as a rigorous description of the operations implemented in the EHDM environment, which together will provide a solid foundation for EHDM. This work will be completed during 1989.
4. The approach to reasoning about imperative features employed in EHDM is more general than the traditional VCG paradigm and allows users to reason directly with program fragments (using Hoare logic). So far, there is insufficient experience with this technique to determine how it compares to the VCG approach in practice. The equivalent of a VCG is available as an additional tool for proof development.
5. The prover component combines powerful heuristics for mechanically generating first-order proofs, together with decision procedures for standard theories. It supports both automated proof generation and interactive, user-guided proof construction. Completed proofs can be captured as proof declarations for inclusion in the specification text and subsequent "replay."
6. The EHDM specification language incorporates several modern features that support reusability of specifications and proofs. Specifications are structured into named modules that other modules can refer to. A general form of module parameterization supports generic specifications; this feature is more expressive than, for example, generic declarations in Ada. The environment also provides a mechanism for grouping standard modules in libraries of reusable concepts, theorems, and proofs; such libraries can be shared among users and projects.
7. The standard user interface of EHDM uses the bit-mapped display of modern workstations and combines a display-

oriented editor with multiple windows, menus, and mouse input.

8. The current version of the EHDM environment is implemented in Common Lisp and runs on Symbolics Lisp machines and on Sun single-user workstations and time-shared computers. Earlier versions of EHDM also exist for mainframes (Multics, TOPS-20).

9. EHDM is implemented as an integrated, interactive environment that supports all activities involved in creating, analyzing, modifying, managing, and documenting specification modules and proofs. An internal database and version control mechanism keeps track of the state of individual modules and proofs, and of the interdependencies among modules and libraries. Thus, EHDM is a fairly complete development environment. However, since it does not support a particular programming language and has no capabilities for compiling and executing programs, it is, strictly speaking, not a "programming environment."

The EHDM environment provides additional tools and features that are not mentioned in the requirements list, but are equally important:

- The EHDM flow tool for analyzing multilevel security (MLS) is based on the noninterference model developed at SRI [8, 7].
- EHDM supports hierarchical structure and hierarchical development of specifications and proofs from high-level requirements to code-level specifications and Ada text (which can be generated from code-level specifications).

2 The EHDM Specification Language and Logic

The EHDM specification language is based on first-order typed predicate calculus, but includes elements of higher-order logic, lambda calculus, and Hoare logic. These enrich the expressive capability of the language. Higher-order terms, for example, are convenient for expressing induction schemas and requirement statements.

The specification language is strongly typed; all entities must be declared with their type before use. The type system includes subtypes and function types. Specifications are written as definitions and formulas (axioms, theorems, and lemmas). The expression language includes all the standard expressions of propositional calculus, polymorphic conditionals, and quantified expressions, including quantification over functions.

2.1 Modules

Modules are the basic building blocks of specifications. A module may represent the theory describing a specification concept, an abstract data type, an abstract state machine, or an (abstract) program. Modules are closed scopes with explicit importation and exportation of names; modules can be nested.

Modules may be parameterized by types, constants, and functions. Semantic assumptions or constraints on module parameters can be expressed; these entail an obligation that must be justified for each module instantiation. This form of module parameterization is very general and powerful; it supports generic specifications and allows many complex constructs to be built from simple language primitives. The module *inductions* shown in Figure 1 exemplifies the use of module parameters, assumptions, and higher-order quantification.

The language has been designed so that it naturally supports a high degree of reusability of specifications and proofs. The main vehicle for reusability are modularization and parameterization of modules. Reusability is also enhanced by the library facility described later.

```
inductions: MODULE [dom: TYPE, first: dom,
                    next: function[dom -> dom]]
ASSUMING

  d1, d2: VAR dom
  measure: VAR function[dom -> nat]

  wellfounded: FORMULA
    (EXISTS measure :
      (FORALL d1 : measure(d1) < measure(next(d1))))

  first_is_first: FORMULA
    NOT (EXISTS d1 : first = next(d1))

  reachability: FORMULA
    d2 /= first IMPLIES (EXISTS d1 : d2 = next(d1))

THEORY

  p: VAR function[dom -> bool]

  induction: AXIOM
    (p(first) AND
        (FORALL d1 : p(d1) IMPLIES p(next(d1))))
      IMPLIES (FORALL d2 : p(d2))

END inductions
```

Figure 1: A Parameterized Module with Assumptions and Second-order Quantification

2.2 Hierarchical Development

An important aspect of EHDM (and its predecessor, HDM) is the support of hierarchical development of specifications and proofs. In a hierarchical development, a system module is specified abstractly at one level of the hierarchy and implemented using the operations provided by the next-lower level. The idea of structuring system designs in this manner has been discussed and used (in the form of "abstract machines") since the late 1960s (cf. [14]), an early form of associated proof techniques is presented in [15]. The EHDM language supports hierarchical structuring of specifications from requirement specifications through (usually several) levels of abstractions down to a level of code specifications. The links between modules at adjoining levels of abstraction are established by mappings, which generalize the notion of implementation. Demonstration of correctness of mappings between levels is supported by the proof system. The *traffic light* example, developed in Appendix A.1, demonstrates the use of hierarchical development in EHDM.

2.3 Code-level Specification

A sublanguage is provided for modeling operational behavior and imperative programs, based on the notions of *state object* and *operation*. State objects correspond to "program variables" in programming languages. Operations express state transformations; they have an effect on state objects by possibly changing their values. The language also includes constructs for composing operation expressions that correspond to the common control structures of programming languages. The combination of all these features forms a sublanguage that is essentially equivalent to a simple subset of the Ada programming language; an example is shown in Appendix A.2. The semantics of operations are defined by Hoare formulas, which express properties of the states before and after the state transformation denoted by the operation.

3 The Theorem Prover of EHDM

The theorem-prover component of EHDM combines powerful heuristics for mechanically generating proofs in first-order predicate logic with efficient decision procedures for the combination of the following standard theories [19]:

- Ground formulas in propositional calculus
- Equality over uninterpreted function symbols
- Presburger arithmetic, i.e., linear arithmetic with the usual ordering relations.

Proofs (more precisely, proof steps) are declared in the proof part of a module; they are expressed as a conclusion to be proven and a list of formulas (axioms and lemmas) from which the conclusion can be deduced. Proof declarations may also specify ground terms to be substituted for the free (technically, non-Skolemized) variables in formulas. A fully-instantiated proof (one for which all necessary substitutions have been provided) can be checked for validity by the ground decision procedures without further input from the user. An example of such a proof (using the induction axiom from Figure 1) is shown in Figure 2.

```
the_result: PROVE closed_form FROM
  induction
    {p <- (LAMBDA z -> bool:
            2*sigma(z) = square(z)+z),
     d2 <- i@C},
  basis,
  inductive_step {i <- d1@P1}
```

Figure 2: A Fully-Instantiated Proof

Proofs that are not fully instantiated can often be completed by the high-level prover (also known as the "instantiator"). The high-level prover employs powerful heuristics in an attempt to construct suitable substitution instances for the formulas involved; this process can be completely automatic, or it can be performed in interaction with the user. Completed proofs can be captured in augmented proof declarations and included in the specification text for later replay as fully instantiated proofs. A proof-chain analysis tool checks for completeness of larger proof trees and helps keep track of dependencies.

Equational reasoning similar to the use of rewrite rules is specially supported by the high-level proof procedure. The prover also implements the main reduction rules of lambda calculus and a fragment of higher-order logic; however, the characterization of the exact extent of this support is still under investigation.

A special procedure for reasoning about state transformations has built-in knowledge of the meaning of Hoare formulas, state objects and state transformations. This procedure provides the main support for code-level verification. It permits users to reason directly with Hoare formulas, without the traditional intermediate step of translating annotated programs into verification conditions (VCs); the procedure also supports reasoning about program fragments, as opposed to complete programs units (like subprograms). In these respects, the paradigm that is implemented by the procedure is more general than the traditional "verification condition generator" (VCG) paradigm. However, a tool equivalent to a VCG is available in the EHDM environment for use in proof development. Examples of proofs with Hoare formulas are included in Appendix A.2.

4 The EHDM Environment

The EHDM environment is implemented as an integrated, interactive system that supports all activities involved in creating, analyzing, modifying, managing, and documenting specification modules and proofs. The standard user interface of EHDM uses the bitmap display and combines a display-oriented text editor (customized and enhanced EMACS) with multiple windows, menus, and mouse input. (A less enhanced editor-based interface is available for remote operation.) All operations can be invoked directly from the editor, including the basic operations of parsing, prettyprinting, and typechecking specification text, invoking the theorem prover, and requesting status information. In addition to these basic operations, the system provides a number of further support tools, including: the MLS Checker (described in Section 5), the context and library tools, the configuration control support, and the EHDM-to-Ada translator.

4.1 The Context and Library Manager

The system maintains an internal database for keeping track of the state of individual modules and proofs (referred to as the *working context*) and of the interdependencies among modules and libraries; the user can manipulate this working context or switch between contexts. Modules are the basic entities around which the EHDM system is organized, and the context feature virtually insulates the user from the underlying file system. The EHDM system creates and manages private files, which the user can ignore completely since all file manipulation happens as a side effect of user interaction with the EHDM system.

The library mechanism permits users to group standard modules in libraries of reusable concepts, theorems, and proofs. The environment offers tools for creating and maintaining module libraries and supports sharing of libraries among users and projects.

Contexts and libraries have been designed so that the novice user can completely ignore these facilities. A user always works within a context, but when a user starts EHDM for the first time, the system automatically establishes a working context; later, when the user leaves EHDM, the system saves the context and restores it when work is resumed. Users need to know about contexts only when they want to work in more than one context; similarly, they can ignore libraries until they want to make use of that facility.

4.2 The Configuration Control Tool

A configuration and version control mechanism ensures that consistent versions of modules are used; status checks report on the status of modules and proofs, and on dependencies among modules and proofs. At any given time, a module specification may be in one of several states. When a module is typechecked or a proof performed, a "version check" is made to see that the typecheck information (type tables) recorded for the transitive closure of all referenced modules is still valid. For example, if module *A* uses module *B*, then changes to module *B* will invalidate the type table for module *A*. This invalidation will be discovered when module *A* is accessed during the typechecking of a module that uses *A*, or during the construction of a proof from *A* or a module that uses *A*.

4.3 The EHDM–to–Ada Translator

Hierarchical development of specifications and proofs from requirements down to the code-level is complemented by an experimental facility for translating code-level specifications or "abstract algorithms" from the EHDM language into Ada. The EHDM -to-Ada translator has been developed to investigate a paradigm of code verification in which the code written in the programming language is regarded as the *target* of a systematic development, in contrast to the traditional view that regards it as the starting point of a verification effort.

The traditional approach is to start with program code, augment it with annotations or "assertions," and attempt to deduce properties of the code. Thus, verification occurs *after* a piece of executable code has been written. In contrast to the traditional approach, a key idea behind hierarchical development as embodied in EHDM is that actual, executable code is the *result* of a development process that involves several layers of abstractions and refinements. Thus, the EHDM way to develop verified programs is to carry out the necessary reasoning at the *design* level, before actual program text is considered. The advantage of this approach is that most, if not all, reasoning is done in the specification language rather than in the programming language, thus avoiding the language complexities that typically result from design considerations such as compiler speed or runtime efficiency. Furthermore, this approach leads to a more natural integration of verification into the process of software development.

The experimental EHDM–to–Ada translator automatically extracts the "operational content" of an EHDM module and translates it into Ada code wrapped in a package. An example of the use of the translator is shown in Appendix A.2. The translator is based on a detailed comparison between the two languages. In many respects, EHDM lends itself to translation into Ada, since central Ada constructs like packages and generics have a direct counterpart in the EHDM language (modules and module parameters). On the other hand, the differences between the two languages are substantial. Many features of Ada that are important for verification cannot be modeled directly in EHDM. For a tool based on this paradigm to become really practical, the specification language must be much closer to Ada than the current EHDM language; for example, it must take such features as the Ada type system and exceptions fully into account.

5 The Multilevel Security Tool

EHDM provides a tool that analyzes specifications for compliance with a notion of multilevel security (MLS). This MLS Checker is based on the noninterference model of security [8,7,17] developed by Goguen and Meseguer at SRI. This is the main tool in EHDM that is specific to security applications.

The MLS Checker examines certain types of specifications and generates formulas (called *verification conditions*), which, if true, establish that the specification complies with the noninterference formulation of security. The verification conditions are collected together in the THEORY section of a new module generated by the MLS Checker, and a simple PROVE or sc verify declaration for each verification condition is placed in the PROOF section of the new module. Often, these mechanically generated PROVE declarations are sufficient to establish their corresponding verification conditions. If not, the user must develop suitable proofs in a *separate* module: modifications to the module containing the verification conditions are not allowed.

The verification conditions generated by the MLS Checker ensure that:

1. The result of an operation depends only on the values of objects whose security classifications are dominated by those of the caller of the operation
2. An operation potentially changes the values of only those objects whose security classification dominate that of the caller
3. Values assigned to an object depend only on the prior values of objects whose security classifications are dominated by that of the object assigned to.

These three properties guarantee that a specification is secure in the sense that it does not *require* an unsecure implementation; they do not prove that it will not *allow* an insecure implementation. Furthermore, since security is a negative property (it is concerned with what must *not* happen), a conventionally verified implementation of a secure specification need not be secure! The properties established by the form of flow analysis embodied in the MLS Checker are therefore quite limited and should not be mistaken for a "proof of security" [9]. Nonetheless, this is a very useful class of tool and the only one capable of detecting covert storage channels [3].

6 Applications of EHDM

6.1 SEAVIEW

Currently, EHDM is used primarily in the SeaView project, jointly conducted by SRI and Gemini, which is developing a design for an A1 multilevel secure database system [6,4,5]. In this project, a formal top-level specification of "secure distributed data views" is being developed using EHDM as the specification language.

This project has considerably advanced the state of the art in multilevel database security. Prior to SeaView no demonstrably viable approaches to multilevel database security existed. SeaView's approach combines element-level labeling with A1 assurance – a feat that was considered impossible as recently as a year ago.

Among SeaView's major achievements are:

- A security policy that defines what security means for a database system
- An interpretation of the security policy demonstrating how SeaView applies the security policy to a relational database system
- A multilevel relational data model that defines extensions to the standard relational model to specifically accommodate element labels
- A formal security model that includes security properties that define secure state and transition properties that further restrict state transitions (to rule out systems such as "System Z" [12]).

The exercise of formally specifying the SeaView properties in the EHDM language resulted in the discovery and clarification of many ambiguities and imprecise or incomplete statements in the original description of the model. Through this exercise, numerous mistakes in the properties of the model were also identified and corrected.

6.2 Other Application Areas

An early version of the EHDM system was used at SRI to extend the design verification for the SIFT fault-tolerant multiprocessor [13] previously carried out with the STP system. This effort involved the specification and verification of rather subtle properties; the transcripts of the specification and proof of the system occupy more than 700 pages.

EHDM is currently also being used to specify and verify the functional behavior of hardware and to formalize completely the published proof of a clock synchronization algorithm [11].

7 Conclusion

The EHDM system as described here has reached a stage where it can support serious specification and verification efforts. However, the development of EHDM is still continuing. We have already mentioned the ongoing work on clarifying the formal basis of EHDM ; both language and system are expected to be refined as a result.

The EHDM system is perceived by some as hard to understand and difficult to use. We are aiming at overcoming these difficulties by developing tutorial materials that complement the existing user documentation [2,1] and that describe the styles and "idioms" of specifications and proofs in EHDM . A library of specification modules for standard concepts is being developed as EHDM is being used more extensively both inside and outside SRI.

Acknowledgements

Previous major contributors to the EHDM development effort include S. Jefferson, P. M. Melliar-Smith, R. Schwartz, R. Shostak, and M. Stickel.

References

[1] *EHDM Specification and Verification System Version 4.1, Preliminary Definition of the EHDM Specification Language.* Computer Science Laboratory, SRI International, June 1988.

[2] *EHDM Specification and Verification System Version 4.1, User's Guide.* Computer Science Laboratory, SRI International, May 1988.

[3] M. Cheheyl et al. Verifying security. *Computing Surveys*, 13(3):279–339, September 1981.

[4] D. E. Denning, T. F. Lunt, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley. A multilevel relational data model. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987.

[5] D. E. Denning, T. F. Lunt, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley. *The SeaView Formal Security Policy Model.* Technical Report, Computer Science Laboratory, SRI International, 1987.

[6] D. E. Denning, T. F. Lunt, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley. *Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database System.* Technical Report, Computer Science Laboratory, SRI International, 1986.

[7] J.A. Goguen and J. Meseguer. Inference control and unwinding. In *Proc. 1984 Symposium on Security and Privacy*, pages 75–86, IEEE Computer Society, Oakland, CA., April 1984.

[8] J.A. Goguen and J. Meseguer. Security policies and security models. In *Proc. 1982 Symposium on Security and Privacy*, pages 11–20, IEEE Computer Society, Oakland, CA., April 1982.

[9] Joshua Guttman. Information flow and invariance. In *Proc. 1987 IEEE Symposium on Security and Privacy*, pages 67–73, IEEE Computer Society, Oakland, CA., April 1987.

[10] Richard A. Kemmerer. *Verification Assessment Study Final Report.* Technical Report C3-CR01-86, National Computer Security Center, Ft. Meade, MD., 1986. 5 Volumes.

[11] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[12] John McLean. Reasoning about security models. In *1987 IEEE Symposium on Security and Privacy*, pages 123–131, IEEE, IEEE Computer Society Press, Washington, D.C., April 1987.

[13] L. Moser, P.M. Melliar-Smith, and R. Schwartz. *Design Verification of SIFT.* Contractor Report 4097, NASA Langley Research Center, Hampton, VA., September 1987.

[14] D. Parnas. On a 'buzzword': hierarchical structure. In *Information Processing '74*, pages 336–339, IFIP, 1974.

[15] L. Robinson and K. N. Levitt. Proof techniques for hierarchically structured programs. *Communications of the ACM*, 20(4):271–283, April 1977.

[16] L. Robinson, K.N. Levitt, and B.A. Silverberg. *The HDM Handbook.* Computer Science Laboratory, SRI International, Menlo Park, CA., June 1979. Three Volumes.

[17] J.M. Rushby. The security model of Enhanced HDM. In *Proceedings 7th DoD/NBS Computer Security Initiative Conference*, pages 120–136, Gaithersburg, MD., September 1984.

[18] R.E. Shostak, R. Schwartz, and P.M. Melliar-Smith. STP: a mechanized logic for specification and verification. In *6th International Conference on Automated Deduction (CADE-6)*, Springer-Verlag Lecture Notes in Computer Science, Vol. 138, 1982.

[19] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.

A Examples

This appendix presents some simple examples to demonstrate the features of EHDM that have been described in the text.

A.1 Traffic Light

The first example demonstrates the use of hierarchical specification and verification. We present a very abstract characterization of a safe intersection controlled by traffic lights, a more concrete realization of such an intersection, and a proof that the one is a valid interpretation of the other.

The module **colors** introduces **color** as an uninterpreted type and **green**, **yellow** and **red** as constants of that type. In this elementary example, we do not include the requirement that these colors should be distinct.

The module **traffic_light** introduces the uninterpreted type **intersection**, and the constant **initial** of that type. **Eastwest** and **northsouth** are functions whose intuitive purpose is to return the current color of the light in their respective directions, while **change_lights** is the function that changes these colors.

The module **safe_intersection** is a *requirements statement* for a safe intersection. It defines the predicate **safe_lights** to be true if a **red** light is showing in at least one of the directions of the intersection, and it requires that the **initial** configuration of the intersection should be safe and that **change_lights** should preserve safety.

The module **light_sequence** defines a particular sequencing of colors, while the module **pairs** defines the theory of pairs, with **first** and **second** as the selectors, and **make_pair** as the constructor. Both these modules use equational specifications, which simplify the subsequent proofs.

The module **traffic_light_adt** provides a concrete realization of an intersection (as an *Abstract Data Type*, or ADT). An **intersection** is realized as a pair of colors, whose **first** component is that showing in the **eastwest** direction, and whose **second** component is that showing in the **northsouth** direction. The **change_lights** function provides a particular sequencing of lights at an intersection, using the sequencing of **colors** defined by the **next** function from the **light_sequence** module.

Finally, the module **safe_lights_adt** establishes a *mapping* or interpretation from the modules **traffic_light** and **safe_intersection** onto the more concrete **traffic_light_adt**. A valid mapping requires that the *axioms* of the higher level modules become *theorems* of the lower one. This is demonstrated in the PROOF section of **safe_lights_adt**. Because all the relevant properties have been defined equationally, the high-level prover is able to complete these proofs without further input from the user.

```
traffic_light_adt : MODULE

  USING colors, light_sequence, pairs[color,color]
  EXPORTING  intersection, eastwest, northsouth,
    initial, change_lights

THEORY

  intersection: TYPE IS pair

  p: VAR intersection
  corner: VAR intersection

  eastwest: function[pair->color] = first

  northsouth: function[intersection->color] = second

  initial: intersection  = make_pair(red,green)
  unsafe:  intersection

  change_lights: function[intersection->intersection]
    = (LAMBDA corner -> intersection:
          IF eastwest(corner)=red THEN
             IF northsouth(corner)=yellow THEN
                make_pair(next(eastwest(corner)),
                          next(northsouth(corner)))
             ELSE make_pair(eastwest(corner),
                            next(northsouth(corner)))
             END
          ELSIF northsouth(corner)=red THEN
             IF eastwest(corner)=yellow THEN
                make_pair(next(eastwest(corner)),
                          next(northsouth(corner)))
             ELSE make_pair(next(eastwest(corner)),
                            northsouth(corner))
             END
          ELSE unsafe  END IF)

END traffic_light_adt
```

```
safe_lights_adt: MODULE

  MAPPING  traffic_light, safe_intersection
  ONTO  traffic_light_adt

THEORY

  traffic_light.intersection: TYPE
    IS traffic_light_adt.intersection

  traffic_light.eastwest:
    function[traffic_light_adt.intersection->color]
                  = traffic_light_adt.eastwest
  traffic_light.northsouth:
    function[traffic_light_adt.intersection->color]
                  = traffic_light_adt.northsouth
  traffic_light.initial: traffic_light_adt.intersection
                  = traffic_light_adt.initial

  traffic_light.change_lights:
    function[traffic_light_adt.intersection
              -> traffic_light_adt.intersection]
                  = traffic_light_adt.change_lights
PROOF

  USING  light_sequence, pairs[color,color]

  WITH light_sequence, pairs

  safe_initially_pr: PROVE safe_initially

  remains_safe_pr: PROVE remains_safe

END safe_lights_adt
```

```
safe_intersection: MODULE

  USING colors, traffic_light
  EXPORTING safe_lights

THEORY

  corner: VAR intersection

  safe_lights: function[intersection ->  boolean]
    = (LAMBDA corner -> boolean:
         eastwest(corner) = red
           OR northsouth(corner) = red )

  safe_initially: FORMULA  safe_lights(initial)

  remains_safe: FORMULA safe_lights(corner)
    IMPLIES  safe_lights(change_lights(corner))

END safe_intersection
```

```
light_sequence: MODULE

  USING colors traffic_light
  EXPORTING next

THEORY

  c: VAR color

  next: function[color->color]

  n1: AXIOM  next(green)  = yellow
  n2: AXIOM  next(yellow) = red
  n3: AXIOM  next(red)    = green

END light_sequence
```

```
pairs: MODULE[firsttype, secondtype: TYPE]

  EXPORTING pair, first, second, make_pair

THEORY

  pair: TYPE
  first: function [pair->firsttype]
  second: function [pair->secondtype]
  make_pair: function [firsttype,secondtype -> pair]

  x: VAR firsttype
  y: VAR secondtype
  p: VAR pair

  mdef1: AXIOM  first(make_pair(x,y)) = x

  mdef2: AXIOM  second(make_pair(x,y)) = y

END pairs
```

```
traffic_light: MODULE

  USING colors
  EXPORTING intersection, eastwest,
    northsouth, initial, change_lights

THEORY

  intersection: TYPE
  initial: intersection

  eastwest: function[intersection->color]
  northsouth: function[intersection->color]

  change_lights: function[intersection->intersection]

END traffic_light
```

```
colors: MODULE

  EXPORTING  color, green, yellow, red

THEORY

  color: TYPE
  green, yellow, red: color

END color
```

A.2 Binary Search

The second example, the module `Binsearch`, demonstrates code-level specification and proof by means of a *binary search* algorithm. Notice how the text of the algorithm (the operation `bsearch`) is broken into smaller pieces, which facilitates reasoning about fragments of the code; for example, the invariant of the loop body (lemma `Inv`) is verified separately (`InvPr`) and then used in establishing the main property (`Main`).

The module `Binsearch` uses two other module that are not displayed here: `IntArrays` declares the type `IntArray`, arrays of integers; `OrdIntervals` introduces the predicates `ordered` and `is_in` on integer arrays and gives their relevant properties in lemmas `IL1`, `IL2` and `IL3`.

```
Binsearch: MODULE [N: int]  (* binary search module *)

  USING IntArrays, OrdIntervals
  EXPORTING bsearch

THEORY

  A          : VAR IntArray
  key, i, j  : VAR int
  lb, ub     : state[int]
  index, x   : VAR state[int]

  div  : function [int,int->int]
  mean : function [int,int->int]
         = (lambda i,j -> int: div(i+j,2))

  newindex: operation
         == BEGIN index := mean(lb,ub) END

  newindex_Lem1: LEMMA
     0<=lb AND lb<ub newindex lb<=index AND index<ub

  body: operation
       == BEGIN
            newindex;
            IF key > A(index) THEN
               lb := index + 1
            ELSE
               ub := index
            END IF
         END

  bsearch: operation [IntArray, int, state[int]]

  bsform: FORMULA
    bsearch(A,key,index)
        = BEGIN
             lb := 1;
             ub := N;
             WHILE lb<ub LOOP
               body
             END LOOP
          END

  Inv: LEMMA
       { ordered(A)  AND  0<=lb  AND  lb<ub  AND
         (is_in(key,A,1,N) IMPLIES is_in(key,A,lb,ub)) }
        body
       { 0<=lb AND lb<=ub  AND
         (is_in(key,A,1,N) IMPLIES is_in(key,A,lb,ub)) }

  Main:  THEOREM
         { N>0 AND ordered(A) }
          bsearch(A,key,index)
         { lb=ub  AND
           (is_in(key,A,1,N) IMPLIES key=A(lb)) }

PROOF

  newindex_Lem2: LEMMA  newindex CHANGES index

  prl2:  VERIFY newindex_Lem2

  InvPr: VERIFY Inv
         FROM   newindex_Lem1, newindex_Lem2,
                IL1 {ky<-key, B<-A, i<-lb@P1,
                     j<-ub@P1, k<-index@P1},
                IL2 {ky<-key, B<-A, i<-lb@P1,
                     j<-ub@P1, k<-index@P1}

  MainPr: VERIFY  Main
          FROM    bsform, Inv,
                  IL3 {ky<-key, B<-A, i<-lb@C, j<-ub@C}

END Binsearch
```

A.3 Example of a Translation into Ada

The binary search module also serves as an example of the translation from EHDM into Ada. The translation ignores the Hoare formulas and proof declarations, which do not convey "operational content." The function div has a "separate body" since no body has been given for it. Note that the operations body and newindex, which merely denote code fragments, do not appear in the Ada text; they have been expanded in the loop body.

```
WITH IntArrays; USE IntArrays;
WITH OrdIntervals; USE OrdIntervals;

GENERIC
  N : Integer;

PACKAGE Binsearch IS
  PROCEDURE bsearch (A : IntArray; key : Integer;
                     index : in out Integer);
END Binsearch;

PACKAGE BODY Binsearch IS
  lb : Integer;
  ub : Integer;

  FUNCTION div (a_0, a_1 : Integer) RETURN Integer
    IS SEPARATE;

  FUNCTION mean (i, j : Integer) RETURN Integer IS
  BEGIN

    RETURN div(i + j, 2);

  END mean;

  PROCEDURE bsearch (A : IntArray; key : Integer;
                     index : in out Integer) IS
  BEGIN

    lb := 1;
    ub := N;
    WHILE lb < ub LOOP
      index := mean(lb, ub);
      IF key > A(index) THEN
        lb := index + 1;
      ELSE
        ub := index;
      END IF;
    END LOOP;

  END bsearch;

END Binsearch;
```

Verifying Implementation Correctness Using The State Delta Verification System (SDVS)[†]

Mel Cutler

Computer Science Laboratory
The Aerospace Corporation
P. O. Box 92957, M1/102
Los Angeles, CA 90009-2957

Abstract

The formal concept of implementation correctness and its embodiment in the State Delta Verification System (SDVS) are briefly described. Described are important features added to a prototype version of SDVS as a result of experience with a substantial implementation-correctness-verification effort. The result is a usable version of SDVS that can be applied to increase confidence in microprogrammed computer implementations; this version is also being expanded to verify the implementation correctness of programs and hardware [1].

1 Formal Computer Descriptions and Verification

A computer system design is hierarchical; that is, descriptions of the system components are developed at different levels (algorithmic, programming, architectural, register transfer, gate, circuit, and integrated circuit mask) that vary in the amount of detail they incorporate. The tools used in the design process often need a formal description of the design at the appropriate level, as the following illustrate:

- A programming language is used at the programming level to specify the input to the compiler
- A Boolean level description language is used to specify the input to logic minimization programs.
- A geometry oriented language is used for each layer at the integrated circuit mask level to specify the input to the pattern generator.

These formal descriptions may also be used to verify the correctness of the design,i.e., that it is consistent with the specifications of the computer system. This verification process is especially important in systems for which a design error is costly, such as in applications in which security is vital or in spaceborne applications.

With most verification systems, the descriptions of a computer system are rewritten by the designers or verifiers in the language underlying the logic of the verification system. Unfortunately, specification errors creep into this process and thereby increase the cost of the verification effort and reduce the confidence in the results.

The State Delta Verification System (SDVS), developed in the Computer Science Laboratory of The Aerospace Corporation [2], takes a different approach. By using the design language rather than the language of the verification system, SDVS has an intrinsic advantage over other verification systems in that it is easily integrated into the design process. SDVS reduces specification errors by translating the designer's descriptions automatically into state deltas, which comprise the underlying logic of the SDVS system. Currently, SDVS supports portions of the design languages ISPS [3] and Ada [4].

The remainder of this paper discusses (1) the formal concept of implementation correctness as it is embodied in SDVS, (2) how SDVS supports implementation-correctness proofs, (3) how SDVS has been applied to a formal proof, and (4) how SDVS is being enhanced to support implementation-correctness proofs at multiple levels of design. These enhancements will result in increased confidence in the correctness of the design throughout the design cycle.

2 Implementation Correctness Using SDVS

Correctness proofs at high levels of design abstraction (e.g., program verification) involve verifying the consistency of a particular design's representation with respect to input and output assertions. As discussed above, these assertions are normally written in the language of the underlying logic of the verification system. Implementation-correctness proofs in SDVS differ from the traditional paradigm in that they deal with two representations of the design at different levels of detail. The levels of abstraction we are considering for implementation correctness correspond to computer program, computer instruction set, computer microprogram engine, and gate level design. All of these levels can be handled within the implementation-correctness paradigm.

Informally, one says that a lower level specification S_1 *implements*[1] a higher level specification S_2 if the significant state changes that S_2 makes are also made by S_1 (usually by a sequence of more primitive state changes). This concept is very general and can be applied to any pair of levels of the computer-system hierarchy. The implementation relation is not necessarily symmetric; there may be state changes of S_1 that have no corresponding significant state changes in S_2

This informal notion of implementation correctness is best captured in a formal correctness theorem. Because, in general, the two descriptions will have different data structures and internal states, the key to the implementation-correctness theorem is to provide the correspondence between upper-level and lower-level data and state universes. This correspondence, called *mapping* in SDVS [6], is the means through which state changes at one level correspond to state changes at the other level.

[†] This research was supported in part by The Aerospace Corporation and in part by the National Computer Security Center under contract F04701-85-C-0086-P00016.

[1] Sometimes the term "simulates" [5] is used for what we call "implements"; however, we prefer the latter term because the former connotes a testing methodology and it is primarily used for a specific portion of the computer-system hierarchy.

To begin this formalization process, we build a semantic base for computation and for the implementation of one computation by another computation. During a computation, the program variables take on values from the *computational domain*; we capture this concept formally below.

Definition 1 *$\langle \mathcal{A}, \mathcal{L}, \mathcal{P} \rangle$ is a* computational domain *if and only if*

- *$\mathcal{A}$ is a model for the typed first-order language $\mathcal{L}$, where $\mathcal{L}$ has types* **place**, **domain**, *and* **architecture**, *where the type* **architecture** *represents a Boolean algebra. $\mathcal{L}$ contains none of the symbols $.$, $\#$, $\sim$, ε, or $\| * \|$; and*
- *$\mathcal{P}$ is a finite set of statements from $\mathcal{L}(.)$, where "." is a function symbol from type* **place** *to type* **domain**. *All occurrences of "." in sentences from $\mathcal{P}$ are applied to constant place symbols from the language $\mathcal{L}$.*

The sentences in $\mathcal{P}$ are called *declarations*. The objects of type **place** are in a one to one correspondence with the variables of the computation; we think of these objects as being actual places or registers in a machine. The variables take on values that are the objects of type **domain**.

Definition 2 *$\langle T, (\sigma_t)_{t \in T} \rangle$ is a* computational model *over a computational domain $\langle \mathcal{A}, \mathcal{L}, \mathcal{P} \rangle$ if and only if*

- *T is a linearly ordered set with a minimal element, and*
- *for each $t \in T$, σ_t is a function from the elements of $\mathcal{A}$ of type* **place** *to the elements of $\mathcal{A}$ of type* **domain**.

T is called the timeline of the computational model. For any $t \in T$, σ_t is called a state in the computation.

For any $t \in T$ and any object p of type **place** in $\mathcal{L}$, the "contents of p at time t" refers to the value of $\sigma_t(p)$.

The following definitions make formal what we mean by an instance of a lower level machine implementing an upper level machine.

Definition 3 *$\langle \mathcal{M}^l, \mathcal{M}^u \rangle$ is a* mapping model *from the upper domain $\langle \mathcal{A}^u, \mathcal{L}^u \rangle$ to the lower domain $\langle \mathcal{A}^l, \mathcal{L}^l \rangle$ if and only if*

- *$\mathcal{M}^l$ and $\mathcal{M}^u$ are computational models over the lower and upper domains, respectively, and*
- *the timeline of $\mathcal{M}^u$ is a subset of the timeline of $\mathcal{M}^l$.*

A mapping *from an upper domain $\langle \mathcal{A}^u, \mathcal{L}^u \rangle$ to a lower domain $\langle \mathcal{A}^l, \mathcal{L}^l \rangle$ is a set of mapping models between those same domains.*

Example 1 *Suppose that the upper domain and the lower domain differ only in that the places in the upper model $\mathcal{A}^u$ are a subset of the places in the lower model $\mathcal{A}^l$. Then there is a unique mapping consisting of mapping models*

$$\{\langle \langle T^l, (\sigma^l_t)_{t \in T^l} \rangle, \langle T^u, (\sigma^u_t)_{t \in T^u} \rangle \rangle\}$$

that satisfy

$$T^u = T^l$$

$$\forall t \in T^u \sigma^u_t = \sigma^l_t|P^u$$

where P^u is the set of objects in $\mathcal{A}^u$ of type **place**

Roughly speaking, in the mapping of Example 1, one can obtain the upper computation from the lower computation by forgetting the contents of the places not in the upper domain.

The mapping comes into play in different ways in the two parts of the theorem of implementation correctness [7]:

- The *declarations* of the upper-level variables, including attributes such as bitstring length, range of arrays, and disjointness, must be implied by the declarations associated with the corresponding lower-level variables.
- The *semantics* of the image of the upper-level description under the mapping must be implied by the semantics of the lower-level description. In SDVS, the semantics are defined by the underlying logic of the verification system.

Whenever we are discussing computational models over the upper and lower domains, we will use $\mathcal{M}^u = \langle T^u, (\sigma^u_t)_{t \in T^u} \rangle$ and $\mathcal{M}^l = \langle T^l, (\sigma^l_t)_{t \in T^l} \rangle$ as the defaults, respectively. We will also use the state delta logic to describe the lower and upper computational models. We will use π^u (π^l) to represent sets of sentences from the upper (lower) logic.

Now we are ready to define formally the notion of implementation.

Definition 4 *π^l* implements *π^u with respect to the mapping μ, denoted by*

$$\mu \vdash \pi^l \hookrightarrow \pi^u$$

iff for any $(\mathcal{M}^l, \mathcal{M}^u)$ in μ one has

$$[\mathcal{M}^l \models \pi^l] \rightarrow [\mathcal{M}^u \models \pi^u]$$

From [7], we know that implementation satisfies reflexivity (with respect to the identity mapping, any sentence implements itself) and transitivity (if $\mu_1 \vdash \pi_1 \hookrightarrow \pi_2$ and $\mu_2 \vdash \pi_2 \hookrightarrow \pi_3$ then $\mu_3 \vdash \pi_1 \hookrightarrow \pi_3$, where μ_3 is the relational composition of μ_1 and μ_2.)

SDVS takes as input formal descriptions of the lower level, the upper level, and the mapping function, and *automatically* creates an implementation-correctness theorem in the underlying mathematical logic of SDVS. This theorem represents the claim that the lower level implements the upper level via the mapping. The command that does this work is called *implementation*; it encapsulates the fact that, under the right restrictions on the mapping, a proof of implementation can be carried out entirely in the lower-level language and logic. This new command embodies a more general and formal mapping definition than that used in previous versions of the system [8]. Having the implementation theorem introduced by the system rather than by the user leads to more reliable proofs, and using the more general mapping leads to broader applicability.

Other verification systems also deal with the issue of correspondence between levels. For example, in FDM, Ina Jo produces a theorem of implementation based on a mapping in a manner similar to that used by SDVS; see [9] for an informal description.

Figure 1 illustrates how SDVS interfaces to the implementation correctness process. These elements are discussed briefly in the following sections.

2.1 State Deltas

SDVS checks proofs of state deltas [10]. For example, SDVS can handle proofs of claims of the form "if P is true now, then Q will

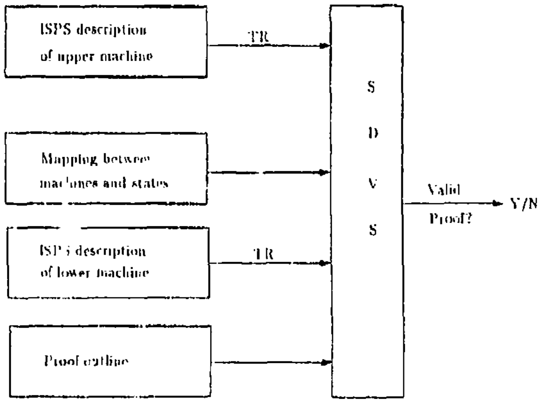

ISPS description of upper machine
TR
Mapping between machines and states
ISPS description of lower machine
TR
Proof outline
SDVS
Valid Proof?
Y/N

Figure 1: Implementation-Correctness Verification as Performed by SDVS. The user provides the two ISPS descriptions and "TR" translates them into state deltas. The user constructs a mapping from variables and states of the upper-level description to variables and states of the lower-level description. SDVS creates the implementation-correctness theorem and the user inputs the outline of the proof, which the system completes and checks for validity.

become true in the future." If P is a program (perhaps with some initial conditions) and Q is an output condition, then the above claim is an input-output assertion about P. SDVS can also prove claims of the form "if P is true now, then Q is true now." In this case, if P is a program and Q is a specification, then the claim asserts the correctness of P with respect to Q.

A computation (a sequence of state changes) is specified by a state delta or set of state deltas. In order to facilitate compact descriptions, the user specifies which variables change value as the state changes from one in which P is true to one in which Q is true. Thus, the true statements involving variables that do not change will remain true in the new state. In particular, if it is specified that *no* variables are allowed to change as the state changes from P to Q, then Q must be true in the state satisfying P, and the meaning is simply the static claim that P implies Q.

This discussion provides only an intuitive notion of state deltas. For a rigorous definition and examples, see the SDVS user's manual [11].

2.2 Describing the Levels of Implementation to SDVS

The mechanics of verifying implementation correctness using SDVS involve the automated translation of specifications written in a programming language to the underlying formal logic. Thus, SDVS can take as input the actual description used by the computer designers, as well as the actual program to be executed on the computer. Other systems require that the description or associated comments be written in the underlying language of the verification system. While SDVS currently only accepts descriptions written in ISPS [12] (which is suitable although not ideal for microprogram verification), we recently developed a formal technique for specifying the SDVS translation of other languages [3], and are applying it to programming and hardware-description languages.

2.3 Proving the Theorem of Implementation

Because there is no general procedure for deciding the truth of theorems in computer verification, many proof strategies are possible. An important characteristic of SDVS is that its proof strategy is a practical compromise between completely automated proofs and completely manual proofs. A powerful proof system that requires no guidance by the user can waste a lot of time trying approaches that an experienced mathematician would easily see are inappropriate. Alternatively, a system that checks step-by-step proofs input by the user executes very quickly, but the tedium of dealing with every minor step of the proof wastes the user's time. With SDVS, the user provides a high-level proof strategy, additional facts useful for the proof, and "hints" to aid the system in specific proof steps, while the system handles the details of the proof automatically. This compromise strategy is also successfully used by other formal proof systems [13][14]. SDVS makes this strategy particularly effective by allowing the user to interact with the system. The interface to SDVS was significantly enhanced in the latest version, giving the system the ability to act as a data-base manager for proofs.

The proof language itself is divided into static and dynamic parts. The static part deals with proving that certain assumptions imply certain conclusions about a given state. The dynamic part controls the state transitions made by the system. It includes constructs for proof by symbolic execution (corresponding to sequential execution), proof by cases (corresponding to branching), and proof by induction (corresponding to loops). When execution has arrived at a new state, a static proof may be needed to verify that new relations do in fact hold (in order to show that the postcondition is true and the goal is reached, or to show that a precondition is true and a new state delta may be applied).

For simple theories where efficient decision procedures exist and are implemented, SDVS derives all conclusions without any user-input proof. Examples of such theories are equality over uninterpreted function symbols and some fragments of naive set theory. For more complicated domains, the system allows the user to write proofs by having the system notice more and more difficult conclusions, where the newly verified conclusions are saved and used as lemmas on which to base the next conclusion. The derivation from a given set of lemmas to the next conclusion may be automatic in some cases, or it may require the user to designate that an axiom or a previously proved lemma is to be applied.

SDVS provides an abstraction mechanism for combining a sequence of state changes into a single state change [15]. This is crucial to managing large proofs. The system may be run in interactive mode, in batch mode, or (as in most real applications) as a combination of the two. In the interactive mode the user writes the proof in SDVS with help from system prompts, the system executing each proof command as it is written. Expressions are written in standard infix notation (e.g., $x + y$). In the batch mode the proof is written by means of the editor and is then executed by SDVS with no further user interaction. Most commonly a proof is written interactively, stored, and later rerun in batch mode.

3 A Case Study

Although several examples were developed and verified (e.g., [16]) during the development of a prototype version of SDVS, the first significant application of SDVS was the C/30 Microprogram Verification Project, begun in October 1984. This project, which was completed in November 1986 [17], used SDVS Version 5 to develop a formal correctness proof of 118 of the 128 instructions (implemented by over one thousand microinstructions) in the C/30 repertoire. Six C/30 instructions were found to have been implemented incorrectly by the microprogram. These errors were confirmed by the responsible engineer (they were corrected in a later release of the C/30 microprogram).

The C/30, specifically designed to serve as a packet switching node on the Defense Data Network (DDN), is one of a family of computers developed by Bolt, Beranek, and Newman (BBN). The C/30 is a 16-bit/word machine with 64K words of addressable memory and three addressing modes. It has a number of special-purpose and general-purpose registers, and a set of 128 instructions, including sophisticated instructions for manipulating queue data structures and controlling multiprocessing. It also operates a polled interrupt system, with clock, I/O and scheduling interrupts. The ISPS description of the C/30 (approximately 30 pages of text) was based on the reference manual and validated through simulation. Ten of the 128 instructions in the C/30 instruction set were not considered for verification, including instructions that manipulate the I/O system of the C/30 (these instructions were incompletely documented and thus difficult to specify formally). As there was not enough time, four other instructions were not verified.

The C/30 is implemented by a microprogram that executes on BBN's Microprogrammable Building Block (MBB) processor [18]. The C/30 implementation was chosen for verification because of an interest in verifying certain aspects of DDN, and because of the existence of a formal ISPS description of a version of the MBB. This description (coincidentally also approximately 30 pages of text) was partially in existence at the time, and was validated through simulation and through discussions with BBN.

The C/30 implementation correctness theorem is a formula that describes how the C/30 is implemented by the MBB. Using SDVS Version 5, we constructed this theorem by hand. Its major components are as follows:

1. the behavior of the C/30 in state deltas (the translation of the ISPS description into state deltas)
2. the behavior of the MBB in state deltas (the translation of the ISPS description into state deltas)
3. the actual microprogram in binary form
4. a mapping of the corresponding registers and other storage elements (called *carriers* in ISPS) of the C/30 description to those of the MBB description

Figure 2 illustrates the functional relationships of the elements of the theorem.

Writing the theorem of implementation and the high level proof in the interactive mode required about 9 MTS months of effort. A final check on the validity of the proof was made by SDVS in the batch mode. This SDVS validation of the proof executed for approximately 85 hours on a Symbolics 3640 Lisp Machine. Even without subsequent improvements that were incorporated into SDVS and upgrades to the Symbolics system, these figures demonstrate the feasibility of verifying large and complex microprograms. If this process were to be integrated into the development cycle of microprogrammed machines, it would greatly improve current debugging/testing practice, which often continues into the applications phase of the product.

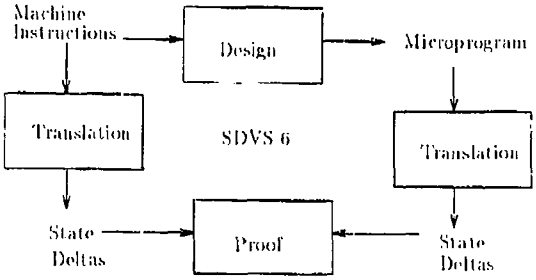


Figure 2: Microprogram Verification Using SDVS. Computer designers implement a computer instruction set by designing a computer (a set of data paths and associated control hardware) together with a microprogram that controls the movement of data within this computer. Descriptions of this instruction set and its implementation are translated by SDVS into formal state-delta semantics. The proof of the implementation theorem is performed in the mathematical domain of state deltas.

4 Towards Implementation Correctness at Multiple Levels of Design Detail

Because errors may be introduced at any level of refinement, a verification system that concentrates on a specific level of design detail is necessary but not sufficient for the design methodology of critical applications. During *any* refinement step it should be possible to verify that the new, lower-level specification that has been developed is consistent with the higher-level specification.

To date, formal verification systems have focused on specific steps in the refinement process. As we saw above, SDVS was developed to verify the microprogrammed implementation of computer instruction sets. Other formal verification systems in use have concentrated on verifying the refinement step of going from an abstract specification of behavior to a system-level design.

Figure 3 illustrates the process we envision of applying a further enhanced version of SDVS to verifying implementation correctness throughout the design cycle. The right arrows in the figure represent steps in the computer-system-implementation (synthesis) process. Some of these steps are performed by means of automated tools (e.g., by the compilation of Ada programs into machine language) and some are performed manually (e.g., by the implementation of microinstructions by means of data paths and associated control). For every synthesis step there is a corresponding implementation-verification step. The SDVS approach allows this verification step to be performed using the actual representation of the design levels that were used in the synthesis process.

We envision the use of formal top-level specifications, Ada programs, ISPS hardware descriptions, and HDL circuit descriptions. We successfully tested the following assumptions in the C/30 case study, and expect that they readily extend to the multilevel verification environment:

- The translation of the design representations into state deltas is faithful.
- The mapping between levels of design representation is expressive enough to capture the correspondence at each level.
- The correctness theorem produced by the *implementation* command represents the formal notion of implementation.

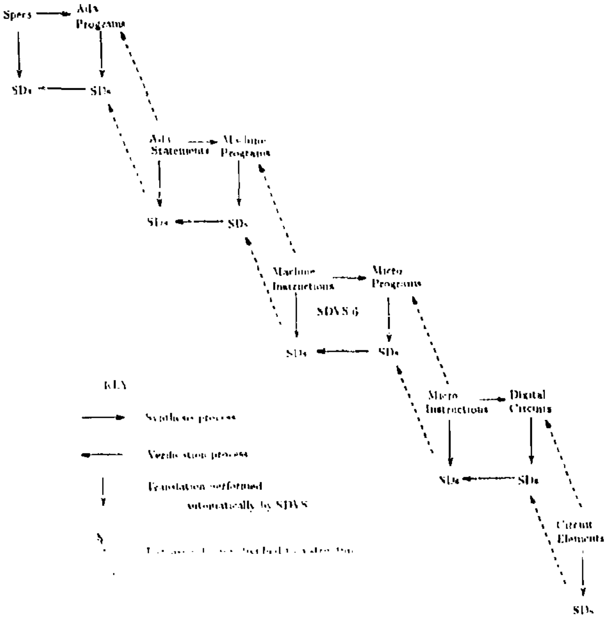


Figure 3: Multilevel Verification Process in SDVS. Each step in the implementation process is verified by a process similar to that in Figure 2, except that the domains of the descriptions correspond to the particular level of detail.

If we apply our microprogram verification approach to developing the implementation-verification environment in other domains, then multilevel verification within the design environment is feasible.

5 Conclusions

Since its inception SDVS has focused on microprogram verification and has been successful in that endeavor. The tangible results are the SDVS tools, which have been applied to verifying the implementation of the C/30. Such a proof of implementation correctness is important in secure or critical applications in which the discovery of errors as late as the operational stage is especially costly.

By its general design SDVS is amenable to implementation proofs at *multiple* levels of design detail. Such proofs are made feasible by our efforts to expand the set of description languages processed by SDVS [19]. Achieving this expansion will enable SDVS to take as input the actual formal design description used in the CAD or CASE system for the synthesis of the computer. SDVS adds logical rigor to this description by translating it automatically into the underlying formal logic of SDVS, and thus implementing a formal semantic definition of the description language. The implementation theorem is also produced automatically by SDVS to reduce reliance on manual construction of the correctness criteria. Finally, the outline created by the user to guide SDVS through the proof of the implementation theorem can be followed by a sophisticated reader. These features make SDVS a promising tool for increasing the confidence in the correctness of a computer design and its embedded programs throughout the overall life cycle.

Acknowledgments

The author thanks Jeff Cook, Beth Levy, Leo Marcus, and Mike Meyer for their editorial contributions to this paper, and thanks the first three above and the other members of the SDVS development group -- Tim Aiken, Dave Martin, and Timothy Redmond -- for their technical contributions to the system that will make implementation-correctness proofs feasible and believable at multiple levels of design detail in the future.

References

[1] L. Marcus, "SDVS 1987 final report," Tech. Rep. ATR-86A(2778)-5, The Aerospace Corporation, 1987.

[2] J. V. Cook, L. Marcus, and T. Redmond, "SDVS version 6 documentation and source code," Tech. Rep. ATR-86A(2778)-6, The Aerospace Corporation, 1987.

[3] D. F. Martin, "A preliminary formal description of the incremental translation of ISPS into state deltas in the State Delta Verification System," Tech Rep. ATR-86A(2778)-7, The Aerospace Corporation, 1987.

[4] D. F. Martin, "A formal description of the incremental translation of Core Ada into state deltas in the State Delta Verification System," Tech. Rep. ATR-88(3778) 1, The Aerospace Corporation, 1988.

[5] A. Birman and W. H. Joyner, "A problem reduction approach to proving simulation between programs," *IEEE Trans. Software Engineering*, vol. SE-2, pp. 87 96, June 1976.

[6] T. Redmond and L. Marcus, "Implementation mapping between levels and state deltas," Tech. Rep. ATR-86A(2778)-2, The Aerospace Corporation, 1987.

[7] T. Redmond and L. Marcus, "Mapping between levels and proofs of implementation," Tech. Rep. ATR-86A(8554)-5, The Aerospace Corporation, 1987.

[8] J. V. Cook and B. H. Levy, "Mapping in versions 5 and 6 of SDVS," Tech. Rep. ATR 86A(2778)-8, The Aerospace Corporation, 1987.

[9] J. Scheid, R. Martin, S. Anderson, and S. Holtsberg, "INA-JO specification language reference manual release 1," Tech. Rep. TM 6021/001/02, System Development Corporation, 1986.

[10] S. D. Crocker, *State Deltas: A Formalism for Representing Segments of Computation*. PhD thesis, University of California, Los Angeles, 1977.

[11] L. Marcus, "SDVS 6 Users' Manual," Tech. Rep. ATR-86A(2778)-4, The Aerospace Corporation, 1987.

[12] M. R. Barbacci, G. E. Barnes, R. G. Cattell, and D. P. Siewiorek, "The ISPS computer description language," Tech. Rep. CMU-CS-79-137, Carnegie-Mellon University, Computer Science Department, Aug. 1979.

[13] R. L. Constable and S. D. Johnson, "PL/CV2 program verifier reference manual," tech. rep., Cornell University, Mar. 1978.

[14] D. C. Luckham *et al.*, "Stanford Pascal verifier user manual," Tech. Rep. STAN-CS-79-731, Stanford University Computer Science Department, 1979.

[15] T. Redmond, "Composition of state changes and program verification," Tech. Rep. ATR-86A(2778)-3, The Aerospace Corporation, 1987.

[16] B. H. Levy, "Microcode verification using SDVS: The method and a case study," in *17th Microprogramming Workshop*, pp. 234-245, IEEE, Oct. 1984.

[17] J. V. Cook, "Final report for the C/30 microcode verification project," Tech. Rep. ATR-86(6771)-3, The Aerospace Corporation, Sept. 1986.

[18] A. Lake *et al.*, "Flexible processor extends design options," *Computer Design*, pp. 181-186, Nov. 1981.

[19] B. H. Levy, "Inadequacies of ISPS as a specification language for microcode verification," Tech. Rep. ATR-86A(2778)-1, The Aerospace Corporation, 1987.

Code Verification

Richard Platek

Odyssey Research Associates, Inc.
301A Harris B. Dates Drive
Ithaca, NY 14850

Unlike many hard core verification people, I believe higher level verification is important in order to derive the relevant code level specifications and relate them to system properties. I don't believe any existing system does a good job of this at all. Without formal support for such modeling, analyses, and proofs how does one know what one needs to prove at lower levels? Also, higher level verification might show which program units need to be verified to ensure a given property. Why verify more than you need to?

High level verification is not really distinct from code verification. In both cases, there is some mathematical object representing "the system", some formal axioms on that object representing what is known or being assumed about "the system", and some formal statement of what one wants to prove about "the system". The difference between what I'm calling high level verification and code level verification is that at the code level, one has more axioms about the system than one has at the high level (namely, a formalization of the axiom "the system is running the following code ..."). I see verification as a continuum from high levels of abstraction to low levels (where by "low levels" I mean microcode and hardware, not just programming language code). The problem with a lot of "high level verifications" in the past is not that they're too high level, but that the axioms which were assumed about the system at the high level were simply *not true* of the actual system. (Of course, doing everything at a high level does make it harder to see when your axioms are false).

The danger with code verification is that it's hard to see at the code level whether the properties you're proving about your code are the ones you're really interested in. With high and low level verification in a continuum, the properties of the system that the verifier really wants can be written down, and the proof of this property can be reduced to establishing certain facts about the code which, at the high level, are assumptions. Code level verification then discharges these assumptions on the basis of other assumptions about compilers, microcode, etc. The other benefit of such a process is that the verifier might find that he can establish the desired high level property without making any assumptions about certain parts of the code. He then knows that he need not verify those parts.

My answers to the specific questions proposed will focus on those about the value of verification and about the best places to put our efforts.

Is Code Verification Desirable?

Yes, as long as we know what we're proving. Code verification should not be a source of false confidence.

Is Code Verification Feasible?

A number of attempts to verify code (the Gypsy EPL and Message Flow Modulator to name two) and others to verify hardware have been successful. As has been pointed out before, the chief problem with many of these projects is not that they weren't successful but that the resulting code never got used and compared with unverified code to do the same jobs. I think these projects demonstrate the feasibility of code and even lower level verification.

What are Suitable Applications for Code Verification?

I don't know what applications are most suitable. It's clear that certain ones such as hard real time and floating point data types have received only limited attention. In the area of verifying security properties, something smaller than a full operating system, like a network gateway, would be a prime candidate.

Other Questions

How close are we? The key word in your question is useable. Useable by who? Unlike you, I won't go on record at this point.

When could something useable be available? I think that is largely a question of funding and the latter is largely a question of how people perceive the benefits.

There is no doubt in my mind that if IBM were to think verification was important we would have very useable systems in five years **and none of the people you addressed your call to would have worked on them.** It's all a question of perceiving the value. Perhaps we should add to your list the question:

What kind of near term demonstrations would positively affect the evaluations of verification within?

The issues of integrating verification into software engineering environments is important but demands an hour itself.

CODE-LEVEL VERIFICATION

(A Position Paper)

Friedrich W. von Henke
SRI International

What is "Code Verification"?

Formal verification of programs started in the 1970s as verification of programs (or subprograms) written in a programming language such as Pascal and augmented by specification constructs; thus, program verification originally meant verification of program code. Later, the main focus of verification work shifted to the specification and formal analysis of software at more abstract levels, and it became common to make a distinction between *design verification* and *code verification.*

For the purpose of this position paper, I want the term *code verification* to mean "formal consistency analysis of *executable, operational* code, usually with the help of mechanized tools." This implies that code verification deals with *real* programs written in a programming language in use for production software. The actual language may be a "higher-level" language like Ada, or a language closer to the machine, like C or even an assembly language.

In contrast to code verification, I use *code-level verification* to include formal analysis of programs in a setting that is more idealized, either by reducing the programming language to an impractical (and unrealistic) subset, or by using a language that has been designed specifically for verification purposes, usually in conjunction with, or as part of, a specification language. For example, the EHDM language [1] includes constructs that closely model the main imperative features of (sequential) Ada, such as objects, assignment, and control flow constructs.

Why Code Verification?

Code (or code-level) verification provides the link between designs and design specifications and what gets actually executed on a computer. Thus, code-level verification is clearly desirable and needed. On the other hand, code-level verification by itself is not very meaningful. The specifications against which the code is to be verified must have been derived from the requirements on the system and the system design. Indeed, most of the modeling and analysis of system properties (such as security) should happen at higher (i.e. design) levels; code-level verification should be restricted to those aspects that cannot be dealt with at more abstract levels, but are sufficient to ensure that the operational code indeed possesses the properties derived at the higher levels. Code-level verification also assumes that the programming language in which code is written has a formal semantics on which the formal analysis can be based. Moreover, the compiler of the language must faithfully implemented that semantics, so that execution of the object code reflects the semantics of the verified source. In short, code-level verification is one of several levels in the specification and verification of a system that ideally extend from the top-level requirements to the computer hardware.

Feasibility of Code-Level Verification

The basic technology for code-level verification has been in existence since the late 1970s when the first verification systems became operational. (In my opinion, little progress has been made since then in the verification of *code.*) Thus, in an elementary sense, code-level verification is "feasible," and has been for quite a while. On the other hand, it would be highly unrealistic to claim that the present state-of-the-art allows us to take an arbitrary program written, say, in Ada and "verify" it. There are quite a number of areas that still require extensive research and development efforts; for example:

- Likely application areas for code-level verification, such as real-time behavior and distributed systems, are lacking the mathematical foundations for formal specification and reasoning and/or the mechanical support for substantial verification efforts. For concurrent and distributed systems, a substantial body of theoretical results exists, but none of the existing verification systems that I am aware of adequately supports reasoning about such systems.
- Code-level verification must support more realistic subsets of the actual programming language.
- Tools and techniques need to be refined, based on extensive experience, to make them really practical.

In short, substantial research and development efforts are still required for realistic code verification.

In addition, alternatives to code verification need to be explored further. For example, the approach taken in the EHDM system [1] is to avoid formal reasoning about actual program text completely. Instead, the specification language itself includes constructs sufficient to specify concrete algorithms at the same level of detail (and, if desired, in the same imperative style) as actual code, and all formal analysis is carried out within the same formal system as the design verification. Once such a code-level specification or "abstract program" has been demonstrated to be consistent with higher-level specifications, it can be converted into Ada text by a translator provided in the EHDM environment.

Such an approach appears more in line with a systematic development process that involves design specifications and verifications; the programming language code is regarded as the *target* of a systematic development, in contrast to the traditional view that takes it as the starting point of the verification effort. However, this approach does not eliminate the inherent complexity of code-level verification; what can be avoided is having to deal with some of the idiosyncrasies of programming languages designed for efficiency (of compilation or execution) rather than logical clarity. (The current EHDM-to-Ada translator is an experimental tool and far from being practical, in particular since the Ada subset that can be modeled in EHDM is still unrealistically small.)

Like all formal verification, code-level verification is very expensive. I doubt that this will change in the near term, even with substantial efforts to develop better tools and techniques. Verification deals with *semantic* issues, which are inherently complex (and often intractable, i.e. unsolvable, in full generality). Thus, fully automated tools that can be used as easily as a compiler will be developed only for small, well-understood - and well-formalized - areas, and formal verification will remain an interactive, labor-intensive activity.

This inherent cost obviously reduces the usability of formal code-level verification. If much of the formal analysis has been carried out at higher (i.e. design) levels, the additional expected benefit derivable from formally verifying the code may not be large enough to justify the cost, except for small, crucial code segment. For example, correct behavior of the separation kernel in a secure system is critical and should be subject to formal verification. This example indicates that it may be more important to focus code-level verification efforts on low-level programming languages, which are more likely to be used for critical code segments than higher-level languages such as Ada.

For general applications, a more cost-efficient alternative may be to use less formal, but nevertheless rigorous, methods such as IBM's Clean Room methodology or VDM. Experience with these methods demonstrates the importance of educating the potential users for the usefulness and acceptance of formal methods.

Code-Level Verification and Software Development

It is obvious that the real benefit is derived from formal specification and verification when it is part of the general system development process and when the verification tools are integrated with the software environment. As a last step in a development based on formal specifications, the specifications derived at the lowest design level can be used to guide the production of the actual code; consistency between the code and the specifications can then be checked by code-level (or code) verification. I'd hope, however, that future development of formal methods and associated support tools will shift the perspective from analytic to synthetic approaches, so that code will be *constructed* from low-level specifications and code verification in the traditional sense will become obsolete.

References

[1] F. W. von Henke, J.S. Crow, R. Lee, J.M. Rushby and R.A. Whitehurst: EHDM verification environment: an overview. These Proceedings.

HOW SOON FOR CODE LEVEL VERIFICATION

A Position Paper

Richard A. Kemmerer

Department of Computer Science
University of California
Santa Barbara, CA 93106

The following are my responses to the questions that were posed to the panel.

Is code level verification feasible? Desirable?

There is no doubt that code level formal verification is feasible. Unfortunately, the conclusion that I made after working on the formal verification of the UCLA Data Secure Unix project in 1978 still holds. That is, the techniques necessary for formal verification (both design level and code level) are available; what is needed is engineering to produce production quality formal verification systems. What I realize now is that to achieve a production quality product will require a profit oriented company to decide that they are interested in formal verification.

The answer to the desirability question is an obvious yes. Of course it is desirable to increase one's assurance that a piece of code will perform as expected.

What applications are most suitable for code level verification?

Formal specification and verification should be used in all software development projects. This does not mean that it is necessary to use a formal verification system, but that one should reason about their programs and use formal notation as their design documentation. Many of the problems that occur in software development projects are the result of programmers rushing off and generating code without thinking about the design. By using formal notation for the design documents one can reason rigorously about the design (again not necessarily with the aid of a verification system) before writing any code. When the code is finally produced, they can then formally verify that the code is consistent with its specifications.

How close are we to having usable verification systems for code level verification?

As was mentioned in the response to the first question, the necessary techniques have been available for more than a decade. However, what is needed is an interested party, that is willing to take the research tools and turn them into a product. None of the existing formal verification tools can be considered to be a product (whether they can accommodate code level verification or not).

I am not necessarily suggesting that one of the existing tools be used as a base, but rather that the experience gained from using these tools be applied to a new product. I also feel that this will be carried out by a profit oriented corporation and not by one of the research groups that are responsible for the existing tools. The stress here is on "development". The know-how already exists, what is needed is the development of a friendly human interface, good documentation, production quality packaging, marketing, etc.

Regarding the time, three to five years is needed to produce a product. However, what is needed first is the desire to have a product or the vision to see its marketability.

How will code level verification fit into the software development process?

As was mentioned in the response to the first question, the only way to develop systems is to use formal specifications as the design notation. This can be in the form of a rigorously defined language whose specifications are machine checkable or in more general notation, such as the set notation of Z, which is not currently machine checkable. These formal specifications allow the designers to rigorously answer questions about their designs. The higher level design specifications should be refined as the design progresses. Finally, the lowest level design should be used to generate the necessary entry and exit assertions for the code. This would be similar to the approach I used for the secure terminal with Ina Jo (See the Verification Assessment Report Volume IV.).

How expensive will it be in people time, machine time and money?

I believe that one of the biggest expenses will be in training software developers to use formal techniques. However, once they are trained the increased reliability of the software will counter any added development cost. In addition, when the system is in the maintenance phase the cost of maintenance should be reduced due to the availability of the unambiguous formal documentation. Therefore, the cost over the lifetime of the system should not increase significantly, if at all.

Afterthought

I believe it is worth mentioning that one can not expect to take an already existing piece of code and formally verify that it does what it is supposed to. If a system is to be formally verified it is necessary to plan for the formal verification from the start of the project. Before any code is written it is necessary to sit down with the developers and discuss the coding practices that should be adhered to to simplify the formal verification process. For instance, if the developers are planning to use pointers in their implementation the cost of formally verifying the code is going to increase, and if they plan to use pointer arithmetic the task may be infeasible. In like manner, using a language feature like the Pascal variant record will increase the cost of the formal verification process.

How Soon for Code Level Verification?

Stephen D. Crocker

Trusted Information Systems
11340 Olympic Blvd.
Los Angeles, CA 90064

Twenty years have passed

This panel was born out of sense of frustration and wonder. It's been two decades since Floyd published his landmark paper on program verification. [Floyd 67] Shortly thereafter, King completed the first working program verification system [King 69], and the field was in full bloom. Further landmark work followed from Hoare, e.g. [Hoare 69] and a number of verification systems were built. With that start, we might expect that twenty years later program verification would be an established technology. *However, to my knowledge, not a single program operating in the field has been verified at the code level!*

To be fair, there has been considerable progress in verification over the past twenty years. A very substantial fraction of the financial and intellectual resources of the field have been focused on two particular aspects of verification, design verification and programming language semantics.

"Design verification", as most of this audience knows, is verification of the consistency between an abstract design of a system and a formal statement of the security policy the system must adhere to. Code level verification can only prove the consistency between the code and a specification. If the specification is incorrect, the code level proof will not be of much use in establishing the trustworthiness of the code.

One community that recognized the importance of verification was the security community. The key concern within the security community is that systems protect information that is entrusted to them. Hence, no matter what else the system is supposed to do, it is obligated to protect the information in it.. It is no surprise, then, that considerable attention was given to the assuring that the overall design of a system adhered to the security requirements in particular.

This attention to design level verification has been so substantial that it completely overshadowed the original focus on code level verification. When the Trusted Computer System Evaluation Criteria [TCSEC] was drafted, the technology to support design level verification appeared mature enough to include in the criteria, but code level verification technology was not. The Criteria contains a place holder called "Classes Beyond A1." The criteria applicable to these classes have not been determined fully, but verification down to the code level is clearly expected: "The TCB must be verified down to the source code..." [TCSEC, page 53]

The verification community also turned its attention to the semantics of programming languages. Fortran, Cobol and PL/I dominated the set of programming languages in common use in the late 1960s and early 1970s. Different branches of DoD were using languages of their own invention -- JOVIAL, CMS-2, CS-4, TACPOL, etc. In all cases, the semantics of the languages were difficult to formalize. The only definitive authority for the meaning of a particular program was how it executed when compiled by a particular compiler and executed on a particular computer. A fraction of the verification community focused on how to formalize the definition of programming languages and how to design programming languages with precise semantics. It is perhaps instructive to note that while these efforts have yielded major new insights into the mathematical foundations of programming languages, the two most widely used new languages are Ada and C, neither of which is widely regarded as having a significantly better defined semantics than older programming languages.

Meanwhile, experience has been gained with using design verification to provide assurance that a computer system will protect information entrusted to it. The experience is mixed, and some question whether the process adds much assurance at all. See, for example, [Schaefer 88] for a discussion of this point. But disregarding the question of whether it's beneficial, the process is undeniably expensive and error-prone. Design verification is necessarily disconnected from the code. This means that efforts to maintain correspondence depend on a great deal of labor and a great deal of discipline. And, of course, even with full attention and discipline, the formal basis for the design and the code semantics may be incompatible in some respects. That is, the assumptions in the formal design may not match the facilities provided by the hardware and/or compiler.

Focus of this panel

With this background, it is perhaps time to re-examine the original vision of code level verification. To stimulate discussion on this point, I proposed this panel and invited a number of leading verification researchers to participate. The invitation included the following:

How Soon for Code Level Verification?

The exciting payoff for verification will be its application to code level proofs where it will provide strong correspondence between operational code and top level specifications. The purpose of this panel is to bring together practitioners of the art -- particularly verification system developers -- and to describe the steps toward achieving wide-spread, efficient code level verification. Contrary views, e.g. code level proofs are impossible, code level proofs are undesirable, code level proofs will always be too expensive, will also be explored. To provide a coherent basis discussion, each panel member will address the same set of questions. The floor will then be open for the panel members to debate with each other and to respond to questions from the floor. The common questions include:

o Is code level verification feasible? Desirable?

o What applications are most suitable for code level verification?

o How close are we to having usable verification systems for code level verification?

o How will code level verification fit into the software development process?

o How expensive will it be in people time, machine time and money?

I asked further that prospective panelists agree to prepare short position papers, and that some discussion and debate take place prior to the conference so that we would present related and comparable views, although hopefully not blandly identical views.

Each of the panelists has played a major role in the development, assessment and/or use of verification systems in recent years. Specifically:

> Dan Craigen has been leading the development of the m-EVES program development system and previously participated in the development of Euclid.
>
> Dick Kemmerer worked on a major effort to verify a version of Unix, has contributed various techniques for analysis of information flow properties and analysis of Ada programs, and led an extensive effort to assess the current technology in verification, resulting in a five volume report [Kemmerer 86].
>
> Richard Platek is the founder and president of Odyssey Research Associates (ORA). Under his direction, ORA has become a major builder of verification systems.
>
> Friedrich von Henke is the prime architect of the Enhanced HDM (EHDM) at SRI International, and has been a leading contributor to the theory and development of verification systems for several years.

The views of these people are sketched in their position papers in these proceedings. Many other researchers have strong and relevant views, but the limitations of space and time restricted us to this panel. If the issues discussed by these panel members are deemed appropriate for further discussion, perhaps the debate will continue in another forum.

My own view

It should come as no surprise that I am optimistic about the prospects for code level verification.

Is code level verification feasible? Desirable?

The "desirable" part is easy, and answered in part in the description of design verification. A very large class of bugs will cease to exist when code is verified before it is fielded, and the resulting increase in reliability will be extremely valuable.

How feasible is code level verification? I take the somewhat radical view that programmers, on the average, know why they write the code they do, and that their knowledge, although informal and often unexpressed, constitutes the elements of a proof. Therefore, only the "trivial" task remains to provide tools to these programmers to express their knowledge in a formally acceptable way.

This view has a number of ramifications. First, it implies that the vast set of existing programs are fair game to specify and verify. The argument often heard that only programs written in new languages and following new paradigms are verifiable is misdirected. The fundamental reasoning processes for understanding programs already exist within the minds of work-a-day programmers. New languages and new paradigms may provide modest help, but the more useful challenge is to find ways to express what programmers already know.

Another ramification is that it is both possible and necessary for program verification systems to work with the same languages that programmers use to build real systems. If the verification community insists that new languages are needed, it has taken on the extremely hard problem of convincing the entire programming community to use those languages. Generally -- Ada excepted -- programming languages have been widely adopted only when they have provided a new level of expressive power and have had efficient translators.

Third, it seems entirely possible that "ordinary" programmers will use verification systems. Verification specialists -- at least in the sense of people who prepare specifications and proofs for verification systems -- may not be necessary in the future.

What applications are most suitable for code level verification?

Given that the central issue in verification is explicating and formulating the knowledge programmers bring to bear in the design of their programs, it makes sense to start with the domain of programs that embody the least knowledge: systems programs. All programs embody considerable knowledge of computers and computing. Applications such as weather prediction, inventory control, stock market analysis, etc. also embody considerable knowledge about domains well outside of computer science. System programs, on the other hand, have far less connection with the real world and are much more easily formalized.

How close are we to having usable verification systems for code level verification?

First, it is important to understand what's meant by "usable." I have in mind criteria similar to the use of a compiler or other tool. A verification system is usable only if...

...It can verify programs written in widely used programming languages, with essentially no restriction on the use of the language, i.e. without restricting the programming language to a weak subset.

...It comes with a specification language that permits pleasing and concise expression of the important behavioral and performance properties of programs.

...It comes with an algorithmic, i.e. predictable, proof system of sufficient power to keep the proofs short.

> Interactive theorem provers may be helpful for the same reason that interactive program development systems are, but a programmer should never be in doubt as to whether the proof system is smart enough to see the trust of a particular assertion in a given context. At the same time, it is not acceptable to achieve predictability and lose conciseness. A proof system that requires the user to supply every *modus ponens* and every instantiation is not usable.

...It is fast enough to be used unhesitatingly, i.e. as often as the user would ordinarily use the compiler.

> In specific terms, this means a modern day verification system needs accept a combination of specification, code and proof and check the proof within a factor of 3 or so as fast as the compiler would compile that program. I haven't check the speed of modern day compilers, but I believe they compile tens to hundreds of lines per minute. The factor of three is my rough guess as to what the user will perceive as comparable to the speed of compilation. Perhaps the factor should be less than 2 or as much as 10. In any case, a verification system is not usable if it takes hours to verify a program that compiles in minutes or seconds.

With all of these provisos, I believe it is possible to build usable verification systems in three to five years. To do so, it is important to limit the attention to existing programming languages and not design a new language in tandem with designing the verification system. It is equally important to set as a ground rule that the verification system will be predictable. Many important algorithms are known for implementing decision procedures, and many more are yet to be discovered, but these are, I believe, less important than providing an understandable interface for the programmer.

The last two questions, how will code level verification fit into the software development process and how expensive will it be to use such systems, have been answered at least partly. Well constructed verification systems will fit into the software development cycle at the same place and in the same manner as compilers. Code will be verified as it is written or changed, and if desired, reverified as part of acceptance procedures. More people and money may be needed to formalize what programs are supposed to do, but NO additional time or money should be needed to verify programs as they are being written. And, of course, the increase in quality of programs will lead to a substantial decrease in the cost of maintenance of programs.

References

Floyd 67 — Robert W. Floyd, Assigning Meaning to Programs, Proceedings of Symposia in Applied Mathematics, Vol. 19, American Mathematical Society, (1967).

Hoare 69 — C. A. R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, 12(10), October 1969.

Kemmerer 86 — Richard A. Kemmerer. Verification Assessment Study Final Report, Office of Research and Development. National Computer Security Center, C3-CR01-86, Library No. S-228, 204, March 1986.

King 69 — J. C. King A program verifier, PhD Thesis, Carnegie-Mellon University, 1969.

Schaefer 88 — Marvin Schaefer. Design Specification and Verification Issue Paper: Symbol Security Condition Considered Harmful, Technical Report TIS-R 152, Trusted Information Systems, Inc., March 1988.

TCSEC — Department of Defense Trusted Computer Systems Evaluation Criteria. DoD 5200.28-STD, National Computer Security Center.

Position Paper for Code Verification Panel

Dan Craigen

Research and Technology
I. P. Sharp Associates Limited
265 Carling Avenue, Suite 600
Ottawa, Ontario K1S 2E1
CANADA

The first point that struck me about the description of the problem to be discussed is the apparent merger of two orthogonal concerns: the use of formal methods during the programming development process, on the one hand, and the application of systems, which support the application of formal methods, on the other.

It is my belief that any self-respecting computing scientist should be able to informally (but rigorously) justify that their program satisfies the specification describing their task. Techniques have been available for well over a decade and new ways of reasoning about programs are continually being developed.

From this first perspective I would answer the four relevant questions thusly:

- **Is code level verification feasible? Desirable?** Yes and Yes.

- **What applications are most suitable for code level verification?** Most work to date has been based on variants of the pre-post form of verification. Other techniques such as CSP, CCS and Constructive Type Theory, are extending our manipulative capabilities. More work needs to be performed!

- **How will code level verification fit into the software development process?** There is nothing antithetical to good software development processes in the application of formal methods. One expects a powerful synergism.

- **How expensive will it be in people time, machine time and money?** I would expect that disciplined and rigorous development of systems would be more cost effective. However, I do not know of any studies which have investigated this contention.

On whether we should be applying verification systems to code level proofs. This is a more contentious issue and generalizes to whether current verification systems should be used at all. In other papers [1] it has been argued that the application of verification systems have three (putative) benefits: soundness, magnification and tracking. These properties are not as fundamental as the advantages that should accrue from disciplined thought but are, nevertheless, important.

So, returning to the listed questions:

- **Is code level verification feasible? Desirable?** With currently endorsed tools, both the feasibility and desirability is questionable. Desirability is for the customer to make an informed decision based upon their organization's mission. It is obvious that more powerful tools are necessary; especially in terms of specification and programming languages, mathematical logics, and supporting mechanized proving systems.

- **What applications are most suitable for code level verification?** Most work to date has been based on variants of the pre-post form of verification. Other techniques such as CSP, CCS and Constructive Type Theory are extending our manipulative capabilities. More work needs to be performed!

- **How close are we to having usable verification systems for code level verification?** Obviously, I have my own axe to grind here [1]. It is my contention that some current developments are moving towards systems which are based on sound mathematics and apply state of the art techniques in theorem proving and language design. Undoubtedly, since fixed formalisms and languages are chosen there will be problems which either cannot be addressed or will be handled in an unwieldly manner.

- **How will code level verification fit into the software development process?** Verification systems must be integrated with general software development tools.

- **How expensive will it be in people time, machine time and money?** I do not know of any studies that have investigated this issue.

Finally, while I have responded in terms of the panel, I should note that it is clear that there is more to formal methods than specification and code proofs. Formal Methods can be viewed as the application of mathematical discipline to the entire system development process. One of the more interesting projects taking this view is the Trusted Systems work at Computational Logic, Inc., with their work on micro-Gypsy, Piton and the FM8501. Others are better qualified to report on that work.

References

[1] D. Craigen, S. Kromodimoeljo, I. Meisels, A. Neilson, B. Pase, M. Saaltink. *m-EVES: A Tool for Verifying Software*. In Proceedings of the 10th International Conference on Software Engineering, Singapore, April 1988.

DESCRIPTION OF MULTILEVEL SECURE ENTITY-RELATIONSHIP DBMS DEMONSTRATION

William R. Shockley, Dan F. Warren
Gemini Computers, Inc.
P.O. Box 222417, Carmel, CA
(408) 373-8500

Abstract

This paper describes a demonstration, implemented as application code executed by untrusted subjects in the environment provided by a high-assurance security kernel (GEMSOS) that provides multi-level database management services using a significant subset of the entity-relationship data model. The demonstration includes the essential data management services one would expect to find in the run-time nucleus of a DBMS, supporting the concept of entities, relationships between entities, and the ability to merge data (as viewed by the user) from all visible sensitivity partitions. The merged data is logically integrated into a single data model for query and manipulation (consistent with the security policy) by the user. The significance of the demonstration is that this is one of the earliest demonstrations proving that significant programs managing complex data structures containing data of differing sensitivities can be efficiently implemented as an application executed by an untrusted subject constrained by a genuine high-assurance security kernel.

1. Introduction

The Multilevel Secure Entity-Relationship DBMS Demonstration (ER/DBMS) is a demonstration of certain of the technical concepts developed for the Secure Entity-Relationship Database Management System design produced by AOG Systems, Inc. and Gemini Computers, Inc. under contract to the Rome Air Development Center, (RADC), United States Air Force (Contract F30603-86-C-0117). The technical goals of the demonstration were as follows:

- To demonstrate that untrusted applications can share and manipulate data in the form of complex, multilevel, dynamic data structures with a high apparent granularity of classification.

- More specifically, to implement the critical components of the multilevel GTERM data model [1], including a demonstration of entities with attributes of differing sensitivity, relationships between entities of differing sensitivities, maintenance of an index structure for types of entities (namespaces), and the polyinstantiation of attributes (that is, support for multiple versions of the same attribute, visible to appropriately cleared users only).

- To demonstrate the transitions between a user's interaction with a Trusted Computer Base (TCB), untrusted applications, and trusted applications, including use of a trusted path, secure login and logout procedures, and changes in session level.

- To gain experience in the design of screen-oriented human interfaces implemented as an untrusted application presenting, to the user, data copied from repositories of various sensitivities.

As important to the design of the demonstration as its objectives are those capabilities chosen not to be demonstrated. The design simplifications chosen included the following:

- The database uses a fixed schema of three entity types and two relationship types representing a typical application: there is no capability to modify the schema.

- Only a single user is supported, so that concurrent manipulations of the database are not encountered.

- The database is low-volume (a maximum of 100 items of each type is supported) so that storage management is very simple.

- The only concern with algorithm selection and data structure design for processing efficiency was to avoid a demonstration that appeared slow.

- No support for multiple, concurrent transactions is provided: essentially, each logical query, update, or addition is treated as an atomic, sequentially-executed transaction.

- No support for the deletion of data is provided.

- Only two levels of data sensitivity (SECRET and TOP SECRET) are supported. The issues related to the potential existence of very large numbers of access classes are avoided.

- No discretionary access control policy is supported.

Although many of the design simplifications chosen allowed issues to be avoided that are clearly of concern for the design of a commercial quality multilevel DBMS, (notably, the need for concurrency and transaction support) the selection of those capabilities to be demonstrated, and those not, was deliberately made to support the primary goal of the demonstration -- to explore data structure solutions allowing data of multiple sensitivities to be re-assembled into an integrated view for an appropriately cleared user, and to allow the data structure to be consistently updated without violating the security constraints by an untrusted subject.

Some of the design simplifications (such as those avoiding concurrency controls) were made because we felt that adequate solutions are already known -- e.g., controls based on the use of eventcounts and sequencers or their equivalents ([2], [3], and [4]).

Other problems not examined (e.g., the ability to deal with large numbers of potential access classes) are important, but are not DBMS-specific problems and are currently being explored in other contexts.

In particular, untrusted algorithms for accessing and manipulating complex data structures containing data of varying sensitivity, linked into a complex semantic network, have not previously, to our knowledge, actually been implemented using untrusted processes executing on top of a genuine security kernel. At the time the demonstration was designed, it seemed of greatest engineering importance to demonstrate the feasibility of such algorithms.

The importance of actually implementing such algorithms on a genuine security kernel, is that one may be sure that no hidden assumptions or programming "workarounds" invalidating the design have been introduced into the demonstration code, thereby gaining a high degree of confidence that the algorithms and ideas being demonstrated are valid.

Our criterion of success for meeting this goal we defined as the ability to demonstrate the central features that differentiate the entity-relationship data model from other data models: namely, its presentation to the user of a "reference" from one item to another (rather than the implicit linking of data records by means of common data values), and its ability to present polyinstantiation in a natural way as an attribute with multiple values (rather than, as needed for the multilevel relational model, by introducing additional tuples with replicated data in the non-polyinstantiated fields). Thus, although only a subset of the data model was supported, it was that subset that includes the core concepts of the data model (entities, data representing relationships between entities, and the use of references to link together data items) and that distinguish it from the relational model.

The remainder of the document is divided into the following sections. The section entitled "Functional Architecture" provides an overview of the complete system. The section entitled "User View" summarizes the operations and capabilities of the demonstration from the users' point of view. The section entitled "DBMS Modules" describes the database management modules themselves in enough detail to provide a basic understanding of how data of different access classes is distributed into physical segments of the right access class and efficiently re-assembled in response to user queries. The final section contains our conclusions.

2. Functional Architecture

The demonstration was targeted for a single-processor Gemini system running the GEMSOS Mandatory Security Kernel, configured with a single user's terminal.

The demonstration was developed in the (untrusted) Unix V programming environment running on Gemini hardware. Although this is now the standard programming environment provided with Gemini systems for the development of GEMSOS applications, at the time the demonstration was implemented the Unix V environment was still under development (in fact, our project was the first "user" of the Gemini Unix V environment.) As the work necessary to provide the full run-time support needed for C programs in the GEMSOS environment was in progress at the time the demonstration was implemented, the demonstration software includes some components that would now be unnecessary because their functionality is available in the form of run-time library functions.

As an established goal of the demonstration was to embed the DBMS human interface in the context of a realistic "trusted" interface managing logon/logout, trusted path interactions, and so on, a relatively large subset of the demonstration software is devoted toward providing this functionality. As the DBMS is actually executed by genuinely untrusted subjects, much of this TCB functionality is "real" (although not meeting the rigorous software engineering standards expected for a Class A1 or B3 system), in the sense that a complete environment must be maintained that allows the DBMS subjects to execute without trust. For example, the terminal port had to be configured as a single-level device (i.e., not capable of associating labels with input-output data) with a multilevel range (as both SECRET and TOP SECRET sessions must be available from at the terminal.)

Because all interactions with the terminal are mediated by a genuine security kernel, and we needed to be able to have the terminal place its input data in either a SECRET or TOP SECRET input buffer (depending upon current session level) so that inter-process communication with the current untrusted DBMS would work, the correct labeling of the terminal port could not be "faked". Similar considerations pervaded the design of the "supporting environment" for the DBMS, and for all practical purposes, this environment may be regarded as a "real" TCB from a functional point of view. Although the pragmatic requirement to develop a "mini-TCB" for the security kernel was not specifically DBMS-specific work, it proved valuable in allowing the resulting demonstration to be used to demonstrate various issues concerning trusted path, login, and session level changes as a side benefit.

2.1 Process and Subject Structure

The demonstration is implemented using four processes. They are:

- an initial process (in execution as the kernel comes out of initialization);
- a (trusted) TCB process, which manages the terminal, provides input/output services to the DBMS processes, and manages all direct user interaction with the TCB (e.g., for logon/logoff);
- an (untrusted) SECRET DBMS process, which provides user access to the database during SECRET sessions, and
- an (untrusted) TOP SECRET DBMS process, which provides user access to the database during TOP SECRET sessions.

Both the initial and TCB processes exist for the lifetime of the demonstration (i.e., from the time the system is booted to the time it is turned off.) The SECRET and TOP SECRET processes exist only for the lifetime of an untrusted session (i.e., from the time the user requests such a session to the time the "secure attention key" is pressed). Although their stacks, code, and data segments are reused, logically when a new SECRET session is begun, a new SECRET process is begun: the previous SECRET process (if there was one) was actually terminated at the end of the last SECRET session.

The initial process is essentially a loader setting up the initial segment structure in main memory for the remainder of the system and spawning the TCB process, then deactivating. It will not be described in further detail here.

The TCB process is trusted over the range SECRET to TOP SECRET. The TCB process attaches and controls the terminal during the lifetime of the system: all I/O requests from an active untrusted process are served by the TCB process using interprocess communication and shared buffers of the appropriate sensitivity. Between untrusted sessions the TCB process provides a direct interface to the user in order to manage logons, logoffs, and requested changes in session level. When the user requests an untrusted session, the TCB process creates a child process of the requested level (consistent with the user's clearance) and acts as an I/O server for the child process. When the user presses the secure attention key this is detected by the TCB process, which orchestrates an orderly shutdown of the child process before returning to the user for a trusted interaction.

The untrusted session processes (whether SECRET or TOP SECRET) are event-driven transaction processors, providing complete access (consistent with session level) to that portion of the database the user is authorized to view and/or update. Both the SECRET and TOP SECRET processes execute exactly the same code: the sensitivity level of the process itself is available at run-time as an entry parameter to the process. Although the untrusted processors use this information to "navigate" through the usable segments of the database, they are not responsible in any way for enforcing the accessibility of data. If, for instance, the SECRET untrusted process attempted (erroneously or maliciously) to access a segment containing TOP SECRET data, the underlying security kernel would simply refuse the request. (In fact, a fair amount of debugging time consisted of determining why such a trap had occurred and modifying the untrusted process code to remove the offending request.)

Terminal input/output requests are passed to the TCB process for service when needed. The result of any request for service (whether input or output) may be an indication from the TCB that the trusted path was invoked -- in effect, notification to shut down. When this happens, the untrusted process swaps the database back to secondary storage and conducts an orderly shutdown. GEMSOS provides the mechanisms required for handshaking between child and parent to coordinate process termination. Although the untrusted process has responsibility for helping perform an orderly termination of itself, failure to meet these responsibilities are not insecure (they lead, at worst, to a denial of service or loss of data.) Once trusted path has been invoked, the TCB process will not perform I/O until the shutdown is complete.

3. User View

In this section, we present a brief summary of the view of data and human interface presented to users by the demonstration. Although many viewers of the demonstration (who have never actually operated a high-assurance TCB) found the operation of the "trusted path" with regard to boot operator authentication, user login/logout and authentication, and session-level negotiation fairly interesting, only the capabilities presented to the user during an untrusted DBMS session will be described, as it is these capabilities that are central to the demonstration goals.

3.1 Data Schema

As previously discussed, the schema for the data managed by the demonstration is not modifiable. Three types of entities (ordnance, aircraft, and city) and two types of relationships between entities (aircraft-carries-ordnance and aircraft-attacks-city) are supported. The data schema (presented in an ad hoc data definition language) is detailed in the Figure 1.

Although the schema is relatively self-explanatory, a few words about the entity-relationship data model seem in order. An "entity" is meant to represent an "object" (real or abstract) with a unique name (used much as is a primary key in queries) and some number of attributes. In the multilevel version of the data model, we allow each attribute to be polyinstantiated: that is, different values may exist in the database for a given attribute for an entity, corresponding to different sensitivities. (The name of an entity is always understood to implicitly contain a sensitivity -- so there may be distinct entities with the same name in the database, at different access classes, as well.)

```
define entity type AIRCRAFT
  primary key SIDE NUMBER
  attribute   CLASS
  attribute   PILOT
end AIRCRAFT

define entity type ORDNANCE
  primary key ID
  attribute   EXPLOSIVE
  attribute   TYPE
end ORDNANCE

define entity type CITY
  primary key NAME
  attribute   POPULATION
  attribute   LATITUDE
  attribute   LONGITUDE
end CITY

define relationship ARMED-WITH
  role          AIRCRAFT
  role          ORDNANCE
  attribute     NUMBER
end ARMED-WITH

define relationship ATTACKS
  role          AIRCRAFT
  role          CITY
  attribute     DATE
end ATTACKS
```

Figure 1. Data Schema

A "relationship" is meant to represent the association of one distinct entity with another. In our version of the data model, attributes may be associated with instances of relationships, as well. Each particular instance of a relationship has its own sensitivity (that of the subject inserting it into the database), and its attributes may be polyinstantiated. Thus, the collection of relationship instances a user will see as existing in the database will depend upon the session level the user has specified. The relationship instance is uniquely defined by the particular entities it refers to: thus, even though there might be two entities with the same apparent name (because of polyinstantiation), internally a given relationship instance will refer to just one of them. (This is rather more convenient than the analogous case for the relational model, where data is "linked" only be means of common, stored data values.)

The human interface presented to the user is screen-oriented and interactive. Essentially, the user may insert or update individual records (for a single entity or relationship instance) or cause the entire visible contents of a whole entity or relationship type to be printed on the screen at once. Data may only be updated (and/or inserted) at the user's current session level: if the user attempts to update an attribute at a lower level, the attribute is polyinstantiated instead.

A typical display that might exist for a user cleared to TOP SECRET, and in a TOP SECRET session, when the AIRCRAFT table is printed is shown in Figure 2.

AIRCRAFT Session Level: TOP SECRET

Side Nr.	Class	Pilot
304 (S)	B-52 (S)	Kelly (S)
320 (TS)	Stealth (TS)	O'Hara (TS)
327 (S)	B-52 (S) Stealth (TS)	Murphy (S)

(a) in TOP SECRET session

AIRCRAFT Session Level: SECRET

Side Nr.	Class	Pilot
304 (S)	B-52 (S)	Kelly (S)
327 (S)	B-52 (S)	Murphy (S)

(b) In SECRET session

Figure 2. Initial Views of AIRCRAFT

These views are what we would intuitively expect: the SECRET view of the database looks like the TOP SECRET view, but with all of the TOP SECRET data filtered out. It is emphasized, however, that what is really going on is that the (untrusted) DBMS is assembling this view based upon **all of the AIRCRAFT data it can find** -- it is the security kernel underneath that makes it impossible for the DBMS to find any TOP SECRET data during a SECRET session, because that data is stored in TOP SECRET physical segments. Thus, with high assurance, there is no way the SECRET view can encode or allow the value of any TOP SECRET data to be inferred -- nor does the DBMS code itself need to be analyzed in order to know this.

The views presented in the last figure also illustrate a few points of interest. Flight 304 represents a SECRET flight -- that is, all of the data about this flight was entered by some user during a SECRET section. Similarly, Flight 320 is a TOP SECRET flight. Flight 327 is more interesting: at a SECRET level, the Flight appears to be an ordinary SECRET Flight. However, at a TOP SECRET level (viewable only to users with TOP SECRET or greater clearances) we see that the "Class" attributed is polyinstantiated: that the aircraft class is "B-52" is (apparently) a "cover story": the aircraft's real class is "Stealth" -- a TOP SECRET datum. This database state could be reached in the following order by a user cleared TOP SECRET. First, the user asks for a SECRET session and enters the SECRET record using the "cover" data. Then, that user obtains a TOP SECRET session and enters the TOP SECRET information as an update to the SECRET record. The database polyinstantiates the value (because it is untrusted, it would be unable to modify or delete the SECRET attribute in any event.)

Let us now suppose that there is a change of pilots for Flight 327 from Murphy to O'Neil. A user cleared SECRET is responsible for entering this update. That user (who has no idea that there is TOP SECRET data associated with this flight) obtains a SECRET session and makes the update in the expected way -- by calling up the record for Flight 327 and changing the pilot data. The new view of the record is as shown in Figure 3 for SECRET and TOP SECRET users.

The most important point to note is that although the data modified is SECRET, the change is instantly propagated to the view of data given to the TOP SECRET user. The point is that we are dealing not with replicated data, but with one version of data properly arranged by access class being integrated dynamically by the DBMS in response to queries. It is our belief that to be useful, a "multilevel" DBMS must have exactly this sort of semantics on update: the ability for a "multilevel record" as viewed by a high-level user to respond "instantly" to changes made to its lower-level components.

AIRCRAFT Session Level: TOP SECRET

Side Nr.	Class	Pilot
327 (S)	B-52 (S) Stealth (TS)	O'Neil (S)

(a) In TOP SECRET session

AIRCRAFT Session Level: SECRET

Side Nr.	Class	Pilot
327 (S)	B-52(S)	O'Neil

(b) In SECRET session

Figure 3. Updated views of Flight 327

Similar examples drawn from a relationship type will not be given here, as relationship data and modifications to it work in a similar way.

4. DBMS Module Descriptions

In this section, the heart of the demonstration system (the untrusted application modules responsible for actually maintaining and retrieving information from the database) are described, essentially in "bottom up" order. Details concerning those untrusted modules responsible for providing I/O and high-level terminal services, as well as modules of the TCB emulation, are available in the technical report describing the system, [5].

4.1 Kernel Interface Module

The lowest level module, called the Kernel Interface Module, hides much of the detail of how GEMSOS segments are named and accessed from higher level modules. The primary goal in designing this module was to provide a means for performing unit tests of the remainder of the DBMS in the UNIX environment, as project contention for the target development machine was relatively high. The module represents the database as a small number (eight) of fixed-length, numbered segments. Functions are provided to open, close, read, and write each segment. The UNIX version of the module maps each segment to a file. "Reading" a segment means that the file is copied into a local array and the physical address of the array passed to the invoking function. The GEMSOS version of the module "opens" a segment by making it known (translating the number into a preassigned pathname) and "reads" it by swapping it in and returning its address. All of the segments are "opened" and "read" by the main program during the initialization phase of the process, and "written" and "closed" when the process is terminated.

4.2 EN Table Manager

The next layer of the DBMS software manages tables (stored in segments) representing entities and their names (keys).

Logically, there is a separate "entity table" (E Table) for each access class. The E table may be thought of as a table of descriptors, with one descriptor for each entity currently in the database. The descriptor for a given entity is stored, of course, in the particular E Table for the access class of the subject that added the entity to the database, so any entities added by a SECRET subject are represented by descriptors in the SECRET E Table, while any entities added by a TOP SECRET subject are represented by descriptors in the TOP SECRET E Table.

The "internal identifier" for an entity has two components: the access class of the E Table it is found in, and the index into the table for the entity's descriptor. Throughout the implementation, the numerals 0 and 1 were used to encode the two access classes supported, SECRET and TOP SECRET, respectively. Given the pair <access class, index>, finding the actual record representing a particular entity is straightforward. (For a SECRET process, a "unique id" of the form <TOP SECRET, x> is not useful as attempting to access the TOP SECRET table will result in a trap. Each function in the DBMS is written to check the validity of input parameters, including access class, to avoid such traps.)

Each E Table descriptor contains two fields. The first is a numeral encoding the type of the entity. The second is an index to an N Table descriptor. There are also two N Tables, one for SECRET and one for TOP SECRET. Each N Table entry is a descriptor representing the (human-readable) name of an entity. (From the point of view of the data model, the name of an entity is the value of a designated property of the entity, depending on its type.) Each N Table descriptor contains two fields: the name (key) value itself, and an index to the E Table record for the same entity.

Taken together, the E table and N table records for a given entity may be thought of as doubly-linked parts of a single logical record. This whole record, as a consequence of the data model definition, has a uniform access class so those parts can be stored in the same physical segment. Knowing where either part is, the other part can quickly be found by following a link.

The motivation for separating an entity descriptor into two physical parts was the following: downward references to some entity records will be made (for instance, TOP SECRET properties for SECRET entities may be entered). The records representing such properties, which must be stored in a TOP SECRET physical segment, will contain an <access class, E Table index> reference to the particular E Table record representing the entity having the property. No SECRET process can be allowed to move the E Table descriptor from one place to another within the SECRET E Table, because that would invalidate the downward reference. There is no way that the SECRET process could modify the downward reference to reflect a new location for the E Table descriptor, because the SECRET process cannot read or write a TOP SECRET table. Therefore, the E Table descriptors, once entered, are never moved: they serve as stable targets for downward references to the entity they represent.

On the other hand, in order to provide for the rapid location of entities by (human-readable) name, some sort of index must be provided into the E Table. This is the function of the N Table. Its records are maintained in alphabetical order by name. (A more sophisticated indexing technique would be used in a genuine DBMS, but the principle is the same.) That is, as new entities are entered, the N Table is rearranged and both the forward and backward links to the E table are updated as required. This is always possible because the two parts of the entity record (a motionless one in the E table, and a movable one in the N table) are always of the same access class. If a process is allowed to move the N table record, it is allowed to update the E table record to match the new position in the N table.

Functions provided to maintain and use the E and N Tables (in concert) include a function to add a named entity, a function that locates an entity given its access class and name, and a function that locates an entity given its access class and E Table index.

4.3 Relationship Tables

The next layer of the DBMS software implements "Relationship Tables", or R Tables. As for E and N Tables, there are two R Tables, one for SECRET relationships and one for TOP SECRET relationships. Each relationship is represented by a record containing an integer representing the type of the relationship, and a reference to each of the two entities related. Each such reference has two parts: an access class code and an E Table index. The access class is required because a reference may be downward: it may be that a TOP SECRET relationship exists that relates a SECRET entity to another entity. Without the access class field, it would not be possible to tell whether the E Table reference in a relationship was to be applied to the SECRET E Table or the TOP SECRET E Table.

As an implementation choice, no index was created for relationships as the technique had already been demonstrated for entities. One could easily conceive of tables indexing the R Table by their effective keys (the two references). just as the N tables serve as indexes into the E tables, that would function in much the same way. Functions provided for manipulating R Tables include one to add a new relationship, and to locate a relationship given its access class and references.

4.4 Property Tables

Entities in the demonstration database have non-naming properties as well as names, and relationships have properties as well. For the demonstration schema, polyinstantiation of non-naming properties and properties of relationships is allowed: that is, there may be both SECRET and TOP SECRET values for a given non-naming property of a SECRET entity, or for a property of a SECRET relationship.

Non-naming properties, or properties of relationships, were uniformly stored in one of two Property Tables, or P Tables. The approach taken was "brute force": a record was preallocated corresponding to each possible entry in each E or R table for each access class. As non-naming properties were entered, their values were simply stored in the appropriate slot. A more compact representation would have involved explicit references to the entity or relationship having the property (possibly a downward reference), and an index into the property table so that the properties of a given entity or relationship could be located quickly.

4.5 DBMS Interface Module

The topmost layer of the DBMS consisted of an interface module that mapped the defined programming interface for a selected subset of the entity-relationship data model to the appropriate sequence of calls to the the EN, R, and P table managers.

5. Summary and Conclusions

It is our belief that the demonstration meets or exceeds all of its objectives. In particular, it demonstrates and exercises techniques for creating and maintaining downward references from one data element to another of lower sensitivity, and the provision of high apparent granularities of classification by storing small data elements in large repositories.

(It must be emphasized that intrinsic to this approach is that the security indicators presented to the users are not safe with respect to downgrading.) All of the data presented in a TOP SECRET session must be treated as TOP SECRET. However, this is not as cumbersome a property as may, at first, appear. If, for instance, a user with a TOP SECRET clearance is browsing the database with a session level of TOP SECRET, and, as the result of issuing a query, decides that a "sanitized" version of that query should be prepared for release at a SECRET level, the user can simply press the "Secure Attention" key to negotiate a session level of SECRET and re-issue the desired query. (It might be noted that there is no requirement, even in the demonstration, to logout and login again, as the TCB is designed to "remember" the clearance of the currently logged-in user.)

The demonstration also serves, in our opinion, as a proof of concept that an adequate degree of functionality for a multilevel DBMS can be attained by implementing the DBMS as an untrusted application on an existing TCB. The major criterion, we suggest, that a "multilevel DBMS" must meet to be useful is that a user should be able to view all of the data classified at the user's session level or below, and that it must be possible to relate high-sensitivity data with data of lower sensitivity in such a way that when the lower sensitivity data is modified, these modifications are reflected in the higher level view of the related data.

For instance, if a SECRET data item (say a Flight with a particular Pilot) has a TOP SECRET property, (say its cargo), and the SECRET item is modified (say the Pilot is changed), the change ought to be reflected in the TOP SECRET view (the TOP SECRET cargo ought to remain associated with the modified SECRET item). This requirement precludes implementations that make redundant copies of low sensitivity data for high sensitivity subjects but cannot then maintain those copies. The demonstration is specifically designed to show that meeting these requirements with an untrusted DBMS is possible.

The following observations resulting from our experience in implementing the demonstration might also be of general interest.

- The implementation effort involved programmers not familiar with the implementation and design of trusted systems. By designing the DBMS as an application to be built on top of a DBMS, this lack of specialized expertise was finessed. In effect, the use of a security kernel as a "virtual machine" transforms the problem of designing and implementing a provably secure system into the much easier problem of designing and implementing an "ordinary" application that will function as specified. Working within the limitations of the Basic Security and Confinement properties was not a great deal more difficult than working within the limitations imposed by a run-time environment that does dynamic bounds checking on arrays.

- It also was not particularly difficult finding specific data structure and algorithmic solutions to the problems posed by the demonstration requirements (e.g., how to represent a "downward reference"). The knowledge that the use of a "trusted process" to perform any critical functions was forbidden was a useful and immediately productive discipline.

- The most troublesome aspect of the design and implementation was in the area of human interface design. This we attribute to a fairly ambitious desire to provide a screen-oriented, event-driven interface with insufficient thought given to its abstract specification. The resulting interface, while reasonably nice for this particular demonstration, does not scale up in any reasonable way to support for large numbers of potential access classes.

- Polyinstantiation proved easy to implement (you just let it happen) but hard to deal with from a DBMS application point of view. Application logic that deals with polyinstantiated properties on a case-by-case basis is difficult to write, particularly for a semantically "rich" data model such as the entity-relationship model. This suggests that high level operators that deal with potentially polyinstantiated items and properties need to be devised for use by applications. The generalization of report generators and screen generators to cope with polyinstantiated values is likely to be quite difficult.

REFERENCES

1. H.C.Lefkovits, R.R.Schell, G.G.Ganjnak, W.R.Shockley. **The Multilevel Secure Entity-Relationship DBMS.** AOG Systems Corporation, Harvard, MA. In preparation.

2. T.H.Hinke and M.Schaefer. **Secure Data Management System.** RADC-TR-75-266, System Development Corp., Nov., 1975.

3. D.P.Reed and R.K.Kanodia "Synchronization with eventcounts and sequencers." In **Comm. of the ACM**, Vol. 22, Issue 2, Feb., 1979.

4. P.A.Bernstein and N.Goodman. "Timestamp-based algorithms for concurrency control in distributed database systems." In **Proc. 6th Int. Conf. on Very Large Data Bases**, IEEE Comp. Soc., Oct., 1980.

5. W.R.Shockley and D.F.Warren. **Multilevel Secure Entity-Relationship Database Management System: Demonstration Description and User's Manual.** Tech. Rep. GCI-88-2-01, Gemini Comp., Inc., Jan., 1988.

A Distributed Architecture for Multilevel Database Security

James P. O'Connor and James W. Gray, III

Unisys Defense Systems, McLean Research Center
8201 Greensboro Drive, McLean, VA 22102

Abstract

A distributed architecture for a multilevel secure database management system, based on the distributed architecture developed at the 1982 Air Force Summer Study on "Multilevel Data Management Security", is presented and described in terms of how it implements the relational operators. There are two notable aspects of this architecture. First, it factors the effect of the security class of the query into the classification of derived data. This allows tuple level labels to be safely used for mandatory access control. Secondly, it provides reliable tuple level labeling without requiring the relational operators to be trusted. This makes this architecture a suitable basis for a near-term solution to multilevel database management using existing DBMS components to implement the relational operators.

1. Introduction

The majority of the databases in the Department of Defense (DOD) and the intelligence community are computerized. These databases commonly contain data that are classified at multiple security levels and must be restricted for different levels of user access. The most common means of restricting access to these data is to store them in an untrusted database management system (DBMS), and operate the system in *system-high* mode. In this mode, every user is cleared to the level of the most highly classified piece of information in the system, and all data leaving the system are assigned the highest classification until a human reviewer assigns the proper classification. Since all users must be cleared to system-high, the cost of system operation and the risk of a security breach is much greater than it might be.

The 1982 Air Force Summer Study on "Multilevel Data Management Security" made several recommendations on the development of multilevel database management systems [1]. Group 1 of the study focused on near-term solutions to multilevel database security. One of the recommended approaches was a Distributed DBMS (D-DBMS) architecture that utilized one untrusted DBMS per security level supported. It was recognized that, by employing physically separate, untrusted DBMSs, a D-DBMS architecture could provide a high level of security and performance in the near-term, while lowering development costs and risk.

Unisys is currently involved in an effort, funded by the U.S. Air Force, Rome Air Development Center (RADC), to design a Multilevel Secure (MLS) DBMS that meets the requirements for a Class B3 trusted computer system [2]. Our approach is a variation on the D-DBMS approach recommended by Group 1. This paper describes the Secure D-DBMS (SD-DBMS) architecture that was developed as a first step in our design. Although discretionary security was an important consideration in the development of this architecture, this paper will focus primarily on the enforcement of mandatory security.

In this paper, it is assumed that the reader is familiar with the relational data model [3,4] and the concepts of multilevel security [5,6].

2. Security Classes, Subjects, and Objects

The SD-DBMS is being designed to support a set of three *security classes*. These security classes can be hierarchical levels (e.g., TOP SECRET, SECRET, and CONFIDENTIAL), non-hierarchical categories (e.g., NATO, NOFORN, NUCLEAR), or a combination of the two. The design can, however, be easily extended to support four or more security classes. The limiting factor is that the design requires one DBMS per security class supported. Note that the set of security classes does not necessarily form a lattice, as described in [7]. For example, if the system is configured to support three non-hierarchical categories (e.g., A, B, and C), then data from different categories cannot be mixed since the result would constitute a new (fourth) security class. The set of security classes is partially ordered by a relation called *dominates*. If C_1 and C_2 are security classes, C_1 is said to dominate C_2 if and only if the hierarchical classification of C_1 is greater than or equal to that of C_2, and the categories in C_2 are a subset of those in C_1. C_1 is said to *strictly dominate* C_2 if and only of C_1 dominates but is not equal to C_2.

Mandatory security is enforced in terms of *subjects* and *objects*. In the SD-DBMS, subjects are assigned two security classes called a *read class* and a *write class*. A subject's read class must dominate its write class. A subject is permitted to read an object at a particular security class if the read class of the subject dominates the security class of the object. A subject is permitted to write an object at a particular security class if the security class of the object dominates the write class of the subject, and the read class of the subject dominates the security class of the object. If a subject's read class strictly dominates its write class, it is said to be a *trusted subject.*

When it does not cause a loss of generality, the discussions and examples in this paper will use the terms *high* and *low* to refer to any two security classes where the first strictly dominates the second.

3. Multilevel Relations

The function of the SD-DBMS is to manage and control access to multilevel databases. A multilevel database is defined to be a collection of logically related multilevel relations.

> **Definition 3.1:** A multilevel relation R is defined as having two parts: a *time-invariant schema* $R(A_1, A_2, ..., A_n, SC)$, where $A_1, A_2, ..., A_n$ are attributes over some domains $D_1, D_2, ..., D_n$, and SC is an attribute over a set of security classes CL; and a *time-varying set of tuples* T, such that $T \subseteq D_1 \times D_2 \times \cdots \times D_n \times CL$. The set CL is called the range of R.

Note that the schema for a multilevel relation includes a security class attribute. For each tuple in the relation, this attribute is used to indicate the classification of the data contained within the tuple. This information is used by the system to enforce mandatory access control.

The above definition permits tuples to be *polyinstantiated.* Polyinstantiation refers to the simultaneous existence of multiple data objects with the same name, where the multiple instantiations are distinguished by their security class [8, 9, 10]. In the SD-DBMS, this means that there may be two or more tuples in a multilevel relation with the same primary key. These tuples are uniquely identified by the combination of their primary key and their security class. Although it presents an integrity problem, polyinstantiation must be supported in an MLS DBMS in order to prevent the appearance, disappearance, or perceived presence of data from being used as an inference or signaling channel.

When a relational operator is applied to one or more multilevel relations, the result is another multilevel relation, called a *derived* relation. An important consideration is how the tuples in a derived relation are labeled. One possible approach is to label each tuple at the least upper bound of the security classes of the tuples that entered into its derivation. For example, when a tuple is selected or projected, its security class is unchanged; when two tuples are joined, the resulting tuple is labeled at the least upper bound of the security classes of the original tuples. This is the approach taken in the SeaView effort [9]. However, as pointed out in Appendix A of [9], these labels do not accurately reflect the security classes of parameters embedded in relational expressions, or the security classes of data upon which the decision to evaluate a particular expression might have been conditioned. This led the SeaView project to the conclusion that tuple level labels are inherently advisory. Our conclusion differs in that we believe that tuple level labels can be perfectly reliable if the security class of the query is considered when determining how to label derived data. For this reason, in the SD-DBMS the query is a labeled object, and each tuple in a derived relation is labeled with the least upper bound of the security classes of the tuples that entered into its derivation and the security class of the query.

Since queries are labeled objects, there must be a way to determine the security class of a given query. If the query is submitted by a untrusted application program, its security class is set to be that of the application (i.e., equal to its read class and write class). This has the effect that, for all untrusted applications, results are returned at the security class of the application. It should be noted that, if there is a need for advisory labels, they can be explicitly stored in relations as an additional column. If the query is submitted by a multilevel application, the multilevel application can supply the SD-DBMS with the security class of the query (providing it falls within the application's security class range). The correctness of these labels can be verified through an information flow analysis of the application program [6].

An important special case occurs when the application is an interactive user interface. In this case, users enter queries directly. If the interface is untrusted, the level of the query must be taken to be the level of the interface. If the interface is trusted (multilevel), the user can supply the SD-DBMS with the security class of the query. This would require that the interface include a mechanism that permits the user to reliably communicate the security class of the query. It is assumed that the user is aware of the security class of the query being entered. This assumption is consistent with those made about users of multilevel operating systems.

The above approach to labeling tuples is attractive for two important reasons. First, it is firmly based on an information flow model of security (i.e., it accounts for information flows that can result from sequences of the relational operators, parameters in relational expressions, and flows in application programs). This allows tuple level labels to be safely used for mandatory access control. Second, it maintains the important closure property of the relational model (i.e., the result of all relational operators are multilevel relations).

4. SD-DBMS Abstract Architecture

The SD-DBMS architecture, shown in figure 4.1, consists of three types of components: User Programs, the Data Manager, and back-end DBMSs. A User Program is a user interface or application program permitted to issue queries against a multilevel database. User Programs can be trusted (i.e., multilevel) or untrusted (i.e., single level). Trusted User Programs are permitted to issue queries at multiple security levels and receive multilevel results.

The Data Manager is a trusted component that performs the reference monitor functions in the SD-DBMS. A *reference monitor* is an abstract machine that mediates all accesses to objects by subjects [2]. In the SD-DBMS, the subjects are

User Programs and the objects are tuples in multilevel relations and user queries. The Data Manager component also performs query decomposition and execution control functions. These functions, most of which can be implemented in untrusted code, are discussed in detail in section 5.

Finally, the back-end DBMSs are untrusted single-level relational DBMSs used to store and process portions of the multilevel database. There is one back-end DBMS per security class supported by the system.

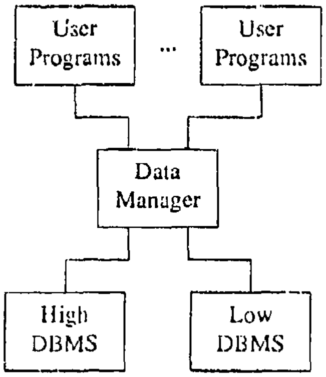


Figure 4.1. SD-DBMS Abstract Architecture

The SD-DBMS stores multilevel relations by horizontally partitioning them into single-level fragments, which are then stored under the appropriate back-end DBMSs. When a user creates a multilevel relation, the system creates a set of single-level fragments in which to store the relation. This is done using the following algorithm.

> **Algorithm 4.1:** Given a multilevel relation $R(A_1, A_2, ..., A_n, SC)$, with a range CL, for each security class c ∈ CL, create a fragment $R_c(A_1, A_2, ..., A_n)$ on the back-end DBMS with security class c.

For example, a multilevel relation R with a range {high, low} would be mapped into two fragments, R_{high} and R_{low}. The fragment R_{high} would be created on the high DBMS and used to store the high tuples in R, and the fragment R_{low} would be created on the low DBMS and used to store the low tuples in R. The multilevel relation R is part of an example multilevel database, summarized in table 4.1, that will be used in examples throughout this paper.

Relation	Primary Key	Range	Fragments
R(x,a)	x	{low,high}	R_{low}, R_{high}
S(x,b)	x	{low,high}	S_{low}, S_{high}
M(x,c)	x	{low}	M_{low}

Table 4.1. Example Multilevel Database

To retrieve data from a multilevel database, subjects (User Programs) submit queries to the Data Manager. The Data Manager decomposes each query into a sequence of subqueries that operate on single-level fragments. This decomposition is done in such a way that each subquery produces a single-level result, and the union of these single-level results forms the result of the original query. To decompose queries in this way is not difficult since all relational operators, except for union, produce single level results when applied to single level fragments. The motivation for this decomposition strategy is that, if the relational operators always return single level results, the back-end DBMSs that implement them can be untrusted.

Once a query is decomposed into subqueries, each of the subqueries is executed on the back-end DBMS having the same security class as its result. Since queries executed on the high DBMS often require access to low data, the SD-DBMS supports the transmission of data from the low to the high DBMS. To assure that no data flow in the opposite direction, all such transfers are constrained to go through the reference monitor (implemented as part of the Data Manager). Once the execution of the subqueries is complete, the Data Manager retrieves the results and labels them at the security class of the back-end DBMS on which they were computed. The reference monitor then mediates the return of these results to the user.

5. Query Decomposition and Execution

The previous section presented an architecture for the SD-DBMS. This section presents a notation for describing query decomposition and processing algorithms, an algorithm for the recovery (recomposition) of multilevel relations, and algorithms and examples that describe how the relational operators are implemented within the architecture.

5.1 Notation

We have adopted the following relational algebra notation to express the queries processed by the SD-DBMS [3]: $\pi_{a_1,a_2,...,a_n}(R)$ denotes the projection of the relation R, on the attributes $a_1, a_2, ..., a_n$; $\sigma_P(R)$ denotes selection (restriction), where P is the selection predicate; $R \times S$ denotes cross product; $R \cup S$ denotes union; and finally, $R - S$ denotes difference. The operator ← is the relation assignment operator. These relational operators can be nested to reduce the need for temporary relations and to form more compact expressions (e.g., $\pi_{a,b}(R \times S)$).

To express update queries, we use the following notation which is a modified version of that used in the query language SQUARE [11]. The insertion operator has the form: $\downarrow R(a_1, a_2, \cdots, a_n; v_1, v_2, ..., v_n)$ denoting the insertion of a tuple into R, where $a_1, a_2, \cdots, a_n$ are attributes of R, and $v_1, v_2, ..., v_n$ are values for those attributes in the inserted tuple. The deletion operator has the form: $\uparrow R(P)$, where P is a predicate. The relation R is searched for tuples that satisfy P, and all matching tuples are deleted.

The modify operator has the form: $\rightarrow R\ (a_1 = expr1, a_2 = expr2, ..., a_n = exprn; P)$, where $a_1, a_2, \cdots, a_n$ are attributes of R, $expr1, expr2, ..., exprn$ are expressions (possibly involving $a_1, a_2, \cdots, a_n$), and P is a predicate. The relation R is searched for tuples that satisfy P, the specified attributes in each of those tuples are replaced with the result of evaluating the corresponding expression.

Since a basic function of the SD-DBMS is distributed query processing, two operations have been defined to describe the distributed execution of queries and dynamic routing of results. The operation $exec\ (Q, C)$ is used to execute query Q at component C. The operation $trans\ (R, C1, C2)$ is used to transfer (copy) relation R from component C1 to component C2. The components of the architecture are indicated by: L for low DBMS, H for high DBMS, D for Data Manager, and U for User Program.

The above notation is summarized in table 5.1.

Symbol	Meaning
π	Project
σ	Select
$\times$	Product
$\cup$	Union
$-$	Difference
$\leftarrow$	Assignment
$\downarrow$	Insert
$\uparrow$	Delete
$\rightarrow$	Modify
trans	Data Transfer
exec	Subquery Execution

Table 5.1. Notation Summary

5.2 Recovery of Multilevel Relations

The execution of a relational operator results in the creation of a new multilevel relation. This multilevel relation can be an intermediate or final result. Intermediate results are relations that are to be used as operands in subsequent relational operations. Final results are relations to be returned to the subject. The last step in processing any query is to *recover* the final result (multilevel relation) and return it to the subject. A multilevel relation is recovered by copying each of its fragments to the Data Manager, labeling the tuples in the fragments at the security class of the back-end DBMS from which they were retrieved, unioning the fragments together, and sending the result to the requesting User Program. This recovery process is given by the following algorithm.

> **Algorithm 5.1:** Given a retrieval request on a multilevel relation R, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject. For each fragment $R_i \in F_R$, copy R_i from the back-end DBMS c to the Data Manager, where c is a security class that is the least upper bound of i and the level of the retrieval request. Then execute the query $T_{ml} \leftarrow \overset{*}{\bigcup}_{R_i \in F_R} R_i$ on the Data Manager, where T_{ml} is a temporary relation, and return the result to the subject.

Since this union operation returns a multilevel result it must be trusted. This is indicated by the star in the symbol $\overset{*}{\cup}$. It should be noted that (depending on the security class of the query) algorithm 5.1 may require fragments to be transferred from a low to a high back-end DBMS. This will occur when c dominates i. As mentioned in section 4, such transfers are constrained to go through the reference monitor to assure that no data flow in the opposite direction.

The recovery of a multilevel relation is illustrated by the following example. This example, and the other examples in this section, assume the security class of the submitted query is low. In appendix A, examples are given with queries at different security classes. Suppose a subject with a read class of high submits a query that results in a request for multilevel relation R to be returned. The system would recover R as follows.

$$trans\ (R_{low}, L, D) \tag{1.1a}$$

$$trans\ (R_{high}, H, D) \tag{1.2a}$$

$$exec\ (R_{ml} \leftarrow R_{low} \overset{*}{\cup} R_{high}, D) \tag{1.3a}$$

$$trans\ (R_{ml}, D, U) \tag{1.4a}$$

If the same query were submitted by a subject with a read class of low, the system would recover R as follows.

$$trans\ (R_{low}, L, D) \tag{1.1b}$$

$$trans\ (R_{low}, D, U) \tag{1.2b}$$

5.3 Retrieval Operators

There are five basic relational operators: select, project, product, union, and difference -- from these, other useful relational operators can be derived (e.g., join, intersection, and division). This section discusses the five basic operators (since all other operators can be derived from them) and a generic alpha operator used to illustrate the handling of aggregate operators (e.g., MIN, MAX, and AVG). Query decomposition and processing algorithms for these operators that assure correct labeling of derived results are presented in the following sections.

5.3.1 Select. A select on a multilevel relation is processed by decomposing it into a set of selects on single-level fragments.

> **Algorithm 5.2:** Given a select query $\sigma_P\ (R)$, where R is an n-ary multilevel relation, and P is the select predicate, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject. For each fragment $R_i \in F_R$, if T_c exists, create the subquery $T_c \leftarrow T_c \cup \sigma_P\ (R_i)$; otherwise, create the subquery $T_c \leftarrow \sigma_P\ (R_i)$; execute the subquery on the back-end DBMS c, where c is the least upper bound of i and the secu-

rity class of the query.

When the execution of these subqueries is complete, the answer to the original query is the multilevel relation T. T can be recovered (using algorithm 5.1) and returned to the user, or it can be used as an operand in further relational calculations.

The decomposition and processing of select queries is illustrated by the following example. Suppose a subject with a read class of high submits the query $\sigma_{a=10}(R)$. The system would decompose this query into the following set of subqueries and operations:

$$exec\ (T_{low} \leftarrow \sigma_{a=10}(R_{low}), L) \qquad (2.1a)$$

$$exec\ (T_{high} \leftarrow \sigma_{a=10}(R_{high}), H) \qquad (2.2a)$$

If the same query were submitted by a subject with a read class of low, the system would decompose it into the following subquery:

$$exec\ (T_{low} \leftarrow \sigma_{a=10}(R_{low}), L) \qquad (2.1b)$$

5.3.2. Project. A project on a multilevel relation is processed by decomposing it into a set of projects on single-level fragments.

> **Algorithm 5.3:** Given a project query $\pi_A(R)$, where R is an n-ary relation, and A is a set of project attributes, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject. For each fragment $R_i \in F_R$, if T_c exists, create the subquery $T_c \leftarrow T_c \cup \pi_A(R_i)$; otherwise, create the subquery $T_c \leftarrow \pi_A(R_i)$; execute the subquery on the back-end DBMS c, where c is the least upper bound of i and the security class of the query.

When the execution of these subqueries is complete, the answer to the original query is the multilevel relation T.

The decomposition and processing of project queries is illustrated by the following example. Suppose a subject with a read class of high submits the query $\pi_{x,a}(R)$. The system would decompose this query into the following set of subqueries and operations:

$$exec\ (T_{low} \leftarrow \pi_{x,a}(R_{low}), L) \qquad (3.1a)$$

$$exec\ (T_{high} \leftarrow \pi_{x,a}(R_{high}), H) \qquad (3.2a)$$

If the same query were submitted by a subject with a read class of low, the system would decompose it into the following subquery:

$$exec\ (T_{low} \leftarrow \pi_{x,a}(R_{low}), L) \qquad (3.1b)$$

5.3.3. Product. The product of multilevel relations is processed by decomposing it into a set of products of single-level fragments.

> **Algorithm 5.4:** Given the product query $R \times S$, where R and S are n-ary relations, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject, and F_S be the set of fragments S_i of S such that i is dominated by the read class of the subject. For each pair of fragments, $R_i \in F_R$, $S_j \in F_S$, if T_c exists, create the subquery $T_c \leftarrow T_c \cup R_i \times S_j$; otherwise, create the subquery $T_c \leftarrow R_i \times S_j$; execute the subquery on on the back-end DBMS c, where c is the least upper bound of i, j, and the security class of the query.

When the execution of these subqueries is complete, the answer to the original query is the multilevel relation T.

The decomposition and processing of product queries is illustrated by the following example. Suppose a subject with a read class of high submits the query $R \times S$. The system would decompose this query into the following set of subqueries and operations:

$$exec\ (T_{low} \leftarrow R_{low} \times S_{low}, L) \qquad (4.1a)$$

$$trans\ (R_{low}, L, H) \qquad (4.2a)$$

$$trans\ (S_{low}, L, H) \qquad (4.3a)$$

$$exec\ (T_{high} \leftarrow R_{low} \times S_{high}, H) \qquad (4.4a)$$

$$exec\ (T_{high} \leftarrow T_{high} \cup (R_{high} \times S_{low}), H) \qquad (4.5a)$$

$$exec\ (T_{high} \leftarrow T_{high} \cup (R_{high} \times S_{high}), H) \qquad (4.6a)$$

If the same query were submitted by a subject with a read class of low, the system would decompose it into the following subquery:

$$exec\ (T_{low} \leftarrow R_{low} \times S_{low}, L) \qquad (4.1b)$$

5.3.4. Union. The union of multilevel relations is processed by decomposing it into a set of unions of single-level fragments at a single security level.

> **Algorithm 5.5:** Given a union query $R \cup S$, where R and S are n-ary multilevel relations, for each security class i $\in$ range(R) $\cup$ range(S) that is dominated by the read class of the subject, if T_c exists, create the subquery $T_c \leftarrow T_c \cup (R_i \cup S_i)$; otherwise, create the subquery $T_c \leftarrow R_i \cup S_i$; execute th. subquery on the back-end DBMS c, where c is the least upper bound of i and the security class of the query.

When the execution of these subqueries is complete, the answer to the original query is the multilevel relation T.

The decomposition and processing of union queries is illustrated by the following example. Suppose a subject with a read class of high submits the query $R \cup S$. The system would decompose this query into the following set of subqueries and operations:

$$exec\ (T_{low} \leftarrow R_{low} \cup S_{low}, L) \qquad (5.1a)$$

$$exec\ (T_{high} \leftarrow R_{high} \cup S_{high}, H) \qquad (5.2a)$$

If the same query were submitted by a subject with a read class of low, the system would decompose it into the following subquery:

$$exec\ (\ T_{low} \leftarrow R_{low} \cup S_{low}, L\ ) \qquad (5.1b)$$

5.3.5. Difference. The difference of two multilevel relations is processed by decomposing it into a set of differences.

Algorithm 5.6: Given a difference query $R - S$, where R and S are n-ary multilevel relations, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject, and F_S be the set of fragments S_i of S such that i is dominated by the read class of the subject. For each fragment $R_i \in F_R$, if T_c exists, create the subquery $T_c \leftarrow T_c \cup (\ R_i - \bigcup_{S_j \in F_S} S_j\ )$; otherwise, create the subquery $T_c \leftarrow R_i - \bigcup_{S_j \in F_S} S_j$; execute the subquery on the back-end DBMS c, where c is the least upper bound of i, the security classes of the fragments in F_S, and the security class of the query.

When the execution of these subqueries is complete, the answer to the original query is the multilevel relation T.

The decomposition and processing of difference queries is illustrated by the following example. Suppose a subject with a read class of high submits the query $R - S$. The system would decompose this query into the following set of subqueries and operations:

$$trans\ (\ R_{low}, L, H\ ) \qquad (6.1a)$$

$$trans\ (\ S_{low}, L, H\ ) \qquad (6.2a)$$

$$exec\ (\ T_{high} \leftarrow R_{low} - (\ S_{low} \cup S_{high}\ ), H\ ) \qquad (6.3a)$$

$$exec\ (\ T_{high} \leftarrow T_{high} \cup (\ R_{high} - (\ S_{low} \cup S_{high}\ )\ ), L\ ) \qquad (6.4a)$$

If the same query were submitted by a subject with a low read class, the system would decompose it into the following set of subqueries:

$$exec\ (\ T_{low} \leftarrow R_{low} - S_{low}, L\ ) \qquad (6.1b)$$

5.3.6. Aggregate. An aggregate computed on a multilevel relation is processed by computing the aggregate over the single-level fragments of the relation and labeling the result at the least upper bound of the fragment security classes and the security class of the query.

Algorithm 5.7: Given an aggregate query $\alpha\ (\ R\ )$, where α is an aggregate operator and R is an n-ary multilevel relation, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject. Execute the subquery $T_c \leftarrow \alpha\ (\ \bigcup_{R_i \in F_R} R_i\ )$ on the back-end DBMS c, where c is the least upper bound of the security classes of the fragments in F_R and the security class of the query.

When the execution of these subqueries is complete, the answer to the original query is the multilevel relation T. T consists of a single fragment at the security class of the least upper bound of all the data used in its derivation.

The decomposition and processing of aggregate queries is illustrated by the following example. Suppose a subject with a read class of high submits the query $\alpha\ (\ R\ )$. The system would decompose this query into the following set of subqueries and operations:

$$trans\ (\ R_{low}, L, H\ ) \qquad (7.1a)$$

$$exec\ (\ T_{high} \leftarrow \alpha\ (\ R_{low} \cup R_{high}\ ), H\ ) \qquad (7.2a)$$

If the same query were submitted by a subject with a low read class, the system would decompose it into the following subquery:

$$exec\ (\ T_{low} \leftarrow \alpha\ (\ R_{low}\ ), L\ ) \qquad (7.1b)$$

5.4. Update Operators

The three basic update operators are insert, delete, and modify. This section covers algorithms for these operators that assure that updates get reflected in the appropriate fragments. The fragments to be updated depends of the level of the update, which is indicated by the security class of the query. Examples are not given for these algorithms since their operation is straightforward.

5.4.1. Insert. An insertion into a multilevel relation is processed by inserting the data in the appropriate fragment of that relation. The fragment in which to insert the data is determined by the security class of the query. Note that the security class of the query always dominates the write class of the subject.

Algorithm 5.8: Given an insert query $\downarrow R\ (\ a_1, a_2, \cdots, a_n; v_1, v_2, \ldots, v_n\ )$, where R is a multilevel relation, execute the subquery $\downarrow R_c\ (\ a_1, a_2, \cdots, a_n; v_1, v_2, \ldots, v_n\ )$ on back-end DBMS c, where c is the security class of the query.

5.4.2. Delete. A deletion from a multilevel relation is processed by deleting tuples in the the appropriate fragments. These are the fragments having a security class that dominates the security class of the query and is dominated by read class of the subject.

Algorithm 5.9: Given a delete query $\uparrow R\ (\ P\ )$, where R is a multilevel relation, and P is a predicate, let F_R be the set of fragments R_i of R such that i is dominated by the read class of the subject, and i dominates the security class of the query. For each fragment $R_i \in F_R$, execute the query $\uparrow R_i\ (\ P\ )$ on back-end DBMS i.

5.4.3. Modify. A modification to a multilevel relation is processed by modifying the tuples in the appropriate fragments. These are the fragments having a security class that dominates the security class of the query and is dominated by read class of the subject.

Algorithm 5.10: Given a modify query $\rightarrow R\ (a_{1=expr1}, a_{2=expr2}, \cdots, a_{n=exprn}; P\ )$, where $a_1, a_2, \cdots, a_n$ are attributes of R, $expr1, expr2, ..., exprn$ are expressions (possibly involving $a_1, a_2, \cdots, a_n$), and P is a predicate. let F_R be the set of fragments of R_i of R such that i is dominated by the read class of the subject, and i dominates the security class of the query. For each fragment $R_i \in F_R$, execute the query $\rightarrow R_i\ (a_{1=expr1}, a_{2=expr2}, \cdots, a_{n=exprn}; P\ )$ on back-end DBMS i.

6. Data Replication

The query processing performance of the SD-DBMS architecture is likely to suffer because of the need to transfer low fragments to the high back-end DBMS in order to process queries. In order to lesson or eliminate this performance penalty, the SD-DBMS permits some or all of the low fragments to be replicated on the high back-end DBMS. Low fragments replicated on the high back-end DBMS can be used in high queries instead of fetching a new copy from the low back-end DBMS each time it is needed. Partial replication (i.e., replicating the low fragments of some, but not all, relations) is considered to be a viable alternative because it is expected that certain low fragments will be needed on the high back-end DBMS more frequently than others. The decision of which fragments to replicate could be based on statistics collected by the SD-DBMS. If a particular low fragment is frequently needed on the high back-end DBMS, then the overhead of replicating it is justified. However, if a particular low fragment is infrequently (or never) needed on the high back-end DBMS, then replication is not justified, and the fragment should be transferred to the high back-end DBMS on an as-needed basis. Thus, the partial replication approach provides a method of tuning the physical database structure.

The main disadvantage of replicating data (besides the obvious storage overhead) is the problem of synchronizing updates between replicated copies of low data. The solution to this problem in the SD-DBMS is to make one copy (the low copy) the primary copy, and all other copies secondary copies. Only the primary copy is permitted to be updated by users. User updates to the primary copy are then propagated to all secondary copies by the system. This technique avoids the problem of simultaneous updates to replicated copies of the data resulting in an inconsistent database. Since all updated must be applied to the primary copy first, the concurrency control mechanisms on the low back-end DBMS can prevent conflicting user updates. Using this approach, if a serious loss of synchronization should occur, it could always be corrected by removing the replicated fragment and recopying the (low) primary copy.

7. Covert Channels

A serious problem with back-end DBMS architectures is that a Trojan horse in a high user program could use queries sent to the low back-end DBMS as a high bandwidth covert storage channel [1]. This attack could take a number of forms. The most highly recognized, and potentially the most devastating, is that a Trojan horse in a high user program could encode arbitrary high data in the qualification portion of the selection query to be leaked to the low back-end DBMS. A cooperating Trojan horse in the low back-end DBMS could then extract the high data from the query and release it to a low user. The following example, borrowed from Hinke [12], illustrates this problem. Suppose the following query is submitted from a high user program:

retrieve NAME where ADDRESS = "504 Pershing Square".

If this query is directed to the low back-end DBMS, the Data Manager cannot determine whether it is a legitimate query asking for employees at a particular address, or whether it has been sent by a Trojan horse in the user program, with the intent of leaking the Top Secret fact that 504 pershing missiles are to be deployed. This threat is not limited to selection queries, since other types of queries (e.g., projections, joins, etc.) can potentially be used as a covert leakage channel when sent down in security class (e.g., by modulating the attribute and relation names in the query).

The architecture presented in this paper does not suffer from this vulnerability. This is because in SD-DBMS architecture assigns each query a security class and strictly enforces the constraint that a query is never executed on a back-end DBMS strictly dominated by its security class. In the example above, since the query was entered from an untrusted high user program, the level of the query would be high, and therefore the query would be executed on the high back-end DBMS and the results would be returned as high. However, if the high user program were trusted, it could reliably communicate to the Data Manager the level of the query as low, and the query could be sent to the low back-end DBMS to return low results.

It should be noted that a covert channel may still exist. The execution of queries that access low data on the high back-end DBMS requires that data be copied from the low to the high back-end DBMS. These copy requests can be used as a covert channel, albeit one of substantially lower bandwidth. This channel can be eliminated by fully replicating the low data on the high back-end DBMS thereby eliminating the need to copy data. This approach eliminates the covert channel at the cost of increasing the overhead of propagating updates. Another approach would be to use partial or no replication, and simply audit the channel and prevent it from exceeding some predetermined threshold (e.g., by metering the flow of copy requests within the Data Manager).

9. Summary and Conclusions

This paper presented a distributed architecture for multilevel database security. The architecture was defined in such a way that it provides trusted tuple level labeling without requiring the relational operators to be trusted. This permits a near-term implementation of this architecture that uses existing DBMS technology to implement the majority of database operations. Another important aspect of this architecture is that it recognizes the role the query plays in the security class of derived results. Reflecting the security class of the query in the security class of results allows this architecture to provide tuple level labels that can be safely used in mandatory access control decisions.

Acknowledgements

We are grateful to the U.S. Air Force, Rome Air Development Center, who supported this work under contract F30602-87-C-0154. We would also like to thank Lee Badger, Dick Creps, Cathy Jensen, Robin Medlock, and LouAnna Notargiacomo for their comments and criticism on earlier drafts of this paper.

References

1. Air Force Studies Board, *Multilevel Data Management Security*, National Research Council, National Academy Press, Washington, DC, 1983.
2. Department of Defense, National Computer Security Center, *Trusted Computer System Evaluation Criteria*, 5200.28-STD, December 1985.
3. Jeffrey D. Ullman, *Principles of Database Systems (Second Edition)*, Computer Science Press, Rockville MD, 1982.
4. C. J. Date, *An Introduction to Database Systems (Fourth Edition)*, Addison-Wesley, Reading, Massachusetts, April 1986.
5. Morrie Gasser, *Building a Secure Computer System*, Van Nostrand Reinhold, New York, New York, 1988.
6. Dorothy E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, Massachusetts, 1982.
7. Dorothy E. Denning, "A Lattice Model of Secure Information Flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236-243, May 1976.
8. Dorothy E. Denning, Teresa F. Lunt, Peter G. Neumann, Roger R. Schell, Mark Heckman, and William Shockley, *Secure Distributed Data Views, Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database Management System*, A002: Interim Report, SRI International and Gemini Computers, Inc., November 1986.
9. Dorothy E. Denning, Teresa F. Lunt, Roger R. Schell, Mark Heckman, and William Shockley, *Secure Distributed Data Views (Sea View), The Sea View Formal Security Policy Model*, A003: Interim Report, SRI International and Gemini Computers, Inc., July 20, 1987.
10. Dorothy E. Denning and Teresa F. Lunt, "A Multilevel Relational Data Model," *Proc. 1987 IEEE Symposium on Security and Privacy*, pp. 220-234, April 1987.
11. Raymond F. Boyce, Donald D. Chamberlin, W. Frank King III, and Michael M. Hammer, "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage," *Communications of the ACM*, vol. 18, no. 11, pp. 621-628, November 1975.
12. T. H. Hinke, J. O. Solomon, and J. P. Dempsey, *Design Considerations for Secure Database Management Systems*, Technical Report TM-7526/100/01, System Development Corporation, January 13, 1986.

Appendix A

Query Processing Examples

This appendix presents three query processing examples that illustrate how a simple query is decomposed and processed in the architecture. The examples were selected in such a way that they also illustrate the effect of the security class of the query, and data replication, on query decomposition and processing. All three of the examples are based on the query shown in figure A.1. The multilevel database used in these examples is the database shown in table 4.1.

$$\pi_{a,b,c} \,(\, \sigma_{a=10} \,(\, R \times S \,)\,)$$

Figure A.1. Example Query.

A.1 Example 1: High User/Low Query

This example illustrates how the query, shown in figure A.1, would be processed if it were entered by a high subject and was labeled low. Note that, for a query to be entered by a high subject and be labeled low, the subject would have to be trusted (multilevel).

$$exec\ (\, T_{low} \leftarrow \pi_{a,b,c} \,(\, \sigma_{a=10} \,(\, R_{low} \times S_{low} \,)\,),\, L \,) \tag{1.1}$$

$$trans\ (\, R_{low},\, L,\, H \,) \tag{1.2}$$

$$trans\ (\, S_{low},\, L,\, H \,) \tag{1.3}$$

$$exec\ (\, T_{high} \leftarrow \pi_{a,b,c} \,(\, \sigma_{a=10} \,(\, R_{low} \times S_{high} \,)\,),\, H \,) \tag{1.4}$$

$$exec\ (\, T_{high} \leftarrow T_{high} \cup \\ (\, \pi_{a,b,c} \,(\sigma_{a=10} \,(\, R_{high} \times S_{low} \,)\,)\,),\, H \,) \tag{1.5}$$

$$exec\ (\, T_{high} \leftarrow T_{high} \cup \\ (\, \pi_{a,b,c} \,(\sigma_{a=10} \,(\, R_{high} \times S_{high} \,)\,)\,),\, H \,) \tag{1.6}$$

$$trans\ (\, T_{low},\, L,\, D \,) \tag{1.7}$$

$$trans\ (\, T_{high},\, H,\, D \,) \tag{1.8}$$

$$exec\ (\, T_{ml} \leftarrow T_{low} \overset{*}{\cup} T_{high},\, D \,) \tag{1.9}$$

$$trans\ (\, T_{ml},\, D,\, U \,) \tag{1.10}$$

In step 1.1 the low part of the result is computed. This result is stored in the fragment T_{low}. In steps 1.2 and 1.3, the low fragments of R and S are transferred to the high back-end DBMS so that they can be used to compute the high part of the result. Note that, if the data were fully replicated, steps 1.2 and 1.3 could be eliminated. In steps 1.4-1.6, the high part of the result is computed. This result is stored in T_{high}. At this point, the execution of the query is complete and the result is the multilevel relation T. This multilevel result was possible because the read class of the subject (high) strictly dominated the security class of the query (low). In steps 1.7-1.10, T is recovered and returned to the subject.

A.2 Example 2: High User/High Query

This example illustrates how the the query, shown in figure A.1, would be processed if it were entered by a high subject and was labeled high. In this case, either the subject could have been untrusted (single-level) and the query was taken to be its read class, or the subject could have been trusted (multilevel) and could have indicated to the Data Manager that the security class of the query was high.

$$trans\ (\,R_{low},L,H\,) \qquad (2.1)$$

$$trans\ (\,S_{low},L,H\,) \qquad (2.2)$$

$$exec\ (\,T_{high} \leftarrow \pi_{a,b,c}\ (\,\sigma_{a=10}\,(\,R_{low} \times S_{low}\,)\,),H\,) \qquad (2.3)$$

$$exec\ (\,T_{high} \leftarrow T_{high} \cup (\,\pi_{a,b,c}\ (\sigma_{a=10}\,(\,R_{low} \times S_{high}\,)\,)\,),H) \qquad (2.4)$$

$$exec\ (\,T_{high} \leftarrow T_{high} \cup (\,\pi_{a,b,c}\ (\sigma_{a=10}\,(\,R_{high} \times S_{low}\,)\,)\,),H) \qquad (2.5)$$

$$exec\ (\,T_{high} \leftarrow T_{high} \cup (\,\pi_{a,b,c}\ (\sigma_{a=10}\,(\,R_{high} \times S_{high}\,)\,)\,),H) \qquad (2.6)$$

$$trans\ (\,T_{high},H,D\,) \qquad (2.7)$$

$$trans\ (\,T_{high},D,U\,) \qquad (2.8)$$

In this example, the entire result will be high because the security class of the query (high) is equal to the read class of the subject (high). In steps 2.1 and 2.2, the low fragments of R and S are transferred to the high back-end DBMS so that they can be used to compute the high part of the result. Note that, if the data were fully replicated, steps 2.1 and 2.2 could be eliminated. In steps 2.3-2.6, the high part of the result is computed. At this point, the execution of the query is complete and the result is the multilevel relation T which consists of the single fragment T_{high}. In steps 2.7 and 2.8, T is recovered and returned to the subject.

A.3 Example 3: Low User/Low Query

This example illustrates how the query, shown in figure A.1, would be processed if it were entered by a low subject. In this case, the query obviously must be labeled low.

$$exec\ (\,T_{low} \leftarrow \pi_{a,b,c}\ (\,\sigma_{a=10}\,(\,R_{low} \times S_{low}\,)\,),L\,) \qquad (3.1)$$

$$trans\ (\,T_{low},L,D\,) \qquad (3.2)$$

$$trans\ (\,T_{low},D,U\,) \qquad (3.3)$$

In this example, the result will be low since the low user can only see low data. In step 3.1, the low (and only) part of the result is computed and stored in T_{low}. At this point, the execution of the query is complete and the result is the multilevel relation T which consists of the single fragment T_{low}. In steps 3.2 and 3.3, T is recovered and returned to the subject.

A Summary of the RADC Database Security Workshop

Teresa F. Lunt

SRI International, Computer Science Laboratory
333 Ravenswood Ave., Menlo Park, CA 94025

1 Introduction

On May 24-26, 1988, Teresa Lunt of SRI led a database security workshop at Vallombrosa Conference and Retreat Center in Menlo Park, California. About 25 researchers working on multilevel security for database systems attended. The workshop was the first extended technical interchange among those participating in the following projects, most of which were inspired by the 1982 Air Force Summer Study [1]: SRI's and Gemini Computer's SeaView A1 multilevel relational database system [2]; TRW's A1 Secure Prototype DBMS [3]; the Unisys B3 secure database system project; Honeywell's LOCK Data Views (LDV) project [4]; MITRE's Kernelized Trusted DBMS project; the Naval Research Laboratory's Secure Military Message System [5]; AOG Systems' secure entity-relationship (E-R) project [6]; MITRE's Integrity Lock project [7]; and the Hinke-Schaefer secure database project [8].

The workshop began with short presentations on the research projects currently under way, and most of the rest of the sessions were devoted to discussions of the advantages and disadvantages of the various approaches to multilevel database security that have been tried, difficult issues that have resisted solution, and new approaches to multilevel database security. Below we review some of the discussions on these topics.

2 Labels

The group debated the issue of trusted versus advisory labels for data. Some projects (e.g., SeaView) return an advisory label with the data because, according to the star property, the data returned to a user are classified at the subject class; moreover, any internal labels on the individual data elements have been handled by untrusted (relative to mandatory security) code, so they cannot be guaranteed to be correct for the stored data. Some projects (e.g., TRW's A1 prototype) do not return any labels with the data, on the theory that advisory labels could be harmful if they cannot be guaranteed to be correct. Other projects (e.g., the Unisys project) return a trusted label, with the mandatory TCB extending out to the user's screen.

The participants expressed diverse opinions about these approaches. Some said the lack of labels could be confusing if data are polyinstantiated. Others believed trusted labels are essential for trusted multilevel applications, and yet others doubted that users would make any use of trusted labels, whether returned directly by the database system or through a trusted application, because trusted applications are too complex and difficult to build.

3 Aggregation

The aggregation problem arises when a set of items of information, all of which are classified at some level, become classified at a higher level when combined. The group debated the approaches taken in LDV and SeaView. The LDV approach is to store all the data forming the aggregate at the low level, detect when the aggregate has been retrieved, and mark up (or withhold) the result. In the SeaView approach, all (or some) of the items forming the aggregate are stored at the aggregate-high level, and subsets can be retrieved at the lower level only through sanitization. Some participants agreed that storing all the aggregate data at a level lower than the aggregate level may not be safe. In addition, one attendee noted that withholding data from one file when a related file has been accessed (as in LDV) can create an unnecessary denial-of-service problem.

Another issue between LDV and SeaView is that many of the so-called aggregation problems that LDV is designed to detect can be readily solved through appropriate data design. When the intent is to hide sensitive relationships between data items that individually are not sensitive, the data can be stored at a low level and the sensitive relationships can be stored at a high level. The data are thus available to low subjects while the sensitive relationships are automatically protected by the underlying mandatory security mechanisms, without relying on complicated trusted software to detect violations. With adequate database design tools (such as the inference control tool proposed by Matt Morgenstern of SRI [9]), a proposed database design can be analyzed for such problems and restructured to eliminate or minimize the problems.

4 Discretionary Security

Dorothy Denning raised the question of whether a database could be partitioned by discretionary permissions as well as by classification. If so, stored data (i.e., storage objects) could be appropriately marked with their access control lists, and the discretionary authorizations for relations and views would be derived from those of the underlying storage objects. This offers the possibility of achieving greater assurance for discretionary security than if discretionary security attributes are associated with views because of the complex mechanism involved in view evaluation. Views do not partition a database because they may overlap, and views defined using conditional clauses may map to changing subsets of the database as the data are modified. Helena Winkler of Sybase explained that Sybase's secure database product (currently under development) associates access control attributes not with views but rather with the base relations, which are mapped directly to storage objects. Sybase believes that although access to the program implementing the view could be controlled, because the view compiler is untrusted no assurance exists that the compiled view maps to the same set of data that is described in the view definition.

Some participants believed that partitioning the database with discretionary security attributes would mean that a repartitioning would be required when authorizations were granted and revoked. Others pointed out, however, that no repartitioning or modifying access control lists is necessary if users are moved in and out of groups.

Lunt raised some other discretionary security issues for A1 and B3 database systems that stem from the requirements in the Trusted Computer System Evaluation Criteria [10] for support for group authorizations and specific denial of authorizations. Because the set of users and groups authorized for an object is independent of the set denied authorization, apparent conflicts between the two sets may exist, raising the questions: What does denial of authorization mean, and how do we reconcile the authorizations granted to users individually and as members of groups?

Denial of authorization can be used merely as a convenience in forming access control lists. For example, granting group G authorization and denying user U in G authorization can make the object in question available to everyone in G except U. With this interpretation, if user U is denied authorization by one user but is later granted authorization by another, then U becomes authorized. A stronger interpretation of denial is that denials take precedence over authorizations: One user's denial operation could not be negated by another's later grant operation.

In some applications, a user may belong to more than one group. In assigning privileges to subjects acting on behalf of a user, one must decide whether the subject should operate with the union of the user's individual privileges and the privileges of all the groups the user belongs to, whether the subject should operate with the privileges of only one group at a time, or whether some other policy should be adopted. These questions should be examined further [11].

5 The Homework Problem

Experience with applying MITRE's Integrity Lock secure database system to a particular secure database application motivated Rae Burns of Kanne Associates to devise a homework problem, which she posed to the group. The group broke up into three teams to work on it. They worked late into the evening and on the following morning animatedly discussed the homework problem, as the team leaders presented their approaches to the problem. Many of those attending considered the homework problem the single most valuable part of the workshop. The homework problem is appended to this paper.

6 Classification Semantics

Gary Smith of George Mason University led a discussion on the semantics of data classification. Smith believes that information should be classified at a level that reflects its contents, not its derivation. He introduced three dimensions to security semantics: content (e.g., flights to Iran can be classified because of the value 'Iran'), description (e.g., the fact that flights to Iran are classified may itself be classified), and existence (e.g., the existence of a flight to Iran may be classified, or the existence of classified flights may be classified). For data, he enumerated the following security semantics: data values classified by themselves (e.g., self-describing data or implicit associations), data values classified in association with an attribute name, multiple attribute associations, functional dependencies, temporal associations, and quantity aggregations.

The group also discussed automatic classification of text. Smith graded the following tasks from easy to hard: keyword search, classifying simple references, classifying disambiguated text, classifying text in limited domains, and classifying free-form text. Some argued that systems such as *Classi*, an automated text classification system proposed by Lunt and Berson [12], should be pursued because humans are not as consistent as machines that classify text and because systems like *Classi* could address the undersupply of qualified human experts. Others cautioned that such consistency may lead to a false sense of security if the inconsistency of the humans can be attributed to rules that were not captured by the expert system. They warned of the risk that the expert system may be used outside its domain of expertise, in which case the system should recognize this and answer, "I don't know." Untrusted subjects in the expert system classifier could tamper with the classification rules. In addition, many ways of signaling through text (e.g., modulating the space width) would be extremely difficult to detect by automatic classifiers or downgraders. Although some participants believed it might be reasonable to assume that *Classi* would not operate in a hostile environment, these issues underscore the need to investigate how to achieve a high degree of assurance for AI systems, such as *Classi*, that are used to assign access classes to text or data. In the absense of high assurance, a human may still be needed in the loop, but, as the group noted, putting a human in the loop is not an answer either.

7 Assurance

7.1 Balanced Assurance

Balanced assurance has been proposed in SeaView (and earlier by Roger Schell of Gemini Computers and others in formulating the Trusted Network Interpretation [13]) as a means of achieving A1 (or B3 or B2) assurance for the system as a whole by applying all the A1 (or B3 or B2) assurance techniques to the portion of the system enforcing mandatory security and less stringent assurance measures (comparable to Class C2) to those portions of the TCB enforcing the less critical security properties, such as database consistency and discretionary access control [14]. According to Schell, the idea of applying balanced assurance to database systems stems from a question raised by Dorothy Denning at the NCSC's Invitational Workshop on Database Security in June 1986 [15]. Her contention was that although views are not appropriate as objects for mandatory security, views could provide an extremely flexible mechanism for discretionary security, especially for content- and context-dependent controls. At that time, she felt strongly that the use of views for discretionary security in IBM's System R [16] pointed to the direction that database systems would take in the future and that requiring A1 assurance for discretionary security mechanisms would rule out view-based discretionary controls because of their complexity. (Denning has since begun to examine whether discretionary controls should be applied to storage objects rather than to views for some applications.)

The argument for balanced assurance is as follows. A system X meets the assurance requirements for Class C2 and operates in system-high mode at class c. Suppose system X is connected across a single-level connection at class c with system Y, an A1 system whose range of classes includes c. In the resultant system, we have A1 assurance that only information whose class is dominated by c can flow to system X. Because system X's C2 assurance was good enough to enforce its (nonmandatory) security policies before the connection was made, no further threat is countered by applying additional assurance techniques, such as formal analysis, to the C2 system. Thus, the overall system (comprising the A1 and C2 components) should meet the assurance requirements for Class A1. If we consider a Class A1 multilevel database system to be a collection of several single-level "virtual machines" (one for each class), each enforcing discretionary and consistency policies, each having C2 assurance, and each executing on an underlying A1 mandatory security kernel, the balanced assurance argument would be that the overall system is A1. The C2 portions of the system are constrained by the underlying mandatory security kernel and thus can introduce no compromise to mandatory security.

The discussion of balanced assurance, led by Bill Shockley of Gemini, was animated. Although some believed that users will not want systems that are "A1 here and C2 there," Shockley emhasized that balanced assurance does not mean just "slapping a C2 on an A1." Rather, the overall system should be well engineered. Just what system engineering requirements should be adhered to still needs to be defined.

7.2 TCB Subsetting

The balanced assurance argument goes hand in hand with the *TCB subsetting* argument [17]. TCB subsetting is a term introduced by Bill Shockley, drawing on earlier work by Marv Schaefer and Roger Schell on extensible TCBs [18], to mean structuring a TCB in layers, with each layer enforcing its own policies and with each layer constrained by the policies enforced by the lower layers. If a previously evaluated TCB is used for the lowest layer, as in SeaView's use of GEMSOS, TCB subsetting allows the addition of a new layer to form an extended TCB without disturbing the basis for the evaluation of the original TCB. With this layered approach, a mandatory security kernel as the lowest layer enforces mandatory security for the entire system without the need to verify the entire extended TCB for mandatory security.

Several workshop participants argued that TCB subsetting is the way of the future. The TCB subsetting approach allows third-party vendors to build independent products to extend a system's TCB to enforce additional non-mandatory policy without having to verify mandatory security. TCB subsetting also allows one to build a system that enforces different discretionary policies in different domains, with the underlying kernel providing domain isolation. For example, a mandatory security kernel could support three different domains — one for a file system, one for a database system, and one for a mail system — with each domain enforcing its own discretionary security policy. Thus, an underlying global discretionary policy need not be enforced by the operating system.

7.3 Layered TCB

The Unisys project is designing a layered TCB (composed of a system TCB plus component TCBs). Honeywell's LDV also has a layered TCB, with the LOCK TCB underneath, plus an additional LDV TCB on top, designed to facilitate proving properties about the LDV TCB. SeaView has a layered TCB, with the GEMSOS TCB underneath and a lesser-assurance TCB enforcing database consistency properties and discretionary access controls on views and multilevel relations on top. Some participants were concerned about enforcing the nonbypassability of the database system. LDV uses the LOCK type-enforcement mechanirm and "trusted pipelines" for this; SeaView uses the GEMSOS ring mechanism. Other participants pointed out that a Biba integrity category could also be used. Another alternative is a dedicated database machine. Several noted that discretionary access controls in the underlying TCB could not guarantee that the database system could not be bypassed.

8 New Approaches

The group discussed several new approaches to multilevel database systems. George Gajnak of AOG described a security model for entity-relationship systems and engendered a lively discussion contrasting his work with the SeaView model. Gajnak introduced what he called the *determinacy principle*, which requires that references be nonambiguous. He used an example to demonstrate the principal advantage of his secure E-R model over a relational model, namely, that in the relational model, one cannot avoid referential ambiguity when data is polyinstantiated. The problem is due to a fundamental weakness of the relational model. Because the relational model matches on value rather than establishing specific references for specific entities, when new data are added, new possibly inappropriate relationships are automatically formed. In AOG's secure E-R model, even though entities may be polyinstantiated, no referential ambiguity exists because a reference is not a relationship but applies to a particular tuple or value in the entity - that is, an explicit link to particular data is required. Thus, unlike the relational model, when a new entity is added in the E-R model, the old references do not apply to it. In the relational model, the higher the access class, the greater the ambiguity.

Another of the new approaches was presented by Bhavani Thuraisingham of Honeywell. In her talk, "Foundations of Multilevel Databases," she advocated applying formal logic to develop a theoretical foundation for multilevel databases. Cathy Meadows discussed multilevel security for an object-oriented data model and sketched how NRL's Secure Military Message System might be modeled as an object-oriented system.

Rae Burns presented what she calls a practical database security policy, that calls for certain features to be built into multilevel database systems to accommodate the requirements of the applications that will be built on top of them. These features include an interface for trusted applications that would provide trusted labels for elements and/or tuples (depending on the classification granularity), transaction authorization controls (as in the Clark-Wilson model), automatic classification and sanitization, automatic enforcement of classification of related data based on foreign keys (that is, the data that the foreign key refers to are constrained to be at least as high as the foreign key itself), no polyinstantiation, and enforcement of entity and referential integrity *inside* the DBMS kernel (that is, ordinary entity and referential integrity, not multilevel versions of them). For example, she feels that if a low user tries to insert a tuple when a high tuple with the same key already exists, he or she should be told that the data cannot be accepted. Although some of these requirements (namely, the automatic classification of related data, the prohibition of polyinstantiation, and the enforcement of ordinary entity integrity) may be in conflict with multilevel security and lead to potentially high-bandwidth covert channels, Burns said she would rather live with the covert channels than inflict polyinstantiation on the applications with which she is familiar.

On the whole, the group had many reservations about her requirements. First, automatic initial classification of data must be distinguished from automatic reclassification: Automatically reclassifying related data can create high-bandwidth covert channels. Also, the advantage of polyinstantiation is that low subjects need no access whatsoever to high data; thus, rejecting a low subject's request based on the existence of high data is not even an option. Moreover, the existence of multilevel secure database systems may change the way world does its business. Instead of mimicking the current way of doing business in the external environment and translating the paper world's low-bandwidth information flow channels into high-bandwidth covert channels, we should be building secure systems and requiring the external environment to adjust to achieve se- operation. In other words, today's research projects should create possibilities for the future rather than build around today's limitations.

9 Classifying Metadata

The group discussed how to classify metadata and views. The group examined whether a query is a labeled object and whether the data in a query, especially strings entered by the user, have classifications. The group agreed that a query is a labeled object classified at the subject class and that a user operating in a range of levels should be able to specify the level of the query. Then the level of the tuples returned should dominate the level of the query object. The group also agreed that a view definition is a labeled object with a classification at least as great as any relation or view it refers to, as in SeaView, and possibly also dominating the level of any strings it contains or the level of the subject that created the definition. A classified view definition is like a classified program: In order for a subject to execute the query defining the view, its access class must dominate the class of the view definition. The group also debated whether the level of the view definition or of the strings in the view definition should contribute to the level of the result of any query using the view. In the Unisys system, the level of the view definition is a lower bound for the result of any query against the view. In addition, in the Unisys system, if a user specifies a level $L1$ for a tuple to be inserted through a view V whose definition has access class $L2 > L1$, the operation is denied because there is information flow from the view definition to the data inserted through the view. Consequently, SECRET tuples cannot be inserted through a TOP-SECRET view, for example.

Asked who should be permitted to browse the descriptions of relations and views in the database, the group agreed that if a user is not cleared for a relation or view, he or she should not be able to read the description for the relation or view (the *description* is the names and types of attributes, as opposed to a view *definition*, which is the query defining the view). The group also believed that if a user does not have discretionary authorization for a relation or view, the user should not be able to read the description (because it would violate least privilege). A separate 'browse' or 'list' access mode, as in SeaView, can be used to allow users to be independently authorized to list the descriptions of relations and views they are cleared to see in a database.

The group agreed that metadata (such as integrity constraints and classification constraints) are classified at least as high as the relation(s) to which they refer.

10 Conclusions

The state of the art in multilevel database security has advanced considerably since the Air Force Summer Study, and the past few years have in fact made the findings of that study obsolete. Projects such as SeaView, originally inspired by the Summer Study, have demonstrated that many of the directions suggested by that study are unworkable. This is not bad news, however, because today's research projects have made possible the introduction of high-assurance multilevel database products in the near future. Moreover, the new research directions suggested at this workshop open up exciting new possibilities for the future.

The general consensus was that this was an extremely productive and successful workshop. Proceedings of the workshop will be published at the end of this year. Readers wishing to purchase copies of the proceedings should send their names to Teresa Lunt, SRI International, Computer Science Laboratory, 333 Ravenswood Ave., Menlo Park, CA 94025. A second workshop is planned for February or March 1989.

Acknowledgments

We are grateful for partial funding for this workshop from the U.S. Air Force, Rome Air Development Center (RADC), under contract F30602-85-C-0243. In acknowledgment of RADC's continued commitment to developing the theory and technology for multilevel database systems over the past fiften years, the workshop attendees unanimously moved to name this workshop the RADC Database Security Workshop. The goup gave a round of applause to Joe Giordano of RADC (who was unable to attend the workshop) for his part in creating today's research program in multilevel database security.

References

[1] Committee on Multilevel Data Management Security. *Multilevel Data Management Security.* Air Force Studies Board, National Research Council, National Academy Press, for official use only, 1983.

[2] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. A Near-Term Design for the SeaView Multilevel Database System. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[3] C. Garvey. ASD Views. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[4] P. Dwyer, E. Onuegbe, P. Stachour, and B. Thuraisingham. *An Implementation Specification for a Multilevel Secure Database Management System.* Honeywell Secure Computing Technology Center Interim Report, 1988.

[5] C. E. Landwehr, C. L. Heitmeyer, and J. McLean. A Security Model for Military Message Systems. In *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August 1984.

[6] AOG Systems and Gemini Computers. *Multilevel Secure Entity-Relationship Database Management System: Security Policy and Interpretation.* Draft Report, August 1987.

[7] R. D. Graubart and J. P. L. Woodward. A Preliminary Naval Surveillance DBMS Security Model. In *Proceedings 1982 Symposium on Security and Privacy*, April 1982.

[8] T. H. Hinke and M. Schaefer. *Secure Data Management System.* Technical Report RADC-TR-75-266, System Development Corp., Nov. 1975.

[9] M. Morgenstern. Controlling Logical Inference in Multilevel Database Systems. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[10] National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria.* Dept. of Defense, DOD 5200.28-STD, December 1985.

[11] T. F. Lunt. Access Control Policies: Some Unanswered Questions. In *Proceedings of the Computer Security Foundations Workshop*, forthcoming.

[12] T. F. Lunt and T. A. Berson. An Expert System to Classify and Sanitize Text. In *Proceedings of the Third Aerospace Computer Security Conference*, Orlando Florida, December 1987.

[13] National Computer Security Center. *National Computer Security Center Trusted Network Interpretation.* NCSC-TG-005, Version 1, July, 1987.

[14] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman and W. R. Shockley. Element-Level Classification with A1 Assurance. *Computers and Security*, forthcoming.

[15] National Computer Security Center. *Proceedings of the National Computer Security Center Invitational Workshop on Database Security*, Baltimore, MD, June 1986.

[16] M. M. Astrahan et al. System R: A Relational Database Management System. In *Computer*, Vol. 12, No. 5, 1979.

[17] W. R. Shockley and R. R. Schell. TCB Subsetting for Incremental Evaluation. In *Proceedings of the Second AIAA Conference on Computer Security*, December 1987.

[18] M. Schaefer and R.R. Schell. Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, April 1984.

Appendix: The Homework Problem

HI-TECH University	Final Exam (Take Home)
Database Security I	Due Date: April 1, 1988

The example database for this exam is taken from our primary textbook, Date's *An Introduction to Database Systems*, Volume 1, Chapters 16 and 27 (p. 279 in the third edition), Addison-Wesley, 1981. The description from the text is as follows:

> In this example we are assuming that the company maintains an education department whose function is to run a number of training courses. Each course is offered at a number of different locations within the company. The database contains details both of offerings already given and of offerings scheduled to be given in the future. The details are as follows:
>
> - For each course: course number (unique), course title, course description, details of prerequisite courses (if any), and details of all offerings (past and planned);
> - For each prerequisite course for a given course: course number and title;
> - For each offering of a given course: date, location, formate (e.g., full-time or half-time), details of all teachers, and details of all students;
> - For each teacher of a given offering: employee number and name;
> - For each student of a given offering: employee number, name, and grade.

Each exam question is based on this database application; each succeeding questions builds on the answers to the prior questions. You are advised to read the entire exam first and then to proceed to answer each question in turn.

1. *(5 pts)* Develop a data model diagram or an entity-relationship diagram for the education database described above.

2. *(5 pts)* Design a relational schema for the database. Include primary keys, foreign keys, and attribute data types. Use an "SQL-like" syntax, with any extensions that seem appropriate.

3. *(30 pts)* Using an"SQL-like" syntax, express the following application security policy, based on the relational schema developed in question 2. The policy statements assume only two levels of security: UNCLASSIFIED and SECRET.

   (a) Some courses are SECRET; all information about a SECRET course (including details of offerings, teachers, and students) is SECRET.

   (b) All course offerings at location "Pentagon" are SECRET.

   (c) If a course has a SECRET prerequisite, then the course is also SECRET.

   (d) Course information may be inserted, modified, or deleted only by a course administrator. At least one course administrator is cleared for SECRET data.

   (e) A course clerk enters offering, enrollment and student grade information; however, once entered into the database, such information may be modified or deleted only by a course administrator. No course clerks are cleared for SECRET data.

   (f) If a student has taken a SECRET offering, then the student's transcript is SECRET.

   (g) A student's grade point average (GPA) is UNCLASSIFIED but may be accessed only by a course administrator.

4. *(10 pts)* Briefly discuss the implications and ambiguities of at least two of the security policy statements above.

5. *(10 pts)* Specify two additional, or alternative, security policy statements that would be appropriate for the corporate education database.

6. *(40 pts)* Informally map four of the application-specific security policy statements from the previous questions (3 and 5) to a general DBMS security policy model. Discuss the relative success of the DBMS model in expressing and enforcing the application-specific policy. Address at least one data entry operation.

EXTRA CREDIT!

The extra credit question is based on the article by Morgenstern ("Controlling Logical Inference in Multilevel Database Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988, pp. 245-255).

Define the sphere of influence (SOI) for the corporate education database. Localize and describe the sources of inference channels, Revise the database schema design and security statements as needed to remove any open inference channels.

STRAWMAN TRUSTED NETWORK INTERPRETATION ENVIRONMENTS GUIDELINE

Marshall D. Abrams
Samuel I. Schaen
Martin W. Schwartz
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

Introduction

This paper provides a snapshot of ongoing support for the National Computer Security Center to produce Environments Guidelines for the Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria (TNI) [1]. The TNI Environments Guidelines (TNIEG) document identifies minimum security protection required in different network environments. Its relation to compater networks parallels the TCSEC Environments Guidelines [2] relation to stand-alone computer systems. The definition of environment is the same in both documents: "The aggregate of external circumstances, conditions, and events that affect the development, operation, and maintenance of a system."

Background

This section briefly describes Department of Defense computer and network guidance and processes.

The NCSC is responsible for establishing and maintaining technical standards and criteria for the evaluation of trusted computer systems. As part of this responsibility, the NCSC is developing guidance on how new security technology should be used. There are two objectives to this guidance:

a. Establishing a metric for categorizing systems according to the security protection they provide.

b. Identifying the minimum security protection required in different environments.

The Trusted Computer System Evaluation Criteria (TCSEC) [3] satisfy the first objective by categorizing computer systems into hierarchical classes based on evaluation of their security features and assurances. The TCSEC Environments Guidelines [2] satisfy the second objective by identifying the minimum classes appropriate for systems in different risk environments. These two documents, however, apply to stand-alone computer systems.

The TNI [1] satisfies the first objective by interpreting the TCSEC for networks. The TNI also provides guidance for selecting and specifying other security services (e.g., communications integrity, denial of service, transmission security). The TNI divides computer-communications networks into two groups: those which can be evaluated as a *Single Trusted System* (also called a network system) according to Part I, and those which cannot. Part I of the TNI is an interpretation of the stand-alone operating system orientation of the TCSEC for networks. These network systems can range from isolated local area networks to wide-area internetworks and are accredited as single entities by a single accrediting authority. Those computer-communications complexes that cannot be evaluated using Part I of the TNI are called *Interconnected Separately Accredited Automated Information Systems*, or *Interconnected AIS*. Interconnected AIS contain multiple systems (some of which may be trusted) which have been independently accredited. In these interconnected AIS networks the accreditor(s) may be forced to accept the risk of assessing network security without the benefit of an evaluation against the TCSEC principles contained in Part I of the TNI.

Purpose

The overall purpose of the TNIEG is to provide guidance to program managers, system security officers (SSO), designated approving authorities (DAA), and others concerned with selecting and operating trusted computer networks. For brevity, this audience is referred to as security managers.

Since the TNI treats network systems and interconnected AIS differently, but includes other security services which are applicable to both views, the TNIEG has several particular purposes. For network systems, one purpose of the TNIEG is to provide security managers with guidance as similar as possible to the TCSEC Environments Guidelines. For security managers of interconnected AIS, one purpose is to explain when systems can be interconnected and what data sensitivity levels are permitted on the connections. For all security managers, one purpose is to provide guidance as to which other security services (from Part II of the TNI) are required for their specific network. Since the guidance for the last two purposes must be less structured than TCSEC guidance, another purpose is to give security managers sufficient material to follow the guidance.

Scope

The TNIEG describes an environmental assessment process that determines the minimum level of trust recommended for specific network environments. The primary focus of this document (and also of the TNI) is on the hardware/software aspects of security; therefore, neither the TNIEG nor the TNI addresses all the security requirements that may be imposed on a network. Depending on the particular environment, communications security (COMSEC), emanations security, physical security, personnel security, administrative security, and other information security (INFOSEC) measures or safeguards may also be required. This document applies to DoD networks that are entrusted with the protection of information, regardless of whether or not that information is classified, sensitive, or national security related.

Two Views of the Network Environment

We begin by considering a series of increasingly complex networks, starting with an Automated Information System (AIS), and ending with complex networks of interconnected systems. This section enables security managers to determine whether to treat their network as a network system or as an interconnected AIS.

Individual Automated Information System (AIS)

The term Automated Information System (AIS) is used in several ways. Two definitions bound the uses. The first, narrower, definition which comes from the TNI states that AIS are "trusted commercially available automatic data processing (ADP) systems." The TNIEG calls such AIS *individual AIS*. An individual AIS may be formally evaluated against the TCSEC and given an evaluation class. An individual AIS has a coherent security architecture and design, and it has a trusted computing base (TCB).

The second broader definition, which comes from DoD Directive 5200.28 [4], *Security Requirements for Automated Information Systems*, states that an AIS is "an assembly of computer hardware, software, and/or firmware configured to collect, create, communicate, compute, disseminate, process, store, and/or control data or information." The directive also states that AIS include "stand-alone systems, communications systems, and computer network systems ...; associated peripheral devices and software; process control computers; embedded computer systems; communications switching computers; personal computers; intelligent terminals; word processors; office automation systems; application and operating system software; firmware; and other AIS technologies...."

This work was sponsored by the National Computer Security Center under contract F19628-86-C-0001. The contents of this paper do not necessarily represent the position of the National Computer Security Center.

The reader is cautioned that any particular use of the term AIS may apply only to an individual AIS, to any system satisfying the broader definition, or to some, perhaps unspecified, subset of the systems satisfying the broader definition.

The Single Network System

A more complex environment, including several AIS networked together with a coherent network security architecture and design, is called a *network system* or a *single trusted system*. A network system has a network trusted computer base (NTCB). The NTCB is partitioned among the components of the network, where a *component* is any part of a system that, taken by itself, provides all or part of the required functionality. A component may or may not have an NTCB partition. The essential point is that the NTCB as a whole satisfies the security architecture and design. Network systems may be evaluated against Part I of the TNI and may be given class evaluations like individual AIS evaluated against the TCSEC.

Network systems commonly provide security services that normally do not arise in individual AISs. These include communication integrity, denial of service countermeasures, compromise protection, and supportive primitives such as encryption and protocols. These services cannot be given a security rating class. They are, instead, evaluated against Part II of the TNI. The Part II evaluation is dependent on the Part I evaluation, in the sense that a Part II evaluation has low assurance if the Part I rating class is low.

The type of network that may be defined as a network system includes, for example, a group of departmental AIS having the same architecture connected by a local area network. Any component that does not enforce the full implementation of all policies must be evaluated as a component, not as a full network system.

Interconnected AIS

The most complex environment is referred to as *interconnected accredited AIS*, or simply, interconnected AIS. The term AIS, in this context, may be an individual AIS, a network system or even an interconnected AIS; i.e., it may be any system satisfying the broader definition given above which has communication capability and which has been previously accredited. Another way of stating the difference between network systems and interconnected AIS is that a network system exhibits a common level of trust at all external interfaces, while interconnected AIS do not. Therefore, interconnected AIS cannot be evaluated and given rating classes. Instead, they are accredited using Appendix C of the TNI to determine what sensitivity levels can be exchanged between the AIS. Part II evaluation applies to all networks.

There are several circumstances that would dictate using the interconnected AIS view, instead of the single trusted system view. These include connecting AIS with different architectures, connecting with an individual AIS that has not been evaluated, and connecting two previously accredited AISs. There are other circumstances where the evaluator/accreditor has a choice: consider one AIS a component of another AIS, and evaluate the whole system; or evaluate each AIS separately and connect them using the interconnection rules (Appendix C of the TNI).

Security Requirement Determination

The TNIEG guides the security manager in determining the recommended minimum security requirements for the network. Following the TNI, the security requirement computation is divided into two parts. During the first part, the security manager determines the type of network and the level of trust required for the given environment. For a network that can be evaluated as a single trusted system, the output is a TCSEC class. For a network that cannot be evaluated as a single trusted system, the output will be guidance concerning interconnection and a list of other possible concerns. During the second part, the security manager determines requirements for additional security services. The second part provides a list of functionalities, strengths of mechanisms, and assurances for TNI Part II concerns.

Protocol Layer Approach

The TNIEG use the Open Systems Interconnection (OSI) reference model [5] because it provides a well-understood terminology. The TNIEG, however, are independent of the actual protocol reference model used; they are applicable to all protocols.

A network system must fully implement all policies; but its NTCB need not be implemented in all protocol layers. The precise security services and their granularity will depend on the highest protocol layer at which the NTCB is implemented. For example, a Network Layer (layer 3) network such as DDN can, at best, distinguish among host addresses in providing discretionary access control. The Secure Data Network System (SDNS) [6], currently being designed, is expected to provide end-to-end encryption (E^3) systems at the Network, Transport and Application Layer. A proposed electronic mail specification expects to use SDNS at the Application Layer to provide access control that can distinguish among individual users.

The network system evaluator may be faced with the choice of evauating a single trusted system or accrediting the interconnection of AIS. There is an advantage to evaluating a group of components as a network system: the design may provide a distributed NTCB where one can show that untrusted or less trusted components provide services that are not critical to security. In contrast, accrediting interconnected AIS is often constrained by the weakest (least trusted) AIS.

Network System Evaluation Guidance

This section provides Part I environmental guidelines for network system, and discusses TNI Appendix C environmental factors for interconnected accredited AIS.

Single Trusted System View We now describe the process for determining the appropriate class of network system (evaluated under Part I) for a particular environment. They can be evaluated against the TNI in the same manner an AIS is evaluated against the TCSEC, and therefore the TCSEC Environments Guidelines applies as well.

To apply the TCSEC Environments Guidelines guidance, the security manager must determine the following:

a. Maximum sensitivity of data processed by the network

b. Whether the network security environment is open or closed. (Open or closed security environment refers to the conditions under which applications programs outside the TCB are developed. Open and closed environments are defined in [2].)

c. Minimum clearance or authorization of the network system users The term "user" must be interpreted broadly in a network system. It can include anyone who may be able to obtain cleartext information from a network and will ordinarily include operational personnel and users on attached hosts.

A table for mapping user clearance and maximum sensitivity of data into Rmin and Rmax respectively is contained in [2]. The algorithm for determining a Risk Index is:

$$\text{Risk Index} = R_{max} - R_{min}$$

Finally, the reference includes a table that maps Risk Index to an evaluation class of the TCSEC [3].

A special situation occurs when E^3 is present in the network. MITRE believes that two possible Risk Indexes should be considered and the larger of the two should be used in determining the required evaluation class for the network. It should be noted, however, that NSA has not yet endorsed this approach.[1]

As indicated in the TNI, an encryption mechanism is evaluated differently than other mechanisms. Evaluating encryption mechanisms has a long history predating the TNI, to which the TNI defers. Evaluation of an encryption mechanism is part of communications

[1] Dr Robert Shirey of the MITRE Corporation is acknowledged for contributing his thoughts on this matter

security (COMSEC). Customarily encryption mechanisms receive a rating of the highest level of classified information which may be protected using that mechanism. The TNI and the TNIEG respect that rating scheme. Therefore, the only rating applicable to an encryption mechanism is the classification level of the information protected. This classification level also establishes the requirement.

A more complicated situation exists when E^3 is employed above the Link Protocol Layer, layer 2. At layers 3 and 4 the protocols are concerned with the end systems or intermediate systems (e.g., hosts, network switches) that the links connect. Higher layers are concerned with other peer entities.

An E^3 system may be provided as (part of) an NTCB. When the E^3 system is integral to the NTCB, it requires evaluation under the TNI.

The TNI evaluation must consider (1) the Risk Index between the highest classification of data on the network and the lowest clearance of user with access to the network, and (2) the Risk Index for the bypass in the E^3 device.

a. **The range of sensitivity levels across the network.** This Risk Index is concerned with the difference between the highest level of information on any host attached to the network and the lowest clearance of a user that could potentially get access to that information. Depending on the characteristics of the network, this Risk Index could be larger or smaller than b. The worst case scenario occurs when some users have lower clearances than the level at which the network backbone is physically protected. For example, there are currently plans to allow some uncleared users on the DISNET segment of the DDN [7] which will be physically protected at the Secret level. In that case, the Risk Index for the bypass works the opposite of the normal case – the ciphertext side will be the higher of the two ratings.

b. **The bypass.** In an E^3 system, protocol control information must be sent around the encryption unit through a bypass. The software and hardware to implement the bypass must be trusted not to send user data through the bypass. A Risk Index can be computed based on the difference between the sensitivity level of the cleartext information and the sensitivity level of the untrusted components of the network.

The components which perform access control and key distribution must also be concerned with this risk range since improper key distribution could lead to compromise across the entire network. An erroneous distribution could potentially permit the lowest cleared user to access the highest classification of information.

Automated Information Systems (AIS) Interconnections Interconnecting AIS is a certification and accreditation issue rather than a commercial product evaluation issue. Accreditation is the management decision that a system can operate with acceptable risk in a particular environment, the TNIEG assist the security manager in the decision process. The interconnected AIS do not receive an evaluation, instead, guidance is given to the accreditor of interconnected AIS

Three factors affect the range of sensitivity levels permitted to flow over an interconnection

a Accreditation range of AIS

b Bidirectional or Unidirectional flow

c Global considerations

Accreditation Range of AIS Each AIS is accredited to operate over a particular range of sensitivity levels. These sensitivity levels are used to determine if communication between AIS is permitted. The accreditation range refers to the information being communicated.

Bidirectional or Unidirectional Flows If bidirectional flow is permitted, any permitted messages must have a sensitivity level within the accreditation range of each AIS. As explained below, this is necessary so that messages can be acknowledged without causing a write-down (viz. writing information at a lower sensitivity level normally a security violation). It follows that no information can flow over an interconnection unless it is in the accreditation range of both AIS. For example, if an AIS accredited to process TS-C data is connected to an AIS permitted to process TS-S data, information permitted over the interconnection must be labeled TS or S.

When only unidirectional communication (no acknowledgement) is utilized between two AIS, write up is permitted if each sensitivity level in the source AIS is dominated by some sensitivity level in the destination AIS. The receiving AIS must change the sensitivity level of the message when the message is received. Although write-up over a communications line is theoretically possible, it is not recommended in general because acknowledgements of packets are a write-down and must be prohibited. A preferred approach is to perform the write-up in a AIS which is accredited for both levels.

Global Considerations There are two global considerations that effect accreditation. The first is called ***propagation of local risk***, discussed below in conjunction with accredited but unevaluated AIS. The other global consideration that may reduce the range further is the "cascading problem". The Cascading Problem has been discussed elsewhere [1,8] and will not be addressed further in this paper.

When an accredited AIS is interconnected with a second accredited AIS, the first AIS treats the second as a device under the TNI. Each AIS is responsible for importing and exporting only messages which are permitted. The permissions depend on the evaluation level and sensitivity levels of the AIS. Finally, the AIS must be connected in such a way that no cascading problem exists.

Managing Propagation of Local Risk Connection of AIS under the jurisdiction of different accreditors requires agreement between these accreditors. This agreement is recorded in a Memorandum of Agreement (MOA). This MOA effectively constitutes a joint accreditation of the interconnected AIS. Interconnection of two or more AIS under the jurisdiction of a single accreditor is a special case, where the MOA may by replaced by a similar accreditation document.

Homogeneous Unevaluated AIS Homogeneous networks may be formed by interconnecting replicated identical unevaluated AIS. The accreditor may decide that no propagation of local risk problem exists within the set of homogeneous AIS. These homogeneous unevaluated AIS may be connected together in a closed network community, thus obviating the propagation of local risk problem. The closed community may be established by physical connection or cryptographic separation.

Hierarchically Related Accreditors As described above, the MOA documents a decision between peer accreditors. Another common case is hierarchically related accreditors. We will refer to one accreditor as the network accreditor and the other as the AIS accreditor. The network accreditor sets the rules to which the AIS accreditor must conform. The network accreditor is responsible for ensuring that each attaching AIS conforms to this set of rules. The responsibility of the network accreditor is analogous to a fiduciary in protecting the security of the attached AIS. The AIS accreditor permits his or her AIS to attach to the network based on an assumption that all other attached AIS conform to the same set of rules. Individual AIS accreditors do not have generally have the opportunity to approve any new AIS attaching to the network. Attaching a non-conforming AIS to the network would jeopardize the security of all attached AIS.

In the special case of a common user network such as DDN, it may be necessary to provide communications capabilities among non-conforming AIS. In general, these non-conforming AIS would be segregated into closed communities which could not directly communicate with conforming AIS. This approach is discussed in detail in [7].

It may be the case that operational necessity overrides the security needs of the attached AIS. The difficult question is: who makes the decision? One solution would be to require approval by cognizant authorities of all attached AIS. Alternatively, the question could be escalated to higher levels of command, until a single individual with authority over all attached AIS was reached. DDN, for example, requires JCS approval for such attachments.

TNI Part II Security Services

Part II of the TNI describes additional security concerns and services that arise in conjunction with networks that do not normally arise in stand-alone computers, and are not amenable to the detailed feature and assurance evaluation prescribed by the TCSEC. These security services provide communications security, denial of service, transmission security, and include supportive primitives, such as encryption mechanisms and protocols.

These concerns differentiate the network environment from the stand-alone computer environment. Some concerns take on increased significance in the network environment, other concerns do not exist on stand-alone computers. Some of these concerns are outside the scope of Part I; others lack the theoretical basis and formal analysis underlying Part I. For example, the TCSEC is based on a well-founded reference monitor concept and formal design methodology; there is no counterpart for this in Part II of the TNI. Part II of the TNI describes services responsive to these concerns and provides a qualitative means of evaluating their effectiveness. This section provides guidance on selecting security services for specific risk and applications environments.

Specification and Evaluation of Security Services Specifying and evaluating Part II security services is quite different from a Part I evaluation even though both parts are concerned with the same three aspects of security services or capabilities: functionality, strength of mechanism, and assurance. For clarity these terms are defined as follows.

Functionality refers to the objective and approach of a security service; it includes features, mechanism, and performance. Alternative approaches to achieving the desired functionality may be more suitable in different applications environments.

Strength of mechanism refers to how well a specific approach may be expected to achieve its objectives. In some cases the selection of parameters, such as number of bits used in a checksum or the number of permutations used in an encryption algorithm, can significantly affect strength of mechanism.

Assurance refers to a basis for believing that the functionality will be achieved; it includes tamper resistance, verifiability, and resistance against circumvention or bypass. Assurance is generally based on analysis involving theory, testing, software engineering, validation and verification, and related approaches. The analysis may be formal or informal, theoretical or applied.

Evaluation Ratings Part II evaluations are qualitative, as compared with the hierarchically ordered ratings (e.g., C1, C2, ...) from Part I. The results of a Part II evaluation for offered services are generally summarized using the terms *none*, *minimum*, *fair*, and *good*. For some services, functionality is summarized using *none* or *present*. The term *not offered* is used when a security service is not offered. For example, if a certain network did not include non-repudiation as one of its security services, that network would be rated "not offered" with respect to non-repudiation. Table 1 represents the evaluation structure of Part II as a matrix. It identifies a set of security services. It also shows the possible evaluation ranges for each service in terms of its functionality, strength of mechanism, and assurance.

Selecting Security Services Part II enumerates representative security services that an organization may choose to employ in a specific situation. Not all security services will be equally important for a specific environment, nor will their relative importance be the same among different environments. Selecting security services is a management decision to which the TNIEG provide input. The TNIEG first address strength of mechanism and assurance, following which there are a series of questions that help the security manager select minimum levels of functionality.

For example, consider the selection of communications integrity service to provide protection against message stream modification. A functionality decision is to select error detection only, or detection and correction; also one may select whether it is sufficient to detect a specific number of bit errors, error bursts of specified duration, or a specified probability of an undetected error.

Strength of Mechanism Strength of mechanism is based on a Risk Index similar to that applicable to Part I of the TNI. One significant difference is that R_{min} may be different than the Part I R_{min}. If the network is protected from access by unauthorized persons, the R_{min} will be based on the lowest cleared user. In general, this means that all end systems, switching processors, and communications lines must be physically protected from unauthorized users. If this condition cannot be met, e.g., because communications lines are in areas open to the public or because the network is attached to other networks with uncleared users, then the minimum clearance is assumed to be Uncleared (U). In this case R_{min}=0.

Table 1

Evaluation Structure for Network Security Services

Network Security Service	Criterion	Evaluation Range
Communications Integrity Authentication	Functionality Strength Assurance	None | present None to good None to good
Communications Field Integrity	Functionality Strength Assurance	None to good None to good None to good
Non-repudiation	Functionality Strength Assurance	None | present None to good None to good
Denial of Service Continuity of Operations	Functionality Strength Assurance	None to good None to good None to good
Protocol Based Protection	Functionality Strength Assurance	None to good None to good None to good
Network Management	Functionality Strength Assurance	None | present None to good None to good
Compromise Protection Data Confidentiality	Functionality Strength Assurance	None | present Sensitivity level None to good
Traffic Flow Confidentiality	Functionality Strength Assurance	None | present Sensitivity level None to good
Selective Routing	Functionality Strength Assurance	None | present None to good None to good

As an example, assume a guarded building with a local area network. The LAN has processors at the Secret level and only those individuals with Secret and higher clearances are able to use the terminals. The LAN is not connected to any other networks or communications lines outside the building. Only persons with at least a Confidential clearance are permitted into the building without an escort. In this case, an R_{min} of 2 (corresponding to the unescorted Confidential personnel) is used. In Part I evaluations of this system, an R_{min} of 3 (corresponding to Secret) would be used.

Table 2 now gives the strength of mechanism requirement based on the Risk Index calculated as

$$\text{Risk Index} = R_{max} - R_{min}$$

Table 2
Minimum Strength of Mechanism Requirement

Risk Index	Strength of Mechanism
0	None
1	Minimum
2	Fair
>2	Good

The Part II Risk Index for the system described above is 3 - 2 = 1. According to the Table 2, a minimum strength mechanism would suffice. If the scenaric described above were changed to include an unprotected communication line between buildings in an open space, the Rmin would be 0. The new Risk Index is 3 - 0 = 3, and a good strength of mechanism is required.

Assurance Assurance is a very important concept in the TCSEC and TNI. This section discusses the need for assurance and the ways in which it may be achieved.

One salient property of trusted network systems is the reliance on an NTCB. In addition to its other responsibilities, the NTCB prevents unauthorized modification to objects within the network system. In particular, the NTCB maintains the integrity of the programs which provide security services, thus ensuring that their assurance is continued. The NTCB provides an execution environment that is extremely valuable in enhancing the assurance of security services. Discretionary and mandatory access controls can be employed to segregate unrelated services. Thus, service implementation that is complex and error-prone or obtained from an unevaluated supplier can be prevented from degrading the assurance of other services implemented in the same component. Furthermore, an NTCB ensures that the basic protection of the security and integrity information entrusted to the network is not diluted by various supporting security services.

The relationship of the Risk Index to the required assurance is expressed in Table 3.

Table 3
Minimum Assurance Requirements

Risk Index	Part II Assurance Rating
0	None
1	Minimum
2	Fair
>2	Good

Assurance of the design and implementation of Part II mechanisms is also related to the assurance requirements in Part I because service integrity depends on protection by the NTCB. Table 4 expresses this dependency. The second column identifies th · minimum Part I evaluation which supports the Part II assurance requirement.

The reader should note that it is not valid to attempt to join Tables 3 and 4 to relate the Risk Index to a Part I rating class because Part I requirements, as expressed in Table 3, include functionality, strength of mechanism, and assurance, while Table 9 only addresses assurance. Furthermore, recall from the previous section that the Rmin for Part II may be different from that of Part I since many of the Part II protections are oriented towards outsiders rather than other users.

Table 4
Part II Assurance Rating

Part II Assurance Rating	Minimum Part I Evaluation
Minimum	C1
Fair	C2
Good	B2

Functionality of Specific Security Services

This section provides questions about each of the security services contained in Part II of the TNI. It devotes a series of questions to each security service. These questions are designed help the security manager identify the functionality required for each security service. In considering these questions, the security manager may wish to substitute a weaker noun for "requirement." The questions should be answered in sequence, unless the answer to one question contains an instruction to skip ahead.

Authentication

1. Is there a requirement to determine what individual, process or device is at the other end of a network communication?

   If no, skip to Communications Field Integrity.

2. Can you identify the basis for this requirement in a policy, concept of operations, or similar document?

   If not, you should confirm and document the validity of this requirement.

3. Do you have a requirement to identify and authenticate the specific hardware device at the distant end-point involved in the network communication?

   If yes, then you have a functionality requirement for authentication. This functionality may be implemented at one or more protocol layer. For example, a specific control character, ENQ (enquiry or who-are-you) may be used to immediately return a stored terminal identifier.

4. Do you have a requirement to identify and authenticate the location of the hardware at the distant end-point or in any intermediate system involved in the network communication?

   If yes, then you have a functionality requirement for authentication at protocol layer 2, the Link Layer or layer 3, the Network Layer.

5. Do you have a requirement to identify and authenticate the specific operating system or control program at the distant end-point or in any intermediate system involved in the network communication?

   If yes, then you have a functionality requirement for authentication at protocol layer 4, the Transport Layer.

6. Do you have a requirement to identify and authenticate the subject (process/domain pair) at the distant end-point involved in the network communication?

   If yes, then you have a functionality requirement for authentication at protocol layer 4 or above.

7. Do you have a requirement to identify and authenticate the application or user at the distant end-point involved in the network communication?

   If yes, then you have a functionality requirement for authentication above protocol layer 7, the Applications Layer. The Applications Layer provides an interface to the application. Authentication information may pass over this interface. Authentication of a user is addressed in Part I of the TNI. Application process authentication is outside the scope of the OSI Security Architecture, but does fall within the scope of TNI Part II Security Services.

Have you chosen to use some mechanism other than encryption to provide authentication? If so, your strength of mechanism is shown in Table 2.

If your authentication mechanism is encryption based, see the appropriate encryption authority (e.g., NSA for the DoD). Even if encryption is used some supporting processes may need to satisfy the strength of mechanism shown in Table 2 (depending on the architecture) For example, a database that relates encryption keys to specific users may need to be trusted

Communications Field Integrity

1. Do you have a requirement to protect communication against unauthorized modification?

   If no, skip to Non-repudiation.

2. Are your protection requirements the same for all parts of the information communicated?

   If no, then you should identify the separate parts and answer the rest of the questions in this section separately for each part. Each part is known as a field.

   There are two major fields: protocol-information, wherein the network is informed of the destination of the information and any special services required; and user-data. Not every protocol-data-unit (PDU) contains user-data, but protocol-information is necessary. Each of these fields may be divided into additional fields; depending on your application, protection requirements for fields may differ.

3. Do you have a requirement for detecting unauthorized modification to part or all of a PDU?

   If yes, you have a requirement for at least minimum functionality

3. Do you have a requirement for detecting any of the following forms of message stream modification: insertion, deletion, or replay?

   If yes, you have a requirement for at least fair functionality. In addition, your functionality must be incorporated in a connection oriented protocol.

4. Do you require that, if message stream modification is detected, recovery (correction) should be attempted?

   If yes, you have a requirement for good functionality. In addition, you must implement integrity in a reliable transport (layer 4) mechanism.

Non-repudiation

1. Do you have a requirement to be able to prove that a specific message transfer actually occurred?

   If no, skip to Continuity of Operations.

2. Do you have a requirement for proving that a specific message was sent? *Specific message* means that the identity of the subject sending the message, the host computer and/or mail agent/server, time and date, and contents are all uniquely and unalterably identified.

   If yes, then you have a functionality requirement for non-repudiation with proof of origin.

3. Do you have a requirement for proving that a specific message was received? *Specific message* means that the identity of the subject to which the message was delivered, the host computer and/or mail agent/server, time and date, and contents are all uniquely and unalterably identified.

   If yes, then you have a functionality requirement for non-repudiation with proof of delivery

Continuity of Operations

1. Do you have a requirement to assure the availability of communications service or to determine when a denial-of-service (DOS) condition exists? A *denial-of-service* condition is defined to exist whenever throughput falls below a pre-established threshold, or when access to a remote entity is unavailable, or when resources are not available to users on an equitable basis.

   If no, skip to Protocol Based DOS Protection.

2. Do you have a requirement to detect conditions that would degrade service below a pre-selected minimum and to report such degradation to the network operators?

   If yes, you have a requirement for at least minimum continuity of operations functionality.

3. Could failure of the system to operate for several minutes lead to personal injury or large financial loss?

   If yes, you have a requirement for at least fair continuity of operations functionality.

4. Do you have a requirement for service resiliency that would continue—perhaps in a degraded or prioritized mode in the event of equipment failure and/or unauthorized actions?

   If yes, you have a requirement for at least fair continuity of operations functionality.

5. Could failure of your system to operate for several minutes lead to loss of life?

   If yes, you have a requirement for good continuity of operations functionality.

6. Do you have a requirement for automatic adaptation upon detection of a denial-of-service condition?

   If yes, you have a requirement for good continuity of operations functionality.

Protocol Based DOS Protection

1. Do you have a requirement to probe or test the availability of service?

   If no, skip to Network Management.

2. The TNI suggests that the number of protocol based mechanisms could be used as the basis for determining the required functionality. Do you have an alternative basis for establish functionality requirements?

   If yes, you should employ this alternative basis and skip to Network Management

3. Do you have a requirement to detect a Denial of Service condition which cannot be met by the protocols used as part of normal communications?

   For example, you may require priority schemes that some protocols to not provide.

   If not, you do not have a functional requirement for protocol based DOS protection and should skip to Network Management

4. The TNI suggests the following protocol based mechanisms:

   □ Measure the transmission rate between peer entities under conditions of input queuing, and compare the measured transmission rate with a rate previously identified as the minimum acceptable;

   □ Employ a request-responce polling mechanism, such as "are-you-there" and "here-I-am" messages, to verify that an open path exists between peer entities.

   Have you identified any additional mechanisms required for your system?

   If so, include these additional mechanisms in your list of required mechanisms.

5. Based the previous question, how many protocol based mechanisms do you require? (Continue with the next question.)

6. Do you require that any protocol based mechanism be designed to not aggravate a Denial of Service condition?

   For example, request-response and poling mechanisms have been known to crash or overload packet switching networks.

   If so, add one (1) to your sum of protocol based mechanisms.

   The relationship of number of mechanisms and functionality requirement is shown in Table 5.

Network Management

1. Do you have a requirement for (at least) detecting a denial of service condition that affects more than a single instance of communication, or attempted communication?

   If no, skip to Data Confidentiality.

   If yes, you have a functional requirement for network management denial of service protection.

Table 5
Protocol Based DOS Functionality Mechanism Requirements

Number of Mechanisms Rating	Functionality Requirement
0	None
1	Minimum
2-3	Fair
>3	Good

Data Confidentiality

1. Do you have a requirement to protect any part of transmitted data from disclosure to unauthorized persons?

   If no, skip to Traffic Flow Confidentiality.

2. Is your requirement for confidentiality limited to selected field of user-data within a PDU?

   If no, then you require confidentiality for the entire data portion of each PDU. Continue with Traffic Flow Confidentiality.

3. Is there a reason to encrypt only selected fields (e.g., cost savings, legal requirements)?

   If yes, you require selected field confidentiality. If no, you require full confidentiality on the data portion of each PDU.

Traffic Flow Confidentiality

1. Do you have a requirement to prevent analysis of message length, frequency, and protocol components (such as addresses) to prevent information disclosure through inference (traffic analysis)?

   If no, skip to Selective Routing.

   If yes, you have a functional requirement for traffic flow confidentiality.

Selective Routing

1. Do you have a requirement to choose or avoid specific networks, links, relays, or other components for any reason at any time?

   If yes, you have a functional requirement for selective routing.

Conclusions

This paper has discussed ongoing work being done for the National Computer Security Center by MITRE. Plans call for a distribution of a draft version to a few dozen reviewers during the Summer of 1988 followed by revision and distribution of the document to the various services and agencies before the end of the calendar year. Depending on the nature of comments, a final version may be available in the Spring of 1989.

The prefered approach for accrediting systems is to evaluate a network system that includes end systems and intermediate systems, i.e., the full seven layers OSI plus the applications. The evaluation process of a single trusted system under Part I of the TNI is very similar to that of a stand done system and the environmental guidlines are similar as well. One important difference is that R_{min} may be based on users or operations personnel. The presence of E^3 requires that the Risk Index for maximum classification and minimum clearance be compared to the Risk Index for the encryption bypass; the larger of the two is used for environmental calculations.

Although a single-trusted-system approach is prefered, it is recognized that, at least for the near term, existing accredited AIS will be part of many networks. Since it is unlikely that these systems can be part of a single trusted system, a set of more-lenient interconnection rules has been established. These rules require the exchange of MOAs between cognizant accrediting authorities who are willing to accept the risk of attaching their systems to each other. In the case of a common user network, a network accreditor is under a fiduciary responsibility to protect the other AIS that have been attached previously. Any need to attach an AIS that does not satisfy the network attachment standards must be weighed carefully against the additional security threat which may be propagated to other AIS.

Part II of the TNI is more qualitative than Part I. Its requirements do not have the strong mathematical bases behind the Part I criteria. Part II applies to systems evaluated as a single trusted system as well as to systems of interconnected AIS. Part II uses the Risk Index as does part I, however the Part I and II indexes may differ for the same system because in some cases the R_{min} is different. To a large extent, Part I criteria protect users from other system users, while Part II requirements protect users from outsiders. Providing environmental guidance was difficult. Quite frankly, the guidance was based on measures that seemed right to the authors. It is the authors' hope that readers will consider the measures in terms of their own systems and they will provide the authors with feedback about whether the measures seem right to them in a sort of informal Delphi methodology.

LIST OF REFERENCES

1. National Computer Security Center, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," NCSC-TG-005, July 1987.

2. Department of Defense, "Computer Security Requirements - Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments," CSC-STD-003-85, 25 June 1985.

3. Department of Defense, "Department of Defense Trusted Computer System Evaluation Criteria," DOD 5200.28-STD, December 1985. Supersedes CSC-STD-001-83

4. Department of Defense, "Security Requirements for Automated Information Systems (AISs)," DOD Directive 5200.28, 21 March 1988.

5. International Standards Organization. "Information processing systems - Open Systems Interconnection Basic Reference Model," International Standard 7498, 15 October 1984

6. G. L. Tater and E. G. Kerut, "The Secure Data Network System: An Overview," *Proceedings of the 10th National Computer Security Conference*, National Bureau of Standards and National Computer Security Center, September 1987.

7. G. R. Mundy and R. W. Shirey, "Defense Data Network Security Architecture," *MILCOM '87 Proceedings*, 21 October 1987.

8. J. K. Millen, "Interconnection of Accredited Systems," *Third Aerospace Computer Security Conference*, pp. 60-65, American Institute of Aeronautics and Astronautics, Orlando, FL, December 7-11, 1987.

STANDARDS FOR NETWORK SECURITY

Kimberly E. Kirkpatrick
The MITRE Corporation
Burlington Road
Bedford, Massachusetts 01730

ABSTRACT

Security in the standards arena is emerging as a pressing topic for discussion and for imminent standardization. The International Standards Organization/Open Systems Interconnection (ISO/OSI) protocols are being implemented and accepted as the future standard for all communications networks. The need to provide security for each participant in an open system has emerged as one of the important riddles which must be solved before OSI will be used whole-heartedly.

This paper summarizes the security activities, as of May 1988, of the various standards bodies which are developing security-related standards within the context of the ISO/OSI reference model. In order to provide a coherent description of the various activities, the structure of the standards organizations is shown, the interactions among the organizations are explored, and the work progressing within each organization is summarized. In addition to a description of the security work progressing in the official standards organizations, an overview of programs within the DoD which are promoting the standardization of ISO/OSI security is provided. Finally, conclusions are drawn about the progress of standardization of security.

1.0 INTRODUCTION

There are several international and national United States standards bodies as well as United States government organizations which are working to develop security standards for the OSI Basic Reference Model [1]. All of the organizations exchange information about their work through specific, well-defined channels. Figure 1 shows the various organization and their official working relationships.

The international standards organizations which are working on the security aspects of open systems are the International Standards Organization (ISO), the European Computer Manufacturer's Association (ECMA), and the International Telegraph and Telephone Consultative Committee (CCITT). ISO has been a major contributor in the area of security with standards for security architecture and for security management. ECMA and CCITT have parallel committees to those in ISO; they contribute to the development of ISO/OSI security standards via liaisons.

The United States' national organizations which are associated with the ISO security work are the American National Standards Institute (ANSI), the Manufacturing Automation Protocol/Technical Office Protocol (MAP/TOP), and the National Bureau of Standards (NBS) Implementor's Workshop.

ANSI is the formal U.S. representative to ISO. Within ANSI, there are parallel committees to those in ISO. Each committee is the formal U.S. representative to the corresponding ISO committee.

The Manufacturing Automation Protocol/Technical Office Protocol (MAP/TOP) has user organizations world-wide and has the status of official contributor to the ISO committees. MAP, representing a consortium of factory automation users headed by General Motors, and TOP, representing the office automation community headed by Boeing, are developing specifications of specific subsets of ISO standards. In areas where ISO standards are not yet fully mature, the corresponding MAP/TOP committees are trying to accelerate the process of defining OSI standards by developing MAP/TOP solutions which are being fed into the appropriate ANSI committee. In addition, MAP/TOP is drawing on the standards being developed at the NBS Implementor's Workshop and using the Implementor's Agreements.

The NBS Implementor's Workshop has much the same goals as well as the same participants as MAP/TOP. The goals of the workshop are to promote multi-vendor interoperability quickly by developing Implementor's Agreements for a specified subset of international security standards' options. These agreements will form the basis for security capabilities within the Government OSI Profile (GOSIP), which are the specific

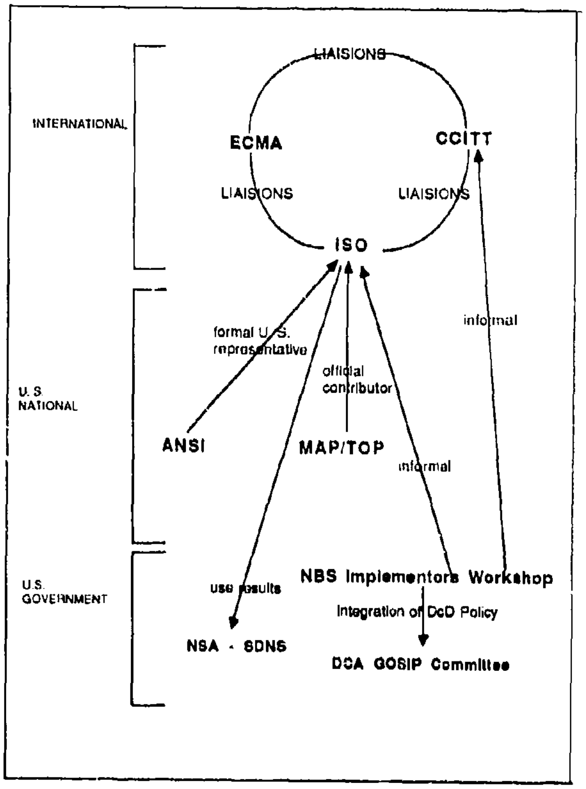


Figure 1. International and U.S. Organizations Working on ISO/OSI Security

subsets of international standard protocols to be procured by the Federal government.

GOSIP is being developed by the Defense Communications Agency (DCA) fo: use in the Internet. The other United States government agency which is developing security standards based on the OSI model is the National Security Agency (NSA) Their program is the Secure Data Network System (SDNS) program.

Both the international and United States national standards organizations are composed of committees which are devoted to particular topics. Within ISO, the Technical Committees (TCs) which have significant security activities are Information Technology Standards and Banking. Recently, the International Electronical Commission (IEC) has formed a specific liaison with the ISO TC97 committee (Information Processing) to sponsor the Joint Technical Committee 1 (JTC1). The IEC is an adjunct of ISO which addresses the standardization for electrical and electronic engineering equipment.

The five Subcommittees (SCs) within these TCs which have active security subgroups are Lower Layers, Data Encipherment, Architecture, Information Interchange, and Electronic Funds Transfer. Figure 2 lists these groups showing which subcommittees belong to each technical committee. The Working Groups (WGs) within these SCs which specifically are addressing security issues in OSI are shown in Figure 3. Within ANSI, there are several groups which contribute to the ISO Working Groups addressing security. Figure 4 lists those ANSI groups, showing the corresponding ISO Working Group.

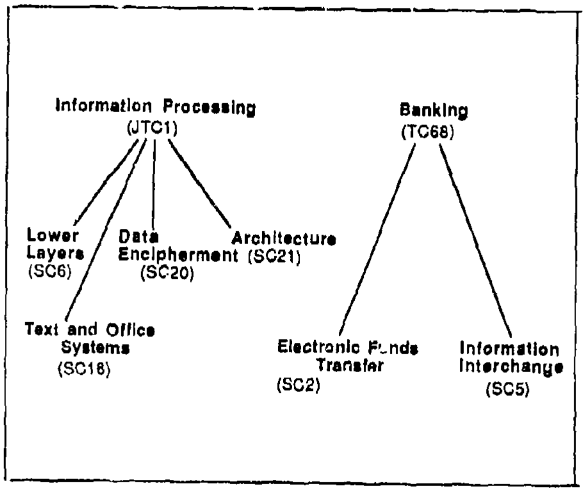
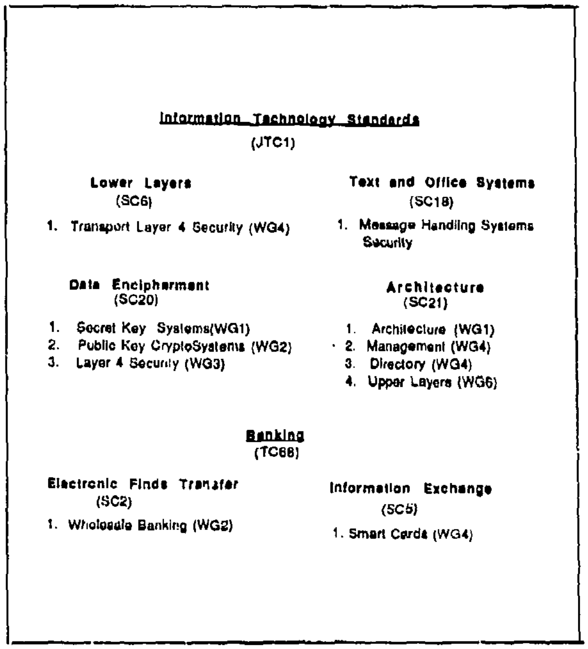


Figure 2. ISO Committees Working on ISO/OSI Security

The list of groups working on OSI security issues is extensive. However, the groups do make attempts to work together so that work is not duplicated unless such duplication is necessary. Thus among the various groups, there are many liaison efforts which are progressing security standards. Figure 5 and Figure 6 show the major, current liaison activities of each group as of May 1988.

The following sections summarize the particular security activities of each working group in the context of their international or national organization, and give a brief summary of any standards which the group is developing.



Information Technology Standards
(JTC1)

Lower Layers
(SC6)

1. Transport Layer 4 Security (WG4)

Text and Office Systems
(SC18)

1. Message Handling Systems Security

Data Encipherment
(SC20)

1. Secret Key Systems(WG1)
2. Public Key CryptoSystems (WG2)
3. Layer 4 Security (WG3)

Architecture
(SC21)

1. Architecture (WG1)
2. Management (WG4)
3. Directory (WG4)
4. Upper Layers (WG6)

Banking
(TC68)

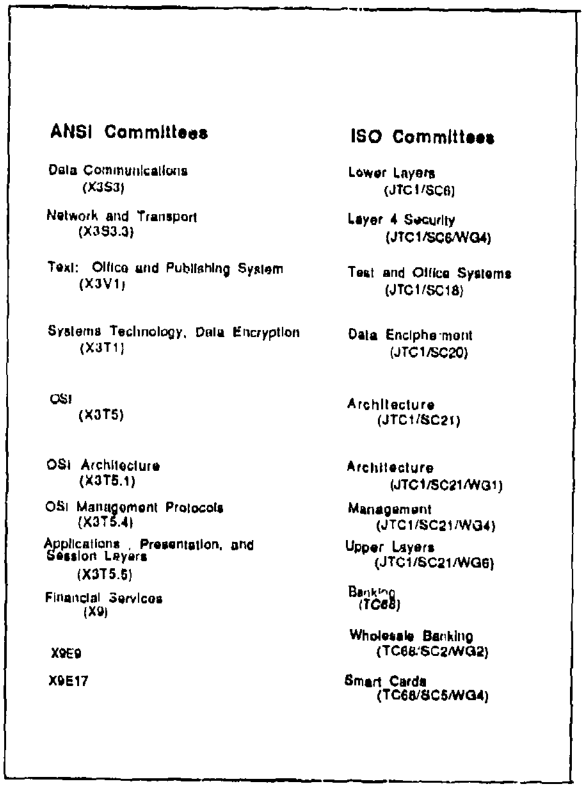
Electronic Finds Transfer
(SC2)

1. Wholesale Banking (WG2)

Information Exchange
(SC5)

1. Smart Cards (WG4)

Figure 3. ISO Working Groups Security Activities



ANSI Committees	ISO Committees
Data Communications (X3S3)	Lower Layers (JTC1/SC6)
Network and Transport (X3S3.3)	Layer 4 Security (JTC1/SC6/WG4)
Text: Office and Publishing System (X3V1)	Test and Office Systems (JTC1/SC18)
Systems Technology, Data Encryption (X3T1)	Data Encipherment (JTC1/SC20)
OSI (X3T5)	Architecture (JTC1/SC21)
OSI Architecture (X3T5.1)	Architecture (JTC1/SC21/WG1)
OSI Management Protocols (X3T5.4)	Management (JTC1/SC21/WG4)
Applications , Presentation, and Session Layers (X3T5.5)	Upper Layers (JTC1/SC21/WG6)
Financial Services (X9)	Banking (TC68)
X9E9	Wholesale Banking (TC68/SC2/WG2)
X9E17	Smart Cards (TC68/SC5/WG4)

Figure 4. ANSI Groups Security Activities and Corresponding ISO Groups

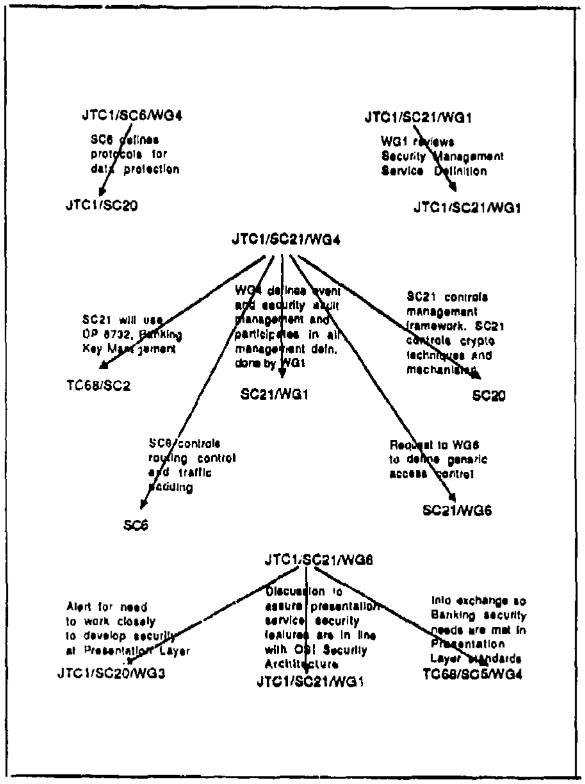

JTC1/SC6/WG4
SC6 defines protocols for data protection
JTC1/SC20
JTC1/SC21/WG1
WG1 reviews Security Management Service Definition
JTC1/SC21/WG1
JTC1/SC21/WG4
SC21 will use DP 8732, Banking Key Management
TC68/SC2
WG4 defines event and security audit management and participates in all management defn. done by WG1
SC21/WG1
SC21 controls management framework. SC21 controls crypto techniques and mechanisms
SC20
SC6 controls routing control and traffic padding
SC6
Request to WG6 to define generic access control
SC21/WG6
JTC1/SC21/WG6
Alert for need to work closely to develop security at Presentation Layer
JTC1/SC20/WG3
Discussion to assure presentation service security features are in line with OSI Security Architecture
JTC1/SC21/WG1
Info exchange so Banking security needs are met in Presentation Layer standards
TC68/SC5/WG4

Figure 5. ISO/JTC1 Liaisons

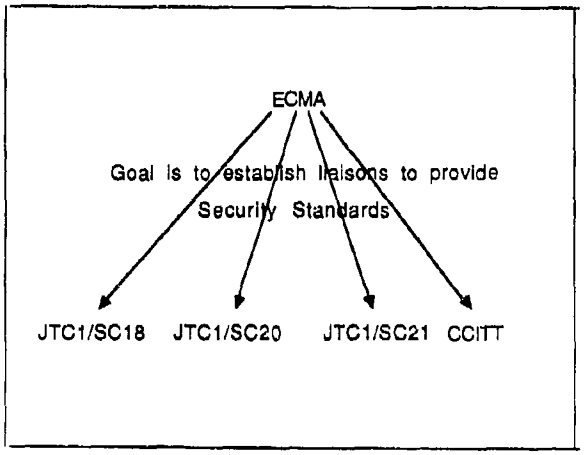


Figure 6. ECMA Liaisons

2.0 INTERNATIONAL STANDARDS ORGANIZATION (ISO)

The goal of the Open Systems Interconnection activity of the International Standards Organization (ISO) is to promote standardization of those functions needed to support communication between open systems. ISO is a voluntary organization whose voting rights are given to the national standards bodies of participating countries. For example, ANSI is the United States representative to ISO.

2.1 Information Technology Standards (JTC1)

The purpose of JTC1 is standardization, including terminology, in the field of information processing systems including, but not limited to, personal computers and office equipment. Figure 7 shows the organization of JTC1 security activities. The following sections describe the activities of each Working Group within each subcommittee (SC) as shown in Figure 7.

2.1.1 Telecommunications and Information Exchange Between Systems (SC6)

2.1.1.1 Transport, Layer 4 Security (WG4)

The working group is developing protocols which incorporate data protection services at the transport layer. The model for this protection of transport data will be taken from the work being done in SC20 or SC21. The members of SC6 do not intend to develop the model in their group.

2.1.2 Text and Office Systems (SC18)

The SC18 group works with ECMA TC32 and with CCITT on the Message Handling System (MHS) standards. The security portions of these standards present a security policy [2], a security model [3], a security service definition for the message transfer service [4], and a security service definition for the message store [5].

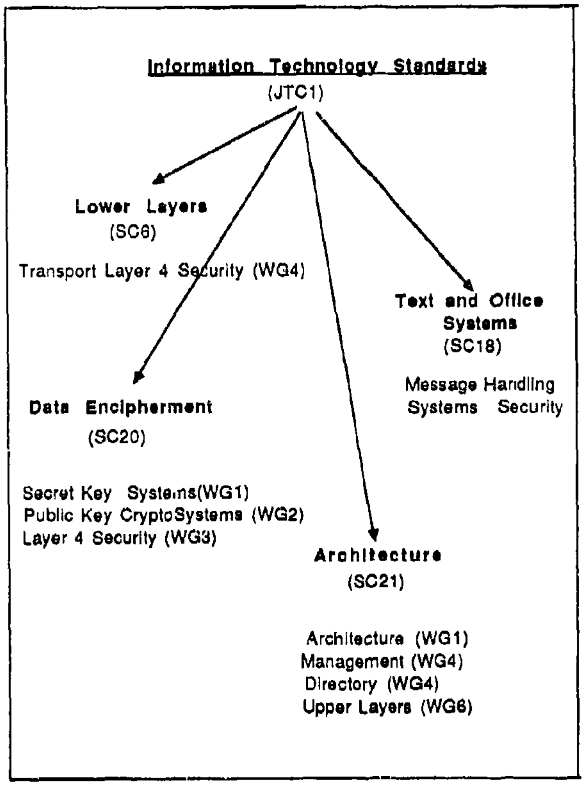


Figure 7. ISO/JTC1 Security Activities

2.1.2.1 Standards for Message Handling Systems

The Message Handling System standards are a set of standards which define the systems and services which allow users to exchange messages using a store and forward architecture. The basic functional entities of the Message Handling System are the User Agents (UAs), the Message Transfer System (MTS) which is composed of Message Transfer Agents (MTAs) and the Message Store (MS). The User Agents are application processes which submit and receive messages on behalf of the user. The Message Transfer Agents transfer messages and deliver messages to the destination specified by the user via the User Agents. The Message Store, which is an optional function, can by used to store and to permit retrieval of messages between the User Agent and the Message Transfer Systems. The security model for the system defines services which would allow these entities or components to be protected.

The security model has two views: Secure Access Management and Administration and Secure Messaging. Secure Access Management and Administration addresses "the establishment of an authenticated association between adjacent components and the setting up of security parameters for that association." Secure Messaging "covers the application of security features to protect messages in the Message Handling System in accordance with a defined security policy [2]."

The services provided for Message Handling security are based on ISO 7498/ Part 2 [1]. These classes of services are: message origin authentication, report origin authentication, probe origin authentication, proof of delivery, proof of submission, secure access management, content integrity, content confidentiality, message flow confidentiality, message sequence integrity, non-repudiation of origin, non-repudiation of delivery, and non-repudiation of submission.

The services within each class are defined in detail in X.402 [3] and are listed in Figure 8. The security parameters for the services provided by the Message Transfer system and the Message Store are defined in X.411 [4] and X.413 [5] respectively. The classes of threats which these services address are access threats, inter-message threats, intra-message threats, and data store threats.

The treatment of security in the Message Handling System is one of the most thorough definitions and set of services in any of the ISO standards.

2.1.3 Information Processing Systems Data Cryptographic Techniques (SC20)

The security activities in ISO/JTC1/SC20 are taking place in WG1, WG2, and WG3 in the areas of protocol development and algorithm registration. The ISO executive committee has voted that algorithms cannot be standardized by ISO committees, thus the work in SC20 may be slowing down. Although there is work progressing in the areas of Secret Key Algorithms and Applications (WG1) and in Public Key Cryptosystems and Mode of Use (WG2), these activities are similar to those of ISO/TC68 in the banking community.

2.1.3.1 Use of Encipherment Techniques in Communications Architectures (WG3)

The security activities of WG3 are the definition of a procedure for registering algorithms and the development of standards for cryptographic techniques in connection-oriented protocols and for public key encryption. The following sections summarize each of these efforts and point out areas where work is still to be done or perhaps redone.

2.1.3.1.1 ISO Register of Encipherment Algorithms. A set of procedures is being defined for registering algorithms. The following list is a first draft definition of what comprises the registration of an algorithm.

1. Unique Identification (assigned by SC20)
2. Proper Name
3. Range of Application (services provided)
4. Modes of Operation
5. Cryptographic Interface
6. Test of Words
7. Patent Information
8. References to Standards
9. Name of Sponsor

Origin Authentication
- **Message Origin Authentication**
- **Probe Origin Authentication**
- **Report Origin Autthentication**
- **Proof of Submission**
- **Proof of Delivery**

Secure Access Management
- **Peer Entity Authentication**
- **Security Context**

Data Confidentiality
- **Connection Confidentiality**
- **Content Confidentiality**
- **Message Flow Confidentiality**

Data Integrity Services
- **Connection Integrity**
- **Content Integrity**
- **Message Sequence Integrity**

Non-Repudiation
- **Non-repudiation of Origin**
- **Non-repudiation of Submission**
- **Non-repudiation of Delivory**
- **Message Security Labelling**

Security Management Services
- **Change credentials**
- **Register**

Figure 8. Message Handling Security Services

Information describing the algorithm and the strength of algorithm is not required for registration.

The work to be done in this area is to define completely the set of procedures as well as to define the terminology, such as "Encipherment Algorithm."

2.1.3.1.2 Standard for Data Cryptographic Techniques. The first working draft of the standard for data cryptographic techniques was developed as of September 1986 and as the introduction states, "the text is provisional and subject to radical change [6]." This standard will apply to the connection-oriented protocol specification DIS 8073 and will define all elements of cryptographic based data protection mechanisms except for the cryptographic algorithms. The work being done in the group is evolving toward a transport to network interface where encryption is a sublayer between the transport and network layers.

The intent of the standard as it is currently drafted is to provide mechanisms to support the security services defined in DP-7498/2. These services include peer-entity authentication, connection confidentiality, and connection integrity with and without recovery all at layer 4.

The elements of procedure or facilities which provide the mechanisms are listed below.

Connection Establishment
Peer-Entity Authentication

Data Encipherment
Integrity
Unique Sequence Numbers

The classes of end-to-end transport protocol layer services to which the standard applies are 1, 2, 3, 4. Class 0 is not included because it is designed for minimal functionality and the required cryptographic parameters cannot be accommodated in the variable header. The following lists the types of cryptographic protection for each transport protocol class.

a. Class 1: Basic Error Recovery Class

   Unilateral Peer-Entity Authentication
   Confidentiality of User Data

b. Class 2: Multiplexing Class

   Mutual Peer-Entity Authentication
   Confidentiality of User Data
   Manipulation Detection for all
   types of Transport Protocol Data Units
   Replay, insertion, and deletion detection
   for normal data stream

c. Class 3: Error Recovery and Multiplexing Class

   Mutual Peer-Entity Authentication
   Confidentiality of User Data
   Manipulation Detection for all types
   of Transport Protocol Data Units
   Replay, insertion and deletion detection
   for all user data.

d. Class 4: Error Detection and Recovery Class

   Same services as for Class 3
   Recovery from detected integrity errors
   is provided using mechanisms from class
   4 checksum failures

The data cryptographic techniques to be defined in this standard are for end-to-end protection of the data. Although no algorithms are specified, only certain algorithms are suitable for use with these techniques (those relevant to protection on an individual end-to-end connection basis). In addition, the techniques defined are only as secure as the security inherent in the management of the cryptographic keys.

2.1.3.1.3 Standard for Public Key Encryption. The standard for public key encryption is DEA 2 which specifies the algorithm to be used for public key encryption protocols [7].

2.1.4 Information Retrieval, Transfer, and Management for OSI (SC21)

The security activities in SC21 are being addressed in the working groups WG1, WG4, and WG6 in the areas of security architecture, security management, directory security, and security in the upper layer protocols such as presentation layer protocols.

There is a proposal for new work to develop an ISO Authentication Framework. Four areas would be investigated for this Meta-Architecture definition: Authentication (WG6), Access Control (WG1), Security Audit (WG4), and Non-Repudiation (WG1) with WG6 tracking the efforts for consistency. WG6 has done some preliminary work in this area which led to the decision that there is a need for a framework in this area.

The current efforts in security standards are listed in the following sections.

2.1.4.1 OSI Architecture (WG1)

WG1 addresses the architecture aspects of OSI. Their present effort is to develop an OSI security architecture.

2.1.4.1.1 OSI Security Architecture Standard. The standard for the OSI security architecture [8] is intended to extend the field of application of ISO 7498, the Basic Reference Model for Open Systems Interconnection, to cover secure communications between open systems. The standard has progressed to the level a Draft International Standard, DIS 7498-2, as of May 1987. this document is being revised to progress to full International Standard (IS) status in 1988.

The security architecture for the Reference Model is defined in terms of the security services and the related mechanisms which can be provided within the Reference Model framework. The definition and placement of these services and mechanisms form the core of the document. In addition, the security architecture standard defines the architecture for security management as well as providing a tutorial on security, a justification of security service placement in particular layers, and a guide for placement of encipherment in applications. The security architecture and the security management architecture are discussed here.

The security architecture supports the following categories of service.

Authentication
Access Control
Data Confidentiality
Data Integrity
Non-Repudiation

The allocation of these categories of service to each layer is shown in Figure 9.
Within each of these categories, specific services are defined. Much of the differentiation between services is to accommodate connection-oriented versus connectionless service provided by an underlying layer. These services are listed, but not described in Figure 10.

The types of mechanisms which can be invoked by the security services are:

Encipherment
Digital Signature
Access Control
Data Integrity
Authentication Exchange
Traffic Padding
Routing Control
Notarization

The mapping of the mechanisms to the categories of services which are allowed to invoke each mechanism is shown in Figure 11.

Layer 7 Application	Authentication, Access Control, Data Confidentiality, Data Integrity, Non-repudiation
Layer 6 Presentation	Data Confidentiality
Layer 5 Session	
Layer 4 Transport	Authentication, Access Control, Data Confidentiality, Data Integrity
Layer 3 Network	Authentication, Access Control, Data Confidentiality, Data Integrity
Layer 2 Data Link	Data Confidentiality
Layer 1 Physical	Data Confidentiality

Figure 9. Allocation of Categories of Services to Layers

Authentication

- **Peer entity authentication**
- **Data origin authentication**

Access Control

Data Confidentiality

- **Connection confidentiality**
- **Connectionless confidentiality**
- **selective field confidentiality**
- **traffic flow confidentiality**

Data Integrity

- **connection integrity with recovery**
- **connection integrity without recovery**
- **selective field connection integrity**
- **connectionless integrity**
- **selective field connectionless integrity**

Non-repudiation

- **Non-repudiation with proof of origin**
- **Non-repudiation with prood of delivery**

Figure 10. List of Specific Security Services

There are some additional mechanisms which do not provide any particular service at a given layer, but which are defined as "pervasive." These pervasive security mechanisms which appear to apply to, or would be used by, services provided by security management are:

Trusted Functionality
Event Detection
Security Audit Trail
Security Recovery

The basic framework for Security Management is included in the standard to guide the development of subsequent more specific standards on managing security [9]. Security management is defined as the management aspects of OSI Security that are concerned with those operations which are outside normal instances of communication, but which are needed to support and control the security aspects of those communications.

There are four categories of OSI security management activities which are defined.

1. System Security Management
2. Security Service Management
3. Security Mechanism Management
4. Security of OSI Management

System security management is concerned with the management of the security aspects of the overall OSI environment such as security policy or event handling. This type of management controls the pervasive mechanisms which apply to all layers, not a particular layer.

Security service management and security mechanism management are concerned with the management of particular services and mechanisms respectively. The management of services and mechanisms also applies to the circumstances under which a service is allowed to invoke a mechanism.

Security of OSI management protects all OSI management functions and the communication of OSI management information. This type of management may be the embodiment of Trusted Functionality and can be defined as the environment.

Service / Mechanism	Authentication	Access Control	Data Confidentiality	Data Integrity	Non-Repudiation
Encipherment	Y		Y	Y	
Digital Signature	Y			Y	Y
Access Control		Y			
Data Integrity				Y	Y
Authentication Exchange	Y				
Traffic Padding			Y		
Routing Control			Y		
Notarization					Y

Figure 11. Mapping of Mechanisms and Services

Although the Security Architecture standard is declared to be ready to progress to the status of a full international standard, there is some work which would make the standard more useable. Currently, the security architecture is described in terms of security services, mechanisms those services can use, and the layers in which the services may be available. There is no attempt to describe which combinations of services provide a particular level of security. For example, if reliable authentication is a goal of the system, then not only is the layer 7 service of peer authentication desirable, but the layer 4 service of connection confidentiality is useful to assure that no one has unauthorized access to the authentication information as it is transferred across the network.

Thus there is a need for an appendix or an accompanying guideline to provide some recommendation of how to combine security services within layers and at different layers to achieve secure communication among open systems.

2.1.4.2 OSI Management (WG4)

The two security efforts within WG4 are standards for security management services and for security in the directory services.

2.1.4.2.1 Standard for Security Management Services. The Security Management Standard [9] defines the specific services required to manage and to control OSI security. Currently, a working draft of the eventual standard exists and is being developed within the Security Management ad hoc group. It is expected to progress to a Draft Proposal (DP) in 1988. The security services are being defined so that they can use the Common Management Information Protocol (CMIP) [10] for information exchange.

The basic definitions and the architectural concepts used in the standard are based on the Security Architecture Standard [8]. The way the service definitions are structured is along the categories of management activities defined in the Security Architecture Standard as

1. System Security Management
2. Security Service Management
3. Security Mechanisms Management
4. Security of OSI Management

For the activities of System Security Management and Security Mechanism Management, the following aspects are defined: a) the facilities used to manage those activities, b) the functional units (general service capabilities), c) the primitives and parameters required by the service. The current list of specific activities which the group is investigating is listed below.

Event Handling management
Security audit management
Security recovery management
Key management
Encipherment management
Access control management
Data integrity management
Authentication management
Traffic padding management
Routing control management
Notarization management
Digital signature management

The activities of Event Handling Management and the Security Audit Management have very similar definitions. Either the activities will be combined or Event Management may be for the control and reporting of events and Security Audit may be for the logging of events and the controlling of groups of related event reports.

Currently, the Management Information Service Definition for security only addresses the management of system security and security mechanisms. The omission of Security Service Management needs to be discussed in the group. The omission of Security of OSI Management is legitimate as this type of management includes Trusted Functionality which will be specific to each system. Other items to be included in the service definition, but on which work has not been completed are: a) model for security management; b) conformance requirements; and c) annexes which provide background on the concepts and requirements of security management.

2.1.4.2.2 Standard for Directory Security Services. The Directory is "a collection of open systems which cooperate to hold a logical database of information about a set of objects in the real world [11]." These objects are application entities, people, terminals and distribution lists. The services offered by the Directory are service qualification, directory interrogation, directory modification, and error messages.

The service qualification services address the security requirements of a directory service. There are three services which make up the service qualification. One service is service control which sets limits on the use of resources such as the extent of a search requested. Another service is security parameters which protects directory information by indicating security level or type of protection a user wishes. The third service is the filter service which defines conditions that must be satisfied so that information may be returned to the user.

The directory interrogation service provides for reading, comparing, listing, searching, and abandoning a query. The directory modification service supplies services for adding entries, removing entries, and modifying entries. The error messages are errors and referrals to another service when one service fails.

Two directory protocols are defined. One is the Directory Access Protocol (DAP) which defines the communication between users and the directory. The other protocol is the Directory System Protocol (DSP) which defines the communication within the directory; this protocol addresses a distributed directory service.

The specific security standards being developed for the Directory Service are not currently included as part of the set of standards, but are in an annex. The rationale for keeping the security portions out of the standard is that security can be considered as a local matter which depends on the particular security policy of the local system. This justification appears to cover up a lack of knowledge about security. Effort needs to be expended to define a useful and useable set of services.

The model of security as proposed currently [12] consists of two aspects: access control and authentication. The control of user access to information is based on the use of access control attributes such as user/application identity, authentication information, or labels and the use of access conditions which relate the attributes to user operations on information.

For access control, no particular services are defined. However, mechanisms are suggested. These mechanisms are information on access conditions and access control lists; authentication information such as passwords; capabilities; and labels.

For authentication, three types of services are defined: no authentication, simple authentication, and strong authentication. For both access control and authentication, an operator defined as BIND is to be used. The BIND operator appears to associate the user-supplied credentials with the user's identity and deny or grant access as appropriate.

More work needs to be done to define a security model and an appropriate set of security services for the Directory.

2.1.4.3 OSI Session, Presentation, and Common Application Services (WG6)

Within WG6, there are two standards activities. One activity is a proposed security addendum to the Presentation Layer Standards. The other activity has been the development of an authentication framework.

2.1.4.3.1 Standards for Security at the Presentation Layer. The Security Addendum to the Presentation Layer [13] presents an argument for placing data encryption at the presentation layer. This proposed addendum is a preliminary discussion paper. The paper asserts that data encryption is a legitimate concern of the presentation layer. The service offered by the presentation layer is protected data transfer. The mechanism suggested for supplying this service is encryption.

The issues which the paper raises are the following.

1. The encryption function should be invoked on all user data in a specific presentation context rather than on all user data on the presentation connection.

2. The transfer syntaxes and mechanisms should be kept separate.

3. Compression should be accomplished before encryption so that data is not duplicated.

4. Well-known algorithms for compression and encryption should be registered.

2.1.4.3.2 Standard for Authentication. The purpose of this standard is to add services for authentication to ISO 8649 which is the Association Control Service Element (ACSE) definition. This first draft [14] outlines the definition of a framework for authentication by providing a model and specific services required for authentication. This draft also will be incorporated into the Authentication Framework. The review of this document is applicable to the Authentication Framework.

The scope of the standard, as stated in the draft document, is to specify the OSI communication services necessary to support Application Layer authentication for connection-oriented operation. The authentication services will be available for use with association establishment and authentication on already established associations. Four levels of authentication are meant to be supported: no authentication; identification of the remote peer entity only, without verification; simple authentication (passwords); and strong authentication (verification using cryptographic techniques).

The problem which this addendum to ISO 8649 needs to resolve is the close binding of services with mechanisms. The definition of the services includes the definition of the mechanisms the services should invoke. If the definitions of services and of mechanisms can be separated, then the standardization of authentication services should be much simpler.

2.2 Banking (TC68)

The security efforts by the banking community are well advanced and moving into new areas of technology. They have addressed the use of encryption for protection of data and are looking at mechanisms for authentication such as smart cards. Figure 12 shows the organization of TC68 security. The following sections describe the activities of each working group within each subcommittee (SC).

2.2.1 Electronic Funds Transfer (SC2)

The security activities in ISO/TC68/SC2 are taking place in WG2.

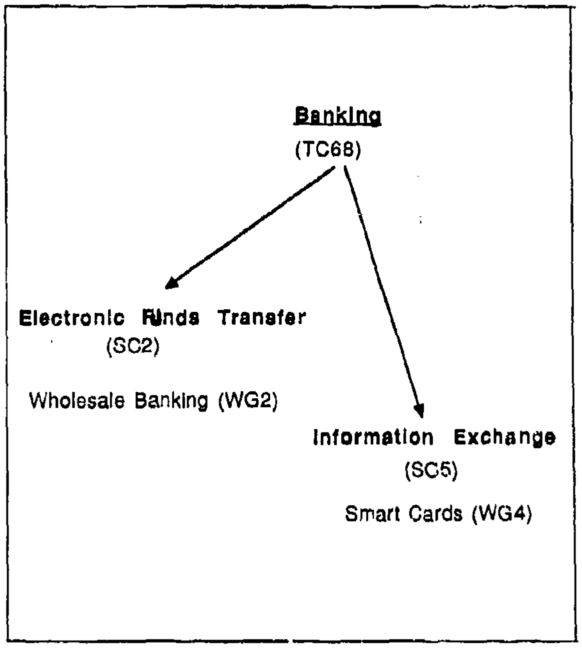


Figure 12. TC68 Security Activities

2.2.1.1 Wholesale Banking (WG2)

WG2 has produced two standards for security to define message authentication using encryption and encryption algorithms to be used for the message authentication. They are working on key management to distribute the encryption keys.

2.2.1.1.1 Message Authentication Standard. This standard [15] specifies how the Message Authentication Code (MAC) is calculated. The standard succinctly defines a MAC as "a data field attached to a set of data (i.e., message) passing between correspondent financial institutions and transmitted along with that set of data." Essentially, the authentication code is an encrypted checksum.

The level of security protection provided by this MAC depends on the protection of the authentication key and the strength of the algorithm used for [illegible] encryption.

2.2.1.1.2 Message Authentication Algorithm Standard. This standard [16] specifies the Data Encryption Algorithm (DEA) to be used in the calculation of the Message Authentication Code. The DEA is also published as ANSI X3.92 which is known as the Data Encryption Standard (DES).

2.2.1.1.3 Wholesale Key Management Standard. WG2 is using the ANSI standard for key management [17] which specifies the key management for wholesale financial institutions. Included is a specification of management of encryption keys and distribution of encryption keys.

2.2.2 Information Exchange (SC5)

This subcommittee has no drafts of documents available. However, some documentation of the work should be forthcoming in six months. The two areas in which SC5 is working are applications for financial messages and security in relation to smart cards.

3.0 EUROPEAN COMPUTER MANUFACTURER'S ASSOCIATION (ECMA)

ECMA is a regional organization in Europe. Its forty-five members are European computer vendors. The security activities within ECMA are paralleling those of ISO. The current focus of security in ECMA is within ECMA TC32. The groups within TC32 which are doing security work are listed below.

TG2 - Security Aspects of Distributed Interactive Processing
TG4 - Security Aspects of OSI Management
TG5 - Security in Distributed Office Applications
TG6 & TG7 - Security Facilities at Lower Layers of OSI Model
TG9 - Security Framework for Open Systems

Other elements in ECMA which are addressing security are listed below.

TC22 - Security of Data Base Systems
TC29 - Security Aspects of Documents

ECMA has stated their goals of establishing liaisons with the ISO groups of JTC1/SC18, JTC1/SC20, and JTC1/SC21 and with CCITT to provide security standards [18].

The most visible aspect of the ECMA work in ISO has been their proposed security addendum to ISO 7498 [1] to address general security services required by distributed applications. The purpose of this work is to define general services in order to avoid the extensive security definitions which had to be done for the specific distributed applications of Message Handling System and Directory Service.

4.0 INTERNATIONAL TELEGRAPH AND TELEPHONE CONSULTATIVE COMMITTEE (CCITT)

CCITT is collaborating with ISO to produce a series of documents which would form parts of a multi-part International Standard in ISO, and a series of Recommendations in CCITT. The current focus has been on the directory. The security aspects of the directory work have concentrated on the authentication framework for the directory.

4.1 Authentication Framework for the Directory

The Convergence Document for Directory Security [19] describes the role the directory plays in user authentications by providing credentials to users. There are two aspects to this authentication: services must be authenticated by the directory in order to obtain credentials of a user and the credentials obtained from the directory are used to authenticate the user.

Two types of authentication are defined: simple authentication and strong authentication. Simple authentication is defined as the use of a user name and an unencrypted password. Strong authentication is defined as the use of public key cryptography, specifically the cryptosystem specified in DP 9307, more commonly known as RSA (named after the authors Rivest, Shamir, and Adelman [20]).

Although the title of the document is authentication framework, the document only describes two particular mechanisms for authentication, a simple one and a strong one. Within the definition of the mechanisms, the types of mechanisms the directory would be required to supply are defined. There is no separate definition of services or of which services the directory could supply.

5.0 AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)

ANSI is the United States industrial standards organization. ANSI does not create standards, but does accredit organizations and committees which develop the standards. These committees are American National Standards Committees (ANSCs). There are four ANSCs: the Computer and Business Equipment Manufacturers Association (CBEMA), the Exchange Carriers Standards Association (ECSA), the Electronics Industry Association (EIA), and the Institute of Electrical and Electronic Engineers (IEEE). The security activities which support ISO are taking place in CBEMA in the X3 committee.

5.1 Data Communications (X3S3)

X3S3 is responsible for developing standards at the lower layers, transport and below.

5.2 Text: Office and Publishing Systems (X3V1)

X3V1 is responsible for standards which have to do with the office environment, such as message handling.

5.3 Systems Technology, Data Encryption (X3T1)

X3T1 develops standards for secure systems based on encryption mechanisms.

5.4 Information Processing Systems (X3T5)

X3T5 is the standards committee responsible for Open Systems Interconnection (OSI). X3T5 corresponds to JTC1/SC21 and is organized into task committees which parallel the SC21 organization.

5.4.1 OSI Architecture (X3T5.1)

X3T5.1 is responsible for the development of OSI Architecture X3T5.1 corresponds to ISO/JTC1/SC21/WG1. The current concern of X3T5.1 in security is to assure that the development of security standards continues and makes the best use of the limited security expertise in networking. To this end, X3T5.1 has suggested that security expertise should be consolidated into a single security group, in the form of a security rapporteur's group, in SC21/WG6. The meetings of this group should be located wherever current work is occurring in security. At present, the work in security services occurs in upper layers in X3T5.5 and SC21/WG6 [21].

5.4.2 OSI Management Protocols (X3T5.4)

X3T5.4 is responsible for the development of OSI Management Protocols and vocabulary for OSI management. X3T5.4 corresponds to ISO/JTC1/SC21/WG4 and works on any issues which surface in WG4. Currently X3T5.4 is submitting revisions to the Security Management Service Definition, Part 7 [9].

5.4.3 Application, Presentation, and Session Layers (X3T5.5)

X3T5.5 is responsible for the development of protocols at the upper layers of the OSI architecture. X3T5.5 corresponds to ISO/JTC1/SC21/WG6.

6.0 MANUFACTURING AUTOMATION PROTOCOL/TECHNICAL OFFICE PROTOCOL (MAP/TOP)

Currently, there is a group within MAP/TOP which is organized to look at security, but it has no output and is not very active.

7.0 NATIONAL BUREAU OF STANDARDS (NBS) IMPLEMENTOR'S WORKSHOP

Within the Implementor's Workshop is a Security Special Interest Group, the Security SIG. The goal of the Security SIG, as stated by the group, is to develop an overall OSI Security Architecture which is consistent with the OSI reference model and which economically satisfies the primary security needs of both the commercial and Governmen. sectors.

The areas currently being addressed by the Security SIG are a general security model (extension to DIS 7498-2), a security management model, security activities at the various layers such as the proposed Security Protocol addendum to DIS 8073 for transport security, and application security. The areas of application security are the Message Handling System application and the Directory Systems application.

The Message Handling System standards, according to the Security SIG's review, will provide a comprehensive specification for message handling comprising any number of cooperating open systems. The Message Handling System and services enable users to exchange messages on a store-and-forward basis.

The Directory Systems standards, according to the Security SIG's review, facilitate the interconnection of information processing systems to provide directory services. The directory provides the directory capabilities required by OSI applications, management, other OSI layer entities, and telecommunications services.

The area where work is complete is the listing of the desirability of the security services defined in DIS 7498-2 in each layer. A rating of High, Medium or Low is determined for each service at each layer. The matrix which describes the desirability can be used as a guide for choosing the appropriate services for various applications.

The Security SIG has the opportunity to influence the security of all the ISO standards. The SIG has established a core of people to follow the wo.k in various areas and to keep the SIG apprised of any developments while they work on the security standards to be included in the OSI Implementor's Agreements [22].

8.0 DOD PROGRAMS

The DoD programs which are going to use ISO/OSI protocols have only committed to use the security architecture document, DIS 7498-2. Thus they are committed to the basic premise of the security architecture, but not to the specific standards being developed within this architecture. In the face of no commitment, it is presumed that the DoD programs will develop their own security standards.

8.1 Secure Data Network System (SDNS)

SDNS is a research program which promulgates the design of the next generation of secure computer communications networks. The current efforts are the architecture, the services, the protocols, and the products. The SDNS products will be developed and fielded under NSA's commercial COMSEC Endorsement Program (CCEP).

SDNS is defining standard security services in layers 2, 3, 4, and 7; these service definitions are claimed to be consistent with those defined in ISO 7498-2. There are many working groups within the SDNS project trying to resolve the various issues such as access control, key management, and protocol definition. The intent is to promote SDNS in the OSI arena so that cleared United States' vendors will build SDNS products.

8.2 Internet and DDN

The Internet and DDN are required by DCA to transition to OSI protocols by 1990. The vehicle for DoD transition to OSI protocols is the Government OSI Profile (GOSIP). This profile is to be the standard reference for all Federal Government Agencies to use when acquiring and operating ADP systems or services and communications systems or services intended to conform to the OSI protocols. Currently GOSIP (Version 1, April 1987) includes two applications: File Transfer and Management (FTAM) and Message Handling (CCITT X.400) and four networking technologies: long-haul network connectivity (CCITT X.25), CSMA/CD (IEEE 802.3), Token Bus (IEEE 802.4), and Token Ring (IEEE 802.5). The specific options and parameters specified for each protocol are based on agreements reached at the National Bureau of Standards Implementor's Workshops.

9.0 SUMMARY OF ISSUES

Standards for security based on the ISO/OSI model are a fledgling area. Work has begun on these standards, but there is much left to do before security standards reach the International Standard status. Figure 13 summarizes the current activities in security standards for the organizations shown in Figure 1. Many of these standards are in preliminary draft paper form; there is additional work to be done before they can be useful even as a guide for how security can be integrated into the Basic OSI Reference Model.

Group / Layer	ISO	ECMA	CCITT	ANSI	MAP/TOP	NBS	DCA	NSA
Application	●	●	●	●		●		●
Presentation	●	●		●				
Session								
Transport	●	●		●				●
Network								●
Data Link								●
Physical								

Figure 13. Security Standard Activity Summary

The areas which have not yet been addressed are: LAN Security and any specific security concerns which are different from Long Haul networks and Network Layer Security or at least little work has been done here except by the United States' DoD.

The areas which need more work are: at the application layer: Security Management, Directory Security; at the presentation layer: more analysis of what security at this means although this does appear to be the appropriate layer in which to perform encryption as it is meant to translate from one representation of data to another; at the architecture level: frameworks or models of the security services.

The work in security standards which is progressing well is the Message Handling Security which makes a valiant attempt at a thorough definition of security. The ECMA work for security service definition of distributed applications is a much-needed effort which should be an area for immediate concentration.

The Banking community has made the most progress at addressing security problems in the area of encryption and in protecting their data using encryption techniques. The Information Processing area should follow their lead in addressing manageable problems and not try to solve everything.

The overall status of standards for security based on the ISO/OSI model is reflected in the NBS Security SIG whose purpose is to develop Implementor's Agreements. The Security SIG is instead discussing models of secure services, because there are no standards on which to base Agreements. This area is moving quickly, but more expertise is needed to contribute to the efforts.

REFERENCES

1. ISO/TC97, Information processing systems -Open Systems Interconnection - Basic Reference Model, ISO 7498, 1984.

2. ISO/TC97/SC18, Message Handling System and Service Overview, CCITT Draft Recommendation X 400/ISO Working Document for DIS 8505-1, Version 4, November 1987.

3. ISO/TC97/SC18, Message Handling Systems:/Information Processing Systems - Text Communication - MOTIS - Overall Architecture, CCITT Draft Recommendation X 402/ISO Working Document for DIS 8505-2, Version 5, November 1987.

4. ISO/TC97/SC18, Message Handling Systems:/Information Processing Systems - Text Communication - MOTIS - Message Transfer System: Abstract Service Definition and Procedures, CCITT Draft Recommendation X.411, ISO Working Document for DIS 8883-1, Version 5, November 1987.

5 ISO/TC97 SC18, Message Handling Systems:/ Information Processing Systems - Text Communication -MOTIS - Message Store/Part 1: Abstract Service Definition, CCITT Draft Recommendation X.413/ISO Working Document for DIS nnnn-1, Version 5, November 1987.

6 ISO/TC97/SC20/WG3, Information Processing - Data Cryptographic Techniques - Transport Layer Interoperability Requirements, September 1986.

7. ISO/TC97/SC20, Information Processing Systems - Data Cryptographic Techniques - specification of DEA 2, a Public Key Algorithm, DP 9307.

8. ISO/TC97/SC21, Revised Text of 2nd DP 7498-2. Information Processing Systems - OSI Reference Model - Part 2: Security Architecture, May 1987.

9. ISO/TC97/SC21/WG4, Third Draft for Management Information Services Definition - Part 7: Security Management Service Definition, September 1987.

10. ISO/TC97/SC21/WG4, DP 9596/2 - Management Information Protocol Specification - Part 2: Common Management Information Protocol.

11. ISO/TC97/SC21, Information Processing Systems - Open Systems Interconnection - The Directory - Part 1: Overview of Concepts , Models, and Services, ISO/DP/9594-1, July 1987.

12. ISO/TC97/SC21, Information Processing Systems - Open Systems Interconnection - The Directory - Part 2: Information Framework, ISO/DP/9594-2, July 1987.

13. ISO/TC97/SC21/WG6, Liaison Statement Concerning Development of Security Feature in Upper Layer OSI Standards, December 1985.

14. ISO/TC97/SC21/WG6, Initial Working Draft Addendum for authentication for ACSE Service Definition, ISO 8649.

15. ISO/TC68, Banking - Requirements for message authentication (wholesale), ISO 8730, 1986.

16. ISO/TC68, Banking - Approved algorithms for message authentication - Part 1: DEA, IOS 8731-1, 1987.

17. ANSI X9, American National Standard for Financial Institution Key Management - Wholesale, ANSI X9.17, 1986.

18. ECMA/TC32-TG9, TC32-TG9 Security in Open Systems, January 1987.

19. ISO/CCITT, ISO/CCITT Directory Convergence Document 3, The Directory - Authentication Framework, April 1986.

20. Rivest, R.L., A Shamir, L. Adelman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communication of the ACM, Volume 21, Number 2, February 1978.

21. ANSI X3T5.1, Comments on Organizing Open Systems Security Work, September 1987.

22. National Bureau of Standards Computer Science and Technology, Stable Implementation Agreements for Open Systems Interconnection Protocols Version 1 Edition 1, NBS Special publication 500-150, December 1987.

Secure Networking
at Sun Microsystems, Inc.

Katherine P. Addison
John J. Sancho

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, California 94043

ABSTRACT

Sun Microsystems is currently developing enhancements to its SunOS operating system to create a Trusted Computing Base (TCB) to be evaluated at the B1 level. Since the Sun system is a distributed collection of workstations and servers connected by a network, network security is a crucial part of the design. This paper describes the problems addressed and solutions found for secure packet routing and for passing mandatory access control labels over a network while remaining compatible with the existing SunOS system and with existing networking standards.

1. Introduction

Sun Microsystems is currently working on a secure version of the SunOS operating system to be evaluated at the B1 level. In this system, security labels are used to provide mandatory access control (MAC). This paper discusses Sun's solution for passing MAC labels over a network. The Secure SunOS architecture considers a collection of workstations as a single distributed TCB. For evaluation purposes, an entire configuration of Secure SunOS Hosts is considered to be a single "system", all of whose hardware must be physically secure. The mechanism used for network communication in this system is sockets. A socket is an end-point for communication that will have a MAC label. Just as MAC in accomplished in the file system by labeling the files, so will labeling the sockets facilitate MAC for networking. In fact, sockets contribute to the single system image of the SunOS system.

Because the system is distributed, there are problems to be addressed that do not need to be addressed by single host systems. While the end-points for communication across the network (the sockets) may be easily labeled, the problem of getting this label information across the network needed further investigation.

While researching secure networking, Sun also found that customer acceptance imposed conditions on the design. A major issue is that of trusting foreign (non-Sun) hosts on a network. Sun has received numerous requests from customers who wish to attach other vendor's hardware in a secure way to the Secure SunOS system. This configuration would not be covered by the NCSC evaluation but it is a highly useful and desired product feature. By attaching foreign hosts, customers have an easy migration path from the existing equipment to an evaluatable Secure SunOS system. However, this addition does raise serious new problems which must be dealt with in order to maintain a secure system.

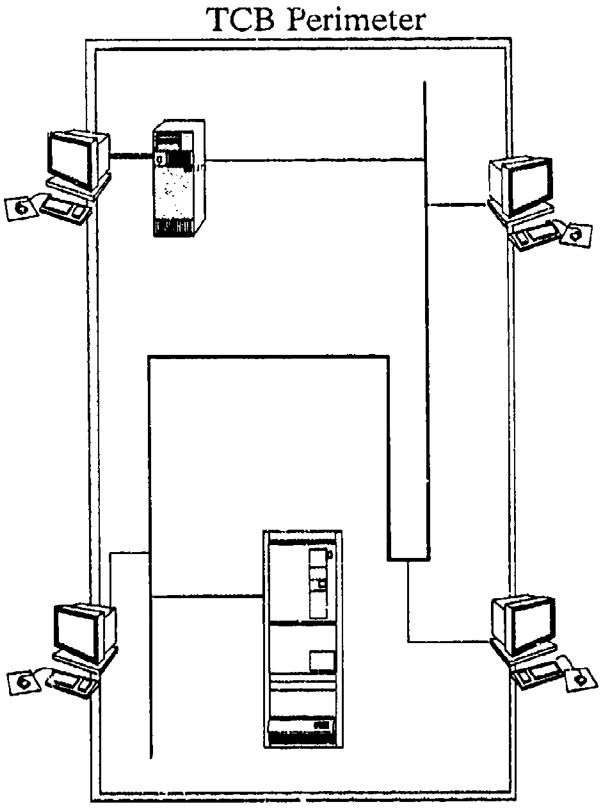


Figure 1: The Evaluated Configuration

SunOS and NFS are registered trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of AT&T.

Another desirable feature is the ability to limit the level of trust of a network. For the evaluated configuration, all networks would be trusted at the entire range of security labels in the Secure SunOS system (system-low to system-high). Other configurations may find it useful to be able to reflect differing degrees of trust on different networks. For example, a network completely made up of Secure SunOS Hosts might have a higher degree of trust than a network which contains Foreign Hosts. Again, this raises new issues. One such issue is secure routing. Given that different networks have different levels of trust, how does one machine send a packet through several gateways (IP routers) to a remote destination and guarantee that all intervening networks and hosts are allowed to carry a packet at that level of security?

2. The Overall Problem

The Secure SunOS TCB is made up of a collection of workstations as shown in Figure 1. It is very difficult to ensure security on a system which is distributed over several hosts. Performing access control decisions between processes on the same host is relatively easy since the operating system has all the information necessary to make such decisions. For processes on different hosts, this is not true. The current SunOS system uses sockets and the ARPAnet standard low-level protocols (TCP, UDP, and IP) to do network communications. The operating system on one host knows about the socket on the other host but does not know about the remote process and its label. This label information must be transmitted from one host to the other. Compounding this "labels over the network" problem is the issue of Sun's stated corporate objective of standards adherence. Thus there are constraints posed by the goals of the Secure SunOS system.

2.1. The Goals of SECURE SunOS

A summary of the goals of the Secure SunOS system are listed below. The goals also formed design constraints. As presented in [Sun87], these goals are

[1] Conformance With NCSC B1 Criteria

[2] Conformance With IEEE P1003 (POSIX)

[3] Conformance To Standard SunOS

[4] Maintenance of UNIX "Look And Feel"

[5] Functionality in Government and Commercial Applications

[6] Minimal External Changes

[7] Operation In Standard Sun Network

[8] Extensibility To Future Security Offerings

2.2. Constraints Driven By These Goals

These goals are in conflict with each other. For example, a POSIX conforming secure system which is backwards compatible to the existing SunOS system with the same UNIX "look and feel" and minimal external changes may not be possible.

The Secure SunOS system must conform to the standard SunOS system and there must be minimal external changes. The current system uses sockets and the ARPAnet low-level protocols, yet the protocols currently defined are not sufficient for a B1 system. The current protocols do not provide the label information necessary to make the correct access control decisions. However, a mechanism must still be found to pass labels across the network using these existing mechanisms. In particular, custom applications which use sockets must be made to run securely with NO modifications.

To compound the problem it must be possible for privileged servers (part of the TCB) to bypass the access control decision of the socket mechanism and instead make access control decision on their own. In other words, label-cognizant servers must have a way to handle multiple clients at different labels. Finally, it is highly desirable to have unprivileged servers be able to respond to multiple clients where the clients may be at different labels, and these servers should be able to run securely with no modifications.

At the time of this writing, Sun knows of no available products that overcome these problems and accomplish these goals.

3. Original Solutions Which Were Rejected

Sun looked at and rejected several solutions to the problem of providing MAC information across sockets. This paper will briefly touch on some of these.

3.1. Restricting Socket Usage To Privileged Processes

One of Sun's first proposals was to restrict sockets to privileged processes only (only the super-user could use sockets). This would have allowed some important SunOS programs to continue working (NFS, *rcp*, *rlogin*, etc.), though in those cases the server and client programs would have needed to be modified to do a label-checking handshake before performing the requested operation. This was not considered a serious problem, since those clients and servers are already privileged and would have required modification regardless of the approach chosen.

The disadvantage of this simple restriction was that it broke (A) all unprivileged users of the Yellow Pages (YP) global database look-up services (such as *ls*), (B) *suntools* and SunView in general, which use sockets to communicate among window-using processes, and (C) all miscellaneous unprivileged uses of sockets (*perfmeter*, user *telnet*, *lpr*, Unify, Alis, etc.). More importantly this solution directly conflicted with the goals of conformance to standard SunOS and minimal external changes.

3.2. Restricting Socket Usage To Single Hosts With a "Forwarding Daemon"

This proposal required that the operating system associate a label with each socket (the label being that of the process that created the socket), and check that this label was *equal to* the label of any process connecting or sending to the socket. This check would have been performed only for actions performed by unprivileged processes, and unprivileged processes would only have been permitted to communicate with sockets on the same host. In this case since both processes would have been on the same host, the operating system would have had all the information it needed.

To extend this mechanism beyond a single host, a "forwarding daemon" was proposed which would have been a privileged process which allowed communication between hosts by doing the mediation itself. The disadvantage of this proposal is that applications would have had to change to use the "forwarding daemon".

3.3. Restricting Socket Usage to system-low Processes

This proposal built on the proposal above by allowing multi-host socket usage if both processes were running with the system-low label. This permitted existing applications to run as long as the users or processes were at system-low. While this was seen as an improvement to the above proposals, it was unduly restrictive and of limited utility.

4. Chosen Solution

To overcome these restrictions, the chosen solution associates labels with packets. Thus, the label information is provided to the operating systems on both hosts. Since the IP standard already provides for options in the packet header, this mechanism has been chosen. The security label is put into a new label option in the IP packet header. This label option is a requisite for secure communications.

Sun's solution sets a label in the socket to the label of the process which creates it. This label is appended to the packets as an IP label option. The destination host then checks the label on the packet. If the received packet label is *equal to* the label of the destination socket (the target process), the packet is delivered; otherwise, it is dropped. If the packet is dropped, an audit message is generated, and the sending process is notified that access is denied. All socket communication is considered to be bi-directional in some sense (either explicitly, or in the form of acknowledgements), so unprivileged processes may only communicate via sockets if their labels are *equal*. Figure 2 shows communication between processes on two hosts. Only the two processes running at the same label are allowed to communicate.

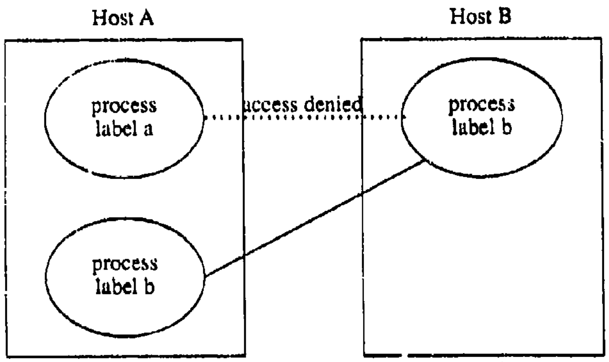


Figure 2: Unprivileged Socket Communication

As mentioned earlier, it is necessary for privileged servers (part of the TCB) to be able to make access control decisions themselves instead of being constrained by the socket label. For example, the network file system (NFS) needs to be able to respond to clients at any label regardless of its own socket label. For this reason, privileged processes are allowed to override the access control decisions made by the socket code by setting a new socket option, the *unconstrained* option. A privileged process is *trusted* to make the appropriate access control decision. A privileged processes is also allowed to send packets at any label. New system calls are provided to send at a different label than the label in the socket. New systems calls are also provided to return the label from the packet so that the privileged process has the information to make the proper access control decisions. The existing routines *getsockopt*() and *setsockopt*() are modified to set and get the socket label and the new unconstrained option. Communication with a privileged process is shown in Figure 3.

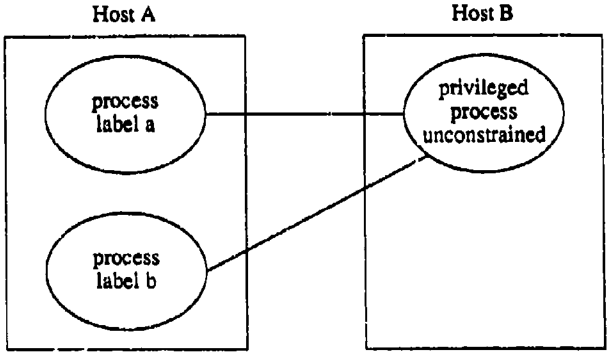


Figure 3: Privileged Socket Communication

It is also desirable for unprivileged servers to be able to respond to clients at varying labels. In the SunOS system, there is an existing server, *inetd*, which connects servers and clients. As it exists, *inetd* gets information about servers from a configuration file. *Inetd* listens for connections to the services in this file. When a connection is found, it invokes the server daemon specified. In the Secure SunOS system, *inetd* is modified to use the new unconstrained option on its sockets. It gets the label from the packet sent by the client and invokes the appropriate server daemon at that label. Thus, any application listed in *inetd*'s configuration file will be able to securely communicate with clients at multiple labels with no change to the existing code. In Figure 4, *inetd* is used to allow communication between an *rlogin* client and its server.

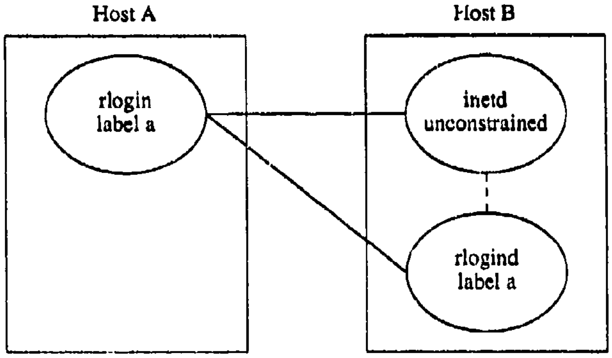


Figure 4: Multilevel Socket Communication Using Inetd

This solution requires no changes to either existing user or server applications. All applications may run if the processes on either side of the socket are at the same label. For servers that serve clients at different labels, again no changes are necessary, since *inetd* will do the access control decision for the application.

This solution has the advantage of using well-understood protocols with fairly localized extensions. This solution will have the look and feel of the UNIX system. The only difference seen by users is that they will get errors if there is a label mismatch between two processes wishing to communicate.

5. Non-evaluated Extensions

The solution stated above is necessary for the evaluated configurations but not sufficient for the existing customer base. Customers require that investment in existing hardware be preserved. In particular, the Secure SunOS system must work safely when connected to other vendors' gear (Foreign Hosts). The desired configuration is shown in Figure 5.

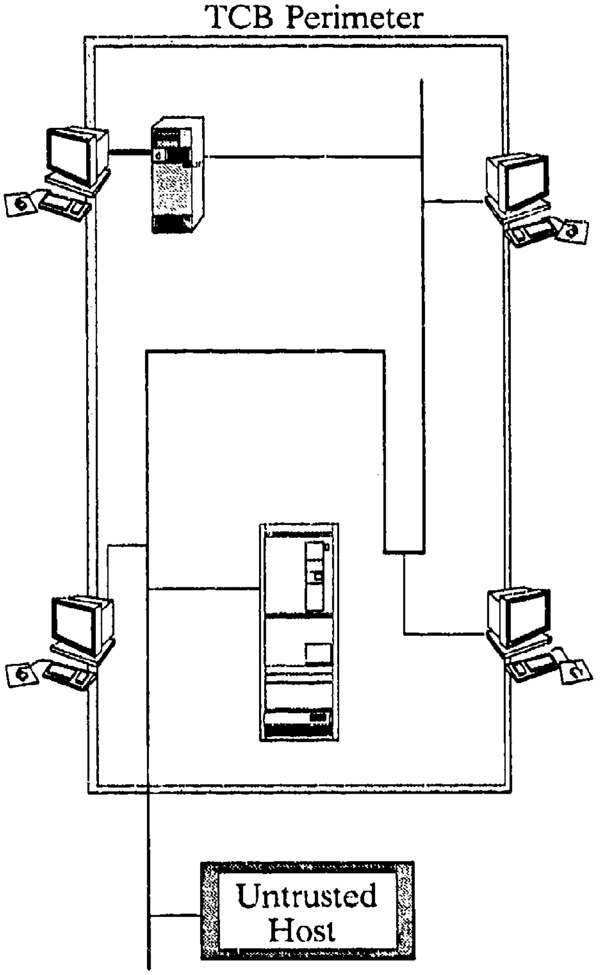


Figure 5: Extensions For Connectivity

Another desired feature restricts the label range of a network. This allows customers to restrict communication with an environmentally less safe network. Communication over the restricted network would only be allowed for originators within its range. Packets at labels outside the network range would never go onto the network.

Both these features raise new issues. The following sections will discuss the issues and how they are dealt with.

5.1. Foreign Hosts

5.1.1. The Problem

Foreign Hosts will not know how to send or receive labeled IP packets. Thus, Secure SunOS Hosts will need to communicate with these hosts without using packet labels. The Secure SunOS Hosts will need to know what label of information to use for each Foreign Host and will need to only send and receive information at this label. Figure 6 shows the desired result of communication between a and a Foreign Host.

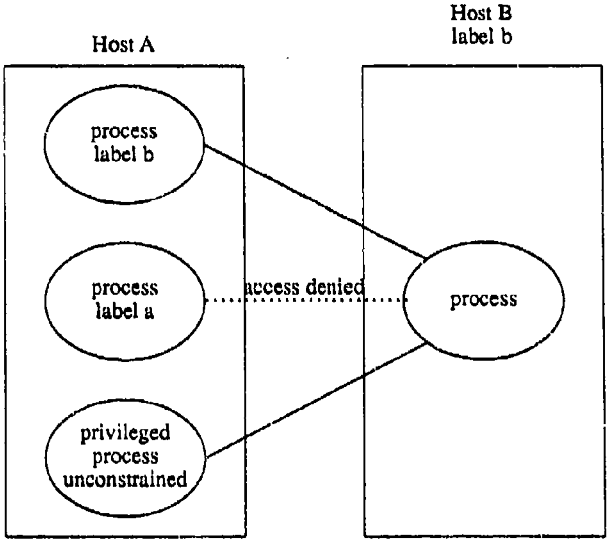


Figure 6: Communication With a Foreign Host

Obviously adding a Secure SunOS system to a network does not make Foreign Hosts secure. There are rules that Foreign Hosts must follow in order to be considered safe. They are trusted not to read packets which are not addressed to them. They must act benignly if a labeled packet is sent to them. Foreign Hosts are also trusted not to communicate with each other if they are at different labels. Foreign Hosts are trusted to be well-behaved in general: they should not pretend to be another host and should not try to act as a server to Secure SunOS Hosts which are booting or asking for Yellow Pages information. Despite these assumptions about Foreign Hosts, the problem of how to coexist safely with hosts that do not understand the IP label option needed further investigation.

5.1.2. Rejected Solutions

The first solution considered for this problem was to define that Foreign Hosts were always at system-low. This seemed unduly restrictive. It also did not address the real problem of determining which hosts were Foreign Hosts and which were Secure SunOS Hosts. Since it was already necessary to keep information about which hosts were the Foreign Hosts, label information about those hosts could also be kept. In fact, information about all the hosts had to be kept in order to distinguish between Foreign Hosts, Secure SunOS Hosts, and hosts which were unknown.

This raised the new problem of how to keep this information. The next solution was fairly obvious: keep a database which had information about every host on the internet. This was also quite restrictive since it meant that a host could only communicate with hosts in its database. Adding a host anywhere on the internet would have required updating every other host. Along with being an administrative nightmare, this restriction would have meant that connection to the ARPAnet would never have been allowed since it would have been impossible to have a database of all other hosts on the ARPAnet.

5.1.3. Chosen Solution

Sun's solution allows broader freedon of communication. Each host is not required to know about all the other hosts on the internet. Instead it uses the Secure SunOS gateways to take care of access control decision for all hosts to which it is not directly connected. If Host A is not on the same network as Host B, Host A will send a labeled IP packet to a gateway. This gateway will forward it on to any additional gateways until the packet reaches a gateway which is directly connected to Host B. That gateway will determine whether Host B is a Foreign Host. If Host B is a Foreign Host, the gateway will determine whether communication with Host B is allowed at the label in the packet's IP label option. If communication is allowed, the IP label option will be stripped from the packet, and the packet will be delivered to Host B. If communication is not allowed, the gateway will return information to Host A that the packet was not delivered. Finally, if the gateway determines that Host B is a Secure SunOS Host, then it will forward the packet to Host B where the appropriate access control decision will be made.

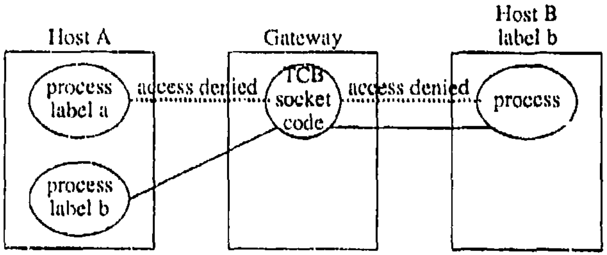


Figure 7: Communication With a Foreign Host Through a Gateway

Correspondingly, if a Secure SunOS gateway receives a packet from a Foreign Host, it will add to the packet an IP label option with the label of the Foreign Host. It will then forward the packet through intermediate gateways until finally it reaches a gateway directly connected to the destination host. This gateway will make the access control decisions as described in the paragraph above. Note that if there is just one gateway between two Foreign Hosts, the gateway would not need to add an IP label option and then strip it.

This allows Foreign Hosts to act as single-labeled gateways. Since Foreign Hosts are trusted to only communicate with other Foreign Hosts at their own labels, Secure SunOS Hosts can assume that any packet received from a single-labe gateway should only be accepted for processes running at the corresponding label. These Secure SunOS Hosts will also only send information at the gateway's label through the single-label gateway.

Hosts are not required to keep information about every host on the internet. Instead, a host only needs to know about hosts on its own network. Gateways provide the needed access control checking for other hosts. The following decision is made by each host and gateway in determining whether to add an IP label option to the packet: if the immediate destination of the packet is a Foreign Host, do not label the packet, otherwise label it. The immediate destination is the final destination if the hosts are on the same network, otherwise the immediate destination is a gateway.

One interesting point is diskless machines. When a diskless machine is booting, it has no knowledge of any hosts including itself. This is solved by treating diskless machines as Foreign Hosts at the system-low label while they are booting. Once the diskless machine has sufficient knowledge about labels, it is returned to its status as a Secure SunOS Host.

5.2. Secure Routing (Allowing Networks With a Restricted Label Range)

The other desired extension to the Secure SunOS system is the ability to restrict the range of communication allowed on a network. This feature might be used for a network which includes a Foreign Host. It might also be used for a link between two building networks as shown in Figure 8.

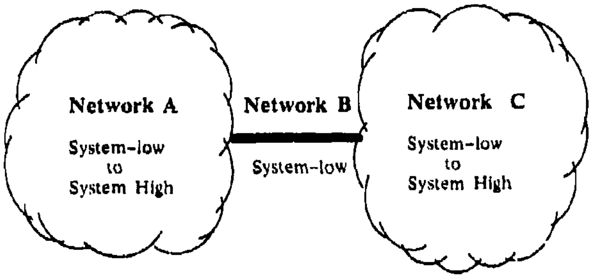


Figure 8: Extensions For Restricted Networks

5.2.1. The Problem

When an IP packet is transmitted from a host, the lowest layer software module must choose a network interface to use. This is easy if the packet is addressed to a host attached to a local network. If the sending host does not have a direct connection to the destination network, it must select another host to use as a gateway. This choice is more difficult: there may be several candidate gateways, each able to correctly deliver the packet, but some of these may provide better (shorter) paths to the destination than others.

A routing protocol is used to help make these decisions. Routing daemons on every host exchange information about known routes at short intervals. When these daemons start up, they only know about networks to which they are directly attached. This information is gradually dispersed among all the routers in an internet so that each router eventually learns about routes to all networks.

In the existing SunOS system, Sun uses the distance to a host, or hop count, as a measure of the cost of a route. Sun can use this information to reduce the size of local routing information, by only storing the best route to a network.

In the Secure SunOS system, each network has a label range associated with it. Hosts attached to the network are only prepared to accept traffic which contains a label within the network range. Herein lies the routing problem in a secure network: a packet which is sent along an otherwise correct route may be discarded before getting to the destination because it encountered a network whose label range did not include the label of the packet in transit. As illustrated in Figure 9, a packet labeled f sent from Host A to Host B will be dropped by Host Y, so it must be sent through Host X. Since both paths have the same cost, the existing route protocol will only know about one of the paths.

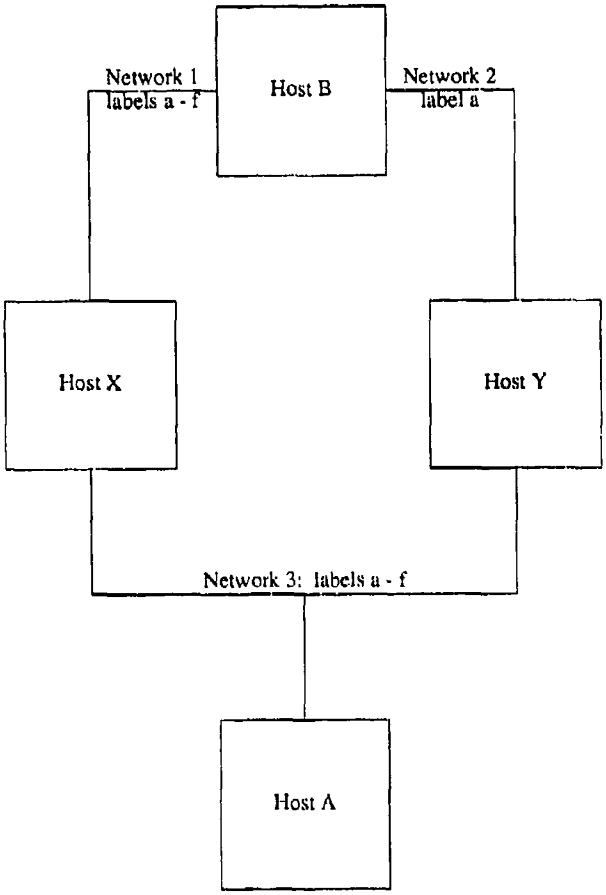


Figure 9: The Routing Problem

5.2.2. Rejected Solutions

Initially, Sun thought the problem could be solved by making all gateways trusted, in some sense. This would have enabled them to pass traffic, which would otherwise not have been allowed, along into the host containing the gateway. Forwarded packets would never have been passed to higher level protocol modules, or out of the operating system, so a user process could never have acquired them.

The problem with this approach was that it disallowed isolation of physical networks using purely software constructs. If an internetwork had a small set of networks which required strong security measures, such as hardware encryption, it would have been necessary to take such steps on *all* reachable networks. Since the gateways would have forwarded packets in an uncontrolled manner, a fault in one of these secure networks might have caused traffic to be redirected through another, less secure, network. For the example shown in Figure 9, traffic from Host B to Host A at label f should go through Host X. Using trusted gateways, it might have gone through Host Y. There would have been no way to isolate Network 2.

Another approach was to use static routes. Each forwarded packet could have used the IP source route option, which would have fully specified the path to be taken. An administrator would have been required to set up routes for particular label ranges. This scheme was quickly discarded due to its inflexibility. If a gateway along a fixed route had gone down, packets would still have been sent to it, and lost, even though an alternate path might have existed.

5.2.3. Chosen Solution

It soon became obvious that the existing routing protocol could also be used to propagate the label range of each reachable network. The routing daemon will initially know the ranges of all directly attached networks and these ranges can easily be distributed to all daemons in an internet.

Each time a router passes along a route that it has learned, it increments the hop count metric, reflecting the extra network that must be traversed. The same idea can be used with label ranges. In this case a new metric is created, which Sun shall call the "route label range". This range is the most restrictive range of all networks along the path; that is, the intersection of each network's label range.

The routing protocol must save all routes to a given network which have different route label ranges. When a packet must be forwarded through a gateway, its label is compared to the ranges of all possible routes. In this way an appropriate gateway can be selected for the next leg.

Network	Label	Gateway
Network 1	a	
Network 2	a - f	
Network 3	a	Host X, Host Y
Network 3	a - f	Host X

Figure 10: The Routing Table for Host B in Figure 9.

This scheme is very flexible. Each gateway is free to make new routing decisions, reflecting local conditions near that gateway. Therefore, a poor choice of initial gateway need not mean that the packet will be discarded.

The new routing protocol will be backward compatible with existing Sun routers, so that coexistence with Foreign Hosts is possible. When routing information is received from such a host the single-label label of that host can be attached to all paths through it, reflecting the restriction placed on the use of Foreign Hosts as gateways.

6. Conformance to Proposed Security Additions to IP

As mentioned above, Sun is using an IP Option to include security label information in the IP packet header. There is a draft underway for revised Internet Protocol Security Options; however, these options are not appropriate for Sun's use. Both of the standard security option specifications ([DoD83] and [RFC88]) use part of the option to indicate an accrediting authority and use the rest of the option as defined by this specified authority. Since the Sun label option would need to be usable by any or all of the authorities, it is not appropriate to chose a particular authority's configuration for the label.

Sun plans to provide translation functions which will allow mapping between the internal label and the security information for each authority. Any mapping which is specified in an RFP can be supported. Note that the label option is only used internally in the Secure SunOS system. The translation would only need to occur for communication with external systems.

7. Summary

Sun Microsystems is committed to excellence and standards. In this particular project, the standards involved are POSIX, the ARPAnet low-level protocols, and most importantly the Department of Defense Trusted Computer System Evaluation Criteria [DoD85]. In many cases at the start of the project, these standards seemed at odds with each other. In particular, TCP and IP were not designed with security in mind. Thus, backwards compatibility and emerging technology became serious design problems. With considerable effort and help from a number of areas within Sun, the Secure SunOS system has come up with a unique solution which fits into existing environments.

8. References

[DoD83]
Internet Protocol, MIL-STD 1777, Department of Defense, Washington, D.C., 12 August 1983.

[DoD85]
Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, Department of Defense, Washington, D.C., December 1985.

[RFC88]
St. Johns, M. *Draft Revised IP Security Options*, RFC 1038, January 1988.

[Sun87]
Addison, Katherine, et al. *Computer Security at Sun Microsystems, Inc.* Proceedings of the 10th National Computer Security Conference. Sept. 1987.

SECURITY ON AN ETHERNET

B.J. Herbison
Secure Systems
Digital Equipment Corporation
85 Swanson Road
Boxborough, MA 01719-1326
Internet: Herbison@Ultra.DEC.COM
Telephone: 1-508-264-5052

Abstract

Local area networks (LANs) are being widely used for a large number of applications. However, LANs are vulnerable to several security threats including wiretapping, masquerading, modification of data, and denial of service.

This paper describes Ethernet Enhanced-Security System, a system that can help provide security for an Ethernet or extended Ethernet. The Ethernet is secured at the Data Link Layer and the system is transparent to network software operating at a higher layer. The system consists of Digital Ethernet Secure Network Controllers and VAX Key Distribution Center software. A DESNC controller is an encryption device that provides node authentication and privacy and integrity of Ethernet frames. The KDC software manages the DESNC controllers on an Ethernet or extended Ethernet and enforce a LAN access control policy. If the access control policy allows, nodes protected by controllers can communicate with nodes not protected by controllers.

LAN Security Threats

Ethernet[1] and other Local Area Network (LAN) technologies provide the means to interconnect computer systems conveniently. LANs are used in many environments, but there are many situations where they cannot be used because of the requirement for a higher level of security than that provided by any commercially available LAN. Some LAN security threats are:

Wiretapping: The most obvious LAN security problem is wiretapping. This security problem is most serious for broadcast LANs where every node connected to the LAN is able to read all of the data that is transmitted on the LAN. It is easy to attach a LAN traffic monitor to any broadcast LAN and read all traffic on the LAN. In addition, some LAN architectures define a mode of operation, typically referred to as 'promiscuous mode', which allows a node to receive all data frames transmitted on the LAN regardless of the destination address of the frame. LAN adapters that implement this feature make eavesdropping easy to accomplish and difficult to detect.

The wiretap threat can be addressed either by physical security for the LAN or through encryption. However, physical security requires close monitoring of the LAN components, including all cables inside and between buildings, and cannot protect against unauthorized monitoring of the LAN by nodes authorized to use the LAN. For Ethernet and other broadcast LAN technologies, standard Data Link Layer encryption techniques cannot be used to prevent wiretapping because a large number of nodes all communicate over a common medium.

Masquerading: LANs are also vulnerable to masquerading attacks. It is easy for an unauthorized node connected to a LAN, or for a node that is authorized to use a LAN but which is untrusted or compromised, to masquerade as another node. Many LAN adapters allow the source address of LAN frames to be selected or changed by the node, making it easy for a node to perform a masquerading attack.

Protection against masquerading attacks requires the authentication of frames transmitted on the LAN. Physical security on a LAN would only protect against masquerading by intruders, not by nodes that are allowed some access to the LAN.

Modification: Another possible attack on a LAN is the modification of data. Nodes may modify frames sent on the LAN and transmit the modified versions. This attack can be used to compromise communication between trusted nodes, even when some level of authentication is used.

Protection against modification attacks requires integrity checks on frames transmitted on the LAN. These checks should be cryptographic functions of the frame, or cryptographically protected. As with masquerading, physical security would only protect against against attacks made by nodes not authorized to use the LAN.

Denial of Service: The operation of a LAN can be prevented or slowed with denial of service attacks. Denial of service includes such attacks as physical damage to LAN components (e.g., cutting the cable), disrupting communication by not using the correct LAN protocol (e.g., sending the wrong signals, or sending signals at the wrong time), or overloading the LAN by flooding it with traffic.

Protection against denial of service attacks requires physical security for the LAN components to prevent physical damage. It is also necessary to use LAN adapters that are trusted to follow the correct LAN standard, and to monitor the LAN for attempts to disrupt communication by flooding the LAN with traffic.

LAN Security Strategies

While all LAN technologies have these vulnerabilities, and especially all broadcast LANs, Digital's LAN strategy is centered around Ethernet. For this reason, Digital has developed the Ethernet Enhanced-Security System consisting of Digital Ethernet Secure Network Controller hardware and VAX Key Distribution Center software.

There are three security strategies that could be used to protect a LAN:

The following are trademarks of Digital Equipment Corporation: DESNC, VAX KDC, DEC, DECnet, ThinWire, VAX, and VMS.

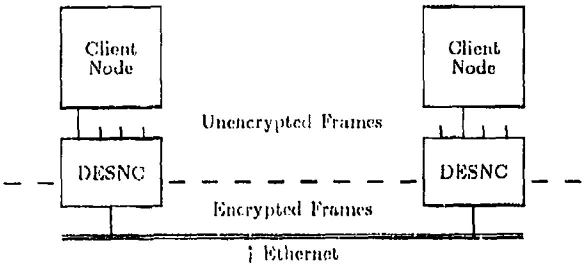


Figure 1: Simple Network Diagram

Physical Security: Maintaining tight controls on access to the LAN communication medium does prevent access by unauthorized nodes, but the security becomes hard to maintain in extended LAN environments and it does nothing to prevent wiretapping, masquerading, modification, or denial of service attacks by authorized users of the LAN.

Separate LANs: Partitioning a LAN into physically separate LANs for different user groups does prevent attacks from nodes assigned to other security levels, but it does not prevent attacks from nodes with a security level and it places limits on communication that are not practical in many environments.

Encryption: A well-designed scheme for distributing keys and encrypting communication on a LAN can prevent wiretapping, masquerading, and modification attacks, and allow a flexible LAN access control policy to be implemented. Unless all equipment on the LAN is designed to encrypt and decrypt frames, additional hardware is required to implement the encryption. It is also necessary to have a facility for generating and distributing the encryption keys necessary for encrypting the frames.

Approach

Of the strategies mentioned above, encryption is the only alternative that authenticates nodes connected to the LAN and allows a flexible access control policy to be implemented. For this reason, encryption is the only reasonable approach to addressing most LAN security threats.

The DESNC controller is a hardware device that sits between a node and the Ethernet and encrypts frames transmitted by the node and decrypts frames received by the node (see figure 1). The frames are encrypted using the DES encryption algorithm. A manipulation detection code (MDC) is added to the frames when they are encrypted and verified when they are decrypted. The controllers also verify the source address on each frame transmitted by the node. Because of the frame processing required, controllers are store-and-forward devices.

Controllers have one port that is connected to the LAN and four ports that are connected to nodes. The four node ports on a controller are separate security domains. A node attached to one node port cannot read frames transmitted to a node on another node port. Multiple nodes can be connected to one node port. The four ports can support up to 20 nodes in any combination.

The interface used by the ports is standard Ethernet (or IEEE 802) format frames. DESNC controllers are designed to interoperate with any equipment that uses the Ethernet or IEEE 802 standards. The controllers operate at the Data Link Layer, and are transparent to any network software operating at a higher layer. For example, DECnet or TCP/IP software does not require modification to be used in a LAN protected by DESNC controllers.

The controllers are managed by VAX KDC software running on specially designated KDC nodes on the Ethernet. These KDC nodes must be running the VAX/VMS operating system. KDC nodes do not perform all parts of the KDC operations, they require the support of an attached DESNC controller. These KDC controllers are physically the same as other DESNC controllers, but they store more information and perform additional functions.

Under normal operating conditions, a node protected by a controller can only communicate with other nodes when the communication is approved by a KDC. When a node attempts to communicate, the controller that receives the first frame requests an 'association' from a KDC node for the pair of nodes that want to communicate. Associations always allow communication in two directions.

Up to five KDC nodes are allowed on an Ethernet or extended Ethernet. Multiple KDCs increase the reliability and availability of the LAN because they increase the probability that a controller will be able to communicate with a KDC when the controller wants to set-up an association.

If an extended Ethernet is used, spreading the KDCs over the extended LAN also improves availability. If a problem causes the extended LAN to be segmented, it is more likely that each controller will be able to communicate with some KDC if the KDC nodes are distributed across the extended LAN.

Protection Against LAN Security Threats

The Ethernet Enhanced-Security System protects against masquerading, wiretapping, and modification attacks, and, to a limited extent, some denial of service attacks.

Masquerading: Nodes protected by controllers are not allowed to masquerade as other nodes because the controllers check the source address of transmitted frames. Because communication must be approved by a KDC, controllers will detect masquerading attempts by nodes not protected by a controller.

If multiple nodes are attached to one node port on a controller, the controller will not be able to distinguish between the nodes nor be able to detect attempts by one of the nodes to masquerade as another. For this reason it is recommended that only one node be attached to each node port, but mutually trusted nodes can be chained on one port.

Wiretapping: Controllers ensure the privacy of communication between any two nodes attached to DESNC controllers by encrypting the Ethernet frames sent between the nodes.

No attempt is made to hide the length of encrypted Ethernet frames. Because the original header of the Ethernet frame is useful for LAN management, it is sent in unencrypted form on the Ethernet. However, the header is protected against modification by placing another copy of the header in the encrypted portion of the message.

Modification: The manipulation detection code that controllers add to Ethernet frames when the frames are encrypted allows the controllers to detect attempts to modify an Ethernet frame. In addition to the unencrypted copy of the header of each Ethernet frame, the header of each Ethernet frame is also included in the encrypted portion of the frame exchanged between controllers, so the header is protected by the manipulation detection code.

The MDC is a 16-bit CRC, but the MDC value is transmitted in the encrypted portion of the Ethernet frames. Because the information being protected and the MDC value are both encrypted with DES, deterministic changes cannot be made to the encrypted data or the MDC. This means that there is a low probability that an intruder will be able to modify an encrypted frame without the modification being detected when the frame is decrypted and the MDC field checked (i.e., a huge number of attempts will be required before one frame is successfully modified).

Denial of Service: DESNC controllers are not intended to provide protection against denial of service attacks. However, nodes protected by controllers are isolated from the LAN and so the effect of any attempts by these nodes to jam the Ethernet or disrupt the Ethernet by not obeying the Ethernet standard will not be allowed to affect nodes other than the nodes protected by that particular controller.

Access Control Policy

The network security manager for an Ethernet secured by controllers can determine which pairs of nodes are allowed to communicate. The ability to communicate is determined by an access class range for the nodes and by the ability of a node to communicate unencrypted.

The network assigns an access class range to each node on the LAN. The access class ranges are from a Bell and LaPadula[2]/ Biba[3] secrecy and integrity lattice, with 256 secrecy and integrity levels and 64 secrecy and integrity categories.

Trusted nodes may be assigned a range of access classes, but untrusted nodes are assigned a single access class. Two nodes are only allowed to communicate if they operate at the same access class, or if the access class ranges for the nodes overlap in at least one common access class.

The access class ranges alone determine if two nodes protected by controllers are allowed to communicate. However, if only one node is protected by a DESNC controller then an additional factor is considered: The network security manager can specify which nodes are allowed to communicate with nodes that are not protected by controllers and which nodes can only communicate when the transmitted frames will be encrypted.

Bypass Operation

For normal operation, controllers must be able to communicate with at least one KDC. If a controller cannot communicate with any KDCs, it will not be able to determine which pairs of nodes should be allowed to communicate.

To handle situations where the KDCs on an Ethernet are not operational, a DESNC controller can be placed in Bypass state. This is done by pressing a Bypass pushbutton on the front panel of each controller.

When the Bypass pushbutton is pressed, a controller will enter Bypass state if it cannot communicate with any KDCs. Nodes are only allowed to send and receive frames when the controller is in Bypass state if the controller has previously been informed by a KDC that the node is allowed to communicate in Bypass state. When a controller is in Bypass state it will not encrypt any Ethernet frames. Even in Bypass state, the controllers check the source address of frames transmitted by a node and only send a node the Ethernet frames that are addressed to the node.

The network security manager may choose which nodes are allowed to communicate when the controller that protects them is in Bypass state. The network security manager should determine, for each node, if it is more important for the node to be able to communicate whenever possible (allow communication when the controller is in Bypass), or to be secure whenever possible (disallow communication when the controller is in Bypass).

Multicast Frames

Because frames sent to a multicast (or broadcast) destination address are intended to be read by many, if not all, of the nodes on a LAN, it is not possible to encrypt multicast frames in the same manner as other Ethernet traffic. DESNC controllers do not encrypt multicast frames.

Controllers pass multicast frames without any modifications, but they allow, at the request of a KDC, a node to be prevented from transmitting multicast frames. In some network environments, it is necessary to allow all nodes to transmit multicast frames because many network protocols depend on multicast frames for correct operation.

Because DESNC controllers are designed to work on LANs where only some of the traffic is encrypted, it would not be possible to simply encrypt multicast messages with a common Ethernet-wide key—it would be necessary to send the messages in unencrypted form as well as encrypted form to reach the nodes that were not connected to DESNC controllers.

Trust

With any security system, it is important to know which components must be trusted, and the degree of trust required. Ethernet Security-Enhanced System was designed to limit the degree that an individual DESNC controller needs to be trusted.

The compromise of a DESNC controller may compromise the nodes protected by the controller, but will not compromise any other controllers or nodes on the LAN. This means that a controller must be protected as well as any of the nodes protected by the controller.

If multiple nodes are connected to the same node port of a controller, the nodes can masquerade as each other. This means that the multiple nodes must be mutually trusting. If this is level of trust is not appropriate, a site can use DESNC controllers with only one node attached to each of the four node ports.

If a KDC node or the controller that supports a KDC node is compromised, the security of the LAN can be compromised. This means that KDC nodes and KDC controllers must be protected as well as any node on the LAN. While KDC nodes can be used for multiple purposes, the security of the network is improved if the KDC nodes are limited to network management functions and access to the nodes is limited to trusted individuals.

Ethernet Enhanced-Security System Architecture

Encryption Keys

Messages exchanged among DESNC controllers and KDCs are encrypted using the Data Encryption Standard (DES) encryption algorithm[4,5]. The messages are encrypted using the Cipher Block Chaining mode of DES.

Several different types of DES encryption keys are used by controllers and the KDC software.

KDC Master Key: This key is used to encrypt keys that are stored on KDC nodes. This key is only known by the network security manager and the KDC controllers.

Key Generation Key: If keys are generated, this key is used as part of the process to generate the keys. This key is only known to the network security manager and the KDC controllers.

Initialization Key: Initialization keys are used to initialize a controller. These keys are used to distribute the master and service keys for a controller, and are then never used again. These keys are only known by the network security manager, the controller initialized with the key, and the KDC that initializes the controller.

Master Key and Service Key: These encryption keys are used to communicate between controllers and KDCs. A different set of keys is used for each controller. The key for a controller is only known by the controller and the KDCs, and they are only stored in encrypted form on KDC nodes. These keys are never handled by any person in unencrypted form.

Association Key: These keys are used to encrypt communication between nodes protected by controllers. A different association key is used for each pair of nodes that communicate. Association keys are distributed by KDCs when controllors request associations.

KDC controllers will generate encryption keys as needed by the KDC nodes, or the network security manager can have the KDC nodes acquire the keys that they need from a user-supplied source. A small amount of user programming is required to use a user-supplied key source, as well as a large supply of keys.

Initializing Controllers

Before a controller can operate, it must be initialized. To initialize a controller, the following steps are required:

- The network security manager enters information about the controller and the nodes protected by the controller into a KDC node. This information includes the addresses of the controller and the nodes and the access control policy for the nodes.
- On the request of the network security manager, the KDC node selects an initialization key for the controller. The key is either generated or taken from a supplied key source.
- The network security manager enters the initialization key in the controller.
- The controller communicates with the KDC and the master encryption key for the controller is distributed. This message is encrypted with the initialization key that was entered into the controller. The initialization key is not used after this step.
- The controller communicates with the KDC and the KDC distributes the information that the controller needs to operate. This information includes:
  - The duration of associations between nodes.
  - The name of the firmware that the controller should be using, and a cryptographic checksum for the firmware image.
  - The addresses of the key distribution centers on the LAN.
  - The addresses of the nodes supported by the controller.
  - Information about the supported nodes. This includes such information as whether each node is allowed to communicate when the controller is in Bypass state.
  - A list of the events that the controller should audit.

After these steps, the controller is operational. The controller can now communicate with any KDC on the LAN. Once a controller is initialized it is not necessary to manually enter any additional information. If the distributed information needs to be changed, the changes can be done remotely from any KDC, DESNC controllers retain the distributed information during power-off or power interruptions for a minimum of 72 hours.

Operational controllers will request associations from KDC node as necessary, and encrypt and decrypt Ethernet frames sent by nodes.

Downline Loading Controllers

The operational firmware images used by DESNC controllers is downline loaded over the Ethernet using the same mechanism used by other DEC products. This allows the controllers to be downline loaded by the same downline load servers that load other products on the Ethernet. The integrity of the images (and the security of the LAN) is protected in the following manner:

1. When a new firmware image is installed on the network, a KDC generates an encryption key and a cryptographic checksum for the image. The KDC generates a different key and checksum for each controller on the Ethernet.
2. When a controller is initialized, the KDC distributes the name of the firmware image and the appropriate checksum information to each controller. If the image changes after a controller is initialized, any KDC may distribute the new image name and checksum information to the controller.
3. When a controller needs to be downline loaded, it requests the appropriate image. After the image is received from a downline load server, the controller calculates the checksum for the image and compares the value against the stored value. If the received image does not have the correct checksum then that image is ignored and a new image is requested.
4. The DESNC controller stores the checksum in memory that is preserved over power failures. Because of this is is not necessary to distribute checksum information to all controllers after a power failure.

Associations

When two nodes try to communicate by exchanging Ethernet frames over the LAN, controllers will not allow the communication unless the 'association' is allowed by a KDC. These associations are granted upon demand by KDCs.

There are three different types of associations:

- Associations between two nodes protected by different controllers. Frames sent under these associations are encrypted while they are on the Ethernet.
- Associations between two nodes protected by the same controller. Frames sent under these associations are never sent on the Ethernet so there is no need for them to be encrypted. The controller only sends the frame to the node port where the destination node is attached.
- Associations between a node protected by controllers and a node not protected by a controller. Frames sent under these associations are not encrypted (because there is no second controller to decrypt the frame), but communication is not allowed unless approved by a KDC.

Examples of these three types of associations are shown in figure 2. When two nodes communicate directly (without any intervening DESNC controllers), controllers and KDCs are not involved in the communication.

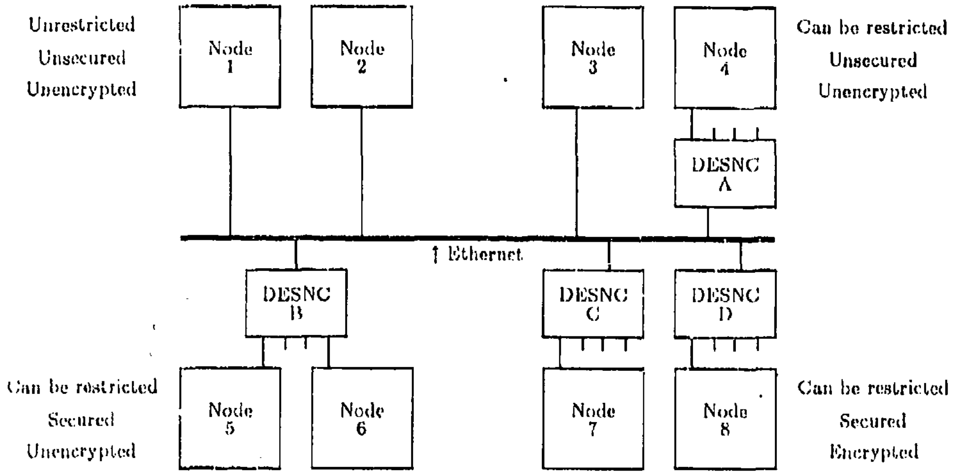


Figure 2: Types of Associations

Association Set-Up: The protocol exchange used between DESNC controllers and KDCs to set-up associations is similar to the protocol used in Voydock and Kent[6]. Here is an example of how an association would be established between two nodes that are both connected to DESNC controllers.

Consider the LAN shown in figure 2. Setting up an association between node 7 and node 8 (with node 4 acting as the KDC node) involves the following steps:

1. Node 7 sends an Ethernet frame to node 8.
2. Controller C receives the Ethernet frame and verifies the source address of the frame.
3. Controller C requests an association from the KDC (node 4).
4. The KDC checks the access control policy and determines that nodes 7 and 8 are allowed to communicate and sends an Association Open message to controller C. The Association Open message is encrypted with the master key of controller C. The message contains an association key, either generated by the KDC controller or taken from supplied keys.
5. Controller C sends an Association Forward message to controller D. The Association Forward message is encrypted with the master key of controller D and was generated by the KDC and included in the Association Open message sent to controller C.
6. Controllers C and D communicate and determine that they share a common association key.
7. Controller C encrypts the message sent in step 1 with the association key and sends the encrypted frame to controller D.
8. Controller D receives the encrypted frame, decrypts the frame, checks the manipulation detection code, and transmits the frame to node 8.

Once the association is established, no further interaction with the KDC is required and all communication between nodes 7 and 8 is encrypted with the association key until the association expires. At that point, another association is requested by the controllers. The duration of associations is determined by the network security manager. If an association is active and approaching expiration, the controller that originally requested the association (controller C in this example) will request another association before the first association expires.

Encrypted Frame Format

When Ethernet frames sent by nodes are encrypted by DESNC controllers, the frames are protected in several different ways:

- The frames are encrypted to prevent disclosure of the contents, and also to prevent predictable modification.
- Sequence numbers are included in the encrypted portion of the frame to detect replay and reflection (sending frames back to the sender with the source and destination addresses swapped).
- A Manipulation Detection Code (MDC) is appended to the frame before it is encrypted. This field is checked when the encrypted frame is decrypted to determine if the frame was modified.

Several other changes are made to the format of the frames when they are encrypted:

- A new IEEE 802 header is added to the start of the frame. The protocol identifier in the frame header allows DESNC controllers to determine which of the frames it receives are encrypted.
- A message type is placed after the header to distinguish encrypted frames sent by nodes from other encrypted control messages. A copy of the message type is placed in the encrypted portion of the frame to allow changes to the message type to be detected.
- An encryption identifier is also added to allow the controller to determine which encryption key should be used to decrypt the frame. (This is just a performance optimization, it would be possible for the controller to determine the appropriate key from the frame addresses and the message type.)
- Two copies of the original frame header (other than the Ethernet addresses) are placed in the frame. One copy is unencrypted and can be used by LAN management tools, the other copy is encrypted so that any attempts to modify the header will be detected.

Destination Address	
Source Address	
IEEE 802 Header	
Message Type	
Encryption Identifier	
Original Header	
Sequence Number	*
Message Type Copy	*
Original Header	*
Original Data Field	*
Padding	*
MDC	*
Ethernet FCS	*

* marks the encrypted fields.

Figure 3: Encrypted Frame Format

- The encrypted portion of the frame is padded to a multiple of the DES block size to allow the frame to be encrypted using CBC mode. The padding is removed when the frame is decrypted.

- The correct Ethernet Frame Check Sequence (FCS) is appended to the newly created frame. The original FCS will be used when the frame is transmitted to the destination node.

The format of an encrypted frame is shown in figure 3. Everything from the sequence number to the MDC is encrypted.

Fragmentation: Because additional fields are added to frames by DESNC controllers, frames that are close to the maximum length will be extended beyond the maximum allowed length for Ethernet frames. To handle this situation, DESNC controllers 'fragment' long Ethernet frames and transmit them in two encrypted Ethernet frames.

This fragmentation and reassembly affects the performance of communication, but the fragmentation is handled by controllers and transparent to the nodes involved. Ethernet performance can be improved for a node if the network software on the node is modified to always send frames that are shorter than 1480 bytes.

Auditing

Controllers and KDCs generate security related audit events. Controllers and KDCs always count significant events as they occur, and detailed information about some events is recorded by the KDC server. The stored events can be displayed by the network security manager.

The level of auditing can be selected by the network security manager by choosing which controllers and KDCs record which classes of events. There are 18 auditable events including:

- Controller status messages and state changes.

- Association requests.

- Rejected associations requests.

- Invalid frame addresses, sequence numbers, or MDCs.

The network security manager can select which events are included in audit reports, and it is possible to have the KDC server send selected classes of events to a physical alarm device (e.g., a terminal or a printer).

Software Design

The VAX KDC software is a layered product that runs under the VAX/VMS operating system. There are two parts to the VAX KDC software:

- A KDC Server that runs continuously. This program responds to association requests for the controllers on the LAN and records audit events generated by controllers and the KDC software.

- A KDC User Interface that is used by the network security manager to enter and display configuration information, manage controllers, and review controller status and audit information.

The following data files are used by the KDC software:

LAN Configuration File: This file contains information about the configuration of controllers, KDCs, and other nodes on the LAN. All KDC nodes must have the same information in their configuration files for correct operation.

Controller Status File: This file contains information about the status of controllers and their communication with this KDC. A separate file is maintained by each KDC.

Audit History File: This file contains all of the audit events recorded by this KDC. The level of auditing can be selected separately for each controller or KDC.

KDC DESNC Controller

The software works together with the DESNC controller attached to the KDC node to function as a KDC. Using this KDC DESNC controller to help the KDC node has several advantages:

- Messages sent by the KDC node are encrypted by the KDC controller. This is much faster than using software encryption.

- Except for the key that is displayed when a controller is initialized, there are no 'red' keys stored on the KDC node. All encryption keys are encrypted when stored on the KDC node.

  When a key is used, it is decrypted by the KDC controller using a key that is only known to the KDC controller. This prevents someone with read-only access to the information on the KDC node (e.g., access to backup tapes) from compromising the security of the network.

- The KDC controller generates encryption keys for the KDC as they are needed. This also prevents unencrypted keys from being present on the KDC node.

Hardware Design

A DESNC controller can be considered to be a combination of an encrypting Ethernet bridge and a ThinWire Ethernet concentrator. The concentrator on the node side of a DESNC controller is design, unlike a normal concentrator, so that a frame transmitted on one node port is not automatically transmitted on the other ports. This allows the controller to enforce the LAN access control policy between nodes on different ports.

Controllers are compatible with the Ethernet and IEEE 802.3 standards to allow interoperability with any products that conform to those standards.

Controllers contains a Motorola 68000 CPU chip and ROM containing the code that is executed when a controller is first used. There are also various memories to store downline loaded code, information about associations code in ROM, RAM for code, RAM for association storage, and RAM for packet storage.

Some of the memory in a controller is preserved over power failures of less than 72 hours through the use of capacitors. This memory is used to preserve enough information so that the controller does not have to be manually initialized after a power failure. The preserved information includes the master key for the controller, sequence numbers used between the controller and the KDCs on the LAN, and the cryptographic checksum for the downline loaded image. If a DESNC controller is opened, power is removed from this memory to reduce the chance that the information will be used to compromise the nodes attached to the controller.

The front panel of a DESNC controller contains lights that indicate the state of the controller, a keypad that is used to initialize the controller, and a keyswitch that is used to activate the keypad. There is also a Bypass pushbutton that can always be pressed to request that the controller enter Bypass state.

DESNC controllers execute complete self-test code when they start operation, and several tests are run regularly during operation of a controller to detect any failures before the security of the nodes connected to the controller.

Relationship to the Trusted Network Interpretation

Although the Ethernet Enhanced-Security System was not designed specifically to satisfy cryptographic requirements for the protection of classified information, the architecture and security policy of the system permits an evaluation in accordance with the Trusted Network Interpretation[7] (TNI) in a variety of different ways. (Even if the system is evaluated, it would only be useful in environments where it is appropriate to use DES encryption.)

Network Evaluation: A LAN consisting of DESNC controllers, a set of KDCs, and several single-user workstations can be viewed as a single trusted system, where the subjects are the users of the workstations and the only supported objects are Ethernet frames. Such a network system satisfies many of the TNI requirements at the B1 class.

In support of mandatory security, users (and their nodes) are assigned explicit secrecy and integrity levels. Ethernet frames do not contain explicit secrecy and integrity labels: the labels are implicit in the key used to encrypt the frame. Because keys are selectively assigned on a node-pair basis the fact that the users of two workstations can communicate means that they are authorized for a common secre - and integrity level. The sensitivity level of a given frame can thus be determined by examining the source and destination addresses in the frame in conjunction with information stored in the KDC database. [When multilevel nodes communicate, it may not be possible to determine the level of frames exchanged, depending on the access control policy used by the LAN. In these cases, it is only possible to determine a possible range of levels for each Ethernet frames.]

Discretionary access control is supported in the sense that DESNC controllers will prevent a frame from being received by any workstation other than the destination node requested by the transmitter.

Specific Security Services: In addition to receiving a rating, the LAN described above contains features that are useful in obtaining positive evaluations for various specific security services including Authentication, Communication Field Integrity, Data Confidentiality, and Traffic Flow Confidentiality.

Component Evaluation: An individual DESNC controller can be viewed as a network component of type 'MA', providing functionality for MAC and audit. It should satisfy the minimum B1 class requirements for MD components. If the component is restricted to connecting one single-user workstation to each port, it would also provide functionality for DAC and identification and authentication, and therefore satisfy the minimum B1 class requirements for MIAD components.

As an alternative, DESNC controllers and a set of KDCs can be viewed as a network component to provide the same functionality.

References

[1] *The Ethernet, A Local Area Network, Data Link Layer and Physical Layer*, Version 2.0, (Digital, Intel, and Xerox), November 1982.

[2] D.E. Bell and L.J. LaPadula, *Secure Compu-er Systems: Unified Exposition and Multics Interpretation*, ESD-TR-75-306, MITRE Corporation, March 1976.

[3] K.J. Biba, *Integrity Considerations for Secure Computer Systems*, ESD-TR-76-372, MITRE TR-3153, MITRE Corporation, April 1977.

[4] *Data Encryption Standard*, Federal Information Processing Standards Publication 46 (FIPS PUB 46), National Bureau of Standards, 15 January 1977.

[5] *DES Modes of Operations*, Federal Information Processing Standards Publication 81 (FIPS PUB 81), National Bureau of Standards, 2 December 1980.

[6] V.L. Voydock and S.T. Kent, *Security in Higher Level Protocols: Approaches, Alternatives and Recommendations*, Report No. ICST/HLNP-81-19, National Bureau of Standards, September 1981.

[7] *Trusted Network Interpretation*, NCSC-TG-005 Version-1, National Computer Security Center, 31 July 1987.

COVERT CHANNELS IN TRUSTED LANS

Albert L. Donaldson
ESCOM Corporation
12206 Waples Mill Rd.
Oakton, VA 22124

Donaldson@dockmaster.arpa

John McHugh
Computational Logic, Inc.
3020 Pickett Rd, Suite 101
Durham, NC 27705

mchugh@cli.com

Karl A. Nyberg
Grebyn Corporation
P. O. Box 1144
Vienna, VA 22180

karl@grebyn.com

ABSTRACT

Trusted distributed systems differ from trusted monolithic systems in that the medium linking the components of the distributed system may be subject to wiretapping threats. This has led to a misconception that the primary concern in covert channel analyses of distributed systems should be the misuse of protocol fields as a means to covertly signal information to wiretappers having access to the communications medium. Depending upon the network's environment this may not be a problem at all. However, all networks do have internal host-to-host channels that must be analyzed in order to satisfy the assurance requirements of distributed systems at the B2 level and higher. This paper describes a layered approach for analysis of these internal channels that is consistent with the way in which communications networks are actually designed and built. Additionally, the use of embedded, network-based access controls is proposed as a means to prevent certain host-to-host channels.

1. Introduction

This paper addresses two distinct ways in which information contained within a secure distributed system or network can be covertly compromised. The first, the threat of wiretapping or compromise of the network medium has been discussed in the literature although it does not seem to fit the traditional definitions of the covert channels. It is pointed out that networks, especially Local Area Networks (LANs) are not necessarily subject to this threat, but if they are, the threat may require the redefinition of the term covert channel as well as changes in the ways that such systems are modeled.

The second mechanism is the traditional covert channel in which a system resource not normally used as an object to contain information is used to signal information between subjects or users of the system. Both the desire to evaluate components for use in building network systems and the complexity of such systems (and even the components) makes it difficult to analyze networks effectively with traditional covert channel techniques. Further, the interconnection of multiple trusted computer systems raises the possibility that a flaw in one system can be exploited by users in other systems leading to the possibility of a covert channel that spans the network.

A layered method of analysis is proposed that closely follows the ways in which such systems are designed and constructed in practice. If such channels must be considered, techniques are available to reduce their bandwidth. This section concludes with the discussion of one such method based on the introduction of host to host access controls.

2. Background

The National Computer Security Center has developed the *Trusted Network Interpretation* (TNI) [1], to be used in the evaluation of trusted network systems and their components. The TNI reflects two different, and sometimes conflicting, perspectives on the overall security problem:

(1) an extrapolation of the trusted system principles from the *Trusted Computer System Evaluation Criteria* (TCSEC) [2] into a distributed environment, and

(2) a communications security perspective that is concerned with protecting the integrity and secrecy of communications within potentially hostile environments.

The TNI addresses both areas, with Part I ratings defining the security policy, authentication, assurance and documentation requirements for loosely-coupled distributed computing systems, and Part II ratings addressing the security of the interconnecting communications paths.

The differing emphases of these perspectives has lead to confusion in the area of covert channel analysis. The TNI continues the requirement to address covert channels within the individual computing systems and observes that there are additional instances of covert channels associated with communications between components: *i.e.*, the exploitation of network protocol information.

The evaluation of trusted systems must provide for the analysis of both *overt* and *covert* channels. Within trusted computer systems, overt channels result from the use of the system's protected data objects to transfer information directly from one subject to another. Analysis of the system must lead to the conclusion that all overt channels conform to the system's security policy.

On the other hand, *covert channels* use entities not normally viewed as data objects to transfer information from one subject to another in violation of the system's security policy. *Storage channels* result from an exploitation of a shared storage resource, while *timing channels* result from the modulation of the system's response time.

The TCSEC and TNI provide the following definitions of such channels:

Covert Channel
A communications channel that allows a process to transfer information that violates the system's security policy. A covert channel typically communicates by exploiting a mechanism not intended to be used for communication.

Covert Storage Channel
A covert channel that involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process. Covert storage channels typically involve a finite resource (*e.g.*, sectors on a disk, device status flags, *etc.*) that is shared by two subjects at different security levels.

Covert Timing Channel
A covert channel in which one process signals information to another by modulating its own use of system resources (*e.g.*, CPU time) in such a way that this manipulation affects the real response time observed by the second process.

Historically, most methods for dealing with covert channels within computer systems have been *ad hoc*. [3] describes the approach used for performing a covert channel analysis during the Honeywell Multics evaluation. Mechanical covert channel analysis tools have been developed for systems characterized by formal top level specifications. These tools, typified by the SRI MLS Flow Tool [4], are based on the assignment of security levels to each TCB resource attribute and the generation of formulas which, if proven to be true, ensure that all information transfers within the specification conform to the system's security policy. The Shared Resource Matrix (SRM) methodology proposed by [5] is an intermediate approach (between the *ad hoc* methods and information flow tools) that can be used at a variety of different levels of abstraction.

The development of covert channel analysis methods, such as those mentioned above, results from the TCSEC requirement to perform covert channel analyses beginning at the B2 assurance level. Since there are only two local area network components known to be under evaluation by the NCSC at the time this paper was written, there is not a significant amount of literature available on the subject of covert channels in LANs.

3. The Wiretap Threat

The TNI seems to assume that the primary covert channel threat to networks results from attacks on the network communications medium by wiretappers who are not "subjects" of the network. Both of the references cited by the TNI in this area, mention the existence of covert channels from an untrusted subject to an external wiretapper. [6] addresses the inability of end-to-end encryption hardware to protect against malicious use of address, length, and timing information by untrusted host software. [7] defines the same three mechanisms within a LAN environment and describes the results of an experiment to measure the bandwidth of the addressing channel.

As described in the following sections, we believe this emphasis on covert channels involving wiretappers is somewhat misleading. Within LAN environments, at least, wiretap threats can be addressed in the same way that they are addressed in any other trusted facility: by physical, procedural, and administrative security mechanisms.

The mechanisms identified by [6] and [7] involve the modulation of protocol fields or other externally visible aspects of the communications packet. Girling identifies the following three methods of exploiting conventional LAN interface devices to covertly signal information to a wiretapper:

LAN Address — A covert storage channel is possible when a high-level hos process can address packets to multiple destinations and a wiretapper can observe the sequence of packets.

Packet Length — A covert storage channel is possible when a high-level host process can determine the length of outgoing packets and the wiretapper can observe the lengths of these packets.

Inter-Packet Delay — A covert timing channel is possible when a high-level host can modulate the delay between outgoing packets and a wiretapper can observe and measure these delays.

3.1. Bandwidths

Both [6] and [7] point out that the bandwidths of such channels may often be in excess of 100 bits per second. It appears that these estimates understate the potential bandwidths that could be achieved using current LAN devices.

(1) The IEEE 802.3 specification defines a six-byte destination address field and packet lengths from 64 to 1518 octets. Assuming all addresses can be generated without detection, the width of the address channel bandwidth is potentially 48 bits/packet.

(2) The increased performance of readily available LAN hardware means that information can be compromised at a proportionally higher rate. Ethernet boards are available with typical throughputs of 500 Kbps, and should be able to generate packets continuously at a rate of 50 packets per second.

From such assumptions, the bandwidth of the address channel in an Ethernet-like LAN could be on the order of 2400 bits/second. As higher throughputs become commonplace, the potential bandwidth of these channels will further increase.

That such signaling mechanisms exist is not in dispute. Whether they need to be considered in every network system is open to question. If the system or component being examined is subject to a wiretap threat, there may be implications that require modification of the system security policy and the definition of a covert channel.

3.2. Is the Threat Universal?

Are some networks exempt from a wiretap threat? We believe that this is certainly true for some LANs, but it may not be the case for geographically distributed network systems. LANs operating within a physically controlled environment or routed through protected wireways should be relatively immune to wiretap attacks.

In order to clarify this issue, consider a series of four different systems[1], as follows:

(1) A trusted monolithic computing system, as addressed by the TCSEC.

(2) A trusted tightly-coupled multi-processor, multi-programmed computing system, as described in Appendix B.4.3 of the TNI.

(3) A trusted, loosely-coupled distributed computing system, with individual host computers operating at potentially different security levels. The host computers are interconnected by a conventional IEEE 802.3 LAN, and the computers and medium are maintained within the same protected facility.

(4) A trusted, loosely-coupled distributed computing system, with individual host computers operating at potentially different levels, but with each host computer separately protected.

For each architecture, consider the implications of allowing an anonymous wiretapper, with arbitrary equipment, access to the backplane of each of the four systems. Does this access constitute a potential for a compromise of information, a violation of the system's security policy? Using the broad definition of "policy", the answer would certainly have to be "yes". Clearly, each example provides the potential for unauthorized disclosure and modification of data.

The significant point is that in the first two cases the vendor and the NCSC assume that the system will be operated according to the assumptions in the vendor's *Trusted Facility Manual*. If the monolithic computer system can be considered a degenerate case of the general computer system, the corresponding intrusion would be tantamount to the removal of the cover from the computer system, and allowing the wiretapper access to bus signals and other backplane activity.

If the operators of any trusted facility allow unlimited access to the internals of their system, then there is significant risk of data compromise at a very high bandwidth. Conversely, if it can be assumed that the facility and its equipment are properly protected, then the covert channel analysis can be limited to potential channels between authorized subjects operating under control of the TCB/NTCB.

3.3. Security Compliant Communications

The TNI references appear to make the assumption that networks are **necessarily** subject to wiretap threats, and ignore the traditional physical, procedural and administrative solutions to physical tampering used in trusted computer facilities, but Appendix B of the TNI describes three different methods for ensuring *security-compliant communications.*

(1) Documenting constraints in the Trusted Facility Manual, thereby deferring an assessment of compliance to accreditation.

(2) Providing suitable end-to-end communications security techniques.

(3) Administrative restriction of use of the channel.

If the assumption is made in the *Trusted Facility Manual* that the network is operated in a protected environment, there should be no need to consider covert channels to wiretappers. This explicit assumption is made implicitly in the evaluation of trusted computer sys:tems. For local area networks, in particular, it appears to be a reasonable assumption to make, considering cost and benefit tradeoffs.

3.4. Policy Implications

This still leaves open the question of whether or not information flows from a subject (user) of a system to a wiretapper constitute covert channels in the usual sense of the term. The answer depends upon the definition of security policy, since a covert channel is defined as "a communications channel that allows a process to transfer information in a manner that violates the system's *security policy*". The TCSEC further defines security policy as "... the set of laws, rules and practices that regulate how an organization manages, protects, and distributes sensitive information". This appears to categorize theft of computer tape containing classified information as being a covert channel; however, that is clearly not what is intended for the covert channel analyses performed by system vendors.

Covert channel analyses focus on the use of entities not normally viewed as data objects to transfer information from one subject to another [subject] [5]. Subjects, in turn, are entities that operate within the control of the TCB (or NTCB) on behalf of human users. This appears to have at least one of the following implications:

[1] The TCSEC/TNI terminology is used here, so that the term "system" refers to a collection of computer and communications hardware, software, and firmware that performs all of the functions defined in Part I of the TNI. Specifically, a system is capable of identifying and mediating access at the human user level.

(1) The wiretapping threat is unrelated to the subject of covert channel analysis (and requires a physical, communications, or administrative security solution); or

(2) Wiretappers can be "subjects", and consequently must be addressed by the security policy model for the network system; or

(3) The manner in which covert channel analyses have been done for monolithic computer systems must be changed to include threats to the physical security of the system.

We dismiss the third alternativo as unduly overloading the notion of covert channels. For a system to be secure, a wide variety of potential threats must be countered. When the evaluation and accreditations are properly carried out, this will be done in such a way as to cover operational and environmental threats as well as architectural ones. There is no need to include all such threats under the umbrella of "covert channel analysis". The choice between the first two alternatives is less clear cut.

At the B2 level of evaluation, both the system's security policy model and the covert channel analysis are rather informal. The search for covert channels is usually based on the system's Descriptive Top Level Specification (DTLS) which may be somewhat imprecise. At the A1 level, covert channel analysis is conducted with respect to the system's Formal Security Policy Model (FSPM) and Formal Top Level Specification (FTLS) using "formal methods", usually with the aid of mechanical tools. In either case, both the security policy and the system specification must define the domain of the analysis. If the covert channel analysis is to include wiretap threats then the policy and specification must include wiretappers. We know of no cases to date in which this issue has been explicitly addressed in network security policy models or specifications.

These observations lead to a conclusion that while information compromise via wiretap channels can be performed using techniques similar to those used for covert channels, the mechanisms are not covert channels under the definition quoted above with the usual policy definitions and specification paradigms. In this case, the fault lies with the policies and specifications. If a system or component will be subject to a wiretap attack and it is desired to evaluate the threat as a part of a covert channel analysis, the policy for the system must clearly consider the existence of wiretappers and define the extent (if any) to which they are permitted access to information contained in the system. A system specification subject to covert channel analysis must also explicitly consider wiretappers as potential subjects and describe their accesses in relation to other system entities. These additions will clearly impact the usual security analysis of the system as well as its covert channel analysis.

4. Covert Channels Between Network Subjects

This section describes a method for identifying and resolving the remaining internal covert channels within a network system. Within a distributed system, internal channels tend to be between processes existing on different hosts, where such channels would occur on a single host in a monolithic computer system. The level of complexity resulting from interconnecting arbitrary hosts running arbitrary applications programs may appear unmanageable at first, because of the potential for large numbers of interactions between heterogeneous host computers, operating systems and processes that must be considered.

4.1. Layering and Abstraction

Fortunately, most communications systems (even untrusted ones) are designed and built in a highly-structured manner that can be used to reduce the complexity of covert channel analyses. The ISO *Reference Model of Open Systems Interconnection* (OSI) provides a framework for defining communications protocols. The actual layers of protocoi that are implemented differ from network to network, however, the purpose of each layer is to offer certain well-defined services to the higher layers, shielding those layers from implementation details. Figure 1 depicts the network architecture used in this paper, based upon the OSI reference model.

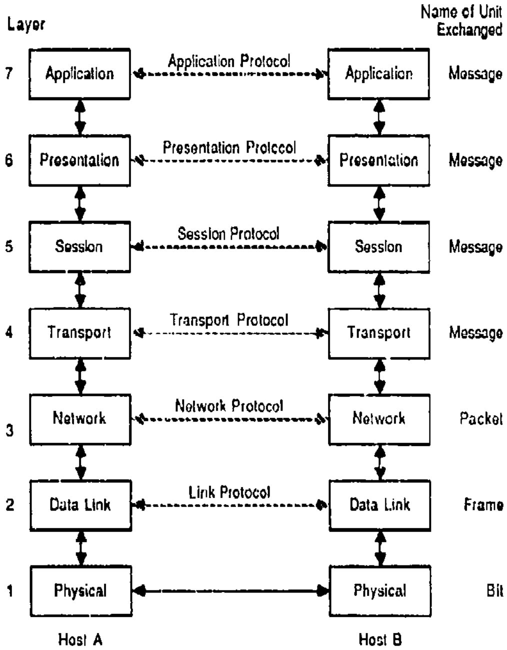


Figure 1. Network Reference Model.

Without going too far afield, it is helpful to review the basic principles that are used in the design of communications networks, for these same principles can, and should, be applied to the development of trusted network systems. As described in [8], the identification of protocol layers is based primarily on the need to deal with a different *level of abstraction*, with each layer performing a well defined function. Strict layering is usually observed, in order to encapsulate the services that are performed and to provide some degree of protocol independence. Without such abstractions and layering, it is unlikely that a network as complicated as the DoD Internet could ever be made to run.

In the following discussion, *layer-n protocols* exist between peer-level entities, while *service interfaces* exist between layer n and (n+1) entities.

4.2. Trusted Protocol Design and Implementation

Unfortunately, the emphasis on protocol layering and abstractions in the communications world is not readily apparent in the TNI[2]. However, such a layered protocol view of the world is not inconsistent with the TNI, and indeed provides an excellent model for analysis of trusted network systems.

Consider a trusted implementation of a layered network component. From a communications perspective, this component would implement protocol layers 1 through n, which would provide layer-n services to layers (n+1) and above. From a trusted system perspective, this component would have its own security policy (and model), which would define access of layer-(n+1) subjects to objects existing at the service interface between layers n and (n+1). For example, a trusted network layer (and below) implementation would mediate access of transport layer subjects to network packet objects. Processes and files do not exist at the network layer of abstraction; only transport-layer protocol entities and packets (or datagrams)[3].

It must be realized that many networking implementations do not necessarily use a separate process for each individual layer of the protocol hierarchy. For example, UNIX systems commonly implement TCP/IP within a single process rather than as separate processes. When a single process is used to implement more than one protocol layer, the combined layers must be treated as a single layer for the purposes of covert channel analysis.

4.3. Covert Channels Within Protocols

This same model can be used for performing covert channel analyses of trusted network systems. Each successive layer of the protocol stack could be analyzed with respect to the covert channels that exist within that layer and lower layers. For example, an layer-n analysis would concentrate on the existence of covert channels through the layer-n implementation that can be utilized by the various layer-(n+1) entities (subjects) through the service interface.

The covert channel analysis for a trusted layer-n component would be concerned only with the identification and analysis of unauthorized information channels between pairs of layer-(n+1) subjects. It would not provide for the identification of covert channels within higher layers, and it can not be responsible for auditing or resolving any channels that might exist within higher layers. It is a basic rule of layered protocol design that a lower-level protocol should not read or modify the contents of higher-level protocols. Similarly, higher-level protocols should not access or modify resources used by lower-level protocols, except through the vocabulary of operations provided by the protocol interface.

4.3.1. Layering Within LAN Systems

Most distributed systems in use today rely on a combination of physical and logical separation at the interfaces between certain layers in the protocol hierarchy. A typical LAN architecture (Figure 2) consists of multiple hosts, each having a dedicated LAN co-processor board that performs packet transmission, reception, limited error control, *etc.* These boards generally implement the physical and link layers, and sometimes the network and transport layers as well. The traditional rationale for this separation has been performance rather than security.

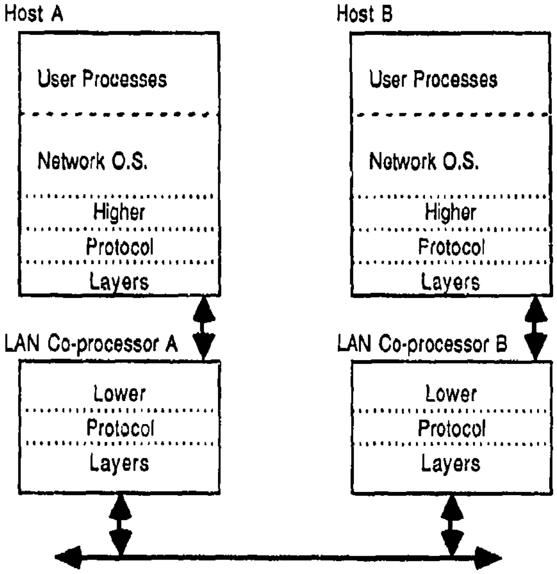


Figure 2. Typical LAN Architecture.

Such physical separation provides assurance that host processes do not have direct access to the physical transmission medium or any data associated with the protocol layers implemented on the co-processor, unless the co-processor interface explicitly provides such access. User-level processes do not have any other means of accessing the internal registers of the co-processor board, observing the contents or addresses of individual packets, *etc.* The net effect of this separation is to limit the available mechanisms for covert communications among legitimate hosts on the network.

Given an appropriate software architecture in which processes are permitted to communicate only through carefully controlled mechanisms managed by the TCB, the same degree of assurance should be possible using logical separation. In either case, careful definition of the service interface between layers is required and the implementation must ensure that it is not possible to bypass this interface. Note that the mechanisms used to implement a layered protocol securely will have much in common with those used to implement TCBs.

[2] For example, Appendix A (Network Components) does not include an example of a layered, distributed component, but rather shows monolithic "boxes" interconnected with wires.

[3] Consequently, it is not possible for a trusted network-layer implementation to qualify as a TNI "system", since it does not deal with the identification and mediation of human-user entities.

4.3.2. Analysis Techniques

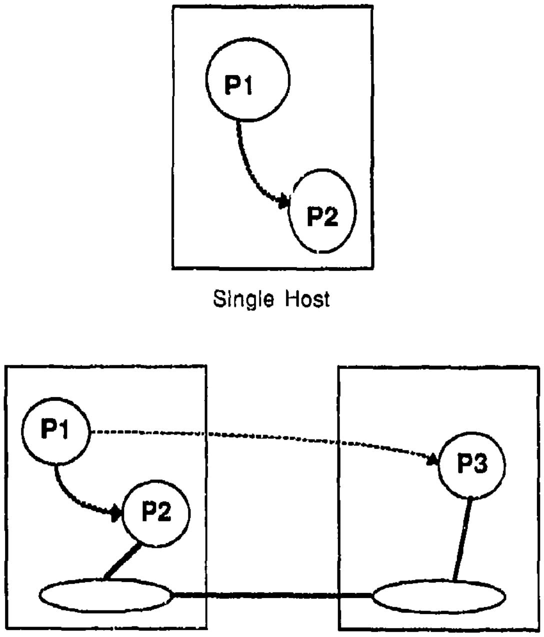
If the Shared Resource Matrix (SRM) methodology [5] is used, we believe the SRM *operations* can be identified by closely inspecting the service interface between the trusted layer-n component and the external layer-(n+1) subjects. A variety of *resources* must be considered, both those visible at the service interface and those embedded within the layer-n component. However, it is not necessary to address any internal resource whose attributes are invisible to the higher-level protocols. For example, it would not be necessary to address the number of retransmissions required to reliably send a packet to a remote peer entity, if the number of retries is hidden within the abstraction of the lower protocol layers. The possibility of composing matrices at the individual levels to provide system level analysis is an interesting subject for further research.

4.4. Host-to-Host Channels

The preceding discussion has presented a general approach to addressing internal covert channels within individual layers of a trusted network system. However, protocol layers may be addressed as a group having a common hardware platform or run-time services, for example, the lower level protocols that normally reside on an Ethernet LAN interface board or the higher-level end-to-end protocols (*e.g.*, FTP, Telnet) that normally reside with the software of individual hosts. Thus, within local area network systems having dedicated LAN hardware, it makes sense to consider separately the covert channels[4] that may exist within the underlying network layers from those within the individual hosts. If this approach is taken, one can then categorize inter-process covert channels as either *intra-host* or *inter-host*:

Intra-Host	Covert channels that exist between subjects on the same host computer can be identified and resolved within that host, independent of any host-to-host connections. The identification of such covert channels is extensively treated in the literature. If a mechanism is identified as providing a potential covert channel, then that mechanism should not be used either within the host or in conjunction with network transfers.
Inter-Host	Covert channels between subjects on different hosts can be addressed in two steps, with host-to-host channels addressed at the lower layer, and process-to-process channels addressed (as above) within each host.

It is possible that a security flaw that has been deemed acceptable within a single host may not be acceptable when the host is connected to a network. This is because the overt communications channels between hosts can be used to extend the number of processes that can take advantage of such a flaw. Consider Figure 3, which shows a covert channel between Processes P1 and P2 in Host A. If Host A is then interconnected with Host B, and peer-level communications protocols are established between the hosts, it is possible that information could be covertly signaled from P1 to P3, using the allowable communications path from P2 to P3. Extended channels such as this (involving multiple transfers) are equivalent to those described by [5] involving multiple attributes, and it is believed that they could be identified by the same method, *i.e.*, a transitive closure of the overall network matrix. In the analysis of covert channels it is not sufficient to determine the security flaw that allows information to work, one must also be concerned with the nature of information that may be leaked through this flaw and the places in the overall system where the information might be extracted.



Single Host

Network System

Figure 3. Example of an Extended Covert Channel.

Even with these additional possibilities for extended covert channels, the layered approach described above has the advantage of being conceptually easier to follow than attempting to address all the channels within a distributed network system at the same level of abstraction. By dividing the potential covert channels, a separation of concerns may be made, and the two distinct cases may be solved individually. As a result of this separation, it is possible to use network-based controls *within trusted lower-level protocols* to significantly reduce the ability of higher-level protocol subjects to interact in a way that violates the system's security policies.

5. Network-Based Controls

The fundamental problem to be solved in the covert channel analysis of a network system is to prevent arbitrary host processes from interacting in ways that may violate security policy. As described above, this process is potentially complex because of the need to address channels between arbitrary processes running in arbitrary host computers.

[4] This is the case if the underlying network is being developed as a trusted component, and is also probably true even if untrusted LAN components are being used with trusted host software.

One way of solving a significant part of this covert channel problem is to embed *network-level* access controls within each LAN interface board, so that only certain host-to-host interactions are allowed. If a particular pair of hosts is not allowed to communicate (at the packet level), then it follows that covert channels cannot exist between processes in those hosts.

This mechanism will suffice by itself in single-user workstation environments where all processes within each communicating workstation are cumulatively allowed (or disallowed) to communicate with all processes in another workstation. However, if some (but not all) of the processes within a host are allowed to communicate with some of the processes within another host, the covert channel analysis must then include an analysis of the mechanisms available within each host system. If there exists a mechanism within one of the hosts that can be used as a covert channel within that host, then the same mechanism can also be used in conjunction with an overt channel provided by the network to covertly signal information to a process in a remote host. This is not a flaw in the proposed method of network-based controls, but rather an unrealistic expectation for the network layer of abstraction.

The use of embedded access controls within the network can also be used to reduce the potential address channel bandwidth in unprotected LAN environments (*i.e.*, to wiretappers). As described in Section 3, the LAN address channel mechanism is possible when a high-level host process can address packets to multiple destinations and a wiretapper can observe the sequence of packets emanating from this host. Reducing the number of authorized destination addresses available to a particular host (and its processes) reduces the width of the channel from the width of the LAN address field to the number of bits that can be signaled using authorized destination addresses. For example, if there are 16 authorized destinations, then only four bits can be signaled per packet, as opposed to the 48 bits otherwise available in the LAN address field. As pointed out in [9], in the absence of ways to force misdelivery of packets on the LAN, there appear to be no host-to-host covert storage channels within the network component itself.

6. Conclusions

This paper has provided an architectural basis for the definition of covert channels within local area network environments. Covert channel analyses for trusted LAN systems must provide for identifying channels between individual host applications running on top of the distributed NTCB. The composition of a network system covert channel analysis from the analyses of individual network and host components is expected to be the primary area of investigation for local area networks that operate in physically protected environments.

In addition, if the network must operate in unprotected environments, the developers should provide mechanisms to protect against covert channels between internal applications and potential external wiretappers. However, in no event should these external channels be the only area of investigation during a secure LAN covert channel analysis.

The architecture of LAN-based systems lends itself to implementing access controls within the network hardware itself in order to prevent unauthorized host-to-host packet flows. This reduces the scope of potential covert channel interactions that must be considered in a network system analysis, Once this capability is provided, it has the additional benefit of eliminating most or all protocol-based covert storage channels by preventing individual host application processes from having direct access to the packets on the LAN medium.

Acknowledgements

The subject matter of this paper is based upon issues which arose during the evaluation of the Verdix Secure Local Area Network (VSLAN). The authors greatly appreciate the technical guidance of the network evaluation group at the National Computer Security Center.

References

[1] – National Computer Security Center, Department of Defense, *Trusted Network Interpretation*, NCSC-TG-005 (Version-1), July 31, 1987, Fort George G. Meade, MD 20755.

[2] – National Computer Security Center, Department of Defense, *DoD Trusted Computer Security Evaluation Criteria* (TCSEC), CSC-STD-001-83, August 15, 1983, Fort George G. Meade, MD 20755.

[3] – Maria M. Pozzo, "Method and Cost of Performing a Covert Channel Analysis", Honeywell Information Systems, Inc.

[4] – Richard J. Feiertag, "A Technique for Proving Specifications are Multilevel Secure", SRI International, Menlo Park, CA, January, 1980.

[5] – Richard A. Kemmerer, "Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels," *ACM Transactions on Computer Systems*, Volume 1, Number 3, August 1983, pp. 256–277.

[6] – M. A. Padlipsky, D. W. Snow, and P. A. Karger, "Limitations of End-to-End Encryption in Secure Computer Networks", MITRE Technical Report ESD-TR-78-158, Mitre Corporation, Bedford, MA, August 1978.

[7] – C. Gray Girling, "Covert Channels in LANs", *IEEE Transactions on Software Engineering*, Volume SE-13, Number 2, February 1987, pp. 292–296.

[8] – Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, 1981.

[9] – John McHugh and Andrew P. Moore, "A Security Policy and Formal Top Level Specification for a Multi-Level Secure Local Area Network", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1986, pp. 34–39.

Secured Communications for PC Workstations

Matthew A. Katzer
Intel Corporation
5200 NE Elam Young Pkwy.
Hillsboro, Or. 97124

Abstract

The iKGM-100[1], Tepache, Advanced Key Generation Module, is a member of the National Security Agency's family of standard embedded COMSEC products. The attached application example ties the iKGM-100 module with Intel's iNA 960 and Microsoft's networks (MS-NET)[2] software to support a COMSEC local area network (LAN). One of the key advantages of using the iKGM-100 is its high-performance, open architecture. In COMSEC LANs, datagram and virtual circuit communications are key to a network's successful implementation. The iKGM-100, as compared to other competitive products, can handle the high demand of LAN communications as well as support datagram and virtual-circuit service

Background

As a result of a "National Security Directive" NSDD 145 signed by President Reagan in 1984, the National Security Agency was given the charter of securing the Nation's tele and data communications network. As a result of this new charter the NSA created the "Commercial Communications Security Endorsement Program" (CCEP). The mission of the CCEP is to provide Communications Security (COMSEC) equipment to the market as quickly as possible. Traditionally, the designs and development of COMSEC equipment was done in total by the NSA, through contract awards. The traditional approach required a 7 - 10 year evaluation effort before the product reached the market. The goal of the CCEP is to reduce that time to less than 2 years. The iKGM-100 is one of the first NSA endorsed CCEP devices.[1]

The purpose of this article is to describe how a personal computer (PC) workstation can be integrated into a secured communications network using the iKGM-100 module. The iKGM-100 integrated component must provide six basic functions for data communications. These functions are; 1) a security fault architecture with complete complements of cryptographic alarms, 2) an optimal architecture to support packet switch and local area networks applications, 3) a controlled cryptographic bypass, 4) a robust instruction set to support key distribution and management functions, 5) a key cache for simultaneous open cryptographic associations and 6) a low development integration and product cost. The key to obtaining NSA endorsement is minimizing the additional security firmware/software to interface the COMSEC component. The iKGM-100, an NSA endorsed device, is fully compliant with this criteria.

The direction of the secured networks is to support commercially available protocols. Even though the mature protocols for LAN's are based on TCP/IP, there is a need to provide secured LAN communications implemented on International Standards Organization (ISO) protocols. The focus of this example is to review Intel's iKGM-100 as it applies to a Type I secured data network architecture.

This application example integrates the iKGM-100 into the iNA 960 (Intel's Networking Architecture) software structure to provide a secure architecture as it relates to an personal computer networking environment. The objective of this application is to use an ISO based LAN with Microsoft's Networking Software (MS-NET).

Overview

The iKGM-100, Advanced Key Generation Module, is a member of the National Security Agency's family of standard embedded COMSEC products. The attached application example ties the iKGM-100 module with Intel's iNA 960 and Microsoft's MS-NET software to support a COMSEC LAN environment. One of the key advantages of using the iKGM-100 is its high-performance open architecture. In COMSEC LAN environments, datagram and virtual circuit communications are a key to a LAN's successful implementation. The iKGM-100, as compared to other competitive products, can handle the high demand of LAN communications as well as support datagram and virtual-circuit service

Intel's communications software supports the Open System Interconnection Model (OSI) at all layers. The MS-NET software is an accepted standard of data communications for Xenix[2], Unix System V, MS-DOS and iRMX (Intel's Real Time Multitasking Executive software). The MS-NET protocols provide transparent access to files anywhere in the network. The MS-NET software is similar to the MAP 3.0 upper layer protocols and represents layers 5-7 of the Open System Interconnection (OSI) model.

Intel's ISO certified (International Testing Institute validated) software is MAP 3.0 or TOP 3.0 compatible. The ISO software from Intel is available in two forms; MAP (layers 5-7), or iNA 960 software executes on any Intel processor (8086, 80186, 80286 or 80386). The MS-NET software communicates with Intel's iNA 960 ISO-complaint software.

This application example is based on a smart PC LAN board. The PC LAN board uses the standard MS-NET software and provides a NETBIOS interface. Modifications are made to the iNA 960 software base to allow the incorporation of the iKGM-100 module into a smart LAN design. Figure 1 shows the implementation of this module.

Hardware Architecture

Four areas need to be addressed in the hardware design. These four areas are:

- Split internal bus design using the iKGM-100 (inline design)
- Duplication of the bus interface to be compatible with PCLINK2
- Maintain or improve system performance.
- Maintain software compatibility.

The overriding architectural design criterion was to maintain software compatibility where possible. In order to fully understand the hardware requirements, each of these areas will be reviewed.

ETHERNET LAN USING IKGM-100

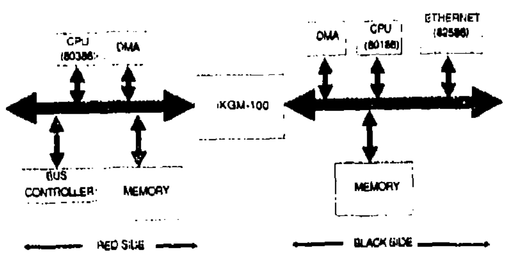


FIGURE 1

In-Line Design:

There are two methods that can be used in a design with the IKGM-100 Key Generation module. The possible methods are a co-processor mode and an inline mode. The co-processor mode is for trusted systems where the module is use as part of the communications kernel. The trusted system environment controls all communications and levels of access. Hence the system environment has the level of trust necessary for the communications traffic. In trusted environments, there isn't the need monitor all traffic from the computer system to the network.

The inline mode is used for systems that do not provide a trusted environment. The inline implementation blocks un-authorize access from transferring data to the communications medium. Hence an inline design need not concern itself with the classification of the user, since classification is a function of the key access. In all cases, the IKGM-100 module blocks the user access is to the outside world for clear text. In some cases, there is a need to transfer clear text, the IKGM-100 management facility allows accountability by controlling the data with bypass features of the module. The IKGM-100 module is designed in such a way that the level of classification is a function of the key initialization process. Typically, with out power supplied to the IKGM-100 module, the device is viewed as controlled cryptographic product, that meets Type I applications. Classification of the device exists once the system is initialized with its key.

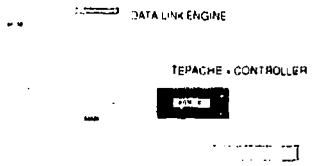
The inline design essentially splits the bus separating clear text (Red side) from encrypted text (BLACK side) with the IKGM-100 module. In the PC design, the bus split is accomplished on the PC-controller card (figure 2). The bus split allows the division of hardware between the Controller card and the Datalink card. The Controller card contains the IKGM-100 module. The inline design does not allow the IKGM-100 module bus access. All data traffic to the datalink card must flow through the IKGM-100 module. The CPU on the controller card is an Intel 80386 with an integrated protection architecture and the appropriate interface devices to the IKGM-100 module.[3]

The controller card design has enough horsepower to handle the secured communications software overhead, and the IKGM-100 interface. The addition of the direct memory access (DMA) unit may allow the use of a less expensive CPU. However, due to the nature of the hardware architecture, a CPU with a built-in protection architecture is key to this design's success. One of the main security functions of the controller card is to handle the key manipulations to/from the IKGM-100 module, and to control any unauthorized access to the module. The 80386 and 80286 central processing units (CPU) meet protected code requirements, providing instruction trap faults when illegal operations are performed.

The datalink monitor card contains the ethernet component (82586) and a 12Mhz 80186 device. This board is designed to handle the reception and transmission of data packets from the Ethernet network and the IKGM-100 device. There isn't a need for a fast CPU at this end, only for a CPU that can handle the network layer and routing overhead. The split bus design using the IKGM-100 allows the ideal separation of processors on the BLACK and the RED side.

Each of the CPU designs contain RAM and EPROM. The control software may be ROM resident or RAM resident. The hardware design can support both situations, but do to the environment, the software is implemented in ROM. Figure 2 shows the proposed implementation using the IKGM-100 module in a split PC board design.

SECURED COMMUNICATIONS



SPLIT BUS LAN DESIGN WITH IKGM-100

FIGURE 2

System Performance:

The system performance must appear to the user as if the IKGM-100 module does not exist. The dual processor design allows for the maximum system performance. The dual processor implementation uses an Intel 80386 at the RED side to handle bus communications, IKGM-100 management and transport packet redirection. The BLACK side contains the 12Mhz 80186 processor. If needed the BLACK side processor could be changed to an Intel 80286 or 80386 which is object code compatible with the 80186 device.

One of the additional features of using an Intel 80386 is the capability to isolated security features from the network applications code. In any endorsement activity, the speed of achieving endorsement is a function of isolating the IKGM-100 access code. The Intel 80386 CPU allows high performance and a protected architecture that isolates application code, and improves the overall efficiency of the communications system.

Software Compatibility:

The main issue for software compatibility is to support MS-NET, NETBIOS and ISO standard software. Figure 3 contains an IBM PC compatible ethernet communications card's (PCLINK2) software architecture. The PCLINK2 architecture allows applications programs to communicate transparently with a network based communications server. This means that any software program that uses NETBIOS or MS-NET on PCLINK2 may execute across the network using ISO standard communications protocols.

The PCLINK2 architecture will be the basis for the secured data network design. Using the PCLINK architecture, current directions in communications security places the data communications encryption engine below ISO transport layer. By using a software architecture as shown, user applications are guaranteed 100 % software compatibility. Hence, any hardware changes to the IBM-PC environment would be restricted to the front end processor. The user would be able to access any off-the-shelf software packages without affecting the security capability of the secured personal computer.

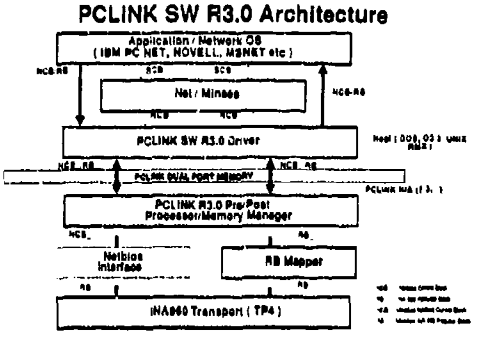


FIGURE 3

The software compatibility may be simply redefined as a hardware communications architecture that does not affect any PC software application packages. The split bus design with the IKGM-100 has successfully isolated the hardware architecture from the applications software.

Software Architecture

The major design effort required is in the communications software. Intel's Networking Architecture (INA) allows a simplified approach. The INA design is around a kernel, with separate protocol units performing the transport and data link functions. Figure 4 shows the protocol environment of INA 960, a subset of INA. INA is the key software design architecture required to implement a secured networked workstation.

The objectives in the software implementation of the secured network are as follows:

maintain programmatic interface to MS-NET and NETBIOS.

- minimize software that interfaces to the IKGM-100
- minimize software required for endorsement
- provide a platform for SDNS development
- absolutely no host operating system changes

Each of the above objectives require that the communications software be flexible; to meet the secured data network requirements for today and tomorrow. In order to fully understand the implication of the design, each area will be reviewed.

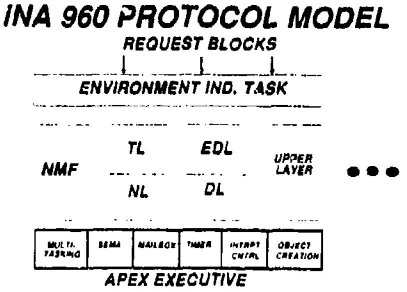


FIGURE 4

Programmatic Interface:

One of the few areas that users tend to ignore is the maintenance of the programmatic interface. All MS-NET, NETBIOS and LAN-Manager implementations are fully compatible with this design. The application software must be able to execute on the system without any changes. For example, in a networking environment, the DBASE III or Word Perfect software packages must be able to make NETBIOS calls, and access data without any problems. In order to accomplish this, the programmatic interface must remain unchanged.

Intel's PCLINK2 Release 3 software allows this capability. The user will have full access to all of the NETBIOS features. This means that the IBM-PC networks program must be able to execute without any errors. Hence, the user will be able to use any applications software from the user community.

Software required for the IKGM-100:

The INA 960 architecture allows the creation of user tasks to perform functions. Figure 5 shows the INA 960 architecture. In this diagram, the INA 960 software was split. One processor handles the transport layer, the other processor handles the network layer. The IKGM-100 module sits between the RED and BLACK sides.

The software required to communicate to the IKGM-100 may be separated into two different tasks. One task, on the RED side, handles all of the communications to the modules as well as key initiation and management. The Applications Programmatic Executive (APEX) task coexists with INA 960 and performs any needed communications to the IKGM-100 module. The RED side task is simplified, handling only the data input to the IKGM-100 module. In the software design, the 80386 CPU ring protection module is used to further segment the user access. The actual network communications software on the RED side is contained in the lowest protection level of the processor. This access level grants all users access to the communications facilities. The highest level of protection is given to the iKGM-100 management functions. In figure 5, the level 0 privilege functions are shown on the RED side, and level 4 privilege functions are given to the ISO communications software (Intel's iNA 960).

iNA960 Architecture with iKGM-100

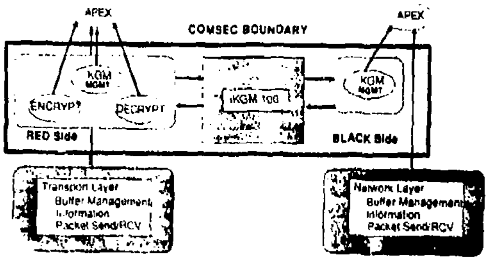


FIGURE 5

The second task, located on the BLACK side, is responsible for transferring information from the iKGM-100 module to the network layer. The BLACK side task is also responsible for handling any re-synchronization with the iKGM-100 module. This entails the transfer of network management information, encrypted keys and data received from the network layer. The BLACK side task does not need to have the same level of trust do to the structure of the inline design. An example of the software architecture is shown in figure 4.

The final area is the issue of Key management. The structure of iNA 960 allows different application tasks to exist. The key management task executes as a function under the APEX kernel, in the 80386 privilege level 0 and outside of the ISO protocol as much as possible. A simplified view shows key management as a function of network management.

Endorsement:

The major advantage of using iNA 960 is to minimize the endorsement process for the secured LAN product. The COMSEC boundary, (shown in figure 5), limits the software required for endorsement to the RED and BLACK tasks. The RED and BLACK tasks are structured to handle iKGM-100 management and various encryption management utilities. Due to the communications structure of iNA 960, all communications protocols are limited to distinct task architecture. Hence, the only endorsement that is required is for the interface software to the iKGM-100 module. This is the case due to the development of the iKGM-100 as an endorsed NSA CCEP device.

In contrast, if a non CCEP module is used, the COMSEC boundary may be extended to the entire software task structure identified in figure 4. As a result of the new COMSEC boundary being defined, the endorsement process will be longer. For example, if a product like iNA 960 was used on a module that is not endorsed by the CCEP program, the complete ISO protocol implementation and the hardware performing the data encryption would need to be endorsed. As a point of reference, iNA 960 contains 20 person years of code development. (18,000 lines of source code). In general, a design that has not been evaluated to the rigorous security analysis, must eventually be evaluated to that standard. The current evaluation standard is a time consuming process and any error noted will require a redesign plus reevaluation of the product. Potential products submitted to NSA that do not conform to the CCEP standards, may take up to 10 years to receive the officially NSA endorsement.

Secured Network Platform:

The secured network communications architecture requires flexibility. The goal of the secured network architecture is to support current communications technology and the next generation systems. In order to meet the flexibility requirements, the software architecture must allows the testing of different communications hypotheses. Flexibility requires that the encryption module must be configurable to support today's communications needs as well as tomorrow's. Intel's products, iNA 960 and the iKGM-100 key generation module are extremely flexible in their design. iNA 960 allows multiple user tasks to interact with the software protocols. The iKGM-100 supports multiple data encryption schemes for greater software flexibility.

The secured data network implementation needs to be able to create a management task, (as shown in figure 5) to perform COMSEC functions. The ISO software product, iNA 960, allows the user to create multiple management tasks. In the creation of these tasks, the user may integrate multiple applications based on transport address ID's. Likewise the network service access ID's may be used on the BLACK side as well. In addition to communicating to the protocol modules, the secured network implementation architecture needs to use a dual tasking architecture. In the COMSEC boundary, one APEX task (transport and network management) can execute on the RED side and the other on the BLACK side (network and datalink).

The use of iNA 960 together with the iKGM-100 module allows maximum flexibility. Any secured network implementation may be readily prototype, tested and put into production. iNA 960 and iKGM-100 meet the needs of any secured communications development.

No Host Operating System Changes:

The major objective of the software design was to not to change the host operating system MS-DOS. All additional commands necessary to initialize the communications module may take place as application software running under MS-DOS or Windows-386. The application implementation described allows the user to purchase an off-the-shelf personal computer, with the COMSEC module, and connect it into the network. The goals in this design are simplicity and flexibility. Since the communications software is restricted to the intelligent LAN board, all software changes are contained on the COMSEC module. Off-the-shelf application software packages may be used to access network information. The applications would only see the MS-NET or NETBIOS interface (figure 3), not the iKGM-100 module.

Summary

Overall, a design with iNA 960 and the iKGM-100 allows the greatest flexibility in a secured data network. Due to the iNA 960 internal design, splitting the communications software can be easily accomplished. The iNA 960 software executes under an APEX kernel. The transport and network interfaces are isolated to a few procedural calls (buffers, information, and packets etc.). These procedural calls may be easily broken down into their individual parts, specific to the communications implementation.

The software environment of the PC would remain unchanged. There would be new utilities for key management, but the base operating system would not be touched. This is extremely critical. The user that would execute software on this workstation would be able to use any off-the-shelf MS-DOS product. Hence, the secured network objectives could be reached with an architecture similar to PCLINK2 which would incorporate the iKGM-100 module: that is a Secured PC LAN card, easy to use, with minimum performance loss, and a cost effective solution.

[1] IKGM-100, iNA 960 and PCLINK2 are a trademark of Intel Corporation.

[2] MS-NET is a trademark of Microsoft Corporation

[3] The iKGM-100 Advanced Key Generation Module, in addition to providing an NSA developed encryption/unencryption capability, also has a robust set of Agency endorsed Security Features. These features are outlined below:

Internal Storage - The iKGM-100 has volatile storage for up to 255 keys. This feature allows simultaneous connections in supporting various key management schemes.

Key External Storage - In addition to having storage on-board the IKGM-100, allows external storage of the keys in the host. This feature allows keys to be stored in the host for later use. As with key storage this feature is useful in key management schemes.

Remake Keys - The IKGM-100 has the ability to electronically distributes initialize vectors to remote users. This feature is useful in key management of geographically dispersed environments.

CIK Support - Support for a "Crypto Ignition Key" is available on the iKGM-100. This feature is particularly useful in the workstation environment. It allows a COMSEC environment to be locked and unlocked cryptologically.

High Bandwidth - The IKGM-100 has 6 cryptographic operating modes plus a Message Authentication Code (MAC) mode. The maximum thruput is 7 Megabits/second. The iKGM-100 transfer rate is based on the cryptographic mode selected. In the application discussed in this paper the cryptographic mode selected will operate at 2.87 Megabits/second.

Alarms - The iKGM-100 provides security protection in the form of internal alarms to detect intrusion and internal failures. These functions can be utilized in various methods to insure system security.

Controlled Bypass - a particularly useful feature incorporated into the design of the iKGM-100 is "controlled bypass". This feature is used to bypass clear text such as header information or control characters.

All of the Features discussed above have been endorsed by the National Security Agency for all levels of classified communication. By utilizing the IKGM-100 in COMSEC designs a majority of the security design criteria have been approved as implemented. This is the major reason for choosing the IKGM-100 for the Secured PC LAN controller.

[4] The current implementation of the Tepache module does not support a firefly exchange. This capability is being reviewed for the next generation module. The goal of the design is to allow firefly exchanges to be processed in less than a second.

[5] Xenix is a trademark of Microsoft Corporation. Unix is a trademark of AT&T Corporation. MS-DOS is a trademark of Microsoft Corporation. IRMX is a trademark of Intel Corporation.

[6] PC/XT and PC/AT is a trademark of International Business Corporation.

Reference Documents

Intel Corporation, iKGM-100 "Advanced Information" Data sheet. March 1988 - Unclassified.

Intel Corporation, The IKGM-100 Advance Key Generation Security Module. March 1988 - Unclassified.

Intel Corporation, iNA 960 Architecture Reference Manual #122194-001. March 1987 - Unclassified

Intel Corporation, Apex User's Guide Internal Product Specification, #460275. March 1987 - Company Confidential.

National Security Agency, Tepache, Interface control Document (ICD), No. 86-31D. March 1988 - Unclassified.

National Security Agency, Tepache Embedding Handbook and Application Notes. November 1987 - Classified.

AN OVERVIEW OF THE GEMSOS CLASS A1 TECHNOLOGY AND APPLICATION EXPERIENCE

William R. Shockley, Tien F. Tao, Michael F. Thompson

Gemini Computers, Inc.
P.O Box 222417
Carmel, CA 93922

ABSTRACT

The Gemini family of high assurance trusted computing base (TCB) products is described. The manner in which these TCB products, which are designed to be Class A1, address different operational requirements is addressed. A selected list of current applications and projects in which the currently available products are being used is presented, in addition to a description of several research projects that illustrate the product's current potential and future directions.

INTRODUCTION

The purpose of this paper is to provide an overview of the GEmini Multiprocessing Secure Operating System (GEMSOS) product line, which includes a variety of hardware and system software components. The major components offered commercially are a family of hardware systems in a variety of configurations, a high-performance multiprogrammable, multiprocessor security kernel [1] to control the hardware systems, and a TCB designed to meet Class A1 that includes the security kernel and hardware systems. This family of products has been under developmental evaluation for several years as the design and implementation has proceeded. Advance versions of the hardware base and security kernel have been available as commercial, off-the-shelf products for several years and have been selected for incorporation as part of several development projects that will be independently evaluated as Class B3 or A1 systems. We anticipate entering the formal evaluation process with the entire TCB within a few months, with completion of a successful formal evaluation later in 1989.

As for any family of commerical products, key architectural characteristics common to the entire family can be discerned that reflect the beliefs of the designers as to what will be needed for a commercially viable system. In the case of GEMSOS, an emphasis has been placed on the use of an identical security kernel throughout the current and future product line, the achievement of a low cost/performance ratio using a multi-microprocessor hybrid CPU architecture, the careful structuring of the TCB into separable, and independently evaluatable TCB subsets [2] enforcing orthogonal security policy components, and the use, wherever feasible, of industry-standard components and interface specifications. The result is a family of highly-assured, trusted components that can be used as stand-alone systems or as parts of more complicated trusted systems, or which can be modified or extended by third party vendors with a minimum impact on re-evaluation. (In particular, the security kernel itself rarely needs to be modified, and can be treated as a high-assurance "sealed unit".) We call this concept an "open security architecture".

Because of the importance of the open security architecture for the design of GEMSOS, the next section discusses our reasons for believing that the concept is appropriate for a commercial product. We then present a technical overview of the GEMSOS architecture, followed by a selected list of applications and projects in which the security kernel has already been used. This is followed by a description of several research projects that indicate more advanced intended applications for the GEMSOS TCB, emphasizing projects intended to provide some measure of compatibility with existing standard (non-secure) system interfaces.

PRODUCT STRATEGY

We view our corporation as a supplier of primary high-assurance (Class B3 and A1) components to system integrators requiring such components to meet the needs of their end-users. We believe that for a viable high-assurance TCB market to exist, it must be based upon the shared use of the critical technical component required for any high-assurance system, a security kernel. In order to be useful as the basis for a self-sustaining vendor/user community, that security kernel must support a wide range of applications, be cost-effective from the standpoint of performance, support the construction of distributed and networked systems, and be available to, and usable by, value-added and third party commercial vendors so that the creation and maintenance of a large body of usable application software can be fostered.

Our overall strategy, then, for participating in the market for high-assurance systems has been to first, develop a security kernel with the required technical and commercial characteristics; second, to build a TCB based upon this kernel and complete its evaluation at Class A1 of the Trusted Computing System Evaluation Criteria [3], and third, to foster its utilization by as wide a variety of vendors, system integrators, and software developers as possible. We have completed the first task, are engaged in the second, and intend to continue the third as vigorously as possible.

Because our security kernel is the key technology around which the remainder of our product and most of our technical activities are organized, it is worth taking a broad look at its design characteristics before proceeding into a more detailed discussion of our activities. The GEMSOS security kernel SP (which includes the hardware as well as a software component) is:

- always invoked, (that is, cannot be bypassed by any non-kernel programs or by users);
- tamperproof,
- small enough, and well-structured enough, to support evaluation at Class A1;
- built, for the most part, from industry standard hardware components (Intel 80286/80386 processors, Multibus I backplane with third-party circuit boards, IBM PC/AT hardware, etc.);
- supports, as feasible, industry standard interfaces (RS-232, Ethernet, X.25, EGA, Centronix, etc.);
- is, therefore, **portable**;
- organizes the remainder of the software system into eight hierarchical **protection rings** which makes it **extensible**, (i.e., TCB subsets enforcing additional access control policies and supporting policies can be erected "on top");
- has a low cost-performance ratio (because it utilizes inexpensive microprocessors);
- has flexible capacity with a high maximum performance (because it utilizes up to eight processors in a proprietary architecture that reduces bus traffic substantially);
- is accompanied by a UNIX * programming environment providing the basic tools needed to program the system using modern high-level languages (Pascal, C);
- and, last but not least, is **done**. (The security kernel has been available and delivered as a commercial, off-the-shelf product, for over two years.)

The complete GEMSOS TCB, which is currently under development, is, of course, based on the GEMSOS security kernel. Our product line, which includes hardware components and the security kernel, and soon will include the remainder of the TCB as well, is designed to support four major categories of use:

- a **dedicated application** market, comprising custom applications written to serve a specific end-user installation or requirement, (i.e., for a sponsored development project),
- a **value added** market, comprising customers who wish to add significant software applications (e.g., message handling systems, file servers, communications software, DBMS software) as secure applications to the TCB, without disturbing the internals of the TCB in any way;
- a **second source** market, comprising customers who wish to market their own TCB but avoid the cost of developing their own security kernel.
- an **embedded component** market, comprising customers who need highly-assured components for larger systems applications (e.g., network components).

All of these potential markets have differing needs: the one thing they have in common is the need for a highly-assured security kernel whose development expense is already being amortized.

The **dedicated application** market needs a wide range of configuration and performance options, so that the delivered system can be tailored to the precise needs of the end application. In addition, a custom application typically will require customized discretionary, identification, authentication, or audit policies, or a combination of these. The GEMSOS kernel is therefore designed to support a wide range of hardware and peripheral options in both loosely-coupled and tightly-coupled configurations, and the GEMSOS TCB is composed of subsets so that individual policy components can be modified without disturbing the software or evaluation evidence already available for the rest.

The **value-added** market needs a widely-used equipment base with a stable interface so that there is a viable market for third-party applications. A secondary (but often, important) need is the ability to bypass general-purpose operating system functions in order to attain performance goals. By making the details of the TCB interface available to third-party developers, both of these needs are fostered: this is the "open software architecture" policy which has been widely successful in the microcomputer industry.

* UNIX is a trademark of AT&T

The dubious benefit of "locking in" end users to a sole supplier (by concealing the details of important system interfaces) is foregone in favor of fostering the development of third-party add-ons and applications in order to build a self-sufficient community of users and vendors.

The second **source** market is served by making our technology available under license. If, for instance, a vendor believes that there is a marketable "better way" to provide discretionary controls than we supply, or can beat our price, it is at least straightforward to procure the requisite OEM license in order to use the security kernel as the basis for a competitive TCB. Licenses for the utilization of our technology are available under a variety of different business arrangements, including source code licenses for the security kernel itself.

Finally, the **embedded component** market is served by providing a high-performance security kernel based on commonly available microprocessor technology that can be ported to new hardware environments. The intrinsic portability of the kernel has already been demonstrated by porting it to the IBM PC/AT * hardware environment, which proved to be a relatively inexpensive effort. Should the end-user community provide sufficient demand for a lightweight (aerospace) or tempested enclosure, for example, the technology could be licensed to a prime contractor or commercial vendor and ported to a "design to specification" enclosure.

PRODUCT DESCRIPTION

The Gemini family of computer systems provides a powerful combination of multilevel security and multiprocessing capabilities. The adaptability of GEMSOS makes the Gemini systems attractive as a trusted base for a wide range of concurrent and real-time computing applications including command and control, communication, intelligence, weapons, networks, and office automation end uses.

Within each enclosure, tightly coupled multiple microcomputers communicate through shared memory segments to provide high-throughput performance. Up to 8 Intel 80286 or 80386 based microcomputers can be served by the same Multibus to provide the required amount of processing power. GEMSOS avoids bus contention by locating data and code segments in local memory of each microcomputer whenever possible.

Between loosely coupled enclosures, processes communicate via ethernet, X.25, or RS-232 based multilevel channels.

GEMSOS Security Kernel

The GEMSOS security kernel supports multiprocessing as well as multiprogramming. The kernel virtualizes all system resources, providing service calls for the required process management, segment management and device management.

The GEMSOS security kernel enforces a label-based Mandatory Access Control policy. The commercially-available GEMSOS kernel supports both secrecy and integrity access class components, each with 16 hierarchical levels. GEMSOS also supports 64 non-hierarchical secrecy categories and 32 non-hierarchical integrity categories. For licensed or OEM systems, the non-discretionary security module of the Security Kernel can be customized to support any lattice security policy, including Clark-Wilson [4], trusted pipelines [5], or policies needing multiple secrecy and/or integrity hierarchies or extended numbers of non-hierarchical categories.

Hardware Configurations

A "Gemini System", in the most general case, consists of multiple loosely-coupled hardware enclosures. A typical configuration might consist of one or more Gemini central host processors, together with as many secure workstations as required, communicating via Ethernet or RS-232 communications channels. Where the cost of an "intelligent" secure workstation could not be justified, a smaller Gemini host configured as a terminal concentrator could be utilized to support communications between multiple "dumb" terminals and the central hosts. A low-end, multi-user configuration might consist of a single Gemini central host, accessed by means of "dumb" terminals connected directly to the host processor.

A single Gemini enclosure is either a Multibus-based, tightly-coupled host processor, or a MLS-AT workstation, which is the GEMSOS configuration that executes on PC/ATs and selected compatables.

The Multibus-based systems are designed to offer a wide range of configurable options: three enclosure sizes are offered, offering support for from one to eight processors (Intel 80286 or Intel 80386). Together with memory options, the throughput rates available span a range of from 0.5 to 24 Million Instructions Per Second. A variety of secondary storage and I/O options are currently implemented. An "open architecture" approach has been designed into the system from the beginning: industry-standard components and non-proprietary Multibus I boards allow customized and tailored applications to be considered. The single Gemini proprietary board for the Multibus system is the Gemini System Controller, which uses proprietary technology to enhance bus performance as well as providing certain peripheral

* PC/AT is a trademark of IBM.

devices (e.g., hardware DES encryption support) required by the TCB.

The Gemini Multibus I systems provide access to supported peripherals through (possibly tailored) third-party interface boards directly connected into the Multibus. Under GEMSOS control, microcomputers share all devices interfaced to the Multibus. The system supports various combinations of hard and floppy disks as well as streaming and half-inch, 9-track tape. Non-volatile memory is available for "core resident" applications. Additional devices include 8-port RS-232 serial I/O cards, ethernet and HDLC. Each Gemini system includes a real-time clock with battery, a data encryption device using the standard NBS-DES algorithm, and a system-unique identifier. The system also contains battery backed up CMOS for storing operator passwords and encryption keys. Business arrangements are available for technology licensees to add application-specific device drivers to the system: the range of commercially available device drivers supported by the kernel is, of course, continually expanding as Gemini adds device drivers to the repertoire of supported interfaces. The TCB and kernel device drivers are typically low-level (as required by a "minimized" TCB architecture for Class B3 and above).

Each single-board CPU includes local memory (allocated and controlled by GEMSOS) as well as a bus interface unit, and several local I/O ports. GEMSOS virtualizes the hardware configuration by making the allocation of local and global memory to segments transparent to applications: it is also relatively straightforward to construct multiprogrammed applications that are independent of the number of processors available as well. Inter-process synchronization is identical for processes executing on the same processor and processes allocated to different processors. The TCB supports the creation of remote processes (in a different enclosure) with the same mandatory and discretionary attributes as the creating process in order to support distributed applications without impacting the validity of the system security controls.

Gemini's family of TCB products includes a MLS AT Workstation, which is a configuration of GEMSOS that executes on IBM PC/AT hardware (and selected compatibles). The intended use of this configuration is to provide a low-cost alternative where a secure, "intelligent" workstation is required by a system architecture. Although the specific I/O devices available at the workstations differ in detail from those available for a Multibus I enclosure, in all other respects the MLS AT Workstation is simply a single-processor GEMSOS system at the programmer's interface. This GEMSOS configuration includes support for the standard Enhanced Graphics Adapter (EGA), keyboard, serial I/O ports and the Centronix parallel printer port.

NON-KERNEL TCB DEVELOPMENT

Gemini is currently implementing the GEMSOS non-kernel TCB elements, which include discretionary access controls, authentication, security administrator support, audit and support for inter-enclosure communications over multilevel channels.

Multi-ring Architecture

The extensible nature of the GEMSOS Security Kernel provides designers great flexibility in the design and implementation of trusted computing base capabilities on top of the kernel [6]. Gemini has separated the non-kernel TCB functions into five distinct protection domains (rings) [7]. In addition to the obvious "least privilege" benefits, this approach allows Gemini to offer customers the ability to tailor specific non-kernel TCB functions with minimum impact on the basis of evaluation for the functions allocated to other rings. In particular, the basis for evaluation of the most demanding component of a high-assurance TCB, the security kernel, is completely preserved. The intended use of this architecture is to support the capability to tailor discretionary, identification, authentication, and audit functions to specific installations or applications (e.g., for a dedicated aerospace or military application) without incurring the complete cost of building a special-purpose high-assurance TCB from scratch.

In addition to the five rings dedicated to the evaluated TCB, three additional rings (for a total of eight) are made available to the applications. Nominally, Ring 5 is allocated to operating system or major system applications (such as DBMS run-time modules), Ring 6 to "approved" applications (i.e., those that have passed site-specific criteria for correctness of behavior), and Ring 7 to "ad hoc" applications (i.e., those that are under construction, or whose trustworthiness is unknown). It would be possible, in many cases, to enforce site-dependent security controls (time of day, separation of duty, etc.) in Ring 5 as a refinement to conventional discretionary and mandatory controls without disturbing the evaluated TCB in any way whatsoever.

The GEMSOS Security Kernel uses the four hardware privilege levels of the 80286/80386 to enforce the ring constraints on a process-by-process bases. Each process may have only three of the eight available rings active at any given time: a typical "application process" will have available two rings for the application code with the most privileged of the three rings dedicated to the TCB. Thus, within the address space of each process one finds code for accessing services through the TCB, code for operating system services, and the application code. Service requests are typically handled without the need for expensive inter-process communication or context-switching between processes. Transfers to and from secondary storage are handled (transparently to application programmers) in the course of making segments accessible.

Multilevel server processes (such as terminal servers and multilevel channel servers) have two active TCB rings and one active ring available to non-TCB functions. The active single-level non-TCB ring in multilevel server processes will typically contain non-security relevant operating system device driver functions that can be customized without effecting the TCB. Application processes communicate with multilevel servers via inter-process communication. Single level devices can be directly attached by application processes.

Distributed TCB Interface

Programs external to the TCB gain TCB services via a program interface with the ring containing the discretionary access control mechanisms. This ring is referred to as the "distributed TCB" in that it is distributed (in the address space) of each of the system's processes. The TCB supports a high degree of concurrency so that more than one application process can be "in" the TCB at the same time where multiple processors are available.

Inter-Enclosure Data Communications

The GEMSOS non-kernel TCB supports communication between enclosures using multilevel communication channels. These multilevel channels are used for both TCB and non-TCB communication between enclosures.

The GEMSOS TCB provides applications with an interface that allows an application process in one enclosure to send and receive information to other application processes executing in another enclosure. The TCB also supports the remote creation of application processes with the same security attributes as the creating process: thus, it is straightforward (from the application programmer's point of view) to provide a remote processing capability, while the security enforcement of the distributed system is not compromised. If, for instance, audit logging is centralized, the creation of a remote process and its subsequent activity will be correctly logged by the TCB and associated with the user identified with the original process by the TCB without the application designer having to make any special provisions for this to happen.

The abstract inter-enclosure communications capability provided to applications programmers may be described as an enclosure-to-enclosure, connection-oriented, transaction based communications system appropriate for use in a distributed architecture. The evaluatable design does not specifically address the needs of more dynamic networks of enclosures linked by unreliable, or physically insecure, communications channels, though the underlying design is extensible to support such systems.

The non-kernel TCB uses the inter-enclosure communications functions for TCB communications among enclosures within a system. In distributed systems (viz., those containing more than one enclosure), certain TCB databases can be centralized in an enclosure that provides TCB services to the remaining system enclosures. Audit records are collected at a central point, and the difficult problems associated with concurrent user permission databases are avoided.

Distributed systems containing multiple loosely-connected enclosures inter-connected by multilevel channels can be viewed as a single trusted system having a single TCB. Thus, a distributed Gemini system possesses a coherent Network Security Architecture and Design, as defined in the Trusted Network Interpretation [8]. The anticipated formal evaluation for compliance with the Class A1 of the TCSEC [3] will be for a generalized multi-enclosure, distributed architecture so that the evaluation results will be immediately applicable for applications making no modifications to the TCB, but requiring a distributed architecture (e.g., if MLS AT workstations are used).

EXISTING AND PLANNED APPLICATIONS

Unisys Defense Systems

In his oral presentation at the 1988 IEEE Symposium on Security and Privacy [9], Clark Weissman of Unisys stated that the GEMSOS security kernel was used by Unisys Defense Systems in their design and implementation of a federal communications system called Blacker, meeting the requirements for Class A1.

In his presentation at the 10th National Computer Security Conference, Jon Fellows of Unisys stated that the GEMSOS kernel, along with other trusted components, is used as the basis for trust for critical cryptographic and key distribution functions that maintain communications separation by cryptographic means. [10]

SACLANT

Multiprocessor Gemini systems and 50 MLS AT Workstations configured in a "star" network were included in the accepted CDR presented by Contel Federal Systems as part of Contel's contract to develop the SACLANT Command and Control Information System for NATO. The design includes the GEMSOS security kernel and a subset of the GEMSOS non-kernel TCB elements [11].

IBERLANT

A configuration of Gemini computer systems similar to that used in the SACLANT architecture was part of Contel Federal Systems' winning proposal to develop the IBERLANT Command and Control Information System for NATO.

Message Editing and Preparation Demonstration

Astronautics Corporation of America (ACA) has developed a multilevel secure message editing and preparation demonstration on the MLS-AT Workstation configuration of the GEMSOS security kernel and a subset of the GEMSOS non-kernel TCB. This demonstration includes message creation with message masks, message editing, transmission and reception of messages over multilevel communication channels and message printing.[12]

Grumman Data Systems

At the 1988 AFCEA International Conference and Exposition, Grumman Data Systems developed and demonstrated a front end secure communications processor that provides users at terminals access to information at multiple levels while maintaining a single system view for the user. The secure communications processor allows users to connect to one of multiple back end host computers that run at different security levels. Users may also establish multiple sessions with hosts at different access classes through the use of multiple "logical terminals".

The demonstration included security administrator support, system manager support and auditing.

Martin Marietta Information and Communications Systems

Martin Marietta has been using Gemini TCB products for over three years for their internal multilevel security development effort, and has developed the following demonstrations and capabilities:

Trusted Network Access Processor with Trusted Ethernet Interface

Trusted Terminal Gateway

Trusted File Transfer

The following projects are in development at this time:

Trusted End-to-End Protocol

Trusted Guards

Integration of Gemini Trusted Products with a Secure Local Area Network

CURRENT RESEARCH

In addition to supporting applications-oriented projects such as those described above, Gemini is involved in several projects oriented towards increasing the number and range of environments supported by the same underlying security kernel. The projects described below differ from routine product maintenance and enhancement because they are oriented toward expanding the base of customers interested in using trusted systems, primarily by providing standard system interfaces emulated using an unmodified underlying security kernel.

Oracle

Oracle, Incorporated, has undertaken an internal research and development effort to upgrade their relational DBMS product to Class C2 and to investigate a further upgrade to the B division. As part of this effort, it is expected that a prototype version of the C2 Oracle DBMS will be ported to the GEMSOS TCB. Later, a follow-on port of the B division prototype will be ported to the GEMSOS TCB. Oracle engineers have also been reviewing the design documentation prepared for the SeaView Secure Data Views project, an Air Force-sponsored effort awarded to SRI International and Gemini to design a Class A1 multi-level secure DBMS. The near-term design produced under this project features the utilization of an architecture similar to that which is expected to result from the Oracle port to provide enhanced security functionality.

If completed, this architectural approach will provide multilevel relational DBMS functionality to customers requiring a Class A1 level of assurance in the form of a conventional Oracle DBMS executing as a secure application on the GEMSOS TCB.

UNIX

Gemini is currently developing a UNIX emulation that will present the Unix kernel interface to application programs. The GEMSOS TCB has been designed from the onset to include those specific features needed to support an efficient UNIX kernel implementation. The UNIX kernel functions will execute in a ring less-privileged than the underlying TCB but tamperproof with respect to UNIX application programs. Because the UNIX kernel functions are less privileged than the TCB, they do not compromise the evaluation of the underlying TCB. Because they are more privileged than applications, the integrity of the UNIX kernel cannot be compromised by application programs, just as one would expect for a conventional UNIX-based system.

It might be noted that at Class B3 and above, the TCSEC requirement to make the TCB "minimal" precludes the competing architectural approach of lower classes of making the Unix kernel and TCB interfaces the same interface: many Unix kernel functions are not security critical and must be implemented outside of a Class B3 or A1 TCB.

MS-DOS

Gemini is currently developing an 80386-based virtual machine monitor and BIOS emulation that will allow selected standard MS-DOS applications to execute in the environment of a dedicated MS-DOS virtual machine. While this approach has some intrinsic limitations (MS-DOS applications that bypass BIOS will not execute correctly) the availability of executable applications is believed to be sufficiently broad, and to encompass a sufficiently wide range of functionality, that this is believed to represent a cost-effective approach to obtaining access to a broad commercial software base for Class A1 or B3 systems. The intended application of this capability would be to (in effect) make a DOS PC/AT workstation, supporting a set of selected DOS applications, available to the user of an MLS AT workstations, at whatever authorized session level the user negotiated. A change in session level would appear, in most respects, to the user as if a new dedicated PC/AT workstation was now in use. Volumes at the same or lower access classes can be shared.

SUMMARY

The Gemini product line is based upon the use of a single, high-performance, general-purpose security kernel designed to be evaluatable at Class A1. The security kernel has been commercially available for several years and is currently being used for a number of important government communication, command, control, and intelligence applications. It has also been found useful by a number of vendors in support of their internal programs to stay in the forefront of trusted systems technology. The usability of the security kernel for such purposes well in advance of the availability of an evaluated, complete Gemini TCB is a direct design consequence of the underlying product decision, made well before the kernel was started, to pursue a market strategy based upon an open and extensible architecture, serving a growing community of third-party and value-added customers. This decision led in turn to a design based upon a TCB composed of subsets with independently-evaluatable layers, the most-privileged of which is the security kernel itself.

In addition to allowing the security kernel to be independently implemented, tested, and marketed before completion of the remainder of the TCB, such a subsetted design allows customers to use unchanged, modify, or replace the remainder of the TCB as warranted by their needs, with a minimum impact on the magnitude of the re-evaluation task. We expect this subsetted architecture, along with the cost/performance advantage intrinsic to a multiple microprocessor capability, to be our major advantages as competing Class A1 and B3 systems emerge.

We hope to concretely demonstrate the additional potential of an extensible security architecture through some of the research efforts described above.

Our product plan includes a UNIX emulation implemented on top of the commercial TCB. At the current time, we also expect to port the Oracle DBMS directly to the TCB interface (not the Unix emulation interface), primarily so that the good performance characteristics of the TCB are made available to the Oracle DBMS.

REFERENCES

1. R. R. Schell, T. F. Tao, and M. Heckman, "Designing the GEMSOS Security Kernel for Security and Performance", **Proc. 8th DoD/NBS Computer Security Conference**, pp. 108-119, 1985.

2. W. R. Schockley and R. R. Schell, "TCB Subsets for Incremental Evaluation". In, **AIAA/ASIS/IEEE Third Aerospace Computer Security Conf.**, pp. 131-140, 1987.

3. **Department of Defense Trusted Computer System Evaluation Criteria.** Department of Defense, National Computer Security Center, Dec. 1985. DoD 5200.28-STD.

4. William R. Shockley. "Implementing the Clark/Wilson Integrity Policy Using Current Technology". In, **Proceedings of the 11th National Computer Security Conf.**, 1988.

5. W. E. Boebert and R. Y. Kain. "A Practical Alternative to Hierarchical Integrity Policies". In, **Proceedings of the 8th National Computer Security Conference**, 1985, pages 19-27.

6. M. Schaefer and R.R. Schell. "Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products". In **Proc. 1984 Symp. on Security and Privacy**, IEEE Computer Society, pages 41-49, 1984.

7. M. D Schroeder and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings". In **Third Symp. on Operating Systems Principles**, October 1971, Association for Computing Machinery, pages 42-54, 1971.

8. **Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria.** National Computer Security Center, Jul. 1987. NCSC-TG-005 Version-1.

9. C. Weissman, "Blacker: Security for the DDN. Examples of A1 Security Engineering Trades". In, **Proc. 1988 Symp. on Security and Privacy**, IEEE Computer Society, p 26, 1988.

10. Jon Fellows, et. al., "The Architecture of a Distributed Trusted Computing Base". In, **Proc. 10th National Computer Security Conf.**, pages 68-77, 1987.

11. W.R. Schockley, et al. "A Network of Trusted Systems". In, **AIAA/ASIS/IEEE Third Aerospace Computer Security Conf.**, pages 140-142, 1987.

12. C. Irvine, et al. "Genesis of a Secure Application: A Multilevel Secure Message Preparation Workstation Demonstration". Gemini Technical Report GCI-88-14-00, May 1988.

A SECURE SDS SOFTWARE LIBRARY

Sara Hadley, Frank Hellwig, Ken Rowe, CDR David Vaurio

National Security Agency
National Computer Security Center
Fort Meade, MD 20755

Abstract

A major challenge facing the Strategic Defense Initiative (SDI) program and the development of the Strategic Defense System (SDS) is the use and distribution of reusable software. The need for reusable software has clearly been established by the ever increasing cost of software versus the cost of hardware. This cost disproportion is magnified in a program with the scope of the SDS. A SDS Software Library will be created in which reusable software can be cataloged, accessed, and distributed. A key attribute of this library is security. The software in the SDS Software Library must be protected from unauthorized access and modification. This paper demonstrates the need for a secure SDS Software Library and the means by which this can be achieved.

Introduction

The Strategic Defense Initiative (SDI) program is an impetus to technology developments on a wide variety of fronts. Two of these fronts are computer hardware and software development. The SDI program is computationally intensive, requiring tomorrow's supercomputers today. This technological pace must be matched by equally intensive software development. The trend over the past decades has shown us that software technology always lags behind hardware technology. Total software costs are rapidly growing as machines become less expensive, and as we uncover more and more problem domains that demand an automated solution [1]. Second and third generation software is hosted on fourth generation machines. Furthermore, when new hardware demands new software solutions, old software is frequently "patched up." More often than not, systems fail in their promise to be extensible and maintainable. In response to this software crisis, the Department of Defense sponsored the development of the Ada programming language and with it, the true potential of reusable software.

The SDI program will make great use of reusable software. There is neither the time nor the money to develop a software system "from scratch" for each new project. In addition to developing new, reusable software, the SDI program will make great use of legacy software, especially in the initial phases of the program. The use of Ada is only a partial solution to the problem of software reuse. No amount of software, whether written in Ada or FORTRAN, will encourage reuse if it is not accessible. For code reuse to be attractive, the overall effort to reuse code must be less than the effort to create new code [2]. Reusable software must be organized in a central repository to which authorized software developers have ready access. It must be organized in such a manner as to provide rapid response to requests for reusable modules to meet the requirements of software developers. Otherwise, software developers will design expensive, single application systems rather than waste time and manpower scouring the countryside for a module that may or may not do the job. This requirement for a repository where reusable software modules are accessible to the SDI software development community has resulted in the Strategic Defense Initiative Organization (SDIO) mandating the establishment of a SDS Software Library at the National Test Facility. [3]

The purpose of this paper is to focus on the need and method of achieving a secure SDS Software Library. This will be addressed in the remaining four sections, each presenting the security issues in succeedingly finer levels of granularity. First, the requirements for a secure library is examined. Second, a conceptual model outlining the required operational capabilities is presented. Third, an implementation is suggested. Finally, the previous material is summarized.

Security

The success of the SDS Software Library can only be assured by the proper application of proven security measures. A failure to do this will result in a library where hostile agents can siphon national secrets without detection. The library will contain a concentrated data base of software revealing a great deal of the capabilities and vulnerabilities of the Strategic Defense System. This concentration of defense software in one location makes the SDS Software Library a high visibility target. The data tables that are included in many classified simulation software packages are a high motivator for illegal penetration into the library. Unauthorized access to the SDS Software Library can result in the compromise of information or corruption of software, thereby resulting in a severe impact to national security.

In a recent tutorial [4], a majority of papers addressed the design and structure of software for reusability. A lesser number were concerned with the actual implementation of a software library. Not a single paper addressed security.

One reason that security is not a principal concern of software library developers is that the principal beneficiaries of reusable software are the software developers themselves. The savings may be passed on to the buyer (i.e., the government), but the actual software development and savings through reusability are an "in house" issue.

If the particular software project is classified, the development is usually accomplished at the system high level where all users are cleared to the highest level of the data. The programmers and the development system are collocated adding to the amount of security which can be enforced through physical means. The situation in the SDS Software Library is quite different due to its handling of material of various classification levels and its distributed user base, therefore making a system high implementation inefficient.

The software stored in the library and the users of the library will span a wide range of classification levels. The flexibility necessary for the library to be responsive to all classification levels eliminates the option of system high operation. Since the SDS Software Library does not actually execute programs, the security treatment is different from that of other computer systems. The true problem is how to enforce security when such a large number of users have access to such a wide range of storage.

Conceptual Model

The SDS Software Library is a storage facility for reusable software serving a widely distributed network of users. The library enables the SDI software development community to efficiently produce complex software systems by providing access to an existing software base. The library is the central point of access to these reusable software modules. A conceptual model of the library is an enumeration of the services and functions the library must provide. The services and functions exist independently of the library's actual physical configuration. It is, however, difficult to create an abstract functional concept without first defining the physical components that constitute a software library. The following will briefly describe the physical elements of the library.

The core of the software library is the computer system, its software, and the associated storage system. The computer system provides on-line library services to the users by executing an application program referred to in this paper as the Software Library Management System (SLMS). The SLMS is the user interface to the information stored in the library and is the system by which the administrative personnel operate the library. The SLMS provides such services as cataloging, tracking, version control, and integrity verification of software items.

It is important to note that the library is not exclusively an electronic storage facility. As in a conventional library, information can be maintained in the form of printed material or microfilm. A user could request the mailing or in the case of extremely sensitive information, the transfer by courier of the requested material. This nonelectronic extension of the library divides the storage media into two groups: on-line and off-line. Material in on-line storage is accessible to the users through the SLMS. Material in off-line storage is transmitted through more traditional means.

The administrative personnel of the library serve two purposes. The first is in the computer system operating staff. The second function is the entry point for software submitted to the library.

So far, we have viewed the library as an information source. Prior to being a source, the library must acquire software items. This acquisition phase is a continuing process. The SLMS is the filter for information flowing from the library to the users. The filter for material being submitted to the library is the administrative personnel. The reasons for a nonautomated mechanism are both technical and philosophical. Current technology does not enable the creation of a system which can examine a piece of code, analyze it, assess its worth in terms of usefulness and reusability, and intelligently catalog it. The second reason involves the human element. The degree of confidence users would have in software obtained from the library is dubious if there is not some intuitive assurance that a human has at least reviewed the software for content and potential.

A conceptual model of the library is derived from a detailed look at the services and functions required of the library to operate smoothly and efficiently. This is formally expressed by a set of required operational capabilities. The required operational capabilities are a functional decomposition of the SDS Software Library. They state what services the library provides to its users and specifies the interaction between the users and the library. The required operational capabilities determine the system level requirements for the library. They are not an all inclusive listing of functions and services. However, the required operational capabilities must be comprehensive so that additional requirements support the composite system without compromising any individual operational capability or degrading one another. The following is a listing of the minimal required operational capabilities.

Fundamental Capabilities

Most computer operating systems enforce a relationship between the users and objects such as files and programs. These relationships are usually expressed as read, write, and execute capabilities. In essence, the SDS Software Library is a system with a large number of users and objects. As a top level specification, the fundamental access capabilities apply as follows:

Read: The library users and administrative personnel will have read access capabilities as specified by their individual access rights.

Write: Only the administrative personnel shall have write access capabilities as specified by their individual access rights.

Append: This is a limited write capability to allow the addition of information to an existing package. The SLMS will account for updates.

Execute: The SDS Software Library shall not have the capability to execute any software item stored in the library. The SDS Software Library is a software repository, not a software development center. The only software the library will execute is the SLMS.

Access Modes

The users of the SDS Software Library shall have two modes of access to the library: system and retrieve.

System: In this mode, the user communicates with the SLMS. A typical function in the system mode is accessing the SDS Software Library Catalog. This mode may however have access limitations (e.g., an unclassified user will not be able to scan classified summaries of classified items).

Retrieve: This mode grants the user direct read access to a software item. Notice that read access is a defacto retrieve mode since it is impossible to control a situation where a user at a remote terminal chooses to download the information appearing on his terminal.

Access Paths

The SDS Software Library shall support the following means through which users may access the library:

Local: Authorized users at the library facility should have direct access to the library.

Remote: Authorized users will be able to access the library via commercial telecommunication links.

Dedicated: The SDS Software Library shall support dedicated electronic communications links. In the case of classified material, these links will be encrypted.

Other: Authorized users may be able to communicate with the library via telephone, mail, courier, and other nonelectronic means.

Access Controls

In accordance with the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) [5], the library shall incorporate discretionary and mandatory access controls and labels for an A1 system. There will be some differences between the TCSEC and the implementation in the library due to the fact that the library users have no write capability and that the library is not capable of executing stored software items.

Accountability

The SDS Software Library shall incorporate mechanisms to enforce the identification, authentication, and audit requirements as specified for an A1 system by the TCSEC.

Integrity Controls

The SDS Software Library shall maintain software integrity. Integrity mechanisms ensure that the state of a software item is identical (i.e. has not been modified) to its state at a previous time. The proper use of access controls and integrity locks ensure that software is maintained and updated under strict control.

To ensure that data integrity is maintained, the access to that data must be controlled. The SDS Software Library access controls are taken from the TCSEC. David Clark and David Wilson, in an IEEE paper, "A Comparison of Commercial and Military Computer Security Policies," argue that the military security policy only protects information from unauthorized disclosure. If, however, a user is authorized to access a particular data item, there is no restriction on how that data can be manipulated [6]. The protection from unauthorized modification can be implemented in one of two ways. The first is to place integrity protection mechanisms between the user and the information. This protection would be in addition to existing access controls. The second is to eliminate the user's ability to modify information. The latter method is inherent in the read only functionality of the library.

Integrity locks determine if information has been modified. An integrity test can be performed to verify that the present state of a software item is unchanged from some previous state. Additionally, after a software item is distributed to one or more users, the same capability must exist at the remote location to verify that the state of that item matches the state of its parent in the library.

Storage

The SDS Software Library will be required to store software modules having different levels of classification (i.e. multilevel secure storage). Also, certain items may be under strict control independent of classification level. Items under less stringent control should have a wide availability while those under strict control must have very limited distribution.

Since the SDS Software Library is a software storage facility, not a software development center, all storage will appear to be of the write-once-read-many type.

Software Entry

Software items are entered into the SDS Software Library only by the administrative personnel. The submitter of the software item must provide the following:

Header: The header contains the authors, organization date, language, system, and system configuration on which and for which the software item was developed.

Pedigree: A clear and detailed development history of the module. This will specify the current version number as well as previous versions, independent of whether or not those previous version exist in the library. The pedigree shall also state the program for which the item was developed and the test and verification/validation history.

Functional Specification: A textual document detailing the precise function of the software item.

Interface Control Document (ICD): A formal document indicating all entry and exit points, data structures, data types and any other operational requirements necessary to execute the software item per its specifications.

Current technology does not permit the formal verification of software at the code level. The ability to automatically analyze software and determine if it contains trojan horses, viruses, or other malicious functionality is still years away. This leaves no other alternative but to distrust all software entered into the library. The saving grace of this bleak fact is the nonexecutable nature of the library. Programs containing a virus cannot propagate to other programs stored in the library since they will not be executed in the library. It is the user who assumes responsibility for the correct or incorrect functioning of a software item obtained from the library.

Distribution

Software distribution is the transmission of a software item to one or more authorized users via an approved access path. The library shall provide for trusted distribution of software.

Catalog

The library will maintain a catalog of all software items stored in the library. "The library shall contain procedures that help construct queries and evaluate the retrieved sample for potential reusability." [2]

Physical Security

The SDS Software Library central host computer system must be situated in a physically secure area. Protection must be offered to prevent unauthorized access to information and to prevent the malicious destruction of hardware, software, or other library elements in an attempt to deny service.

Implementation

In order for the SDS Software Library to simultaneously process information of different sensitivity levels the Department of Defense requires the library to be a trusted system. The TCSEC defines a trusted system as "A system that employs sufficient hardware and software integrity measures to allow its use for processing simultaneously a range of sensitive or classified information." [5] The SDS Software Library must be designed as a secure system while preserving the required functionality. What will be presented here is an informal implementation outline.

This implementation fulfills the operational requirements through the proper application of information security mechanisms. It will examine only the electronic portion of the library. The nonelectronic areas are addressed by existing policies for the handling of classified material.

At the center of the library is the computer system which controls all of the information between the users and the storage. No user has direct access to the library storage facility. All access is mediated by the central computer system. In addition to enforcing access controls, it incorporates other security relevant functions such as audit trails and user authentication. As stated previously, the fact that the library users and the information stored in the library span all classification levels mandates the central computer to be a trusted system.

The requirements for the type of trusted system are derived from the publication Guidance for Applying the TCSEC in Specific Environments [7]. This standard provides guidance to determine the minimum evaluation class required for a system in specific implementation. The evaluation class determination is based on three factors: the minimum use clearance, the maximum data sensitivity, and the type of system (i.e., open or closed). In the case of the SDS Software Library, the minimum user clearance is unclassified. We will assume the maximum data sensitivity to be top secret. The type of system to be used in the library will be considered a closed system. This means the library will not execute applications software from outside sources. The only application software which the library will actually execute is the SLMS. This will be developed by cleared personnel under a tight, configuration controlled environment. Applying the guidance standard to these factors results in a criteria class requirement of an A1 system.

The SLMS is the application package hosted on the trusted system. The SLMS is the interface through which the users and administrative personnel access and manage the library. Its functions include those library procedures not inherent in the operating system of the trusted computing base. These procedures include the cataloging, tracking, and overall control of the software items stored in the library. The security feature of the SLMS is the integrity locking of all information in the library. There is currently no standard algorithm or device available to perform an integrity lock. When an algorithm or device becomes available, it would be integrated into the software library by either software or hardware.

An A1 system affords the data separation necessary to allow an algorithm to be implemented directly within the SLMS. This procedure could append a cryptographic authentication code to all software items entered into the library. The alternative is a hardware sealing system in line between the computing system and the library storage facility. Any nonsealed item passing from the system to the storage is appended with an authentication code. When an item is transferred from the storage facility it must pass through the hardware system where an integrity test is performed. Any item failing the integrity test would be flagged and rendered unavailable until some corrective action is taken.

The final level of security to consider is the communication channels linking the user to the library. These links must be secure in order to transfer classified information. This requires the use of secure gateways to the library. One possibility is to locate the library within an existing classified data communications system. Sections of the National Test Facility provide this capability. Locating the SDS Software Library within the National Test Facility will provide the SDS Software Library with secure data communication.

Conclusion

In this brief treatment of a complex subject, we have stated the necessity for an SDS Software Library, cited its required operational capabilities, and shown the mandatory role that security must play to create a successful system. A software library is the only method by which the SDI program, or any program of such magnitude, can accomplish its challenging software development tasks. The notion of reusable software demands a central access facility. A software library must provide a broad range of services to entice software developers to make good use of reusable code. None of these services should be degraded unnecessarily by the proper incorporation of security.

The SDS Software Library must be a secure, trusted system to allow the library to handle software of various sensitivity levels. This places stringent requirements on the system as a whole spanning the areas of A1 level trusted mainframes to creating and standardizing secure and verifiable integrity locking mechanisms. There are many challenges to be met in achieving a secure SDS Software Library. Failure to commit to the security of the SDS Software Library will result in unusable, or even malicious, rather than reusable software.

References

[1] G. Booch, Software Engineering with Ada. The Benjamin/Cummings Publishing Company, Inc., 1983, p. 7.

[2] R. Prieto-Diaz, P. Freeman, "Classifying Software for Reusability," IEEE Software, vol. 4, no. 1, pp. 6-16, January 1987.

[3] Strategic Defense Initiative Organization, "Strategic Defense System Software Policy," p. 3, April 7, 1988.

[4] P. Freeman, Tutorial: Software Reusability. Computer Society Press of the IEEE, November 1987.

[5] Department of Defense, Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28 STD, December 1985.

[6] D. D. Clark, D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," IEEE Symposium on Security and Privacy, April 1987, pp. 184-194.

[7] Department of Defense, Department of Defense Computer Security Requirements--Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, CSC STD 003 85, June 1985.

INFOSEC IRAD AT MAGNAVOX:
THE TRUSTED MILITARY MESSAGE PROCESSOR (TRUMMP) &
THE MILITARY MESSAGE EMBEDDED EXECUTIVE [(ME2)]

Gregory King
Bill Smith

Magnavox Electronic Systems Company
Northern Virginia Systems Group
120 Ashburn Road
Ashburn, VA 22011

Abstract

The multilevel secure automated exchange of military message; has been the subject of much research over the past fifteen years. During the last decade, several attempts to implement military message systems with Bell and LaPadula security models have been characterized by security kernels with poor performance and a resulting security model which does not accurately describe the real behaviour of a military message system. Consequently, the Naval Research Lab (NRL) has developed, and is promoting, an alternative approach that includes an application based security model for military message systems [1]. These facts have prompted Magnavox to pursue the development of verifiable security kernel alternatives capable of enforcing application based security models.

Our resulting product is the Trusted Military Message Processor (TRUMMP) and the Military Message Embedded Executive [(ME2)]. The TRUMMP is the hardware base that provides the performance and isolation characteristics necessary for the (ME2). The (ME2) is a security kernel alternative that has as goals: enforcement of "configurable" application based security models, and real-time performance. At the heart of the (ME2)'s ability to provide enforcement of a "configurable" application based security model is a state machine architecture that provides for rigid domain separation and strongly typed data flows over secure connections.

INTRODUCTION

The Military Message Experiment (MME) was a joint research effort sponsored by the Navy, Defense Advanced Research Projects Agency (ARPA) and the Commander-in-Chief Pacific (CINCPAC) to produce and evaluate the feasibility of developing multilevel secure military message systems. That project produced SIGMA, an operational system that was used by military officers and staff personnel. Later evaluations of the system by the NRL demonstrated the serious deficiencies that arise when a military message system is implemented with a Bell and LaPadula model. For background, a brief summary of the NRL findings and their solutions are presented here. Our description of the NRL findings are taken from [1].

Reference [1] argues that a security model should enable users to understand how to operate the system effectively, implementors to understand what security controls to build, and certifiers to determine whether controls are consistent with directives and whether controls are correctly implemented. Reference [1] proceeds to describe three deficiencies of the Bell and LaPadula model that prevented the model from providing this guidance to SIGMA users, implementors, and certifiers. Those deficiencies, as described in [1], are as follows:

Prohibition of write-downs. The * (star) property prohibits the writing down of information to a lower classification level. However, under some circumstances this action would be secure for a military message system. It was assumed that user confirmation by SIGMA would prevent security violations when this action was needed but because so few understood the security policy (a phenomenon derived from the numerous exceptions forced by the nature of the application) users tended to always permit these actions without understanding why they had been questioned.

Absence of multilevel objects. The model recognizes only single-level objects when in reality, objects for a military message system are inherently multilevel. An example is the multiple paragraphs of a message, each of which has its own classification. By treating a multilevel object as a single level object, some information is treated as more classified than it really is.

No structure for application dependent security rules. The model contains no structure for application dependent rules. A military message system typically must enforce some security rules that are unique to its application. An example is a rule that allows only users with authority to invoke a release operation.

Reference [1] continues by stressing the deficiencies of approaches that attempt to fit an application on top of the general purpose Bell and LaPadula model. The need for application based security models, as opposed to Bell and LaPadula derivatives, is emphasized. Such a policy is defined for a military message system. Implementations are purposely omitted so that implementors may use current technologies [1]. Our product is one such method that may prove useful in the implementation of application based security models.

Magnavox is coordinating with the NSA/NCSC as to the certification of TRUMMP and which criteria is to be used. The evolution of our product suggests an interpretation of the TCSEC or TNI that is "unique" to the processing requirements of military message systems that contain multiple embedded military message processors. As a result, our security architecture group has prepared a technical report detailing issues associated with an interpretation of the TCSEC for this type of system [7]. In any event, using the context of the TNI. as a minimum, our goal is a component rating . M (Mandatory Access Control) implying the product to be applicable to the B1 through A1 divisions of the TCSEC. Verification issues are not addressed in this paper but we have a security verification and validation group that is working along with us to assess the security assurance aspects of our development [6].

This paper is organized into six major sections: (1) the functional and security requirements of military message systems; (2) an overview of attempts to implement these requirements with security kernels; (3) definition of a security kernel alternative; (4) implementation of this alternative in the Military Message Embedded Executive [(ME2)], including discussion of data and process security enforcement; (5) a description of the Trusted Military Message Processor (TRUMMP); and (6) our conclusions and future plans.

1. FUNCTIONAL AND SECURITY REQUIREMENTS

Systems that are used to process military messages are concerned with the handling of different message types. One type, the formal military message, is at the heart of Command, Control, Communications and Intelligence (C3I) systems. Nearly all military operations and policies are communicated through a formal military message [4]. Military standards which govern the format of these messages typically require a TO, FROM, INFO, DATE-TIME-GROUP, TEXT, SECURITY, and PRECEDENCE field for each message. A formal military message must be maintained for long periods of time and be capable of being quickly retrieved.

Another type of military message is the informal message. In contrast with formal messages, informal messages generally do not have the same storage and retrieval requirements.

The systems used to process these messages consist of both embedded processors and non-embedded processors. An embedded processor refers to those processors that do not contain a direct human-machine interface (HMI) but are still a vital processing component (e.g. a message switch). While most non-embedded processors are concerned with the unauthorized disclosure or unauthorized modification of information to users (data security), the embedded processor must also enforce what [3] refers to as process security.

For an embedded military message processor, a process security requirement might be related to the precedence of the message. In a military message, the precedence specifies the maximum delivery time for a particular type of message. Consequently, a process security requirement for a military message system might state that higher precedence message processing will preempt lower precedence message processing. Enforcement of this process security requirement by the embedded computer allows the system to respond to high priority traffic as quickly as possible. In a C3I environment enforcement of this process security requirement is as important as the security requirements related to the unauthorized disclosure and unauthorized modification of data.

Survivability is another fundamental process security requirement for a military message system. Each computer resource, embedded or non-embedded, must continue to function/recover in the presence of simultaneous jamming, destruction, and nuclear blackout. Ironically, it is in the face of these very conditions that the system will undoubtedly face the largest volume of formal message traffic. Performance under these stressful conditions becomes a process security requirement for the embedded computer resources. An important derivative of this process security requirement is denial of service protection. Denial of service protection ensures that no one process monopolizes resources so as to delay or prevent other system functions.

Finally, the objects of a military message system must reflect the multilevel nature of a military message. A military message is often composed of paragraphs of differing classification with the overall classification for the message equal to the highest classification of any part of the message. To treat the entire message as classified as the most sensitive portion causes some information to be treated as more classified than it really is. As a result, an information structure that is capable of representing the multilevel nature of a military message is required. Reference [1] refers to this type of information structure as a "container".

2. EXPERIENCE WITH SECURITY KERNELS

Until very recently the security kernel, a reference monitor implementation, has been promoted as an appropriate foundation for meeting nearly all security requirements. By enforcing a multilevel security policy, the security kernel creates an abstract machine upon which it is impossible for an application program to commit compromise. By restricting the role of security enforcement to a small and simple mechanism such as a security kernel, security verification is much more tenable.

In a military message environment, however, the first attempts to implement military message systems were faced with a practical problem related to the nature of the application. Although many of the required actions are commonly defined as secure, they violate the general purpose axioms of the security kernel (i.e. the *-property). An example is the classification of the message header information at security levels lower than that of the associated message text. To solve the problem, these systems relied on numerous trusted subjects which are effectively exceptions to the general purpose axioms.

Security kernel performance was an additional problem. Some kernels yielded performance as low as 10-25 percent of their non-trusted counterparts [2]. With good performance a critical requirement, security kernels did not yield adequate results. Yet another drawback of security kernels was their absence of attention to the process security requirements that are present in DoD embedded computers.

Thus, some alternatives for security enforcement in a military message system are: 1) develop a special purpose security kernel that will enforce a model tailored to military message systems, 2) use a Bell and LaPadula model with numerous trusted subjects, or 3) use a security kernel alternative [5]. The first alternative is extremely costly, and the second alternative yields a product with the problems that SIGMA experienced. As a result, TRUMMP uses a security kernel alternative, (ME2), pronounced as "ME TWO", to implement application security models specifically tailored to military message systems. Fundamental to our approach is the concept of a state machine architecture.

3. A SECURITY KERNEL ALTERNATIVE

The security kernel alternative that (ME2) will use to enforce an application security model is based on the concept of a network of communicating finite state machines (CFSM) that operate under the control of a State Machine Executive (SME) [5] . In our development the (ME2) is the state machine executive that provides domain concurrency, domain separation, inter-domain communication via message passing, as well as enforcement of application based data security and process security requirements. The general architecture is referred to as a State Machine Architecture (SMA).

3.1 Definitions

The following definitions are provided to establish the required terminology for discussion of the Military Message Embedded Executive (ME2).

Processing Nodes - The nodes of the graph in figure 1. These nodes represent a processing domain. In the context of a military message system a processing node might represent the processing domain responsible for displaying a message while another node might represent the processing associated with the network interface.

Connections - The arrows of the graph in figure 1. These arrows represent the communication channels between processing nodes. Connections may be external or internal. An internal connection refers to a connection that has a processing node for both its source and sink points. Connections that do not have a processing node for sink and source are external connections.

Source Station - The head of each arrow in figure 1. Data is sent from a source station to a sink station. A connection attached to a source station of a node is called a sink connection for that node.

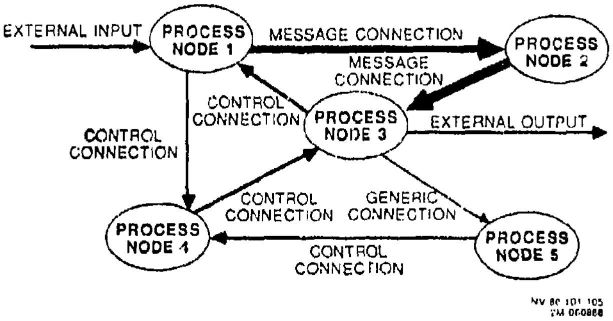


Figure 1. A State Machine Architecture

Sink Station - The tail of each arrow in figure 1. Data is received on a sink station from a source station. A connection attached to a sink station of a node is called a source connection for that node. Each sink station represents an unbounded FIFO queue, the tail of which acts as the sink point for source connections, while the head acts as a source station for sink connections.

Workstation - The heads of the queues associated with the sink stations of each processing node.

Container - A typed information structure that is categorized as single-level or multilevel based on its typed value. For example, a message container is a multilevel information structure while a control container is a single-level information structure.

Data Security Labels - Container labels which indicate the level of damage that might result if the container information is subjected to unauthorized disclosure or unauthorized modification.

Process Security Labels - Process labels and container labels that are used to enforce the process security requirements.

In the State Machine Architecture, the heart of security is the domain separation mechanism known as the State Machine Executive. In our implementation the State Machine Executive is the (ME2). The (ME2) is the foundation that supports the enforcement of application based security models. In addition to supporting application based security models, (ME2) allows an implementation that enjoys performance advantages over traditional security kernel implementations.

4. THE MILITARY MESSAGE EMBEDDED EXECUTIVE

The Military Message Embedded Executive [(ME2)], pronounced as "ME TWO", is an implementation of the state machine executive described in the previous section. This section describes the (ME2) implementation of each of the state machine architecture concepts described above.

4.1 Processing Nodes

Processing nodes, also called logical processors, are the components of the state machine architecture that refer to processing segments which operate independently of each other. In a military message system, one processing node might represent the processing associated with display of a message while another node might represent the processing associated with a network interface. The fundamental requirement that must be supported in an implementation of a processing node is domain isolation. The resources of each processing node must be isolated. In a single computer implementation, this means that the (ME2) must insure that the computer registers, flags, memory, and code of one node are physically inaccessible to another node. Not only must the (ME2) establish domain isolation but it must also ensure that once established that the processing node cannot alter the domain isolation characteristics.

Usually, hardware support for domain isolation and domain switching has been rare. Time consuming operating system software has traditionally been required to save the current processor domain in memory and to establish or load a new domain. The overhead associated with this single, but frequent, operation has been cited as the source of many performance problems in previous security kernel implementations.

Furthermore, the security assurances placed in the system are rooted in the correctness of the software used to implement the domain isolation principles. The more complicated this software, the less likely we can be certain that the software is implemented correctly, and the less likely that the software is worthy of our trust.

To combat these problems, the Intel 80286 microprocessor was chosen as the TRUMMP CPU. The Intel 80286 is unique for its ability to provide single instruction, firmware based, domain switching in as little as 16 microseconds. With the Inte. 80286, the (ME2) can perform an entire domain switching operation by executing a single privileged instruction designed to change the current task register (see figure 2).

As shown in figure 2, the Task State Segment (TSS) is a hardware recognizable, and hardware manipulated structure that defines the contents of all machine registers, flags,code, and memory that are physically visible at any one time. In the (ME2), each processing node has an associated TSS that completely defines th node's domain. Contained in the TSS is the address of another Intel 80286 data structure called the Local Descriptor Table (LDT). The LDT is the TSS component that defines the memory resources and code segments that are associated with a particular processing node. The TSS and LDT data structures are made invisible to applications in two ways. First, these critical data structures are contained exclusively within (ME2) LDTs. Second, their access is governed by a hardware protection mechanism that restricts access to "privilege level 0" code, i.e. (ME2) code.

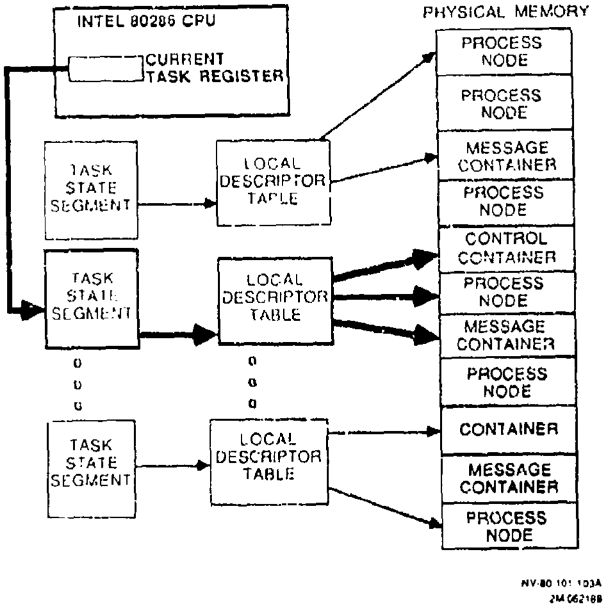


Figure 2. Intel 80286 Domain Isolation

4.2 Connections

Connections, also called virtual channels, are the communication mechanisms for process nodes. Controlled data flow over these connections is fundamental to the (ME2) enforcement of application based security models. Our implementation of connections requires the (ME2) to enforce strong typing rules over the connection, as well as data and process security rules. Verification that data flows are maintained according to typing rules is expected to be useful for verifying certain types of data integrity.

In the (ME2) implementation of connections, each connection has several associated attributes which describe the size of data that the connection accepts, the type (e.g. control, data, or message type) that the channel accepts, and whether or not the connection is internal or external. In addition the container has data and process security labels that must dominate the data and process labels of the sink before a transmission can occur. In the (ME2) implementation, the connections and their associated attributes are contained within a (ME2) data base that has been burned into Read Only Memory (ROM).

4.3 Source Stations, Sink Stations, and Workstations

At the (ME2) implementation level, source and sink stations are the heads and tails of data queues. Because the (ME2) provides first-in-first-out (FIFO) processing of queued data, a source station is always the queue head while sink stations are always the queue tail. Workstation is a more general term that refers to a data depository at which a queue element (the sink station, the source station, or some other element) is accessible.

In the (ME2), a workstation is implemented as a permanent entry within a process node's local descriptor table. Effectively, this permanently defines the virtual address of a workstation. To access data contained at the data depository (i.e. a particular queue element), application programs access the virtual address of the workstation. As an analogy, consider the workings of a photographic slide projector. A queue of slides, present in the slide projector, are viewed individually by the projector's movement of a slide through the lens path. In much the same way, the (ME2) must make individual containers visible at a fixed virtual address for viewing by process nodes.

The descriptor table based memory management of the Intel 80286 microprocessor provides the efficient mechanism for making a new queue element visible at the workstation. Because the local descriptor table is actually a table of physical addresses and access rights associated with each virtual address, the (ME2) can efficiently make a new queue element visible at a workstation by copying the physical address of the queue element to the local descriptor table (see figure 2). As a consequence, the elements of a queue might be at many non-contiguous physical memory segments, but are viewed, because of the actions of the (ME2), through a workstation at a fixed virtual address.

The (ME2) implements application based restrictions on workstation access according to the access rights declared for the queue. A workstation may be declared as read only, write only, read-write, or no access. (ME2) provides these access restrictions by writing the applicable value to the local descriptor table entry associated with the particular workstation.

In the (ME2) implementation, because each work station of a process node implies a unique local descriptor table entry for that process node, the number of work stations is physically limited by the maximum number of local descriptor table entries (approximately 8000). However, we expect the number of workstations to vary greatly based on the input and output requirements of the node. For example, a node responsible for message storage might segregate messages based on data and process security labels and thus require a large number of workstations, while another node might only require one workstation for an internal data structure.

4.4 Containers

As previously described, a container is broadly categorized as either a single-level or a multilevel information structure that is transmitted over a connection. Like the connections that they are transmitted over, containers have attributes that describe their size, type, data security, and process security attributes. Examples of container types include message containers, control containers, and data containers.

In the (ME2), containers are implemented according to user configurable allocations of physical memory based on container type and size. Consequently, a container is actually a contiguous segment of physical memory that contains data of the type declared for the container as well as data and process security labels. Potentially multilevel containers, such as message containers, contain a data and process security label for each individual unit of information.

The (ME2) distributes containers to workstations based on the value of a workstation attribute known as the auto-refill attribute. If a workstation is defined as an auto-refill workstation, the (ME2) will associate an empty container with the workstation at system initialization and at any future time that all workstation containers have been transferred over a connection. An empty container is defined as a container whose contents have been set to zero or some other innocuous value by the (ME2). At system initialization, all containers are empty. Later, used containers are emptied by the (ME2) and become available for reuse as they are sent from the application back to the (ME2).

4.5 Data Sensitivity Labels

In (ME2), data sensitivity labels are implemented with a secrecy component and an integrity component. The secrecy component of the label provides up to sixteen hierarchal levels and sixty-four non-hierarchal compartments within each level. Integrity types will, of course, be different among applications. Specific definition is dependent on a particular application based security model. Two integrity levels, high and low, are supported. As mentioned earlier, (ME2) enforcement of strong typing rules is also expected to be useful for verification of certain types of integrity.

The assignment of values to a data sensitivity label differs depending on whether the data sensitivity label is for a container or a workstation. The data sensitivity labels for workstations are static. They reside in the read-only portion of the (ME2) data base and result from a system design based on a state machine architecture. Because the system design is based on isolating data flows and processing, the set of possible data security labels that a workstation may accept is known pre-runtime.

On the other hand, because containers are reusable, the values for the data sensitivity labels of a container must be assigned as the container is used. Initially, at the time the (ME2) provides a container to an auto-refill workstation, the data sensitivity labels of the container have a value known as obscure. That is, the (ME2) and all process nodes recognize that the container in question has not been labelled. Certain process nodes may request the (ME2) to change the value of a data sensitivity label for a container from obscure to some other value. The (ME2) will perform such a label change if and only if such a change is consistent with the process security requirements. That is, just as only certain individuals have the authority to downgrade a military message, the (ME2) ensures that only certain process nodes may change a container label.

From a (ME2) implementation perspective, data sensitivity labels are always contained in memory that is accessible only to the (ME2). Consequently, data sensitivity labels are only changeable if the process node is trusted and the change is requested through the appropriate (ME2) service request.

4.6 Process Sensitivity Labels

Process security requirements ensure the prevention of undesirable and potentially catastrophic events in an embedded computer system. A general example is the requirement that a certain weapon system be fired only after a sequence of controlled events has been implemented. An example taken from military message system domains is the requirement that the processing of more important messages preempt the processing of less important messages or the requirement that among equal priority messages, messages should be processed in a first-in-first-out (FIFO) manner.

The Naval Research Lab suggests that one approach for enforcing process security requirements is the approach that is currently used for enforcing data security requirements. Specifically, the NRL suggests that to enforce process security, a system be structured into a set of functions that affect process security and a set that do not [3]. By verifying that the functions that have the capability to violate process security do not violate it (they are trusted), process security can be assured.

This approach is the one we have taken with the (ME2). As a result, the (ME2) is the function that enforces data security and process security. As mentioned, an important process security requirement is that the processing of more important messages preempt the processing of less important messages, and that among equal priority messages, messages are processed in a first-in-first-out (FIFO) manner. Just as the (ME2) provides sensitivity labels to enforce data security requirements, a similar label is used to designate the relative importance of a message so that process security requirements can be enforced. In the (ME2), this label is referred to as the container's "Necessity". Five necessity values are supported in the (ME2). The structure of these values is application configurable.

supported in the (ME2).

By assigning necessity labels to each container, containers may be segregated according to message necessity. This eliminates the need for a shared queue of multiple necessity levels. As with data security labels, the (ME2) uses the necessity labels to restrict the flow of containers over connections. The (ME2) will transfer a container over a connection only if the necessity label of the container dominates the necessity label of the sink. Consequently, the segregation of containers coupled with the (ME2) process node preemption based on the arrival of a container at an empty workstation provides assurance that the data that has arrived is of greater importance than current processing, and that the arrival will cause preemption. This enforces the process security requirement that higher priority message processing preempt lower priority message processing.

The other type of process security label that the (ME2) provides is a command label which lists all of the authorized (ME2) commands that a process node can request. An earlier example illustrated how the command label is used to prohibit all nodes, except the authorized nodes, from changing the data security labels of a container.

The final type of process security provided by the (ME2) is denial of service protection. Denial of service protection is provided through application configurable parameters for each process node and a TRUMMP interval timer.

4.7 The (ME2): A Summary

The Military Message Embedded Executive [(ME2)] is an executive which also contains a security enforcing foundation based on the data and process security requirements of military message systems. The key to security in the (ME2) is a state machine architecture which divides the system into process nodes. Information is stored in strongly typed containers which are transferred over connections to strongly typed workstations. The (ME2) implements traditional data security labels as well as process security labels. The deficiencies, as described in [1], of SIGMA, a security kernel for the Military Message Experiment (MME), do not exist. Specifically, (ME2) provides the capability for authorized downgrade, implements multilevel objects (containers), and provides a structure for implementation of application dependent security requirements.

5. THE TRUSTED MILITARY MESSAGE PROCESSOR

When complete, the Trusted Military Message Processor (TRUMMP), pronounced as "TRUMMP" as in playing Bridge, will be a militarized microcomputer specifically designed to support the Military Message Embedded Executive [(ME2)] and its applications. Design objectives include high performance, architectural features that directly support (ME2) functions, and minimum physical size and weight. Mil-Spec and ruggedized versions will be available. TEMPEST and HEMP requirements can be accommodated.

At the heart of the TRUMMP is an Intel 80286 microprocessor. The Intel 80286 is a high performance 16 bit microprocessor that provides on chip memory management, descriptor based segmented virtual addressing, and physical memory addressing to 16 megabytes. These attributes support the functional requirements of a military message processor. From an INFOSEC perspective, the Intel 80286 was chosen for its unique capability to provide the type of efficient hardware enforced domain isolation that is at the heart of the (ME2). Specifically, the Intel 80286 defines data structures that allow the microprocessor to perform firmware based domain switching in as little as 16 microseconds. The TRUMMP also contains interval timers, and interrupt controllers.

Because the usual application of the TRUMMP is intended to be as an embedded processor, particular attention to the expected operational environment is required. Figure 3 illustrates a sample environment. In the figure, the TRUMMP is the primary trusted computer resource responsible for sending and receiving military messages to and from the telecommunications network(s).

Figure 3 shows the expected subsystem interfaces that the TRUMMP accommodates. To meet performance objectives and to simply the external interfaces to the TRUMMP and (ME2), as figure 4

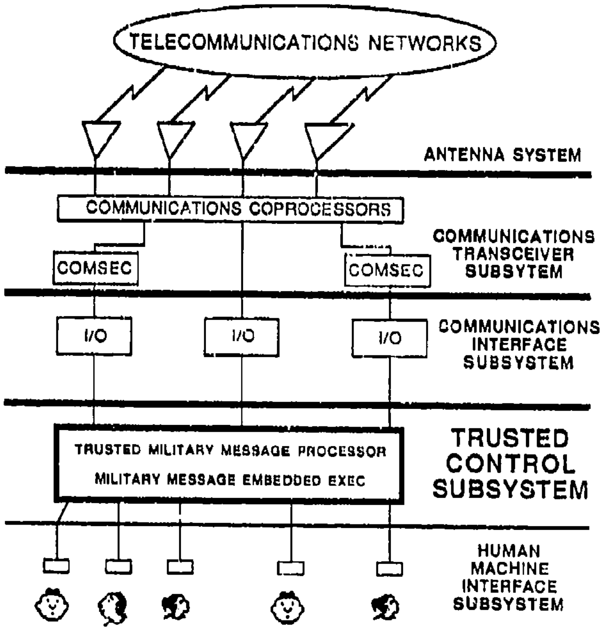

TELECOMMUNICATIONS NETWORKS
ANTENNA SYSTEM
COMMUNICATIONS COPROCESSORS
COMSEC
COMSEC
COMMUNICATIONS TRANSCEIVER SUBSYTEM
I/O
I/O
I/O
COMMUNICATIONS INTERFACE SUBSYSTEM
TRUSTED MILITARY MESSAGE PROCESSOR
MILITARY MESSAGE EMBEDDED EXEC
TRUSTED CONTROL SUBSYSTEM
HUMAN MACHINE INTERFACE SUBSYSTEM

Figure 3. Embedding the TRUMM

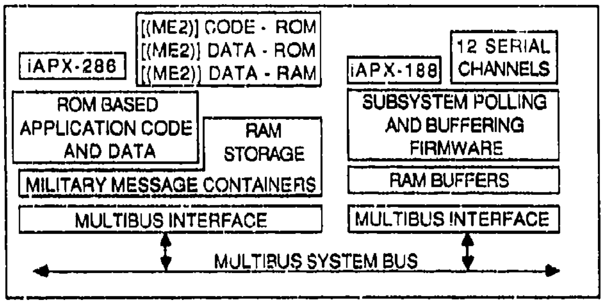


Figure 4. TRUMMP Components

shows, an iAPX-188 component is responsible for polling and buffering data to and from each of the serial channels. The iAPX-188 communicates with the TRUMMP and the (ME2) over the Multibus system bus.

6. CONCLUSIONS and FUTURE PLANS

This paper has described the progress of Magnavox research into the multilevel secure automated exchange of military messages. This work represents new approaches to "designed in security" that are not based on the security kernel and Bell/LaPadula model approaches that have dominated military message systems and the industry for the past fifteen years. Instead, the approach is based on the concept of a set of communicating finite state machines. The (ME2) is an efficient implementation of this concept that corrects the security kernel and Bell/LaPadula model deficiencies that have been cited for military message systems through frequent use of these traditional techniques.

Beyond its security kernel alternative, we believe the (ME2) is additionally unique for its attention to the process security requirements [3] of embedded computers. With the annual maintenance and development costs of DoD embedded computer resources expected to exceed $ 40 billion by 1990 [3], we believe that its time to understand and enforce the security requirements that are unique to these computers. Security in these systems is more than protecting files from users. In the (ME2) development, we have begun to address these requirements. In referring to these requirements, we have used the same terminology as the Naval Research Lab, process security [3].

We are currently testing our concepts with (ME2) and TRUMMP prototypes in our System Security Engineering lab. Our next major milestone is the evaluation of our development against either TNI, TCSEC, or some other standard [7]. Beyond these goals, we expect to address verification issues in a future report [6], and to evaluate our concepts with different hardware and different application domains.

References

[1] Landwehr, C. E., Heitmeyer, C. L., and McLean. J., Naval Research Laboratory. "A Security Model for Military Message Systems," ACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984, pages 198-222.

[2] Landwehr, C. E., Naval Research Laboratory. "The Best Available Technologies for Computer Security," IEEE Computer, July 1983, pages 86-100.

[3] Froscher, J. N., and Carroll, J. M., Naval Research Laboratory. "Security Requirements of Navy Embedded Computers," Technical Memorandum, 17 April 1984.

[4] Heitmeyer, C. L., and Wilson, S. H., "Military Message Systems: Current Status and Future Directions," IEEE Transactions on Communications, Vol 28, No. 9, pp. 1645-1654, Sept. 1980.

[5] Snow, D. W., "State Machine Architectures: An Alternative to Security Kernels for Embedded Systems", Magnavox Electronic Systems Company, Ashburn, VA, 1985.

[6] Smith, B.; Lindsay, K.; Reese, C.; and Crane, B., Magnavox Technical Report TR MX-NVSG-605/SSE.8801, "A Description of a Formal Validation and Verification (FVV) Process", Magnavox Electronic Systems Company, Ashburn, VA, February 1988.

[7] Ladd, S., "Criteria Extension For A Trusted Military Communications System Using Embedded Processors", Magnavox Technical Report, TR MX-NVSG-605/SSE.8804, Magnavox Electronic Systems Company, Ashburn, VA, February 1988.

SENSITIVITY LABELS AND SECURITY PROFILES

John C. Williams and Merlyn L. Day

Ford Aerospace Corporation
Henry Ford II Drive
San Jose, California 95161 - 9041

Abstract. In computer systems designed for high levels of security assurance, sensitivity labels are used to protect sensitive data. As multilevel secure distributed computer systems increasingly replace sensitive hardcopy documents with softcopies and automate their processing, the more comprehensive mechanism of a ***security profile*** will be needed. We illustrate the problems that ***security profiles*** will address with three popular but rarely valid assumptions about labels: 1) labels on data (classifications) and labels on people or processes (clearances) are drawn from the same partially ordered set, 2) labels are relatively simple, having only an hierarchical part and a non-hierarchical part, and 3) sensitivity labels and their associated rules suffice to insure correct handling of the data so labeled. Practitioners in the computer security community are becoming aware of the inapplicability of these assumptions outside defined contexts. We seek to increase that awareness and to explore the implications for future more ambitious systems.

1. Overview and Definition

1.1 Introduction. Institutions and individuals have a need to protect sensitive information from unauthorized disclosure, modification, degradation, destruction, and misuse, while, at the same time, allowing those who have been identified as trustworthy and are so authorized, to read, write, manipulate, store, retrieve, and selectively transmit or share such information according to their legitimate needs to do so.[1]

In some contexts, documents containing sensitive information may be identified as such by simply attaching warning statements. A more elaborate scheme is to classify information according to sensitivity and to "classify" personnel according to trustworthiness. Only the more trustworthy personnel with an identified ***need to know*** are allowed access to the more sensitive information.

In current government and some commercial computer systems, the protection mechanism may involve associating with data, sensitivity labels that indicate the degree (hierarchical level) of sensitivity and the category(ies), compartment(s), or "subject area(s)" of the corresponding information. One sensitivity label is said to dominate another if its hierarchical component is greater than or equal to the hierarchical component of the other and its set of compartments contains the set of compartments of the other [3, 4]. Typically, in the absence of ***privilege***, a secure computer system allows data to flow from one subject/object to another only if the sensitivity label on the recipient dominates the sensitivity label on the sender. (This is a necessary, not a sufficient, condition.) For example, Secret-A/B dominates Confidential-B, while neither is comparable with Secret-B/C. An ***untrusted*** (normal) ***process*** running at the first level could receive data from an untrusted process running at the second level, but not ***vice versa.*** Neither process could send or receive from an untrusted process at the third level. An "object" (e.g., file) containing data at each of these levels would have to have an overall classification of at least Secret-A/B/C.[2] In other words, the classification of a document must be at least the ***least upper bound*** of the classifications of data in the document.

The hierarchy scheme is usually associated with government classification systems (in some communities, "classified" = "government classified"), but it is also used by large corporations for their proprietary information. For example, IBM classifies their internal information as Public (unclassified), Internal Use Only, Confidential, Confidential Restricted, and Registered Confidential.[3] Most of the following is based on the government classification program, but the requirements are just as applicable to a commercial and/or proprietary program.[4], [5]

1.2 Markings. The classification of a document is not the only security-related information needed to handle a document properly. When marking hardcopy documents, the applicable classification labels with modifiers must appear on the pages and portions, and all other markings may be placed on a "title page", which must also be marked with the overall classification label. The other markings (notations and statements) are usually referred to as the title page markings. These include warning notices, information about control channels for dissemination, classification authority, declassification date, etc.

Of course, the title page should also have a title and other means (Document Number) of identifying the document. The title, itself, must be labeled with a classification label and, wherever possible, be unclassified.

The ***Industrial Security Manual for Safeguarding Classified Information*** **(ISM)[6] requires the following**

[1] Although the goal of information security is to protect ***information***, this is usually accomplished by protecting ***data***, whose correct or approximate interpretation yields the information deemed sensitive, and this in turn may be achieved by controlling ***signals*** that transmit, or patterns that convey, the data. It is customary to blur these distinctions.

[2] Unfortunately, in computer security, assuming an "object" is only a passive object is problematic. Given a string of bits in a computer, it is difficult, at best, to determine whether that string is potentially data or process or both. In a Bell-and-La-Padula-like computer security model [2], an executable "object", when executing, is presumably a subject, or at least a surrogate thereof, and thus, as a string of bits, is both data and process [12, 13].

[3] These are the hierarchical levels suggested in "Good Security Practices for Information Ownership and Classification," G360-2705-0, IBM, Nov. 1986. We believe these suggestions are based on IBM's practice.

[4] We also consider primarily sensitivity to disclosure rather than to destruction or modification.

[5] The emphasis in information security programs is usually placed on the upper levels of a hierarchical system, as is appropriate. However, there is a large amount of information that is not identified by any classification scheme that still requires protection because of some legal, business, or ethical requirement.

[6] DoD 5220.22-M, September 1987

markings "for all classified information, regardless of the form in which it appears":[7]

1. Identification Markings.
   a. Name and address of facility responsible for preparation of material.
   b. Date of preparation.
2. Overall Markings. (Classification of document.)
3. Page Markings.
4. Component Markings. (Each appendix, attachment, etc.)
5. Portion Markings. (Section, paragraph, illustration, photograph, figure, graph, etc.)
6. Subject and Title Markings.
7. Downgrading/Declassification and "Classified by" Markings.
8. Additional Markings. (if applicable)
   a. RESTRICTED DATA Notation.
   b. FORMERLY RESTRICTED DATA Notation.
   c. INTELLIGENCE SOURCES OR METHODS Notation.
   d. DISSEMINATION AND REPRODUCTION NOTICES.
   e. FOREIGN GOVERNMENT INFORMATION.
   f. THIS DOCUMENT CONTAINS NATO INFORMATION.

For our purposes, it is helpful to reorganize the above as:

A. Classification Labels (2-6 above)

B. Notations

  1. Warning Notices

  2. Instructions

  3. Control Channels (Dissemination)

C. Authority Statements

  1. Classification Statements (Classified by · · ·)

  2. Declassification Statements (Regrade/downgrade/declassify to · · · by <date/event>)

D. Ownership (Must approve dissemination or justify classification)

1.3 Security Profiles. Sensitivity labels provide at most, the information in **A** above. But no piece of information in a Marking is superfluous. For every component of a Marking, there is some action that may be taken on or with the hardcopy document that requires that component (possibly in conjunction with others) in order to maintain security and ultimately to prevent unauthorized disclosure. (We provide examples below.) Automating these actions in a secure computer system will, therefore, require more information than is contained in sensitivity labels. This information, in a computer system, we refer to as a ***security profile***.[8] (The reader is warned that where we use the term "security label", we always mean a complete security label, i.e., a ***security profile***, not a sensitivity label.)

Definition. The ***security profile*** of a softcopy document is the softcopy equivalent of the complete set of Markings for the corresponding hardcopy document.[9]

1.4 Problems with Hierarchical Levels. For classified information, possible U.S. hierarchical levels are: Unclassified < Confidential < Secret < Top Secret. Possible non-U.S. hierarchical levels are: Unclassified < Restricted < Confidential < Secret < Top Secret. The U.S. translation of "Restricted" is typically "Handle as Confidential" (label and clearance required for [read] access) even though the physical storage requirements may be more akin to FOUO (see below). Some countries have only two levels: Secret < Top Secret.[10] Therefore, any U.S. Confidential document (in whatever medium) provided to those countries might have to be upgraded (in their possession or access) to an hierarchical level (their Secret) equivalent to (U.S.) Secret. Of course, it would also retain its U.S. Confidential label. The actual equivalences and translations of labels is per the security agreements between the countries involved. Automating this may require a Multinet Gateway (MNG), a Trusted Network Interface (TNI), or Trusted Computing Base (TCB) to "know" such security agreements.

A label that is not really part of the U.S. classification scheme is "For Official Use Only". Abbreviated as "(FOUO)" for portion (paragraph) marking, FOUO is used for information that is not classified but still requires some degree of protection: it's for official business, not for general distribution or publication.[11, 12] Some documents specify that unclassified extracts must be marked "For Official Use Only". An unclassified document that contains portions extracted from a classified COMSEC document, where all portions extracted were marked "(U)", must be marked "For Official Use Only", while the extracted portions must be marked "(FOUO)".[13] (See Fig. 1.) In a secure automated information system (AIS), this means that extracting a labeled portion may require changing the label of that portion when it is imported into another file and changing the ***security profile*** of the receiving file, even when the receiving file remains "unclassified". Since, by definition, the ***security profile*** of a document includes all markings, the insertion of a labeled portion constitutes a change in the ***security profile***; the point

[7] *Ibid.*, II-b., p. 52.

[8] Woodward [14, 15] exploits the dual nature of sensitivity labels: (1) sensitivity indicator and (2) Mandatory Access Control Label. As separate labels, the latter must dominate the former. Both, however, are "sensitivity labels", not ***security profiles***.

[9] This definition assumes that inherent in any portion marking is an indication of what portion the marking applies to.

[10] E.g., Finland. Sweden has the one word "Hemlig" (Secret), which may be bounded by a double red border (Top Secret). Haiti has only Confidential and Secret.

[11] Some agencies just say "Official Use Only" and "(OUO)".

[12] FOUO could be used as an added label for classified information, but such use would be superfluous: classified information is, by definition, for official use only, i.e., need-to-know.

[13] From a practical standpoint, FOUO could be positioned between Unclassified and Confidential in the U.S. hierarchy; however, it's more a subset of Unclassified than a separate level.

here is that the overall document marking must change as well. As we shall see, such extraction-import problems tend to be more difficult and, *ipso facto*, more serious, with higher levels of classification.

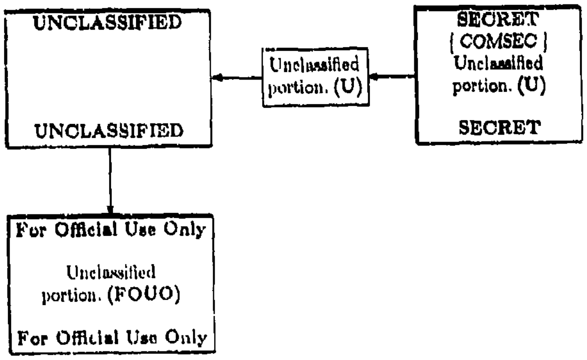


Figure 1. Receiving file remains unclassified, but markings on imported portion *and* on the file must change.

Also, declassified information, marked or unmarked, must still be protected; it's not for public release, unless so authorized by the agency responsible for it. By unmarked, we mean it is not apparent that it was once classified. If it is unclassified, but not releasable to the public, there obviously needs to be some indication: e.g., "For Official Use Only" or "Limited Distribution".

2. Erroneous Assumptions.

One way to illustrate some of the problems that ***security profiles*** will address is to consider three popular but rarely valid assumptions about labels. We refer to these assumptions as ELA 1, 2, and 3. "ELA" (pronounced "Ella") = "Erroneous Label Assumption".

Erroneous Label Assumptions:

(ELA 1) **Sensitivity labels on data (classification) and on users and their processes (clearances) are drawn from the *same* partially ordered set;**

(ELA 2) **Security labels are relatively *simple*, having only an hierarchical part and a non-hierarchical part (partially ordered by set inclusion);**

(ELA 3) **Sensitivity labels and their associated rules *suffice* to insure correct handling of the data so labeled.**

Whether these ELAs ***are*** erroneous or rather, if you will, ***how erroneous***, depends on context, i.e., the demands placed upon a given AIS. We are ***not*** saying that the computer security community as a whole is unaware of problems with these assumptions. Rather, we hope to increase that awareness and believe this is essential as the demands on secure AIS increase. Finally, the above assumptions, even when erroneous, are sometimes conceptually useful simplifications, when focusing on other security mechanisms. But not even Dorothy could remain forever in Kansas. We propose, instead, to tell the truth and nothing but the truth.

Notice we do not propose to tell the whole truth. An exhaustive treatise on the way ***security profiles*** and clearances really work would be longer than the Greater Armageddon phone book and duller than tofu. Such a treatise might also be classified. Nevertheless, we provide sufficiently detailed examples to bring home the most outstanding and interesting (not to say amazing) ways in which each of the three assumptions above fails, singularly and collectively.

Not surprisingly, the three ELAs are related. After all, they provide three views on why secure AIS's will need to use ***security profiles***. Thus, if a real security label, one that suffices to insure correct handling of data so labeled (in other words, a ***security profile***), is complex, contrary to ELA 2, then the simpler mechanism of a sensitivity label does not suffice, contrary to ELA 3.

2.1 (ELA 1) Sensitivity labels on data (classification) and on users and their processes (clearances) are drawn from the *same* partially ordered set.

2.1.1 Clearance & Sensitivity Labels. The degree of trust and degree of sensitivity markings are usually expressed, within an hierarchical scheme, with the same labels: Clearance Label = Classification Label. E.g., a Top Secret clearance is authorized access to Top Secret information. Because of the hierarchy scheme, the higher clearance level can also access (read) lower classification levels, but the lower clearance level cannot access (read) higher classification levels.

2.1.2 Clearance Labels. However, life would be dull without exceptions. Clearance Levels can be modified by labels that have no equivalent classification labels. For example, a clearance can be modified as an interim clearance. (We think you're OK, and you can have some limited access, but not to the good stuff until we finish checking you out.)

A person with an interim Secret clearance cannot access information classified, under the authority of the Atomic Energy Act, as Restricted Data (RD) or Formerly Restricted Data (FRD), but a person with either a Secret or interim Top Secret clearance can access both types of information at the Secret Level or lower [7]. DOE 5631.2 (Chapter I, Section 8) permits a very restricted form of interim access authorization. According to a reliable DOE source, DOE does not recognize interim clearances. But the DoD does. Hence, it appears that a person with an interim TS clearance might be granted access to S-CNWDI (a subset of RD) by the DoD. Whether this actually happens we cannot say. The clear and important point here is that usually no label on a classified document specifies whether a person with an interim clearance can access the document; i.e., there is no "interim" classification label corresponding to an interim clearance. There might be a notation that prohibits interim access, however. (See Section 2.3.2 and the Appendix.)

Similarly, persons with contractor-generated confidential clearances cannot have access to classified foreign government information, RD, FRD, or ACDA (Arms Control and Disarmament Agency) classified information. Again, there is no contractor-generated confidential classification level. These situations are handled by other administrative procedures: just learn the rules—unless "you" are a computer system.

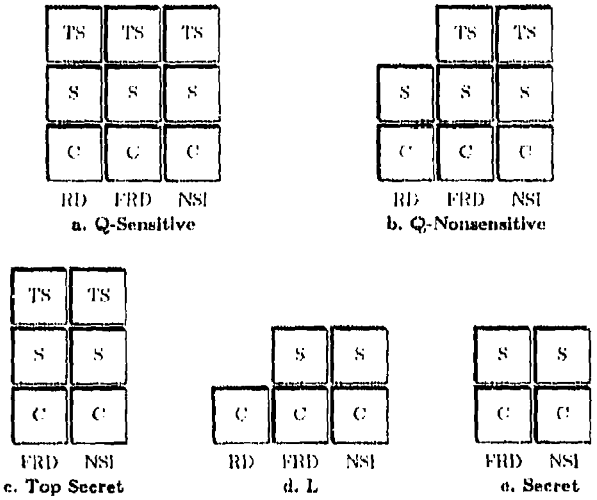
The new Limited Access Authorizations (LAAs) that are now issued to immigrant aliens and foreign nationals impose very strict access and need-to-know limitations. An individual's access under an LAA must be determined on a case-by-case, almost document-by-document, basis. In effect, there are no clearances for non-(U.S.) citizens, only access authorizations for specific purposes. A noncitizen contractor employee could require government authorization for access to specific information, as determined by need-to-know based on job requirements, for a specific contract only.

2.1.3 DOE Example. The Department of Energy (DOE) uses classification labels that are not identical with clearance labels.[14] From the following table, clearances and classifications clearly differ at least in name.

Table 1. Source: DOE 5631.2/11-13-80.

Clearance	Highest Classification (Read) Accesses Permitted
Q-sensitive	Top Secret RD, FRD, & NSI
Q-nonsensitive	Top Secret FRD & NSI Secret RD
Top Secret	Top Secret NSI & FRD
L	Secret NSI & FRD Confidential RD
Secret	Secret NSI & FRD
Q(X)	Secret RD (as specified in the access permit)
L(X)	Confidential RD (as specified in the access permit)

We illustrate the accesses permitted with various clearances in figure 2. Of the three clearances – Q-Sensitive, Top Secret, and Secret – each can be described in the usual way by a level and a set of compartments. Two clearances – Q-nonsensitive and L – however, cannot be so described. Thus clearances and classifications differ more than in name.



a. Q-Sensitive

b. Q-Nonsensitive

c. Top Secret

d. L

e. Secret

Figure 2. Some DOE clearances are *not* just a level with compartments.

Neither the ISM nor DOE 5631.2 is transparent. According to our DOE source, the distinction between QS and QN is largely irrelevant. Most people neither know nor care which Q clearance they have. The difference is not in the background investigation done, but who did it: an Office of Personnel Management (OPM) investigation can yield a QN clearance; a QS clearance requires the FBI. Since the Walker case, the situation is even simpler: DOE Branch Chiefs and above get QS; everyone else gets QN. (We infer that Branch Chiefs and above have had FBI investigations.) The operational difference between QS and QN is not whether TS access is granted, but how often. (Imagine trying to automate this.)

Consistent with Table 1, we have also been assured that the difference between QS and QN is unimportant because "DOE doesn't generate (small grain of salt here) TS-RD." Any TS-RD document is almost surely a compilation of RD (Secret or below) with data that was already TS for some other reason. The portion markings make clear how the amalgamation was formed. DOE does not do portion marking of strictly RD documents.

The DOE has to distinguish between RD/FRD information classified under the authority of the Atomic Energy Act and NSI (National Security Information) classified under Executive Orders. You figure it out—unless "you" are a computer system. DOE also has to protect Unclassified Controlled Nuclear Information (UCNI). The nonparallelism between clearance and classification labels is caused in part by the overlapping, nonhierarchical nature of these classification requirements – which brings us to the next ELA.

2.2 (ELA 2) *Security labels* are relatively *simple* with an hierarchical part and a non-hierarchical part (partially ordered by set inclusion).

2.2.1 Nonhierarchical Systems. Unclassified National Security-Related (UNS-R) (pronounced "unser") information is a broad category of information that is considered to be unclassified but sensitive and, in the interests of national security, should be protected, especially when being processed in AIS or telecommunications systems that are vulnerable to monitoring by unfriendly interests. There is no label for UNS-R information – but perhaps there should be in computer systems. Such information, of course, can have other labels or notations such as Proprietary, FOUO, etc. – some of which could be hierarchical. Large proprietary or private systems could be considered equivalent to "UNS-R Systems".

Level Modifiers. In many cases, the level label must be modified to show owner or added sensitivity of the information:

2.2.2 Ownership Modifiers. In friendly foreign relations, whereby countries or international pact organizations exchange classified information, the document must identify the owner of the information. For example, if the U.S. should be provided with a United Kingdom Secret document, the level label should indicate: UK-Secret. NATO documents, for another example, must be labeled COSMIC

[14] Practitioners, and even more so, researchers, in DoD secure computer systems should not object to a DOE example. Members of one community need to be aware of the practices of the other. Moreover, we believe the trend is toward multilevel secure distributed systems of once isolated communities that will communicate with each other through Multinet Gateways.

Top Secret, NATO Secret, NATO Confidential, or NATO Restricted, as applicable. Notice that modifiers may be inconsistent: Top Secret is modified by "COSMIC", not "NATO".[15] It's a little like learning an irregular verb --unless "you" are a computer system.

A modified level does not necessarily transfer when a document portion is copied into another document. For example, if NATO Secret information is copied into a U.S. document, the U.S. is obliged to classify the document Secret (at least), but not NATO Secret. Instead, the U.S. document must bear the warning notice: "THIS DOCUMENT CONTAINS NATO INFORMATION"; and the portion containing the NATO information must be labeled "(NATO-S)". (See fig. 3.) To import a portion from one "file" into another, you learn the rules --unless "you" are a computer system. Such rules involve more than sensitivity --which is our final ELA.

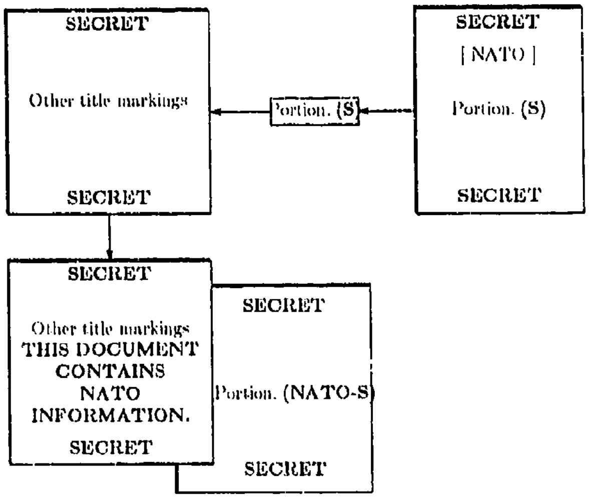


Figure 3. Importing Secret Portion from NATO File into U.S. Secret File.

2.3 (ELA 3) Sensitivity labels and their associated rules *suffice* to insure correct handling of the data so labeled.

2.3.1 Sensitivity Modifiers. As some of the previous examples illustrate, sensitivity labels by themselves do not suffice for a variety of reasons, such as the inclusion of various modifiers, document notations, and the interdependencies of ownership and classification authorities. Frequently, classification labels must be modified to show that the information is extra sensitive and requires additional protection.

In the COMSEC world, encryption keys are considered ultra-sensitive. Material containing a key must be labeled with the appropriate classification level ***modified*** by a CRYPTO label:' Top Secret Crypto, Secret Crypto, or Confidential Crypto. This requirement even extends to unclassified material: Unclassified Crypto and For Official Use Only Crypto. And, of course, the requirement applies to all marking levels: document, page, and portion; e.g., "(U-Crypto)".

Restricted Data (RD) is another label that modifies classification labels. Moreover, this label indicates both ownership ***and*** sensitivity. Restricted Data is applied to information classified under the Atomic Energy Act, and thus "owned" by DOE—even if it is also proprietary to and "owned" by a contractor. RD also indicates that the information is extra sensitive and requires additional access restrictions. The same is true for Formerly Restricted Data (FRD). RD and FRD modify not only document and page labels, but also portion labels: TS-RD, S-RD, C-RD, TS-FRD, S-FRD, and C-FRD. There is no unclassified RD or FRD. As previously mentioned, however, there is an Unclassified Controlled Nuclear Information (UCNI) label. Generally, RD is physics, and UCNI is facilities and operations. UCNI (Section 148 of the Atomic Energy Act) is a Congressional response to the fact that OUC provides no legal standing in resisting a Freedom of Information Act request. UCNI protects sensitive, but unclassified, information about facilities and operations that would previously have been marked OUO.

A subset of RD is Critical Nuclear Weapons Design Information (CNWDI). CNWDI is always labeled Top Secret Restricted Data, Secret Restricted Data, or Confidential Restricted Data. At the document or page level of marking, there is no ***label*** for CNWDI; there is a CNWDI ***notation*** required on the document.[16] (More on notations later.) However, CNWDI can be used as a label to modify clearance labels. A person can be cleared TS-CNWDI; i.e., has a TS clearance with a CNWDI access authorization (and briefed). There is a formal access authorization procedure that must be completed prior to a person's having access to CNWDI. Thus one might understandably infer the hierarchy: $FRD < RD < N$. In fact, in the subset sense, $FRD < RD$ and $N < RD$, but FRD and CNWDI are generally incomparable. According to our source, DOE transclassifies RD to FRD when it needs to be shared with foreign partners and transclassifies RD to CNWDI when it needs to be shared with the DoD. Since FRD does not preclude access by DoD (DoD has "tons" of FRD), we infer that CNWDI, like RD, precludes access by foreign nationals.

ATOMAL is another label associated with Restricted Data and Formerly Restricted Data. ATOMAL is an exclusive designation used by NATO to identify Restricted Data or Formerly Restricted Data information released by the U.S. to NATO. Therefore there has to be a method for translating RD or FRD to ATOMAL: TS-RD ⟷ COSMIC TS-ATOMAL; S-RD ⟷ NATO S-ATOMAL.

In message traffic, the "Unclassified" label can be modified by EFTO, which stands for Encrypted For Transmission Only (but can be stored in plain text). Apparently, you have to explain why unclassified information is taking up space in a classified environment. EFTO can also act as a reminder that the plain text could serve as a source for cryptanalysis.

[15] DIAM 65-19, item 3-07: "COSMIC is a caveat applied to NATO TOP SECRET documents. The caveat denotes requirements for specific procedures in handling and disseminating documents so marked." Our description characterizes practice.

[16] At the portion granularity (paragraphs, etc.), portion labels must be modified with the CNWDI label "(N)": (TS-RD)(N), (S-RD)(N), or (C-RD)(N). "N", however, appears to be an artifact of the ISM not used by the DOE.

Some other labels that may be used to modify classification labels include: NOCONTRACT,[17] PROPRIETARY,[18] and LIMDIS.[19]

Compartments usually have access labels that modify classification labels that modify classification labels. Special Access Programs (SAPs), Special Access Required (SAR) programs, and Sensitive Compartmented Information (SCI) programs usually have code words or other designators that modify the classification: Top Secret/codeword or Top Secret/codeword/codeword/ · · · , where each instantiation of "codeword" is distinct. Usually, these markings are either classified or, if unclassified, considered very sensitive, and protected accordingly. Hence, no samples.[20]

NOFORN is a label modifier that is sometimes used (e.g., Secret-NOFORN), and stands for No Foreign Nationals. When NOFORN is used, the document should be marked with some notation that specifies that access is limited to U.S. citizens and exceptions must be approved by such-and-such agency. Other possibilities are the exclusion, inclusion, or both, of a group of nationals. For inclusion, the group may consist of one country; e.g., NOCONTRACT/REL AUSTRALIA.[21]

The label modifier WNINTEL is sometimes used to label information subject to the warning notation "Warning Notice Intelligence Sources or Methods Involved." When used, this modifier is usually at the portion level (granularity); e.g., (S-WNINTEL).

The relationships between compartmented classification labels and clearance labels can be very complicated. (See Appendix.) Compartmentalization is consistent with need-to-know. It helps prevent all but the most trusted from getting the big picture. In the commercial world, compartments could be considered equivalent to separation-of-duties requirements.

2.3.2 Notations. Notations are document modifiers. They place some warning, restriction, or explanation on a document. Notations are not labels or label modifiers *per se*. There is no such thing as Secret/Notation. The notation must appear once on the document, usually on the cover and/or title page, and "merely" provides additional instructions on how the document is to be protected. (This poses problems for computer systems.) Some notations, however, are specifically required in addition to certain label modifiers. If data or information is extracted from a document to which such a notation applies, then the notation(s) must accompany the extraction. Some of the current notations required by the ISM are as follows:[22]

(a) Restricted Data requires the following notation on the containing document:

RESTRICTED DATA
This material contains RESTRICTED DATA
as defined in the Atomic Energy Act of 1954.
Unauthorized disclosure subject to
administrative and criminal sanctions.

(d) The following Notice that reproduction of any portion of a document is absolutely prohibited without permission may require knowing a chain of command—which might prove difficult for a computer system:

REPRODUCTION REQUIRES
APPROVAL OF ORIGINATOR
OR HIGHER GOVERNMENT AUTHORITY

(e) **FOREIGN GOVERNMENT INFORMATION.** Where appropriate, this marking on U.S. documents ensures that such information is *not* declassified *prematurely* or made accessible to nationals of a third country *without the consent* of the originator. Importing a portion that contains such information into a "file" that did not, poses for a computer system at least the two problems emphasized by the italics: correcting the declassification date (see below) and correcting the dissemination controls.

The notation for COMSEC material is contained in DoD 4220.22-S-1, a supplement to the ISM. The current requirements for COMSEC material include:

a. Keying material requires the caveat **CRYPTO**.

b. Otherwise, the Notation: **COMSEC Material—Access by Contractor Personnel Restricted to U.S. Citizens Holding Final Government Clearance.**

CRYPTO is a label; it modifies a classification. COMSEC is a Notation; thus there is no Secret/COMSEC, for example. Yet in an AIS, such a label may be needed. For clearances, COMSEC *is* a label (ELA 1). A contractor employee must have a Secret/COMSEC clearance (Secret clearance with COMSEC access authorization) to access Secret documents containing COMSEC information. A government employee currently would need only a Secret clearance and a need-to-know. (Will computer systems know what kind of employee each of us is?) COMSEC access also requires U.S. citizenship and a final clearance; no interim clearances, except an interim Top Secret, can have access to S- or C-COMSEC. COMSEC access authorization includes access to CRYPTO; there is no longer a CRYPTO label for modifying clearance labels (ELA 1 again).

A Notation required when information is believed to be, or should be, or it is believed it should be classified, is:

CLASSIFICATION DETERMINATION PENDING.
PROTECT AS THOUGH CLASSIFIED
[appropriate classification label].

[17] Not releasable to contractors/consultants no matter what their clearances.

[18] Proprietary: Government does not own the information. Approval must be obtained from owner for further dissemination.

[19] Limited Distribution: Distribution is limited to some specified control channel or program. LIMDIS should be accompanied by some notation that specifies what distribution is acceptable.

[20] Nor can we offer examples of codewords that are no longer used. To do so might result in this document's being classified. While we are on the subject of who-huh-what-label?, there exist compartments, that we can or could mention, that have always been unclassified and whose meaning we now can or could provide, although at one time their meanings were classified. But, while we can discuss such compartments and their meanings in an unclassified document, if we were to do so, we might not be permitted to indicate that their meanings were once classified. The fact that the meaning of a specific compartment has been declassified tends to be classified or, at least, sensitive.

[21] We are unaware of any one-country *exclusion* group. However, as an (hypothetical) example with (real) labels: NOCONTRACT/REL AUSTRALIA/ NEW ZEALAND/ UNITED KINGDOM could be upgraded to the more restrictive NOCONTRACT/REL AUSTRALIA/ UNITED KINGDOM. (Such an upgrade could be driven either by a change in the document's contents or by external events.)

[22] *DoD Industrial Security Manual for Safeguarding Classified Information*, DoD 5220.22-M, September 1987, paragraph 11.b.(8) ADDITIONAL MARKINGS. The Notations themselves are verbatim, the rest paraphrased from the ISM.

Once notated, the document must be protected at the designated classification level, but it does not have to have any other labels, although it could be marked with appropriate private or proprietary labels. Any apparent parallel between a pending classification and an interim clearance is purely illusory.

A Notation required on some documents (e.g., COMSEC) is: **Not releasable to the Defense Technical Information Center per DoD Instruction 3200.12.** (Will computer systems act on the basis of such Notations?)

2.3.3 Authority Statements. Classified documents must be marked with a statement that specifies what authority classified the documents. Usually this authority statement and the declassification statment are combined into one statement, but here we shall treat them separately.

2.3.3.1 Classification Authorities. Various components of the government are designated classification authorities, some by law (e.g., DOE under the Atomic Energy Act) and some by Executive Order.[23] There are Original Classification Authorities and Derivative Classification (Authorities). The latter is (are) required to respect original classification decisions and to carry forward any assigned, authorized markings.

Anyone who wants to know how many original classification authorities there are can read the Federal Register and then try to determine how many other officials have been delegated the authority. Or they can ask the Information Oversight Office in General Services Administration (GSA): In FY84, the number was 6,900.[24]

"Classified by" line. Original classification authorities are responsible for developing classification guidelines for the derivative classifiers, who actually produce most of the classified information FY84: 4% original, 96% derivative. These guidelines should specify the classification authority statement to be used on documents created under its authority. For contractors the statement should be provided via the Contract Security Classification Specification (DD Form 254). Each classified document (regardless of medium) is required to have a marking: **Classified by <something>,** where **something** could be the name of an agency, the name of a classification guide, multiple sources i.e., too many to list (but a record must be kept somewhere a potential problem for computer systems, especially distributed ones) a DD 254 for Contract <ID>, dated <date>, or, in the case of message traffic, nothing. Messages do not have to have a "Classified by" line; the sender, by default, is considered to be the classification authority for the message.

2.3.3.2 Ownership. Each classified document is required to be marked with the name and address of the originating agency or facility and the purpose (e.g., Contract number) for which it was generated. Ideally, especially in contractors, the document should indicate both the preparer and whom it was prepared for.

2.3.3.3 Control Channels. When applicable, classification authorities may designate certain control channels for the distribution of their information. When deemed appropriate, documents in these channels may be required to be labeled with a statement specifying the channel required, such as **Handle via XYZ Control Channels Only.** Control channels can be implicit: some COMSEC material is controlled through the COMSEC Material Control System, but this is usually not explicitly labeled as a requirement on the material. (It may need to be explicitly labeled in a computer system.)

2.3.3.4 Accreditation Authorities. Accreditation authorities should not be confused with classification authorities. Classification authorities determine what is classified; accreditation authorities (accreditors) determine whether a particular system can process classified information and at what level. Once a system is accredited, the accreditor is responsible for ensuring that any classified information processed is appropriately protected and for issuing the appropriate security rules for operating the system. A system can be accredited by more than one authority.

Some accreditors may be responsible for accrediting all systems that process a particular category of information; such accreditors may also be the original classification authority for the information.[25]

2.3.3.5 Declassification Statements. All classified material should be marked with downgrading and declassification instructions, as appropriate.

Downgrade to <level> on <date or event>.
Declassify on <date or event>.

Abbreviations include: DNG/S/<date or event> and DECL <date or event>.

If there is no date or event, the notation "Originating Agency's Determination Required" or **OADR** should be used. Importing from one "file" into another may pose problems for a computer system if the declassification statements differ. For documents derived from or based on multiple sources, the new documents must be marked with the most restrictive determination.

When material is downgraded or declassified, the classification labels and other markings will be changed as appropriate. Material that has been declassified must still be provided some degree of protection. Declassification does *not* mean releasable to the public. Public release is a separate determination.

Another marking required on classified documents is the Date of Origination. This date obviously forms a base line for any downgrading or declassification.

3. Interdependencies of Security Label Components

Relations among security label (profile) components (figure 4) can be implicit or explicit. Implicit: Change in one item's label can cause a change in another item administratively but without a change in label. Explicit: Change in one item will cause a label for another item to change also. Implicit relations can be handled easily enough by manual operations. For electronic operations, however, implicit relations may have to be changed to explicit ones but not necessarily in all cases.

[23] Executive Order 12356 provides for Original Classification Authorities President, Agency Heads and officials designated by the President in the Federal Register. Officials delegated this authority pursuant to Section 1.2(d) of the order, and Exceptional cases and Derivative Classification.

[24] "Annual Report to the President FY1984", Information Security Oversight Office, GSA, April 26, 1985

[25] Examples of accreditors include the four listed in the IP Revised Security Option (MIL-STD 1777). Accreditors are usually owners of control channels.

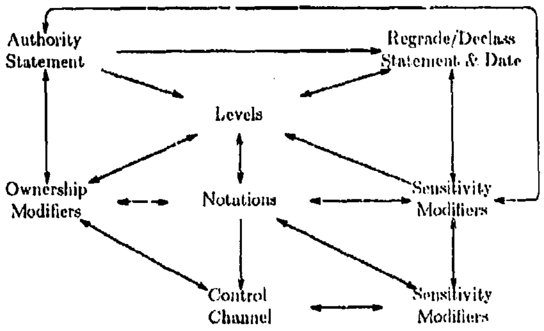
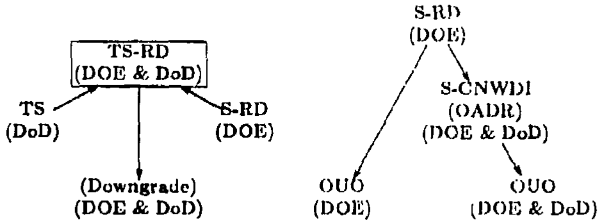


Figure 4. Which Security Label Components Affect Which?

For example, a Warning Notice of Restricted Data implies (1) that the Atomic Energy Act is included in the Classification Authority, (2) that DOE is included in Ownership, and (3)that the hierarchical level (Classification) may also be affected.[26] Thus, in a computer system, importing an S-RD portion into a "file" may affect all these components of the receiving file's ***security profile.***

Because of the interdependencies of components, either transclassifying or downgrading a document (or "file") can complicate its ultimate declassification (fig. 5-b) in the same way that amalgamation often does (fig. 5-a), namely, by including more owners and classification authorities. The DOE could directly "declassify" an RD document to OUO. If, however, DOE transclassifies the document ("file") to FRD or CNWDI, then "declassification" to OUO may require the approval of both the DOE and DoD, where (if) FRD or CNWDI implies that DoD is included in ownership and classification authority.[27] From a DoD viewpoint, "declassifying" a CNWDI document may be more complicated simply because the DOE is the originating agency, as indicated by the required Restricted Data notation, and "OADR" applies even if it does not appear explicitly on the document.



a. Amalgamation Can Complicate Downgrading.

b. Transclassification May Complicate Declassification.

Figure 5. Expanding Ownership Complicates Re-Classification.

4. Further Implications for Computer Systems

4.1 Messages. In messages, the title page and portion markings are usually considered text: the author is required to include them in the body of the message. The page markings are also in the text, top and bottom, but the highest page marking must also be attached to the container, envelope, packet, etc. that transmits the message. In other words, major portions of what may eventually need to be part of a (trusted) ***security profile*** and therefore may need to reside in (and be controlled by) the TCB, resides, with current implementations, in the contents of the message. Ideally, computer security and specifically the avoidance of unauthorized disclosure should not depend on automated message contents, i.e., data integrity.

4.2 Distributed Systems. In a distributed computer system, users may be able to access remote resources. Labeling requirements will vary according to the granularity of transfer. We illustrate the problem in figure 6. Importing a portion of a "file" into another (on a different host or file server) may result in the loss of that security-related information and control provided by the title markings associated with the "sending file". While the same problem could occur in a monolithic computer system, it seems almost inevitable in a distributed system, and its solution more remote (no pun intended).

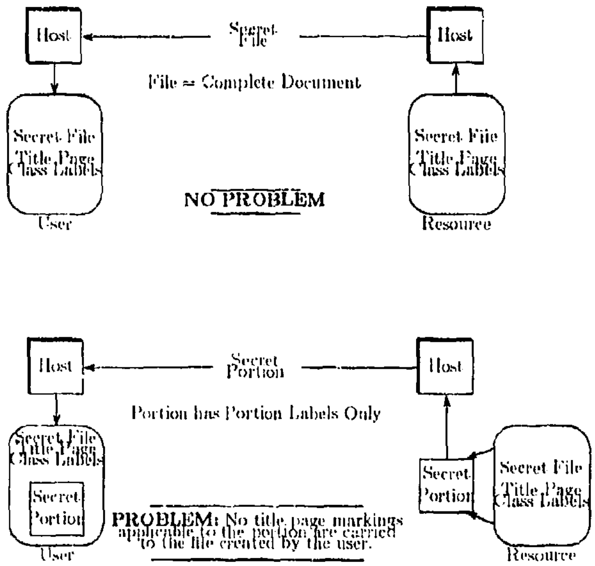

Host
Secret File
Host
File = Complete Document
Secret File Title Page Class Labels
User
NO PROBLEM
Secret File Title Page Class Labels
Resource
Host
Secret Portion
Host
Portion has Portion Labels Only
Secret File Title Page Class Labels
Secret Portion
User
PROBLEM: No title page markings applicable to the portion are carried to the file created by the user.
Secret Portion
Secret File Title Page Class Labels
Resource

Figure 6. Loss of Security Label Components in a Distributed System.

We illustrate the problem more concretely in figure 7. The context could be monolithic or distributed. The "file" resulting from the import may have its Classification Authority expanded to the union of the two separate authorities. The owner might remain the same if no proprietary information, special requirements, or interdependencies (as in our DOE/DoD example) apply. The resulting Control Channel will be per agreement among all channels involved. The resulting Declassification Statement will be the more restrictive. (We assume OADR is more restrictive than 2 years.) Finally, any Notation applicable to the imported portion must be included in the result.

[26] A common misconception outside the DOE is that RD or CNWDI implies at least Secret. There is plenty of C-RD and S-RD, and relatively little TS-RD. CNWDI can also be C, S, or TS. In addition, CNWDI may be categorized as Sigma 1 (thermonuclear), Sigma 2 (fission), etc.

[27] For brevity, we use the portion labels for the document labels.

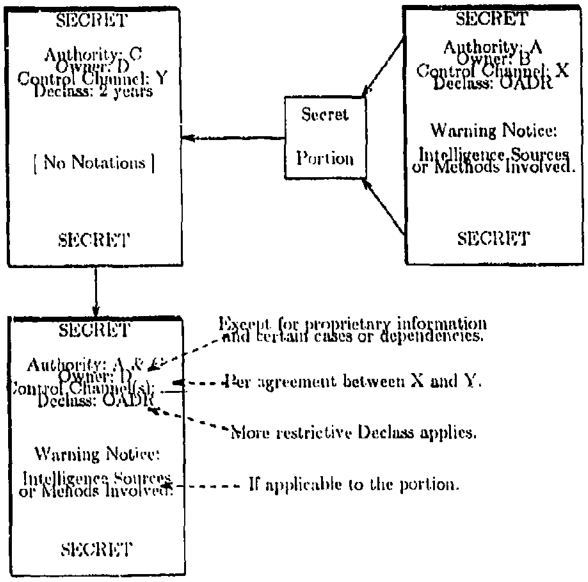


Figure 7. Problem even in a Monolithic Computer System.

5. Conclusions

We hope it is abundantly clear that ***security labels*** are not simple. They provide more needed security-relevant information than sensitivity alone. Even sensitivity is nontrivial. Neither do classifications and clearances necessarily map directly, one to the other. The implications for secure computer systems may be profound, especially for those systems that will seriously and distributively attempt to replace, as far as possible, manual operations on hardcopy.

Of course, if one never generates any documents (in whatever medium) of any significant longevity; if all one's actions are "quick and dirty" and simple (and on a monolithic system), then getting the Declassification Statement wrong on a sensitive document may never actually result in the document's unauthorized disclosure. Under these limiting conditions, perhaps some of our other examples go away as well. But "quick and dirty" contradicts security; and simple, monolithic systems are the past, not the future.

We hope to discuss in detail, in a future paper, our thoughts on possible solutions to some of the problems presented here. At present, however, in our judgement, the community is not "ready" for general solutions. There is not just a lack of consensus on what the problems are or on what some reasonable candidates for solutions might be. There is not yet even much perception that a general solution is needed. Each agency understandably tends to view its own classification and security procedures as ***relatively*** sane and simple. (Familiarity with procedures encourages this view whether it is well-founded or not.) As one person at DOE stated, "The complexity of the issue is in DOE's interactions with the DoD and other foreign governments, not within the department." Here "other" is a redundant synonym for "foreign", but the unintended suggestion that the DoD is a foreign government underscores the person's point that the difficulty lies at the boundary. Currently the DOE exchanges files with DNA, a DoD agency. While the DOE admits that "file handling is messy and the [file] headers are HUGE", they have confidence in the security of the exchange and remain comfortable with a distinction between sharing files and sharing documents. Of course, the DOE is just one example that interfaces with the DoD. As automated, inter-agency traffic increases (both the traffic and number of sharing agencies), each agency may understandably become more concerned about what agencies the agencies they share with share with. A general, standardized security mechanism such as ***security profiles*** may then be demanded by such agencies or mandated from above; and the distinction between sharing documents and sharing files may no longer be viable.

At the top level and for monolithic computer systems, one solution may be stated simply enough: Identify all gaps between security procedures for clearances and classified hardcopy, on the one hand, and current AIS practice on the other; then close these gaps by including the required information in a genuine ***security label*** mechanism. Obviously, this is easier said than done.

Simply what constitutes a complete list of gaps is not, in general, known or agreed upon. The "list" given here can be expanded. Thus, as a parting shot, consider the retention problem. Classified hardcopy may be associated with a specific contract, identified, say, as number N. Said contract has an estimated expiration (completion) date, EED (*cf.* DD form 254). Depending on contractual relations and whether data is to go from one company to another, permission may or may not be needed to export data from the document to a document associated with a different contract. Customarily, at the EED, either the EED is extended or the document must be destroyed within a specified time, typically 90 days. Destruction of hard or soft copy under a different contract (including data imports) may be someone else's problem, but destruction of contract-N identified copy must include ***all*** copies, including automatically generated backups. (We say nothing here about the problem of (magnetic) remanence.) The contract number and the date of the most recent applicable DD form 254 may be indicated in the "Classified by" marking of a hardcopy document, but the EED itself is not on the document. Here again automation may require the ***security profile*** of the softcopy to contain even more than the full set of markings on the corresponding hardcopy.

While solutions to the problems broached here may be tailored to a specific application on a monolithic computer system, such an ***ad hoc*** approach seems inefficient from a software development view. Neither does such an approach lend itself to verification of either the formal or less formal kind.

For distributed systems, we expect the same problems to be harder. Communications protocol standards are only now being agreed upon which leave no room for all the additional data that would be needed to hold a full-fledged ***security label.***

Acknowledgements

We would like to thank Carl Landwehr of the Naval Research Laboratory for his encouragement, and Dave Bailey of the DOE, who steered us away from numerous misconceptions about DOE classification. (For any that remain we accept full responsibility.) We would also like to thank our colleagues at Ford Aerospace: George Dinolt for bringing us, the co-authors, together, and Chris Tucci and Karl Kelley for their comments on earlier versions.

References

1. "Annual Report to the President FY1984", Information Security Oversight Office, GSA, April 26, 1985.
2. Bell, D. E. and La Padula, L. J., "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997, MITRE Corp., 1976.
3. Bell, D. E. and La Padula, L. J., "Secure Computer Systems: Mathematical Foundations and Model," M74-244, MITRE Corp., May 1973.
4. Denning, D. E., ***Cryptography and Data Security,*** Addison-Wesley Pub. Co., 1982, pp. 266 ff.
5. ***Department of Defense Trusted Computer System Evaluation Criteria,*** DoD 5200.28-STD, December 1985.
6. "Good Security Practices for Information Ownership and Classification," G360-2705-0, IBM, Nov. 1986.
7. ***Industrial Security Manual for Safeguarding Classified Information,*** DoD 5220.22-M, Sept. 1987.
8. "National Policy on Telecommunications and Automated Information Systems Security," ***National Security Decision Directive 145,*** (NSDD-145) (Unclassified Version), The White House, Sept. 17, 1984.
9. ***Personnel Security Program,*** DOE 5631.2, Nov. 13, 1980. (This document is superseded by DOE 5631.2B, presumably published in 1988. The authors have not seen this revision.)
10. ***Standard Security Markings,*** DIAM 65-19, July 1984.
11. ***Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria,*** NSCS-TG-005, Version-1, 31 July 1987.
12. Williams, J. C., "Morphism between Bell & La Padula Model and a Sequential, Dynamic, Special, Hybrid Graph," F0041, Ford Aerospace & Communications Corp., 1987.
13. Williams, J. C. and Dinolt, G. W., "A Graph-Theoretic Formulation of Multilevel Secure Distributed Systems: An Overview," ***Proc. 1987 Symposium on Security and Privacy,*** IEEE Computer Society, pp. 97-103.
14. Woodward, J. P. L., "Exploiting the Dual Nature of Sensitivity Labels," ***Proc. 1987 Symposium on Security and Privacy,*** IEEE Computer Society, pp. 23-30.
15. Woodward, J. P. L., "Security Requirements for System High and Compartmented Mode Workstations," MTR-9992, MITRE Corp., May 1986 (also pub. by DIA as DRS-2600-5502-86).

Appendix

The following table shows classification components *vs.* some clearances.

	TS	Int TS	S	Int S	C^G	Int C^G	C^C
TS	+	+	-	-	-	-	-
S	+	+	+	+	-	-	-
C	+	+	+	+	+	+	+
RD	+	S	+	-	+	-	-
FRD	+	S	+	-	+	-	-
NWSC	+	*	+	*	+	*	-
ACDA	+	+	+	+	+	+	-
SCI	+	S	+	-	+	-	-
COMSEC	+	S	+	-	+	-	-
NATO	COSMIC	S	+	-	+	-	-
NATO Restr.	+	+	+	-?	+	-?	+
Foreign	+	?	+	?	+	?	-
Domestic	+	+	+	+	+	+	-

+ Access granted

\- Access denied

* Authors lack need-to-know.

? No specific rules. We assume this means "as per agreement with the Government." May be similar to NATO.

-? Access denied according to the ISM, but this might be chalenged based on the fact that lower clearance (C^C) can have access.

Int = Interim

NWSC = Nuclear Weapons Security Program

ACDA = Arms Control Disarmament Agency

C^G = Confidential clearance granted by Government

C^C = Confidential clearance granted by Contractor

MANAGING THE ACCREDITATION PROCESS: LESSONS LEARNED

Jennie Stevens

Booz, Allen & Hamilton Inc.
4330 East West Highway, Bethesda, Maryland

INTRODUCTION

Accreditation: A policy decision by the responsible Designated Approving Authority (DAA) resulting in a formal declaration that appropriate security countermeasures have been properly implemented for the computer (ADP) system or network.

The process involved in preparing for the accreditation of a computer system or facility is, in many ways, akin to doing your federal income taxes -- it is a labor intensive process, requires a mass of supporting documentation, and is almost certain to frustrate the most eventempered participant. In fact, it seems that the less organized you are, the more frustrating an experience you will have -- in both taxes and accreditation!

The primary intent of this paper is to help others avoid unnecessary frustration by sharing some of the lessons that we have collectively learned -- some the hard way -- through direct participation in a variety of different accreditation experiences. We share a common concern gained through our involvement in computer security as civil servants, military personnel, and consultants who have helped accredit systems for the Department of Defense, selected U.S. civilian agencies, and the Department of Energy. Based on these experiences, we believe that the success of a computer system accreditation (or certification) is, in very large part, dependent upon the way it is managed from the start.

Because of the many different agencies and departments we have either served in or supported, it is hoped that there is broad applicability to the the lessons presented here. They should be useful to anyone inside or outside the government who finds him- or herself involved in planning, managing, or participating in any portion of the accreditation process. The desire to save money and reduce waste is also another reason for this paper: it is hoped that dollar depleting wheel-spinning can be avoided by applying some of the lessons offered. Finally, while accreditation is an experience that the defense community has lived with for a long time, the civilian agencies have not, for the most part, had to comply with its many requirements. Risk assessment, annual loss expectancy, contingency planning, security tests and evaluations, etc., may be foreign to many of the federal government's unclassified yet, critical computer operations.

But the climate appears to be changing and concern for computer security in this sector is slowly increasing. It is therefore also our intent to share our experience with those in that community who are gearing up for an accreditation or certification. The paper is organized according to the key phases involved in the accreditation process: initial preplanning, risk assessment, security test and evaluation, and the final phase of preparing and presenting the formal accreditation package. The remainder of this paper highlights many of the most frequently cited problem areas that can hamper each of these phases, and offers suggested approaches for avoiding these glitches in future accreditation efforts.

THE PLANNING PHASE: LAYING THE FOUNDATION FOR SUCCESS

Solid, up-front planning establishes the foundation for a smooth and successful accreditation experience. While this may seem intuitively obvious, there are enough "war-stories" around of accreditation snags and mix-ups to indicate that the rudiments of this phase are frequently only given lip service or are altogether ignored.

(1) Define the Roles and Responsibilities of All Participants

Accreditation requires the participation of numerous people. Those most intensively involved are the accreditation team leader and team members. The assembly of a multidisciplinary accreditation team is essential if all facets of accreditation are to be addressed with a high level of confidence. The team should, at minimum, have expertise in the following areas of security: computer, physical and environmental, communications, and emanations (TEMPEST). The team leader should be definitive about who will be responsible for each of these areas throughout the process.

But aside from the accreditation team, there are a number of other key people who must be identified early on in the process and with whom accreditation requirements must be precoordinated. Senior decision makers responsible for approving budget expenditures and labor assignments should be briefed at the start with regard to resource requirements, milestones, and any special needs that the team anticipates. If that individual or group of individuals is not fully conversant with accreditation specifically and security generally, a brief overview of the process is also worthwhile. This is particularly true if accreditation resource requirements are larger than the allocated security budget (which in most cases is true since security is rarely a dollar-rich line item in the operating budget).

Other key individuals with whom early contact must be established include the:

- Computer facility manager and principal system operators (to coordinate scheduling of the risk assessment and ST&E with the objective of minimizing system down times, as well as to collect all system/facility background information relevant to the accreditation)

- System security officer (SSO), if one has been selected (to participate on the team or to assist the team in collecting all necessary data during the survey, risk assessment, and ST&E phases)

- Agency or headquarters security office (principally for coordination on physical and environmental security concerns and threats)

- Agency or headquarters engineering office (to address facility-related design, construction, environmental, and electrical issues and/or questions throughout the process)

- Local law enforcement agencies or representatives (in the event that additional threat data is necessary)

- Nearest Military Intelligence (MI) Group (for up-to-date TEMPEST threat and espionage data and survey/test assistance).

Since formal TEMPEST surveys are generally scheduled based on the importance of the computer facility's operations, this last point is particularly crucial if accreditation must be achieved within a certain period of time. One team leader briefed his supervisor that accreditation could be accomplished in 4 months, only to discover that there was a significant waiting period for the required survey! (Fortunately there was a happy ending to this story -- he was able to piggyback his survey with one that had been scheduled for a nearby facility with a higher priority operation.)

(2) Precoordinate and Preschedule As Much As Possible

Based on the preceding discussion of roles and responsibilities, it is clear why precoordination and prescheduling are worthwhile. From the standpoint of the accreditation team, it also provides for well-coordinated interfaces with the numerous offices and personnel with whom they will have contact. No one likes to answer the same questions repeated by different members of the team, nor do they appreciate redundant requests for data which could have been provided during one, single session. In other words, it is critical to pre-plan all survey questions to be asked by each team member in order to minimize repetition, unneeded back/tracking, and successive, costly site visits.

The development of a comprehensive schedule and a companion accreditation "diary" is also integral to a well planned effort. They serve to provide a clear understanding of the accreditation timetable along with the agreed upon responsibilities of each person involved. The diary should include dates when precoordination meetings or contacts took place, the name(s) of the individual(s) with whom the arrangement(s) were established, and the date(s) of the planned meetings, the survey, risk assessment, ST&E, etc. Advance scheduling of critical meetings or events is especially important, particularly when they involve senior decision-makers or entail a significant occurrence such as a system shutdown for the ST&E. And finally, a word to the wise: all meetings should be confirmed a day or two in advance to ensure all key attendees will be available as planned. One team member made a very long trip only to find that a key point of contact was on leave for the week, an advance phone call might have saved the day.

(3) Make Use of The Fruits of Others' Labors

In many instances, a computer facility facing accreditation is located in the same building or on the same base or center where an accreditation has recently taken place. It is perfectly acceptable to coordinate with the participants of the previous accreditation activity to solicit data and information that is pertinent to the ongoing effort. In many cases the threat data that pertained to the accredited facility are applicable to the other, and can be massaged to fit current accreditation needs. It is advisable to precoordinate this step with the DAA's representative to ensure that he/she does not object to this time saving approach.

In this same vein, it is important to acquire as much available data and "history" about the computer facility/system as possible. Of interest would be any records on major and/or minor system down-times and their causes (A/C failure, electrical outages, weather related problems, etc.); security "track records" and incident reports, and all system mission- and organization-related data. This type of information, assembled as part of the daily operational regimen of a facility, can be invaluable to the accreditation team.

(4) Collect, Read, and Analyze All Applicable Instructions, Regulations, and Standards in Their Entirety Early On

While this statement also seems so patently obvious that it should not merit discussion, war-stories dictate otherwise. To avoid surprises, it is prudent to collect and thoroughly read all documented requirements that pertain to the accreditation of the given system before doing anything else. This early familiarization period allows the team manager and members to identify whether methodologies and approaches set forth in the documentation fit the "reality" of their particular accreditation situation. Additionally, it allows them to pinpoint any areas where guidance is ambiguous or non-existant. This will also allow the team to strategize in advance regarding how they will address any special accreditation issues that are not specifically covered in their respective agency's accreditation guidance. Finally, in those cases involving multiple agencies, agency-specific accreditation standards must be integrated into a memorandum of understanding that sets forth the agreed upon accreditation needs of the multi-user system.

(5) Use the "Tools of the Trade" to Maximize Efficiency

The accreditation team can vastly simplify its task if it plans in and precoordinates the use of selected accreditation "tools." For instance, if permissible during the site survey, the use of a hand-held dictaphone for note taking greatly speeds up an otherwise tedious process. The use of a 35mm camera has also proven to be useful in certain situations. Photos are not intended for publication in the final package as a rule, but rather are used by team members to jog their memories when developing survey and assessment results. It can also be invaluable in tracking the progress of major system changes over time that have an impact on security, and provides a head start on the system's accreditation or certification.

The team should also avail itself of the latest Evaluated Products List (EPL) which invertories all computer security products approved for use by the DOD Computer Security Center, as well as the latest Datapro reports which list all available computer products and their capabilities, to include security features for given releases of software.

Finally, the team may find that instructional videos, similar to one that we recently produced for the Navy, can simplify many facets of the accreditation process, help explain key roles and responsibilities to prospective team members and other participants, and provide a cost-effective security training tool for use by the SSO throughout the lifecycle of the system.

THE RISK ASSESSMENT PHASE: STAYING IN THE DRIVER'S SEAT

Perhaps the most frequently mentioned liability during this phase of the accreditation process is the tendency to let the documentation drive you rather than the other way around. The applicable regulations and instructions provide policy and guidance on how best to determine and measure a system's vulnerability to specific risks. But the accreditation team is responsible for actively managing the overall process, and is afforded a certain degree of latitude in meeting this responsibility.

While the degree of latitude in selecting a risk assessment methodology may vary depending upon whether the system will process classified information, the team can generally:

- Select the best risk assessment methodology for the given system (e.g., qualitative versus quantitative)

- Determine whether an automated risk assessment package makes sense for the accreditation situation at hand and, if so, make an appropriate selection

- Select the most suitable format and "packaging" strategy for all documentation prepared as part of this and subsequent phases

- Consider any planned system enhancements in selecting the methodology in order to facilitate future risk assessments.

Finally, if the team is considering use of anything "exotic", it is strongly advisable to touch base with the DAA's representative to determine whether the approach under consideration is acceptable -- before digging in too deeply. In one instance a system developer toyed with designing his own add on security software package, assuming that this was an acceptable approach to managing the system's risks. The DAA had other ideas on this subject, however, and use of only a few software packages with solid security performance records was considered acceptable for this particular classified system configuration. Luckily the question was asked first.

THE SECURITY TEST AND EVALUATION (ST&E) PHASE. PLANNING TO ADDRESS THE OMITTED AND UNMENTIONED

Once the risk assessment is completed and all recommended countermeasures have presumably been implemented, the ST&E team is responsible for ensuring that the system's security countermeasures exist and serve their intended purpose. The accreditation team manager can provide a thread of continuity between these two phases, since, as a rule, survey/risk assessment team members do not participate in the ST&E to ensure full objectivity.

Because a separate team is involved in the ST&E process, the problems that are frequently encountered during this phase are associated with continuity. Specifically, the ST&E team must hope that the risk assessment report, which they will use extensively in preparing for the ST&E, addresses all system deficiencies and associated countermeasures. Frequently, the risk assessment addresses only those countermeasures identified as part of the assessment and omits mention of any that were already in place and effective. Thus, the ST&E team should plan on fully immersing itself in all available documentation -- not only the risk assessment, but also system mission statements, standard operating procedures, etc., to ensure that all countermeasures are tested and discussed in the final ST&E report. This is critical since the report is used by the DAA to make the final accreditation decision.

The need for ample precoordination prior to and during the ST&E is also critical. The necessity for prescheduling the ST&E itself has already been emphasized above. As the date approaches, the test director should coordinate test activities with all involved site personnel. Moreover, any special requirements for the ST&E should be discussed in detail with the system manager (e.g., color changes, system shutdown/start-up, etc.) The test director should also require that each ST&E team member prepare a schedule showing the sequence of their specific test activities. This will cause team members to critically review and plan their tests, eliminate back tracking, and expedite the test.

Presentation of the ST&E results is the final major hurdle in the ST&E phases; it can also bring the process to a standstill if (1) the ST&E uncovers a deficiency which must be corrected before the DAA will accredit the system, and (2) in order to correct the deficiency, additional resources are required. Since accreditation is frequently regarded as "that time-consuming irritant that senior decision maker's learn to live with" -- few team leaders feel comfortable making a presentation that concludes with a request for more money. In the final analysis, this presentation is a sales pitch, not a results briefing that the team leader or his supervisor must present. It should be viewed as such and the content of the presentation must be on- target, high-level in perspective (versus forcing feeding a busy decision maker endless tactical details), and as upbeat in tone as possible. While this suggested approach will not guarantee a commitment of additional resources, it will hopefully make it more likely.

TYING UP LOOSE ENDS AND OTHER FINAL REMARKS

The last major hurdle to cross in the accreditation process is effectively organizing all the team's findings and supporting documentation into one document that will comprise the formal accreditation package. A word to the wise here -- the importance of a well organized presentation and report cannot be over stressed. The presentation should include a detailed table of contents that serves as a "road map" for the DAA in his/her review effort. Again this will not ensure accreditation of the system, but it will certainly establish a good first impression regarding the team's overall professionalism and performance.

In the final analysis, a well managed accreditation effort provides an excellent learning opportunity for all those that it involves, facilitating a greater appreciation for the many risks to which a computer system is exposed and the ways in which such risks can be minimized. However, when the process is unmanaged or ill managed, it can be a costly and nightmarish experience for all concerned. Hopefully the lessons set forth here will help those who might face such involvement to turn the experience into the positive management challenge that it can and should be.

Spiral Classification for Multilevel Data and Rules

Matthew Morgenstern
Senior Computer Scientist

SRI International
Computer Science Lab.
333 Ravenswood Ave.
Menlo Park, CA 94025

Abstract

This paper addresses two multilevel security problems that appear to require write-down of data. However, such write-down would incur risks since it makes visible to users data that is derived from information for which the users are not cleared. The proposed approach essentially avoids write-down in both cases, thereby increasing the level of assurance of the system. The first case arises in a multilevel rule-based expert system, where we need to ensure that a low-level user will not be given grossly inconsistent or harmful advice due to higher level rules and data not being available. The second case arises from use of rules to assign classification labels to new data entering a multilevel database system.

1 Consistency for Multilevel Rule-Based Systems

Rule-based Expert Systems currently operate at a single classification level. However, there will be increasing need for rule-based systems in which the rules themselves may have classification labels, as well as the data on which these rules operate. Examples are discussed by Berson and Lunt [2]. Only rules at or below the user's clearance level would be invoked on behalf of the user, and only data at or below the user's level could be utilized by the rules.

An important problem which arises for such multilevel rule-based systems is the *consistency* of the results. By *consistency*, we mean that these results would not seriously conflict with requirements of the application. Such consistency could be achieved, but at the expense of security, if the results were initially produced with the system running at system-high and then an attempt were made to sanitize these high results. However, run-time sanitization generally would not be acceptable due to the complexity of such a sanitization process and the need for it to handle a very wide range of information. It is not likely that such complex sanitization could be sufficiently trusted.

We propose a method which we call *spiral consistency enforcement* to essentially avoid this write-down problem. A key aspect of this method is independent execution of two *similar* expert system processes at two different classification levels. The separate results are then compared by a *verification process* to ensure that the recommendations formed at the user's level are safe and consistent with the technical requirements of the application.

For example, the expert system should not propose a mission with advice to use sunglasses (a glare shield) when in fact the more highly classified details of the mission actually require thick lead radiation shielding. It would be far better for the system to simply not propose the mission at all, than to risk human life. The verification process has the responsibility to determine whether the result computed at the user's level is valid or invalid, and to produce a yes/no decision as to its use. The primary stages needed for this enforcement of consistency are:

__Spiral Consistency Enforcement__:

1. **Primary Process:** Execute the rule-base of the expert system at the classification level of the user, generating what we call the *primary result* P. Since this process omits rules and data that are not dominated by the user's level, the *primary result* satisfies all security requirements (assuming the original classification labels on existing data and rules were assigned properly).

2. **System Process:** In a separate process, execute the rule-base at system-high or at the highest level deemed necessary to ensure that the *secondary result* S produced by this step is completely *consistent* and acceptable from the viewpoint of the application – i.e., that critical application requirements are satisfied. Call the level of this result L_S.

3. **Verification Step:** At this higher level L_S, execute a *verification process* V whose task is to determine whether the primary result P produced at the lower level violates any essential constraints of the application relative to the more complete secondary result S. This process V only compares the two results but makes no changes to either. The sole output of V is a binary yes/no indication of whether it is acceptable to release the primary result to the user. For example, if the primary result is so far afield that it could endanger human life, then it would not be released.

4. **Release of Results:** If this verification fails (i.e., indicates 'don't release') then either no result is produced by the system or, perhaps, a *cover story result* is presented. If the verification succeeds, then just the *primary result* P — which already is at the user's level — is presented to the user. It is important that these four steps operate as an *atomic* unit, in that no data is released except at the proper completion of this final step. Any system interrupt or suspension of processing during these steps must be safeguarded to not release data directly or indirectly.

This approach avoids the direct use of the higher level rules in creating the mission plan, thereby avoiding a potentially serious downgrading problem. When primary results are released to the user, we know that they were generated only from use of the lower level rules.[1]

Thus this procedure for *spiral consistency enforcement* avoids the problem of downgrading or major sanitization while at the same time providing essentially the same application safeguards. The verification process cannot release data above the user's classification level. Its purpose is to ensure that the recommendations in the primary result are safe and consistent with the technical requirements of the application. While the verification process does operate at the higher level, its output is through the very narrow yes/no channel.

In particular, the only data that is a candidate for release is the *primary result* P, and it was generated at the user's level. When the primary result is not deemed consistent or safe, then it would be withheld. Furthermore, if the simple absence of output is itself a covert channel of concern in the application, then a *cover story* could be offered.

A *cover story* is an explanation that can be presented at a low classification level to explain conditions that actually arise from higher level data. Such a cover story must be generated by a trusted subject. For example, if flights to Iran are not visible due to their being classified at a higher level, some explanation might be offered to divert attention from the actual reason that such flights are not shown. The cover story might be that the airfield for Iran is undergoing repair. In general, a prespecified set of cover stories might be used, with limited run-time tailoring done by a trusted subject. It is worth noting that the use of cover stories is similar to sanitization via adding noise (perturbing the data).

2 A Rule-Based System for Multilevel Classification

As multilevel database systems become available, the process of classifying large volumes of data at appropriate levels will become increasingly complex. Not only will the initial database need to be given classification labels, but new data entering the system will require classification. Manual classification of massive amounts of new data likely will not be feasible. Thus automated techniques will be needed.

A good framework for such automated techniques would be a rule-based system in which the rules recommend classification labels based upon the type of data, its source, value, and possible relationships to other data. Such rules have been called *classification rules* by Denning in the SeaView project [3]. We call the overall system which produces classification labels for data the *Classifier Tool*. This Classifier could assist a security officer or database administrator with the classification process. An analysis of other aspects of such a tool, including the problem of accounting for logical inferences by users, can be found in Morgenstern [4]. This *Classifier* would determine which of the potentially large set of classification rules may be applicable to data about to enter the database, and would evaluate each rule to determine a classification label for the data. In particular, it could find that several such rules apply, and that different rules recommend different labels — thereby giving rise to the problems we investigate below.

The problem of *write-down* will arise if the tool consults data at several classification levels, or if classification rules are themselves assigned classification levels. In these cases, the tool may need to operate at the least upper bound (lub) of such levels. If the tool assigns a classification that is less than this lub, then writing the label at a lower level would be a form of *write-down*, which might be considered a potential violation of security.

2.1 Classification Rules

Classification rules may apply to data objects at several levels of granularity, including relation, tuple, or element level. Mandatory security requires that in order for information D to be available to user U, the authorization of U must dominate the classification label of D — which may be represented as $U.level \geq D.level$, where $U.level$ denotes the authorization level of user U and $D.level$ denotes the classification of data D. The term *level* is traditional terminology although it should be noted that the set of classification "levels" actually forms a lattice due to compartmentalization of data.

We assume completeness of the classification rules, so that at least one rule is applicable. The new data to be classified will be referred to as the *target data*. To determine which rules need to be executed for some new target data, *relevant rules* first are selected based upon the nature of the target data, such as the the type of the data, etc. Rules are selected as relevant without referring to current data values stored in the database. Then the condition part of each relevant rule is evaluated relative to actual data. In general, the condition parts for only some of these rules will be fully satisfied — these are the *executable rules*. We assume here that several rules could be executed for one data object.

If the executed rules access existing data at different levels — or if the rules themselves are classified differently — then additional questions arise:

- If high level rules and data need to be consulted to recommended a lower level classification label, we have the problem of *writing down* when generating the low label. Can this write-down be avoided?
- If the recommended labels are different, which label or combination of labels should be used? Generally, taking the least upper bound of the labels would be appropriate, although other possibilities might be to prioritize the rules or to signal an inconsistency requiring manual intervention. The decision may be specific to the application.

In the next section, we focus on the first question above, namely can we avoid write-down when high level rules and data need to be consulted to determine that a lower level classification would be appropriate?

[1] Although we are addressing mandatory security here, this procedure can be generalised to include discretionary security. The *primary result P* depends upon data and rules to which the user legitimately has access, while the verification process can operate at a higher classification and with additional access privileges.

2.2 The Write Down Problem

Since the Classifier will need to write labels at all levels including system high, one might wish to operate the Classifier at system high. However, writing labels at lower levels then would present the write-down problem.

There are two cases where the classification process may *need* to operate at a high level in order to properly determine that the label for some data should be at a lower level. For the first case consider that all the rules are unclassified. If a rule is given some target data, then the label it assigns may depend upon the classification levels of related data that the rule needs to access. Thus if the rule is run at a low level, it may not be able to see such related data, and thus may incorrectly conclude that the label should be low.

It might seem necessary then, for the rule to be run at a higher level, so that it could detect this related data and make the correct decision to assign a high label to the target. However, if the rule is run at this higher level but there is no high data which is related, then the rule may appropriately recommend a low level label. Since the rule is operating at a high level, generation of a low level label creates the write-down problem.

The second case arises if the classification rules themselves are classified, with different rules having potentially different classifications. For example, sensitive values may be represented in some rules in order to assign higher labels when these values arise. Such rules might be classified higher than other rules because of the contents of these rules.

In this case then, the relevant high level rules must be invoked to determine if their value-dependent conditions are satisfied and whether they recommend a high label. If such rules do *not* recommend a high label, writing a lower label would be a form of write-down from this higher level process.

The approach presented below addresses both of these cases. It considers the level at which the classification process is executing regardless of whether this arises from the level of related data which needs to be inspected or whether the rules themselves are classified at this level.

2.3 The Spiral Classification Process

One might consider as a first step in addressing the write-down problem the creation of a separate Classifier process at each security level. Each Classifier process would be allowed to write labels only for its current level of operation. A Classifier process could utilize rules and access data at lower levels but not at higher levels [3].

However, this approach does *not* solve the write-down problem. First, the decision as to whether some target data warrants a low or a high classification could logically depend upon the presence or absence of higher level data. Secondly, the labeling decision may depend upon whether any higher level rules exist that apply to this target data. That a low level rule is applicable does not prevent a higher level rule from knowing more and mandating a high label. In both cases only a high level process can determine whether high level data and/or high level rules apply. Hence it would still appear that write-down from a Classifier process operating at a high level is needed to fully ensure that the labels assigned to data take into account all relevant classification criteria.

We propose a spiral classification procedure which essentially avoids the write-down problem. Since it executes a classifier process at successively higher levels, starting with the lowest level, it creates a "spiraling" effect. The properties necessary for spiral classification are:

<u>Spiral Classification:</u>

1. **Monotonicity**: A classifier process is executed first at the lowest classification level, and then executed at each of the other classification levels. The order of considering the levels must be in a *monotonically non-decreasing* order in the classification lattice. That is, each subsequent classification level to be examined must either dominate or be non-comparable with each preceding classification. A separate executable process could be created at each classification level.

2. **Atomicity**: The data to be classified is *not* made available to users until the entire spiral classification procedure is completed at all levels. This requirement will be satisfied if the overall process is atomic. That is, either it completes without error, or else the system ensures that there is no evidence of an incomplete execution. Atomicity ensures that if high level rules assign a higher classification label than previously executed lower rules, then this higher level label will take precedence before any data access is allowed. This approach maintains the *tranquility* property of Bell and LaPadula [1], because users will be presented with an unchanging classification for each data object.

3. **Least Upper Bound**: When the classifier recommends a new classification label, it should be at the level the classifier is then operating at. Thus there is no write-down from any iteration of the classifier. If the data already has a current label (c), then the least upper bound (lub) of the new label (h) and the current label (c) should be used. If we have a strict hierarchy of levels, or if the h dominates c, then the lub is just h. Thus any writing of classification labels by the classifier raises the data object to at least the level at which the classifier is executing.

4. **Trusted Kernel**: Since each iteration of the classifier is at a single level there is no write down. The outermost supervisory process that initiates each iteration must be trusted to execute the levels in monotonically non-decreasing order, to prevent release of data during the process, and to allow labels to be revised only upward.

We observe that atomicity guarantees that data is not released until it has been given the highest level label that is applicable, and then it is released only at that level. The trusted kernel guarantees that intermediate stages of the iterative process do not release data. Thus lower level rules do not release data, they only could cause the classification labels of the data to rise. We also note that the trustworthiness of a rule is essentially the *integrity level* of the rule rather than its sensitivity.

The spiral classification process can be applied during initial loading of the database to label all the data. During run-time it can be reexecuted periodically to label new data — each such execution must be in a trusted partition and must be atomic. An external concern is that new data should arrive through a secure route so that it is not accessible until it is classified appropriately (for example, sensor data could be encrypted at the source and decrypted when it enters the classifier).

Thus far it has been assumed that a new Classifier process is created at each level. One might wish to iterate a single Classifier process at successively higher levels, rather than creating a new process at each level. Since the spiral is upward to higher levels, it might be considered adequately safe for some systems. However, due to the partially ordered nature of the classification lattice, care must be taken regarding non-comparable levels. In particular, use of the same process at such non-comparable levels creates the danger of *write-across* – which is the counterpart of write-down but for non-comparable classifications.

3 Conclusion

Both spiral classification of data as well as spiral consistency of output from rule-based expert systems share the spiral process of iteratively executing at two or more levels. The monotonically non-decreasing order of executing the levels ensures that data is not passed from a high level to a low level because the higher level executes later.

The spiral process further decomposes the computations at different levels in such a manner that interactions among levels are essentially eliminated. Only the trusted kernel persists over the multiple iterations.

We have discussed how the spiral process can ensure consistency of the results from a multilevel expert system in that a low level user will not be given grossly inconsistent or harmful advice due to lack of access to higher level rules and data This spiral process executes at two levels, the user's level and a higher system verification level. We noted that the remaining low frequency binary (yes/no) channel, with one bit per problem solution, could be further reduced by use of cover stories.

We then considered how a spiral process could be utilized to execute classification rules without write-down so as to assist a security officer or database administrator by generating classification labels for data.

Acknowledgments

This work has been sponsored in part by the U.S. Air Force, RADC, Rome, N.Y., under Contract No. F30602-86-C-0088. I would like to thank Teresa Lunt and Peter Neumann for their useful comments. Thanks also to Teresa for the cover story example.

References

[1] Bell, D. E., and L. J. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, The MITRE Corp., Bedford, Mass, Report ESD-TR-75-306, March 1976.

[2] Berson, Thomas A., and Teresa F. Lunt, *Multilevel Security for Knowledge-Based Systems*, IEEE Symposium on Security and Privacy, Oakland, CA, April 1987.

[3] Denning, D. E., S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell, *Views for Multilevel Database Security*, IEEE Trans. on Software Eng., SE-13(2):129–140, Feb. 1987.

[4] Morgenstern, Matthew, *Logical Inference Channels in Multilevel Database Systems*, IEEE Symposium on Security and Privacy, April 1988, Oakland, California.

[5] Morgenstern, Matthew, *Security and Inference in Multilevel Database and Knowledge-Base Systems*, ACM International Conference on Management of Data (SIGMOD-87), San Francisco, May 1987.

[6] Trueblood, Robert P., *Security Issues in Knowledge Systems*, Proceedings of 1st Int'l Workshop on Expert Database Systems, ed. L.Kerschberg, October 1984, vol.2, pp.834-840.

BUILDING A SECURITY MONITOR WITH ADAPTIVE USER WORK PROFILES

Lawrence R. Halme
Arca Systems, Inc.
2860 Zanker Road #210
San Jose, CA 95134

Brian L. Kahn
Odyssey Research Associates
525 Middlefield Road #250
Menlo Park, CA 94025

Abstract

Issues relevant to construction of a security monitor for use as Intrusion Countermeasure Equipment to detect system intrusions and abuse by legitimate users are presented. The monitor compares current activity against adaptive user work profiles and system security policy, and alerts the security operator to any significant deviations. This approach discriminates data aggregation attacks and insider abuse as effectively as it detects intruders, and also supports a standard commercial system as well as a system customized for security. Design details cover principles of an analysis engine to extract system policy violation and historically abnormal usage patterns from the audit trail, a high level design of the security monitor, a discussion of installation specific concerns, and directions for semi-automatic application as a detective device.

1 Introduction

While the technology needed to design and verify secure machines is advancing steadily, there now exists a massive installed base quietly and efficiently performing the everyday tasks needed to support the government and commercial infra-structure. It is not practical to replace all sensitive systems subject to intruder or aggregation threats with secure systems, yet neither is there much hope for retrofit security on commercial operating systems. Integrating a reference monitor [1] which is pervasive enough to prevent bypass is difficult and disruptive, robbing both performance and functionality.

The challenge is to develop peripheral Intrusion Countermeasure Equipment (ICE) for the mainstream DP shops which has a minimal impact on the system, yet is deeply rooted in the system to discourage bypass. Analysis of the information provided by existing audit facilities fits these criteria. Modification of the operating system or hardware is not required, performance is not degraded severely, and the mechanisms are extremely hard to bypass; accounting information is collected deep within the operating systems and has been supported by special hardware features for decades. This is especially appropriate for installations with sensitive data vulnerable to aggregation attacks but with scant resources to spare.

The authors have worked on design and development of two audit analysis ICE systems. This paper contains our thoughts on building a security monitor with adaptive user work profiles, based upon our research and experience and considering the fine work of our peers presented at this and other forums[10]. The goal of this paper is to lend ideas and assistance to current and future development of security monitors or ICE.

2 Approach

This paper presents a design for ICE based upon a usage monitor which compares current activity against adaptive user work profiles and system security policy. Profiles of work patterns characterizing individual users can be derived from audit trail analysis. Deviation from these work profiles may indicate an intrusion or insider abuse. The monitor constructs and maintains the historical work profiles for each user and compares them with current activity in real time. The system security policy is regarded as a universal profile to which all users must conform.

Users have recognizable usage patterns which leave a distinctive signature on the audit trail. Naturally some users are more regular than others, but for some large percentage of users the patterns will be clear enough to be extracted by some analysis engine[6] [9]. Unfortunately, detecting intrusions and insider abuse by analysis of audit trails is not a trivial exercise. A human cannot readily review a system audit trail and isolate characteristic user work patterns. What is needed to make audit trails useful for security purposes is ICE to assist in the detection of anomalous activity. Ideally such a tool would reliably alert the operator when intrusion or abuse was actually occurring on the computer system. Realistically, the tool will alert the security operator to improper or abnormal activity, in close to real time, and assist in review and investigation of anomalous events.

Review of statistically deviant activity is tractable and manageable for large systems in real time. An analysis engine performing this task can detect abnormal or improper behavior by 1) checking the audit records for direct violation of the system security policy and 2) comparing a statistical analysis of the audit trail against historical work profiles. A user work profile might consist of a broad description derived from his or her job description, augmented with a continually updated summary of the user's individual historical activity. There may also be group or system wide usage profiles. Some actions or sequences are implicitly suspicious, clearly deserving the operator's attention without reference to the user profile.

Audit analysis ICE should allow the security officer to create templates defining the filters and statistical measures to apply to the audit trail, and the relative significance of deviations. Further, the basic analysis engine built for audit reduction and profile updating can serve as a powerful detective tool for investigation of suspicious activity.

The realm of information useful to a comprehensive audit reduction and analysis tool can be separated into the following five categories:

System Specific
/dev/ttya is connected to a call out modem.
/lib is a directory of read-only libraries.

Policy
passwd is a security-related command.
Users may not send system data files to a printer.

User Activity
Mr. Smith has never previously logged in late at night.
Mr. Jones, in accounting, has never used the compiler before.

User Specific
Mr. White is on a three week vacation.
Mr. Black is terminated as of July 4th.

Worldly
The company will be closed over the Christmas Holiday.
Research dept. is finishing up a proposal, expect late nights.

3 Theoretical Framework

Audit trail analysis ICE allows a System Security Officer to develop a rule base describing system policy, user job descriptions, and templates for construction of historical user profiles. These rules define a set of conditions which match some audit records, what data is desired from these records, how to interpret this data, and where to record the derived information. A rule base for ICE need not be the colossal web of experience that is seen in an expert system, and the rules can be laid down at a high level which is accessible to human review or discussion.

3.1 Characterizing Intrusions and Insider Abuse

Intrusion and insider abuse is defined here as the use of a computer system's resources from which you would normally be prevented or for which you do not have privilege and ethical reason to use. The following is a taxonomy of misuse we are concerned with:

Trespassing	*Access the system*
tourist hacker	*breaks in as hobby*
Browsing	*Look at the system*
passive	
active	*scavenging with goal in mind*
aggregation	*accumulation increases value*
inferencing	*distillation increases value*
Malfeasance	*Misuse the system*
leakage of classified data	
non-work related use of system	
Theft	*Steal from the system*
general software	
specific sensitive data	
Modification	*Change the system*
data	
data diddling	
false data entry	
programs	
Trojan horses, logic bombs	
round-off attacks	
viruses	
system behavior	
access rights/password files	
ownership	
accounting	
Destruction	*Destroy the system*
data	
programs	
accounts	
Denial of service	*Degrade service*
locking accounts	
degradation of system response	

Note that misuse may be accidental. A system flaw may give unexpected access to a well meaning user. Mistyping could produce an unintentional malicious result. A new user may even be unaware of certain aspects of the system policy. We make no attempt to divine intent or malevolence; the System Security Officer must use the investigative tools to determine this.

3.2 Features

Identifying these kinds of misuse requires a sophisticated review of the audit trail. The idea is to measure particular *features* of the audit trail, and compare these measurements with the historical activity record for the same user. This comparison does not directly reveal misuse, but rather indicates a degree of concern or a *warning count*. Some combinations of violations are cause for more concern than the sum of the parts would indicate, so detected features should be considered in context.

This list describes the kind of activities which should be recognized, measured, and evaluated for detection of intrusion or insider abuse. These features are examples, not as a definitive list or a taxonomy.

Time	
time activity takes place	
time of day	*off shift, lunchtime*
day of week or month	*status reports*
date	*end of quarter activity*
length of session	
time between actions	
last use of this command	
last action in a class	
last action of any type	*idle time*
Command	
first use of a command	
frequency of a command in a session	
job rate	
job mix	
ratio of one command versus another	
ratio of one command versus total activity	
multiple login	
specific command sequences	
access to certain files or directories	
other common command sequences	
permission denied, file or command	
login denied	
invalid password	
unusual terminal	
multiple login	
Resource Usage	
CPU cycles per minute	
CPU cycles per command	
Disk space	
Virtual memory	
Printers	
General I/O	

3.3 An Abstract Model for the Analysis Engine

The key to successful audit trail analysis is constructing an applicable system security policy and locating the key discriminators in user work histories. Building a successful analysis engine requires a broad, rich language for description of general system activity. Our effort started out with an approach laid out by Denning [3] and we adapted it to fit our experience on actual systems. This work outlines a small language for processing of audit records, including definition of *variables* which apply some statistical analysis to audit records selected by pattern matching. The variables describe which *features* of the audit trail we want to measure and evaluate. The reader should be familiar with Denning's work to get the most out of this section.

Our initial research was based on our knowledge of audit information available under Unix and IBM MVS systems, considering a dozen scenarios of intrusion and insider abuse. We found that analysis of individual actions (single audit records) would not reliably discriminate many of our sample scenarios. We reworked the feature definition language to better support sequences, combinations, and timing. These capabilities extend the descriptive power to effectively deal with problems one encounters during system modeling. We also added histograms, a special storage class to capture historical time relevance.

3.3.1 Events

Events are the actions which may be measured. Events are composed from audit records. Events may be represented by a single audit record or a sequence of records.

Single. Records will be treated as events when one or more of the data fields match a specified pattern. The pattern description must be flexible, including wildcards, character ranges, and alternates.

Sequence. Sequences are meant to capture a single event which is broken into many audit records. Some systems associate a string of actions with a single parent process, and this information may be available in the audit trail.

3.3.2 Metrics and Measures

Metrics are the "yardsticks" for various events we are interested in. Measures are numbers which quantify activities. Applying a metric to a record sequence produces a measure, a single number indicating how much or how many.

Counter Cycle. A counter metric tallies the occurrences of an event within a time period *(see Cycle Structure)*. A counter could track sessions during a day, number of times a transaction is executed during a session, or failed logins during three minutes.

Quantity Cycle. A quantity metric measures the amount of resource consumed by events of a specified type during a time period *(see Cycle Structure)*. Resources are reported in a variety of units. Quantities are in like units, for evaluating anything from CPU milliseconds to printer page counts.

Ratio Cycle. Ratio of one record type compared with another over a time period or session *(see Cycle Structure)*.

Cycle Structure. A cycle structure is the structure used to store counters and quantities. A cycle refers to the occurrences of some event within a repeating time interval. These intervals are expected to be stored as segments within a cyclic period such as hours in a day, days in a week, months in a year. A cycle may also be the variable length of time between login and logout called a *Session*. The period is divided into equal segments, and a measurement is made for each of these segments. We often refer to these as Histograms because the structure is easily displayed or conceived as a strip chart.

3.3.3 Models

Models are the functions which compare the profile value with the measured values, and produce a violation. The violation is multiplied by the Weighting (below) to produce a scaled *warning count* for the time period. Separate time periods in the cycle are added together to produce the feature warning count.

Limit. The measure value for each feature is compared against the value in the profile.

Deviation. The basic average is maintained to determine deviation from the mean. To save space, a close approximation can be computed with the old average and the new value. The length of the history needs to be recorded to ensure proper weighting of the new value into the average.

Variance. Variance is a computation of consistency over time. Whereas **Deviation** compares the measure for one time slot against the profile value, **Variance** compares the time slot against adjacent time slots.

3.3.4 Profile Updates

Fixed. A fixed value may be used for **Limit** processing.

High Water Mark. A fixed value may be used for **Limit or Variance** processing. Remember that the SSO is expected to verify substantial changes. Useful for locating reasonable fixed limits.

Decaying. A decaying value may be used for **Limit or Variance** processing. A decaying value is a high water mark that slowly moves back down.

Average. The average over time is maintained for **Deviation or Variance** processing.

3.3.5 Evaluation

Evaluation displays violations, determines degree, and provides weighting between the features.

Violation. There is a *violation* when a measure exceeds the profile. The warning count of the violation increases as the measure grows.

Weighting. When a violation occurs, the significance is computed by a simple formula. This formula includes a scaling factor, which takes into account the domain of the measure and the relative importance in this installation of the violation. The weighting formula produces the warning count for each feature.

Alarms. Some violations deserve more than simply increasing the significance. Alarms are presented separately, in addition to the warning count.

Grouping. Metafeatures work with the computed *warning counts* of other features, compared against limits. This grouping of features allows increasing the warning count when multiple related violations are in evidence. Grouping increases the session warning count whenever several features in a set are present.

3.4 Writing system and user profiles

Constructing and updating profiles of user work patterns and the system security policy is a sophisticated and detailed task. A database query language such as SQL is convenient for setup and investigation, but the overhead of a database query language is too high for runtime analysis and profile updating. The installation and maintenance of this design needs a small language for definition of the desired features. The paragraphs below are not a complete language, but a terse presentation of the concepts involved.

The elements of profile description are audit records, event definition, feature definition, group definition, and profile definition. These elements build upon each other as follows:

The structure of an *audit record* is:

Subject:	[user] [terminal] [modem]
Objects:	[files] [directories] [ports] [peripherals]
Action:	[command] [parameters] [error codes]
Resources:	[CPU time] [I/O counts] [page counts] etc.

The structure of an *event definition* is:
[Event-name] [Field(s):Pattern(s)]...
or
[Event-name] [Event(s)]...

The structure of a *feature definition* is:
[Feature-name] [Event] [Metric] [Model] [Update]
[Cycle] [Period] [Weighting] [Initial]
or
[Feature-name] [Feature(s)]...

The structure of a *group definition* is:
[Group-name] [Feature(s)]...

The structure of a *profile definition* is:
[User-name] [Feature(s)-or-Group(s)] ...

Audit records are reformatted by the ICE before processing. Pattern matching on audit record fields is not relevant; the authors prefer the extended regular expressions familiar to users of UNIX *egrep*, but should include wildcards, character ranges, and full regular expression alternates. The first form of an *event* pattern matches one or more fields in an audit record. The second form is a *sequence* of audit records, each defined as an single record event. A *group* is an alias for multiple features (features may be in several groups). The profile definition is used to create the actual *profile* data structure. This data structure is then maintained by the profile updater.

A new user profile contains a list of the features and a copy of the Limits referred to in the profile definition. These limits may be changed over time by the **Profile Updater**, depending upon the Update field from each feature definition *(see Profile Updates)*. The profile also contains a running total of the warning count in evidence at the end of each session, which is decayed over time and usage in the same manner as a Decaying profile update.

The session warning count is the sum of all feature warning counts during a session. This value indicates the calculated significance of all profile violations. The further out of profile a user is on any particular feature, the higher that feature warning count will be. *Out of profile* is defined as being over Limit, outside Deviation from historic mean, or excessive Variance from adjacent time periods, depending upon the Model. Since these differing counts are summed for evaluation, the Weighting of each feature serves two purposes: weighting indicates the relative significance of each feature and also converts each feature to the same scale.

3.5 Sample Feature Definitions

Login Time, System Policy. Users at Acme Widget, Inc. login during shift hours only, but may log in and out several times a day. This feature watches for the extraordinary late night login. The **event** is the time stamp from a login record. The **measure** is a counter, on a daily cycle, half hour periods. The **update** is decaying, so that the profile adjusts to the work habits. The **model** is variance because we are searching for a an event well away from other recorded events. The **weighting** is high, as this is expected to be a strong discriminator on the target in question. A login at a new time but close to the historical times would be flagged, but with a lower warning count; thus a worker who stays a bit late one day to finish one last thing will be noticed, but not heavily stressed.

Job Frequency, Accounting. The accounting staff at Acme uses the database for inventory control and does a far amount of report generation, but they are not sophisticated computer users. This feature monitors the transaction rate per minute, looking for out of profile work. The **event** is any job or process. The **measure** is a counter, on a session length cycle, twenty minute periods. The **update** is average, so the profile adjusts the mean and standard deviation automatically. The **model** is deviation, looking for significant difference from the user's norm. The **weighting** is low, because the users may deviate from their routines somewhat on occasion and we are not interested in minor violations of this limit. If a trojan horse was activated invisibly, scanning the file system for executables which it might attach itself to, the transaction rate would soar and the feature warning count would go high.

Job Mix, Order Entry. Order entry at Acme is a dull job. The procedure is pretty much the same all the time: query the inventory database once or twice, fill out the order form online, and post an inventory transfer request. This feature looks for extreme variation in the routine. The **events** under scrutiny are the queries versus the order entry sequence. The **measure** is a ratio, on a 60 minute cycle, ten minute periods. The **update** is fixed, because we know the pattern. The **model** is a limit, fixed at 3; under no circumstances should the ratio of queries to orders exceed 3 over a ten minute period. The **weighting** is medium. If an intruder found his way into the system, and began doing a large number of queries into the inventory database, the feature warning count would jump.

3.6 Summary

A rule base for ICE is determined by defining the system security policy and finding discriminating features in user work patterns. This process is crucial to successful ICE installation, yet remains difficult and subtle. In general the content of the audit trail will affect the rule base.

The audit trail can be used to construct and update a distinctive profile for each user, characterizing both personal work habits and tasks normally performed on the job. Minor deviation from a work profile is more common and of less concern than a combination of violations. The more stable a user's system usage pattern is, the more likely that any aberrance from this pattern is evidence of intrusion or abuse. Users in a production environment more likely to exhibit these consistent and stable work patterns.

The audit data collected on most computer systems is not useful in its raw form for manually identifying intrusions. Existing commercial audit facilities are difficult to bypass, but are a possible burden for the target system; expect to negotiate with the system administrator.

The audit trail features described above cannot be derived from a quick reading of the audit records. The audit trail must be distilled, summarized, and cross referenced to derive useful information. Locating features which most discriminate users from intruders (or proper use from abuse) is more difficult still *(see Usage: Installation)*.

4 A Design: PUMICE

The design of an ICE usage monitor is demonstrated through presentation of a Proper Usage Monitor for Intrusion Countermeasure Equipment, or PUMICE. "Proper" implies compliance to system policy and user historical norms. Design objectives for PUMICE are:

Interactive Usage Analysis. A graphic and flexible user interface that alerts an operator to suspicious activity and facilitates effective investigation. A summary of system activity, policy violation, and anomalous user work patterns coupled with a flexible means in which to investigate anomalous activity – for instance the ability to graphically compare current versus profiled activity.

Evolving Profiles and Rulebase. Continuous aging and updating of user profiles to reflect evolution of the user work pattern. As the operator learns more about the system characteristics and the users' habits, the rulebase and profile templates may be fine tuned.

Minimal Impact on Target. No modifications to the target machine's operating system, and minimal modifications to target machine's auditing mechanism. There should be little performance degradation on the target machine.

The high level design (Figure 1) shows subjects as ovals and objects as boxes. Audit Format interprets the target system audit trail and enters it into the long term Audit Database. The audit records are checked against system policy by Policy Review and compared with each user's profile by Profile Review. Both review processes modify the session summaries in the Summary Database. A summary is created when a user opens a session and is updated until the session closes. The **Profile Updater** processes each normal session as it closes and updates that owner's profile in the Profile Database. Anomalous sessions must be manually approved before the profile can be updated. Extreme deviant behavior may activate Countermeasure. The operator can access all four of the databases, and may choose to initiate (or inhibit) countermeasures.

4.1 Audit Data

By using existing accounting and audit facilities, PUMICE invokes no changes to the target system. Because these facilities are well established (especially in production environments), they do not present a formidable political hurdle if a system without them already installed is encountered. Also, because these accounting facilities are old technology, they have settled deep into the operating system, and are reasonably robust and tamper proof in their data collection.

The audit trail produced by the target system will not be just as desired for our analysis engine. The Audit Format process transforms the raw audit trail to a more suitable format. Performing this work on PUMICE is more secure *(see Hardware Requirements.)*, lessens the impact on the target, and allows the monitor to be installed for various

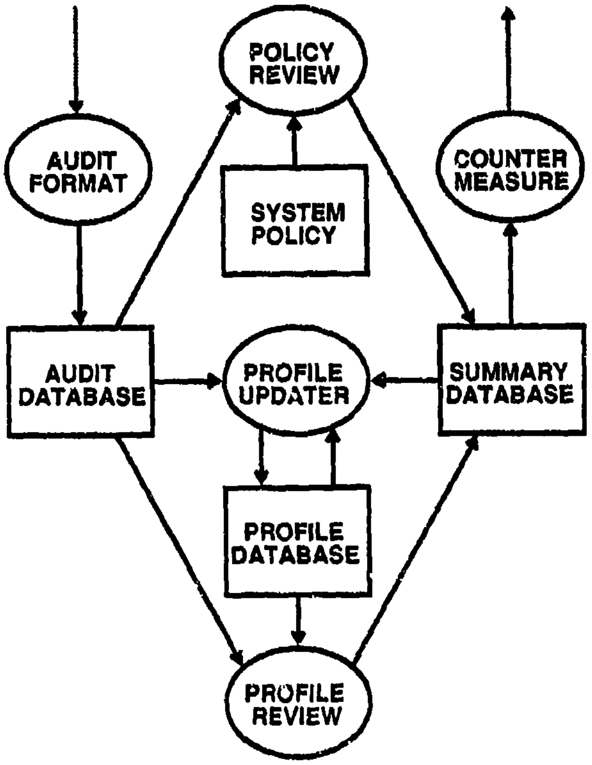


Figure 1: A high level design for PUMICE

machines with minor software modification. Sometimes an enhancement of the audit trail may be in order. On some IBM systems, the audit records are associated with a control job instead of a user; the Audit Format could associate job IDs with user IDs in the formatted audit trail. The information available in the audit trail will of course affect the nature of the rule base.

4.2 Hardware Requirements

PUMICE is designed to run on a workstation class machine with high resolution graphics, a mouse, and windowing support. These are requirements for this particular design, not ICE or this approach in general. The use of PUMICE as an investigative tool is greatly enhanced by a strong graphical interface. The hardware should have enough power to perform most or all of the analysis in real time for the best countermeasure capability. By running the security monitor on a separate processing platform, security is enhanced and the target system is not burdened. PUMICE must have a solid communications channel to the target system, since the analysis is only as strong as the audit trail.

4.3 Database

The database requirements differ substantially in two phases, system analysis/installation and monitor operation. During system analysis a large audit trail database is surveyed for usage patterns which discriminate between proper usage and abuse. Possible features which show a small variance over time are entered into the rule base. This phase is facilitated by the high level database query language SQL. During the operational phase, processing proceeds at a much lower level. Computational overhead inhibits use of database mechanisms and puts a high premium on usage summaries and simple indexing for retrieval of related records. Several database packages support both high level analysis and high speed indexed access to the same database records. For performance gains, the analysis engine may be implemented in more aerodynamic access method code instead of at the traditional SQL programming level (thus escaping all the requisite baggage of a full DBMS).

4.4 Analysis Engine

The analysis engine accesses the DBMS at a low level and checks current activity records against limits (absolute as from policy statement, or derived from the profile). Current user activity is compared statistically against the users' respective profiles and any deviations are reported to the PUMICE operator for review. The analysis engine updates profiles as high water mark, average, or deviation from the mean. System behavior is kept on a site, group, and user specific basis. The profiling must be initialized by running in a learning mode over an appropriate quantity of data. New users to the system need to be given a generic profile from users with similar job responsibilities, and watched closely as their profiles harden. User sessions and histograms are used as the basis for counts and percentages.

4.5 PUMICE Security Precautions

Although it is presumed that at most sites the system console will be physically secured, security features have been designed into PUMICE itself. Each operator is issued a unique account and password that is used for access mediation and for auditing the activity of the operators. An appointed system manager periodically reviews operator activity. Non-privileged accounts are dedicated to run only the monitoring software, and do not permit access to sensitive files or any other workstation resource. When an operator takes a break from active monitoring, he/she may lock the console screen without closing the session. This locked screen displays a non-specific dynamic indicator of the security of the target: if particularly anomalous activity occurs on the target, keyboard bells and the locked screen reflect that the immediate attention of an SSO is required. An SSO must re-enter his/her password to unlock the console. Locking of the console screen also occurs automatically if the keyboard/mouse remain idle for some configurable number of minutes. PUMICE maintains a complete audit trail of significant activity on the security monitor itself. The audit trail includes both SSO activity and frequent posting of the system and display status.

4.6 User Interface

The manner in which to best distill information on hundreds of users over thousands of audit records and tens of features to one console screen is directly related to the success that a PUMICE operator has in discriminating anomalous activity. Effective analysis of anomalous activity for intrusionary intent is dependent upon evaluating this activity against the assimilated knowledge of what is truly abnormal for the target system.

The design of the user interface must direct full attention to abstraction of information, human engineering and effective visual presentation, ease of use, and flexibility to meet differing environments. The powerful windowing environment afforded by current workstation technology is essential to an effective console screen interface. The following concepts optimize the human interface so that the SSO may quickly and effectively assess the security status of the host machine:

Abstraction of information. PUMICE may be viewed as an audit reduction tool, which implies that there will be activity flagged that is not alarmingly abnormal and which is not evidence of an intrusion. Suspicious activity is ordered on warning count, and site specified alarm thresholds enforced so as not to completely inundate the SSO with a console screen full of false alarms.

Direct access to target system audit trail. If an SSO believes that a user does represent a potential intruder, he/she is able to easily examine this user's activity in detail down to the transaction level (regardless of whether PUMICE believes it to be suspicious).

Multiple windows. Separate windows permit simultaneous access to system status, complaints of abnormal activity, and results of querying the database for investigative information.

Stockpiling events until manual release. The SSO can not be expected to be present at all times to immediately investigate flagged activity. PUMICE displays and saves all suspicious activity until the SSO directs a manual release. If the SSO comes back from lunch, he/she should be able to glance at the monitor console and quickly assess to what degree the system has been subjected to abnormal activity. Activity flagged as being suspicious is not used to update the user's profile unless the SSO manually releases it.

User files. A description of each user including: job description, department, physical description, location, phone number, manager, etc. is available. The SSO may append notations characterizing the user's usage habits.

Investigative capability. A flexible means in which to graphically compare current versus profiled activity, and the ability to issue manual queries against the database is available.

Online help. PUMICE provides access to localized help facilities from within each window. A new operator should require only a short supervised training period to become proficient.

PUMICE's windows are designed to quickly focus the operator's attention to the most suspicious activity on the target without distracting the operator with details about normal or only slightly out of profile activity. The windows use screen layout, color, and highlighting to display information in a quickly comprehensible form. Activity considered potentially serious enough to present a threat of system compromise is energetically highlighted to alert the operator to take immediate action and hopefully confront the user in question while still in the act or at least the facility. If the operator requires more information about a session flagged as demonstrating intrusionary activity, elements in the windows may be expanded by clicking on them with the workstation mouse. Feature weighting and system variables may be configured to meet the requirements of a particular installation.

Operations are separated into a number of windows that are initiated by clicking a mouse on the appropriately labeled button of a main option window. PUMICE may be considered to have two basic modes: a monitoring mode that displays status and security relevant information generated by current activity on the target system, and an investigative mode for when the SSO requires additional information from the target or from the audit DBMS in order to investigate discovered deviant behavior. Any combination of these windows may be opened on the console at one time and positioned or overlapped as the SSO desires.

4.6.1 Monitoring Windows

The Security Summary, Feature Grid, Selected Session, and User Data windows of this mode are closely tied to each other (Figure 2). When a cell on the Row column of the Security Summary window is mouse clicked, the related session row is highlighted in this window, as well as the matching column of the Feature Grid window. A summary of the features that have been flagged for the selected session is summarized in a Selected Session window, and information on the owner of that session is displayed in a User Data window.

Security Summary. This window displays an ordered list of sessions that are flagged as diverging from their owners' respective profiles. Sessions are flagged as suspicious and displayed in this window when the session count, total count, or any individual feature count exceed the threshold established by the system administrator.

The first field of each row displays an index number for that session. The second and third fields display the session ID and how long that session has been closed, or that the session is still active. The fourth and fifth fields display the warning counts accumulated for that session and for the owner over all sessions. The sixth field displays an interest value (Note) manually tagged to the owner of the session. The seventh field gives the ranking of that session over all current flagged sessions based on the session warning count. The Security Summary window is normally updated every 10 seconds, but this value may be adjusted by the administrator. The screen may be locked during investigation to prevent reordering of the display.

Ordering of the Security Summary may be based on session warning count, total warning count, session name (alphabetical), imposed interest level, or time of last warning count update. This window is normally configured to order the flagged session rows on session warning count, and may be scrolled up and down using the workstation mouse or keyboard. The ordering of the session list may be changed by clicking on any of the middle five column headings to cause reordering on that respective field (e.g., clicking on *Session Status* causes the sessions to be ordered on time of last warning count update). The ordering chosen is signified by highlighting of the respective column heading. Clicking on the SPLIT button in this window splits the twenty row list into two ten row lists, each with their own scroll bar mechanisms. Initially both new lists are exactly the same, displaying the top ten rows of the original twenty row list ordered in the same manner. Clicking on a column heading now, however, only affects the top list of ten sessions. Any pair of orderings is possible by choosing an ordering for the entire list, splitting, and then choosing a new ordering for the top half.

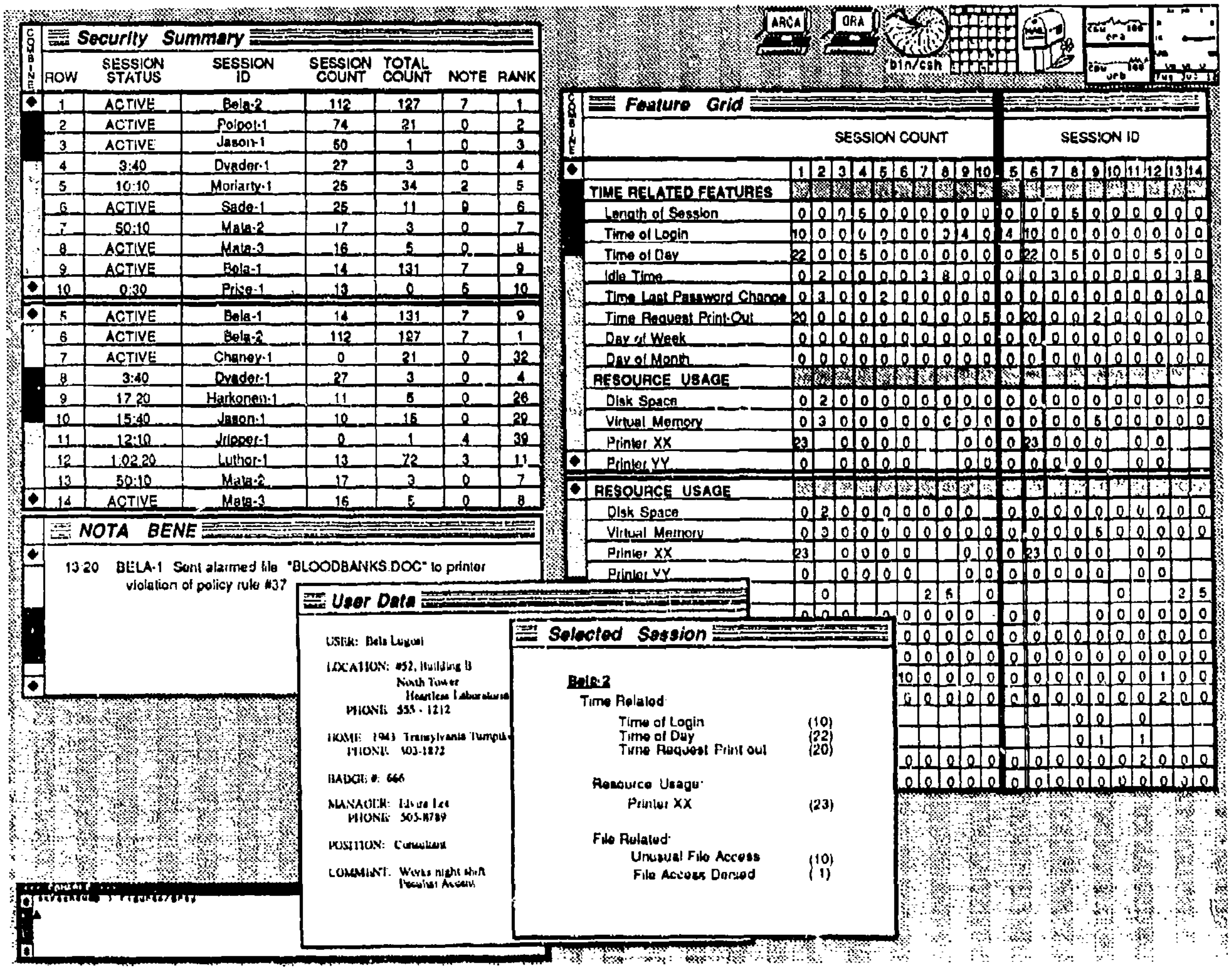
The PUMICE operator may investigate any flagged session, but the row in this window and the session data destined for the **Profile Updater** may not be *released* until that session is closed by the owner. A closed session may be released by clicking on the *Row* cell of a session. In releasing a session, the operator will have the opportunity to tag the owner with an interest value and attach comments to his record. Subsequent sessions by users with non-zero interest values will always be flagged regardless of whether they fall outside of profile or policy rules. This mechanism allows for increased monitoring of users who are new or who are thought to present an increased risk for external reasons. If the operator believes that the session reflects an intrusion attempt (particularly a masquerade), or if the session is verified to be atypical yet valid, the operator will not release the session data to the Profile Updater so as not to skew the user's profile.

Feature Grid. This window is a matrix displaying *features* down the left side and sessions displayed in the Security Summary window across the top. It consists of a grid made up of colored cells labeled with warning count values for each feature of the flagged sessions. The higher the feature warning count, the warmer the cell is painted. Sessions that have been selected and highlighted in the Security Summary window are also highlighted in this window. Clicking on a feature cell in this window opens a subwindow containing the results of a query to the database for a summary of the transactions in this session that increased that feature's warning count.

Selected Session. This window summarizes the flagged features of a session highlighted in the Security Summary and Feature Grid windows. Each feature listed specifies the last time that it caused an increase in that session's warning count due to transactions found to be out of profile by this feature. This provides immediate information on when intrusionary activity last occurred.

User Data. The User Data window displays general information about the owner of the currently selected session. This information includes such items as name, office location, phone number, manager, and authentication data. This window might even display a digitized picture of the user's face.

Nota Bene. This window acts as a warning window of activity that clearly violates established limits, either from system security policy or from the user work profiles. Explanatory text is printed to this



Security Summary

ROW	SESSION STATUS	SESSION ID	SESSION COUNT	TOTAL COUNT	NOTE	RANK
1	ACTIVE	Bela-2	112	127	7	1
2	ACTIVE	Polpot-1	74	21	0	2
3	ACTIVE	Jason-1	50	1	0	3
4	3:40	Dvader-1	27	3	0	4
5	10:10	Moriarty-1	25	34	2	5
6	ACTIVE	Sade-1	25	11	9	6
7	50:10	Mata-2	17	3	0	7
8	ACTIVE	Mata-3	16	5	0	8
9	ACTIVE	Bela-1	14	131	7	9
10	0:30	Prise-1	13	0	5	10
5	ACTIVE	Bela-1	14	131	7	9
6	ACTIVE	Bela-2	112	127	7	1
7	ACTIVE	Chaney-1	0	21	0	32
8	3:40	Dvader-1	27	3	0	4
9	17:20	Harkonen-1	11	5	0	26
10	15:40	Jason-1	10	15	0	29
11	12:10	Jripper-1	0	1	4	39
12	1:02:20	Luthor-1	13	72	3	11
13	50:10	Mata-2	17	3	0	7
14	ACTIVE	Mata-3	16	5	0	8

NOTA BENE

13:20 BELA-1 Sent alarmed file "BLOODBANKS.DOC" to printer violation of policy rule #37

Feature Grid

	SESSION COUNT										SESSION ID									
	1	2	3	4	5	6	7	8	9	10	5	6	7	8	9	10	11	12	13	14
TIME RELATED FEATURES																				
Length of Session	0	0	0	5	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
Time of Login	10	0	0	0	0	0	0	0	4	0	4	10	0	0	0	0	0	0	0	0
Time of Day	22	0	0	5	0	0	0	0	0	0	0	22	0	5	0	0	0	5	0	0
Idle Time	0	2	0	0	0	0	3	8	0	0	0	0	3	0	0	0	0	0	3	8
Time Last Password Change	0	3	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Time Request Print-Out	20	0	0	0	0	0	0	0	0	5	0	20	0	0	2	0	0	0	0	0
Day of Week	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Day of Month	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RESOURCE USAGE																				
Disk Space	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Virtual Memory	0	3	0	0	0	0	0	C	0	0	0	0	0	0	6	0	0	0	0	0
Printer XX	23		0	0	0	0			0	0	0	23	0	0	0		0	0		
Printer YY	0		0	0	0	0			0	0	0	0	0	0	0		0	0		
RESOURCE USAGE																				
Disk Space	0	2	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0
Virtual Memory	0	3	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0
Printer XX	23		0	0	0	0			0	0	0	23	0	0	0		0	0		
Printer YY	0		0	0	0	0			0	0	0	0	0	0	0		0	0		
		0					2	6		0					0				2	5
						0	0	0	0	-	0	0			0	0	0	0	0	0
						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
						10	0	0	0	0	0	0	0	0	0	0	0	1	0	0
						G	0	0	0	0	0	0	0	0	0	0	0	2	0	0
														0	0		0			
														0	1		1			
						0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
						0	0	0	0	0	0	0	0	0	0	0	0	0	J	0

User Data

USER: Bela Lugosi

LOCATION: #52, Building B
North Tower
Heartless Laboratorie
PHONE: 555 - 1212

HOME: 1943 Transylvania Turnpik
PHONE: 503-1872

BADGE #: 666

MANAGER: Elvira Lee
PHONE: 505-8789

POSITION: Consultant

COMMENT: Works night shift
Peculiar Accent

Selected Session

Bela-2

Time Related:

Time of Login	(10)
Time of Day	(22)
Time Request Print out	(20)

Resource Usage:

Printer XX	(23)

File Related:

Unusual File Access	(10)
File Access Denied	(1)

Figure 2: Sample Monitoring Screen

window chronologically. Upon review the operator can send individual nota bene items to be archived. It should be fairly rare that nota bene items are discovered, but a scrolling mechanism assures that all can be viewed.

4.6.2 Investigative Windows

Upon discovery of anomalous activity, the System Security Officer uses these windows to investigate (Figure 3).

Graphical Display. PUMICE supports the construction of graphical comparisons between current activity vs. profiled norms, bar graph displays of Deviation features, or strip chart histograms of Limit features such as number of transactions per minute. Specifications for different graphs may be drawn on top of each other and color coded.

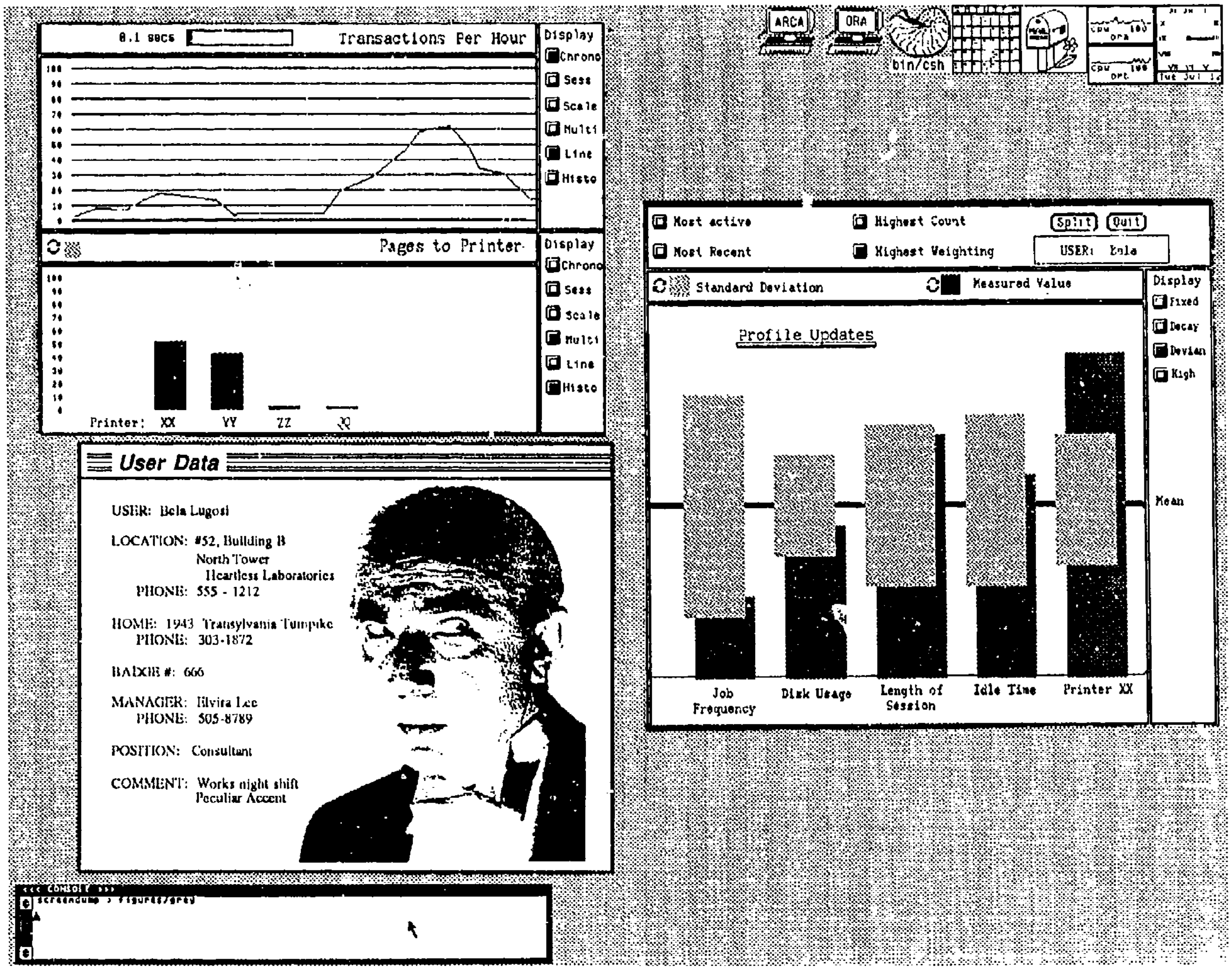
The following are possible graphs for a feature: over the user's current session, over a recent or long term profile of that same user, over that user's group's profile, over another user's current session or profile. Any of these might be compared against each other or the feature might be compared against another similar feature.

Database Query. This window permits direct interface to the DBMS. Canned queries are provided for expected common requests for analysis of past system usage. These include displaying a user's profile vs. current session parameters, displaying all audit records of a user from some starting time, and displaying reported anomalies by userID, warning count, and feature. Highly specific individualized queries into the database may also be issued from this window. This window is non graphical and more appropriate for such comparisons as terminal IDs against a user's profile, and for sequences.

Terminal Emulation. This window allows an operator to open a session onto the target machine from the PUMICE console. Investigation and necessary immediate action to prevent damaging intrusionary activity may be made directly from a (privileged) account on the target.

4.6.3 Other Windows

System Administration. This window permits the PUMICE administrator to configure intrusion threshold values, the time between screen updates of the Security Summary and Feature Grid windows, length of idle time before the console is locked, update times of features, transactions that will always be flagged as possible security hazards, specifying users whose sessions will then always require manual release regardless of their accumulated warning count, fields to be masked in the audit records coming from the target machine that have no applicability to this target machine, the number of raw audit data records that are to be accumulated before records are copied to archive, and initialization profiles. Database maintenance tasks are also performed using this window. Note that the PUMICE administrator has control over the functionality of the security monitor; it



User Data

USER: Bela Lugosi

LOCATION: #52, Building B
North Tower
Heartless Laboratories
PHONE: 555 - 1212

HOME: 1943 Transylvania Turnpike
PHONE: 303-1872

BADGE #: 666

MANAGER: Elvira Lee
PHONE: 505-8789

POSITION: Consultant

COMMENT: Works night shift
Peculiar Accent

Figure 3: Sample Investigative Screen

may be desirable to separate the operator and administrator jobs at some sites.

Memo and Report. This window supports the informal noting of expected future deviant behavior as foretold by the user *("I'll be working over the weekend.")* or circumstance *("Group X will be exchanging office space with Group Y, so expect many terminal ids to be flagged as abnormal for members of these groups")*. This window also supports the more formalized reporting of witnessed abnormal behavior or the status of ongoing investigations from one shift's SSO to the next, or external information *("Employee A. Waters resigned and will leave at the end of this week")*. It is expected this latter reporting has representative usage standards in the traditional facilities guard arena.

Status. The Status window provides general operational information about PUMICE and the target system and is designed to display information that is not constantly required for the security functions. This reduces the clutter and optimizes the performance of the other security monitoring windows. The Status window is normally examined when the operator returns from an absence and wants to check that both PUMICE and the target system have been operating without problems.

Information displayed in the Status window includes the date and time that the security monitor became operational, the time that the last console activity occurred (other than activity within this window), and the time that a Status window was last invoked. Statistics on the total number of audit records received, the total number of audit records rejected (because of line noise or encryption problems), and the total number of audit records that have been processed, and the number of audit records received, rejected, and processed since the last time a Status window was opened are displayed. The window also displays information on the target machine, operating system, and particular configuration, and how long the target system has been operational.

5 Usage

This section describes how to use a security monitor like PUMICE once it is fully implemented and connected to a live target. A version of this intrusion countermeasure equipment may be tailored to a wide range of targets if an appropriate rulebase is constructed. We envision ICE applications for Unix development environments and IBM mainframe MIS and dedicated airline reservation systems. ICE is even an appropriate tool to monitor the security of a network: with the narrowed functionality and more tightly constrained usage standards of a network, the profiling would be simpler and more reliable.

5.1 Installation

Installing PUMICE requires establishing an initial rule base, formalizing system policy, entering system data, collecting initial profile information, setting thresholds, and defining generic profiles by job description and group. The trail from an appropriate period of auditing of the target system will need to be run through PUMICE to be *learned.* Additional tuning using SQL will be required to customize PUMICE's understanding of what is desired to be considered normal for the target.

The initial installation should not be expected to fit perfectly with the target system and user base. False alarms are normal until adjustments, both manual and automatic, are well underway. Features with no alarms should be reviewed, even if expected to be good discriminators, as the limits may be set too high for appropriate triggers.

The security monitor will increase in effectiveness the longer that it is operational because the user and command profiles develop. The local system security officer also continues to gain experience and become more proficient at the "art" of distinguishing innocuous abnormal system activity from truly offensive activity for this site. The experience the SSO's gain from their investigations will gradually become embedded in the system policy and profile templates as the rulebase is fine tuned. Training of new operators to the nuances of PUMICE and site specifics is important.

5.2 Operation

ICE should be expected to change and adapt as the targeted system and the activity on it changes. The initial rule base and thresholds should also be expected to develop and mature as the administrator gains experience with the user base and job mix on the target system.

Intrusion attempts may progress slowly over months of activity. The bookkeeping of separate events of deviant behavior in the Memo and Report Window may well prove valuable as an evidence gathering tool and as an aid to subsequent legal action.

5.3 Vulnerabilities

The PUMICE operator must be well versed in the vulnerabilities inherent to the adaptive user work profile approach and the dependence on the integrity of the target's audit data. *Weak profiles* allow too much latitude for the user work patterns. In addition to attentive installation of limits and thresholds, the SSO must be cognizant of how the profiles change over time. The investigative windows support display of the profile limits, averages, and deviations. These displays should be cross checked with a variety of users to detect profiles that have worn loose over time. Individuals who must have a loose profile due to the variety of their work should be watched more carefully because their accounts are more vulnerable to intrusion and masquerade.

The sophistication of an intruder is of course a major factor in the probable success of an attempted intrusion. The following ordering may be made on increasing potential that an intruder can keep within the profile associated with an account: familiarity with system, familiarity with particular installation, familiarity with particular user account, familiarity with particular usage pattern.

The central concept of our approach to audit trail analysis is that intrusion and insider abuse look different than established legitimate use. This admittedly will not be the case for *all* incidents of intrusion or abuse. Our experience indicates that artful development of system and user profiles produces a security monitor applicable to a broad range of systems, detecting many different kinds of misuse or intrusionary attacks. Detection of intruders is more certain on machines with a stable and consistent user base.

5.4 Countermeasure Capability

When ICE discovers anomalous activity it may be instructed to respond with different levels of autonomy. We have presented PUMICE as foremost an investigative tool that first alerts the SSO to sessions with summaries of anomalous activity and then provides him/her with the tools for further investigation. PUMICE is not designed to normally run unattended with expert system software making Orwellian decisions about proper usage. Our approach does not assume that anomalous activity will always be understood entirely from data available on the target system. We believe that external verification such as phone calls to the offenders *("George, is that you working late?")*, their managers *("Has Jane been assigned work on Project X?")*, etc. will be standard procedure.

Nevertheless, many sites may desire ICE with the capability to take action on its own if system activity attains an overwhelming level of deviance and there is no human authority available. If ICE runs unattended at night it may be instructed to take action to prevent system compromise while attempting to alert off-site personnel. The following is a list of possible responses:

- Challenge the user to confirm identity.
- Slow system response.
- Pretend to execute commands.
- Lock the account.
- Lock the entire system.

5.5 Privacy

PUMICE is bound to raise privacy issues especially if users learn of the active profiling. Privacy is an especially valid concern when macroscopic conclusions about users become readily apparent ("J. Biafra isn't as fast a worker because he posts 70% fewer transactions than H. Rollins."). A possible solution to this is to map usernames to pseudonyms up until intrusionary activity forces unveiling. Legalities may be addressed by having users sign consent forms when they apply for the account.

6 Related Work

Auditing computer systems and reviewing the resulting trails has a well established history – though primarily for the purpose of accounting and job charging purposes, and generally by manual means. In the past year, interest in intrusion detection has increased greatly with a number of separate efforts being funded. In March, 1988, SRI International facilitated round table discussion between these efforts by hosting a workshop: there was a healthy exchange of ideas and further workshops are scheduled.

Sytek Automated Audit Analysis This project was conducted in 1984-5 and was funded by the Space and Naval Warfare Command. It used audit data collected at the Csh level of a research environment UNIX machine. Sytek's research established the legitimacy of profiling user work patterns constructed from simple and readily collectible audit data. It demonstrated the ability to discriminate between normal and abnormal system usage[6][7][8]. This project also showed the utility of using database tools for hand analysis, prototyping, and investigation of suspicious activity.

A set of *features* was defined, with each feature, made up of one or a combination of audit data parameters, describing an aspect of system usage. An Automated Audit Analysis tool was developed using an Ingres DBMS to evaluate the viability of user profiling and to choose the feature set. Hard ranges from a "learning period" were used instead of ongoing profile updating and statistical analysis. A set of intrusion scenarios was developed and performed on the audited machine, and the resulting data from the simulated intrusion attempts examined.

SRI Real-Time Intrusion Detection SRI's Intrusion Detection Expert System effort showed the practicality of audit trail analysis performed in real time on a separate, dedicated computing platform[5][9]. Modifying a TOPS-20 operating system to collect audit data, the SRI implementation examined three features: source terminal location, length of a user's session, and number of sessions in each of three work shifts, and displayed the amount of anomalous activity system wide. Analysis and reporting is organized by feature rather than by user. Their results show that much can be gained from modest data collection efforts. The effort found a graphical interface to be an effective means of highlighting intrusions.

Tracor Haystack Tracor Applied Sciences, Inc. is developing an audit trail analysis system called Haystack for the Air Force Cryptologic Support Center at Kelly Air Force Base. Audit trail volume is about one million events per week generated by Sperry 1100/60 mainframes running 1972 vintage operating systems. Haystack processes audit trail events using statistical measures and pattern recognition, and classifies violations with deterministic and heuristic rules.

TRW Discovery TRW Information Services Division is developing Discovery, an expert system based design for "detective and preventive control in the on-line environment"[11]. TRW's goal is to review daily inquiry activity and detect unauthorized inquiries (out of some 400,000 inquiries per day from a base of 120,000 customer access codes). TRW has found Discovery to be useful for non-security purposes such as marketing and risk analysis.

NCSC MIDAS The National Computer Security Center is developing the Multics Intrusion Detection and Alerting System (MIDAS) to take audit data from a Multics system and compare it to statistical user profiles maintained in LISP structures. Forty heuristic rules in an expert system shell define straight limits, expected user behavior, expected system-wide behavior, and sensitive command sequences (similar to known attacks). MIDAS runs on a Symbolics machine using archived audit tapes.

7 Conclusions

A standalone security monitor with adaptive user work profiles and policy rules is a powerful, low-impact means for addressing the threat of intrusions and insider abuse. By using existing auditing facilities, disruption of the target operating system is avoided, and demands on host performance and resources are minimized. Implementing the ICE on a separate workstation platform provides security advantages as the audit data is immediately moved off of the target system, and the monitor software is isolated.

The more narrowly directed the activity on the audited system and the more regulated the user population, the more stable the resulting user profiles will be; a production environment presents less of challenge to this approach than does a research environment. Sophisticated profiling is facilitated by an information rich audit trail, but even very limited audit trails have been shown to be successful in flagging intrusionary work patterns.

The intelligent compounding of feature violations offers much more information than the respective individual component elements. The system's discriminatory capability will improve over time as the user profiles mature. Statistical methods to mimic the evolution of a user's work profile over time and job task changes are necessary for adaptive profile updating.

Profiling will not detect a break-in if the intruder does not diverge noticeably from the account's normal pattern of system usage, or if the host auditing mechanism is subverted. The ICE may not be able to discern an intrusion that is performed so gradually that the user's work pattern is never divergent enough from his/her evolving profile so as to be found suspicious. The ICE approach will not be able to catch an insider familiar enough with normal patterns and the data elements that might be audited, that he/she is successful in avoiding exceeding thresholds.

The design presented for a system called PUMICE demonstrates the summary and investigative aspects possible with this approach. Techniques to best condense a large amount of analysis to abbreviated displays of anomalous activity that are held for later review are aided by the window and graphical capabilities of workstations. Interactive tools enable immediate investigation of anomalous activity. Knowledge of active profiling may in itself act as a deterrent.

> "Bobby was a cowboy, and ice was the nature of his game, ice from ICE, Intrusion Countermeasure Electronics... Legitimate programmers never see the walls of ice they work behind, the walls of shadow that screen their operations from others, from industrial-espionage artists and hustlers like Bobby Quine.... Bobby was a cracksman, a burglar, casing mankind's extended electronic nervous system, rustling data and credit in the crowded matrix, monochrome nonspace where the only stars are dense concentrations of information, and high above it all burn corporate galaxies and the cold spiral arms of military systems."
> - *Burning Chrome*, William Gibson

References

[1] J.P.Anderson, *Computer Security Technology Planning Study*, ESD/AFSC, Hanscom AFB, Bedford, Mass., October 1972.

[2] Allen R. Clyde, "Insider Threat Identification Systems," *Proceedings of the 10th National Computer Security Conference*, September 1987.

[3] Dorothy E. Denning and Peter G. Neumann, *Requirements and Model for IDES - A Real-Time Intrusion-Detection Expert System* Technical Report, Computer Science Lab, SRI International, August 1985.

[4] Dorothy E. Denning, "An Intrusion Detection Model," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987, pp. 222-232.

[5] Dorothy E. Denning, David Edwards, R. Jagannathan, Teresa Lunt, and Peter G. Neumann, *A Prototype IDES - A Real-Time Intrusion-Detection Expert System*, Final Report, Computer Science Lab, SRI International, August 1987.

[6] Lawrence R. Halme and John Van Horne, *Analysis of Computer System Audit Trails - Final Report* Sytek Technical Report TR-86007, Mountain View, California, May 1986.

[7] Lawrence R. Halme, Teresa F. Lunt, and John Van Horne, "Results of an Automated Analysis of a Computer System Audit Trail," *Proceedings of the Second Annual AFCEA Physical & Electronic Security Symposium and Exposition*, Philadelphia, Pennsylvania, August 1986.

[8] Lawrence R. Halme and John Van Horne, "Automated Analysis of Computer System Audit Trails for Security Purposes," *Proceedings of the 9th National Computer Security Conference*, September 1986.

[9] Teresa F. Lunt and R. Jagannathan, "A Prototype Real-Time Intrusion Detection Expert System," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[10] Teresa F. Lunt, "Automated Audit Analysis and Intrusion Detection: A Survey," *Proceedings of the 11th National Computer Security Conference*, October 1988

[11] William T. Tener, "Discovery: An Expert System In the Commercial Data Security Environment," *Proceedings of the IFIP Security Conference*, Monte Carlo, 1986.

USE OF AUTOMATED VERIFICATION TOOLS IN A SECURE SOFTWARE DEVELOPMENT METHODOLOGY

Margaret Murray, Roxanne Berch, Sally Caperton,
Phil Costello, Alexander Enzmann, and Frank Minjarez

COMPUSEC, Inc.
4909 Murphy Canyon Rd., Suite 500
San Diego, CA 92123
(619) 260-1881

ABSTRACT

In performing verification tasks on several different secure software projects, the authors have been required to address issues concerning software quality, size and complexity. Many lessons were learned in the course of these efforts about how to efficiently specify and verify operational systems. Additionally, while evaluating characteristics of both programming and specification languages, we have identified syntax and style that either enhances or obscures security analysis. In the real world of large, complex systems where documentation often provides imperfect inputs to the verification process, we have devised methods for clarifying specification style, automating security analysis, and improving the communication that must occur between designer and verifier. Much of this work has focused on the use of Ada both as a PDL and as an implementation language.

The authors used the COMPUSEC Verification Toolset to formally verify both the Army/Air Force AN/GSC-40 Series Command Post Terminals, and the Army's Regency Net system (under contract to Magnavox's Northern Virginia Systems Division). For Regency Net, this involved generation and then analysis of a hierarchy of software design documentation consisting of three tiers of specifications as required by MIL-STD-490. Ada was used as a program design language at both the B5 and C5 levels of specifications. A combination of automated and manual methods were used to rigorously analyze Ada PDL. For this system, the "distance" between the verified C5 and the Pascal implementation code was very small. This leads us to believe that the subset of Ada analyzed in this C5 PDL could be expanded into one sufficiently rich to be used for verification of Ada implementation code. We are currently pursuing this line of research under Air Force Small Business Innovation Research (SBIR) contract F19628-86-C-0203.

APPLICABILITY of AUTOMATED VERIFICATION TOOLS

We have developed a methodology for formal verification of MLS properties based on the HDM methodology and the work of Richard Feiertag [1]. Variations of this methodology are being used to verify both the AN/GSC-40 Series Command Post Terminals and Regency Net to achieve levels of assurance approaching those described at the A1 level of the Criteria [2]. This methodology has been applied to the preliminary design, detailed design, and deployment life-cycle maintenance phases of system development. Several iterations of a combination of automated and manual steps were used to find logical inconsistencies in design documents at each phase. Extension of this methodology to cover both MLS and proof-of-correctness analysis of either Ada PDL or source code is currently under development [3].

Verification methodology development has been an evolutionary process. Automated portions were developed in an attempt to circumvent both human and resource limitations while meeting project deadlines. Manual efforts required comprehensive training and were best applied to fails analysis. Both extensive explanations of the causes of failed proofs as well as justification of the methods employed were often required in the face of a skeptical attitude towards the worth of formal verification. The timeliness and relevance of both input documentation guidelines and output discrepancy/failure reports were also often at issue. Much has been learned in the process.

We believe that the feedback loop between software designer and verifier must be shortened so that more iterations of the verification process can economically occur at each stage of the software life-cycle. It is expected that responses to security feedback reports will act to increase software quality assurance while reducing cycles of formula generation and therefore ultimately reducing the number of failed proofs. Once fails have been located, it is the job of fails analysis to then quantify the bandwidth of the channels discovered and determine the degree of risk vs. the cost of a fix.

Automated tools are particularly valuable when they allow achievement of greater accuracy and throughput than is possible using manual analysis. We utilize automation at a variety of points in the overall verification process. Descriptions of several currently used elements of our verification toolset follow:

VERIFICATION TOOL DESCRIPTIONS

COMPUSEC-Enhanced HDM. The HDM tool set (also called "old HDM") was originally developed by SRI International. Initially intended to structure the overall software development process, it has found a specific application in MLS security analysis. HDM includes the language SPECIAL (SPECIfication and Assertion Language), a theorem prover, and the MLS formula generator for multilevel security [1]. We have ported HDM from the Digital Equipment Corporation's TENEX operation system to VAX/VMS. We also modified the user interface to the MLS tool, added a label propagator, and corrected how the Verifier counts failed proofs.

ATOS. ATOS (Ada-to-SPECIAL) works as a security cognizant Ada parser/translator. In addition to translating a subset of MIL-STD-1815A Ada into an FTLS (Formal Top Level Specification) written HDM's specification language SPECIAL, ATOS generates valuable discrepancy reports that provide feedback relevant to software quality assurance. Problems identified include:

- Parsing problems
- Undefined subprograms
- Undefined types
- Undeclared variables
- Type mismatches
- Formal and actual parameter mismatches
- Illegal assignments to constants
- Missing referenced files

ATOS also identifies local and global scopes of data items in preparation for further processing by the Labeler tool. ATOS handles security ramifications of Ada's modularity: The constructs "with" and "separate" are addressed in a consistent way by searching both current and configuration management directories for the referenced files. Once a match is found, it is then instantiated into the correct scope. To account for capabilities not easily represented in Ada, ATOS recognizes annotations in the Ada PDL. Annotations include representations of the data items transported in low-level procedures and the locations of referenced files.

BTOS. BTOS (Bubble-to-SPECIAL) is used for data flow analysis and produces FTLS from data flow diagrams [4]. Data flow diagrams (DFDs) can be developed from the top-level PDL in order to show the definition of the functional interfaces of the system. Given DFDs that reflect all information flow for these interfaces BTOS identifies the following:

- The shortest path between two nodes
- All possible paths between two nodes
- Paths between two nodes that contain labeling conflicts

Discrepancies are identified as they occur, and DFD components can be adjusted dynamically. When labeling conflicts occur, BTOS examines the security attributes of the entities involved and either relabels them through use of a label propagation algorithm, or shows an unresolvable conflict. If only external inputs and outputs of a module are given labels, the setting of all internal labels to the system's least-dominant level followed by propagation of these labels across the DFD allows BTOS to find the least-dominant conflict-free set of labels of all entities in the module under analysis.

TAT. TAT (Trustedness Analysis Tool) is used to determine allocation of trust in a secure system design. Given an input of formatted tables representing all possible paths between components in the system, TAT will identify which components of the system need to be trusted with respect to secrecy and integrity. The formatted tables that are used by TAT are consistent with the Ada PDL and reflect all inter-component flow of data specified in the PDL. TAT performs a data flow analysis of these tables that includes checking of inter- and intra-component I/O consistency as well as consistency with Data Dictionaries. TAT determines which design modules potentially violate the system security policy and therefore must be trusted to carry out their functionality in a secure manner. Other modules are secure by induction.

Labeler. The Labeler accepts as input an unlabeled SPECIAL FTLS and local and global Data Dictionaries, and produces labeled FTLS as well as discrepancy reports. It determines the scope of data items and assigns correct labels to them based on definitions found in the local and global dictionaries. Discrepancy reports identify problems with consistency in the Data Dictionary itself, as well as disconnects between the Dictionary and the FTLS. If errors of omission or commission exist between records and their components, the Labeler flags these cases and then relabels record components according to rules which preserve the meaning of the security model. The Labeler can also format the Data Dictionary and its security labels into a file suitable for processing by the TAT tool.

Labeler-Propagator. We added a propagator tool to HDM's environment for use in conjunction with HDM's MLS tool. The Propagator assigns security labels to data items not assigned fixed labels in such a way as to minimize security conflicts. The Propagator assigns default security labels to any data items within the FTLS which were not already assigned labels by the Labeler. It then uses a data-dependency recognition algorithm and a property violation algorithm equivalent to those implemented by Compusec-Enhanced HDM in recognizing potential security violations. The Propagator writes a new FTLS file in which all data items have been propagated. It also reports labeling conflicts that occur any time data flows between data items with incompatible security labels. Use of the Propagator has proven to be quicker and more accurate than previous efforts using manual propagation.

Splitter. The Splitter has enabled us to handle the formal verification of a large system in a timely manner. If an FTLS module is too large to be processed by HDM within system memory constraints, it must be broken down into appropriate subunits. The Splitter allows verifier-defined limits to be put on the size of these units while maintaining an accurate representation of the original large module. The Splitter correctly preserves scoping and interdependencies and produces the following:

- Hierarchical Structure - This shows how the module is split and enables ease in seeing relationships between high and low level procedures.
- Recursion Identification - HDM's MLS tools do not accept recursive functions. Instances requiring manual response are identified.
- Smaller SPECIAL Files - Resulting files are not only more manageable, but they remain parseable.

STOF. The Source-to-Formulas tool provides an integrated verification environment that operates on Ada code or PDL. STOF directly translates code or PDL into provable formulas. It also translates security policy into axioms and rules. The security policy to be applied to analysis of the code or PDL can specify desired system behavior with respect to either multilevel security or correctness.

Multilevel Security Concept. STOF's verification of MLS properties is based on use of its security-cognizant parser in conjunction with a labeling mechanism. The parser extracts type and scoping information from the source input and creates unique names that are formatted in its parse tree. The parser recognizes subjects and objects and performs path analysis to determine the shortest path between any subject/object pairs. The collaborating labeling mechanism allows data dictionary information to be added to path information. The data dictionary used supplies all fixed security labels. Labels themselves may be multi-attribute and of a project-specific format. The labeling mechanism transcribes the fixed labels to each instance of that data item found in the parse tree. Once all fixed labels have been transcribed, remaining unlabeled or floating label data items are subjected to label propagation. Propagation is a multi-pass operation that attempts to find the least dominant conflict-free path between all subject/object pairs. The number of propagation passes corresponds to the radius of a call graph of the system. Unresolvable conflicts are flagged as potential security violations.

Proof-of-Correctness Concept. STOF's handling of proof-of-correctness is based on definition of a desired transformation that can be described using formal notation. Source is analyzed by deriving and then formally notating its properties. Proofs involve demonstrations showing that valid translations of source properties imply only the desired transformation.

STOF Components and User Interface. STOF is currently implemented in SUN/Ada and runs on a SUN Microsystem Model 3/60. Inputs, outputs, and major components are shown in Figure 1. Low-level commands to STOF treat operations as predicates to be evaluated. Although a PROLOG syntax and semantics are sufficient to describe all operations, a visually-oriented, window-based user interface is under development for ease of use.

Example: Star Network Simulator (SNS). STOF has been tested and demonstrated by using it to verify a small network simulator program. SNS models identical terminals as an array of Ada tasks managed by a terminal controller that is a single Ada task. Terminal tasks request a security level at login. From this point on, the terminal controller handles message input and output in a secure fashion and issues an audit trail. The current simple security model only allows terminals at the same level to pass messages. A known security violation exists in the SNS in that all message passing occurs using the same routine WRITE_WIRE. The "wire" is unprotected and therefore has been labeled at level "1" (low). When SNS handles messages at level "2" (high), a violation occurs because messages are sent over the level "1" wire. STOF discovers and reports this by processing input SNS source code and a simple data dictionary. Figures 3 - 6 contains fragments from SNS source and STOF output that demonstrate discovery of this security violation.

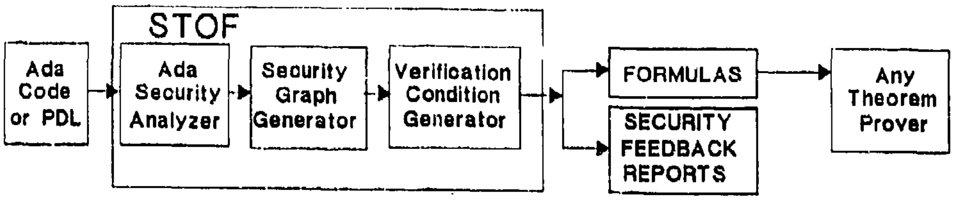


FIGURE 1. STOF Inputs, Major Components, And Outputs

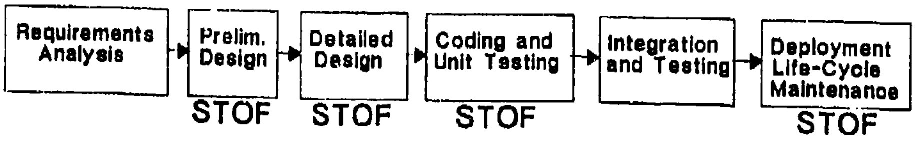


Figure 2. Life-Cycle Applicability of STOF

```
--// WRITE_WIRE:
--//
--// Writes data to the simulated wire, in a real
--// application this routine would interface to a
--// low level output routine.
--//
--// MESSAGE is output to the DESTINATION_TERMINAL
--// wire.
--//

procedure WRITE_WIRE (
                DESTINATION_TERMINAL : in LONG_INTEGER ;
                MESSAGE : in SHORT_STRING ) is
begin
  WIRE (DESTINATION_TERMINAL) := MESSAGE ;
end WRITE_WIRE ;

STOF PARSE TREE:

procedure_body(write_wire,null,
[in((destination_terminal),name(long_integer),null),
 in((message),name(short_string),null)],
              [],
[
  statement(null,
    name(wire,args(name(destination_terminal)))
                               := name(message))],
              null)
```

FIGURE 3. Ada Source and STOF Parse Tree for Procedure WRITE_WIRE

```
function FIND_ACCESS (MESSAGE : in SHORT_STRING)
                                return LONG_INTEGER is
RESULT  : LONG_INTEGER ;
begin

case MESSAGE (2) is

when '1' =>
  RESULT := ACCESS_ONE ;

when '2' =>
  RESULT := ACCESS_TWO ;

when others =>
  RESULT := NO_ACCESS ;

end case ;

return (RESULT) ;

end FIND_ACCESS ;

STOF PARSE TREE:

function_body(find_access,(long_integer,null),
[in((message),name(short_string),null)],
[variable_decl((result),name(long_integer),null)],
[
  statement(null,
    case(name(message,args(integer(2))),
      [
        ((character(49)),
          [
  statement(null,name(result) := name(access_one))]),
        ((character(50)),
          [
  statement(null,name(result) := name(access_two))]),
        ((others),
[statement(null,name(result) := name(no_access))])])),
 statement(null, eturn_statement(name(result)))],
            null)
```

FIGURE 4. Ada Source and STOF Parse Tree for Procedure FIND_ACCESS

```
--// PROCESS_SEND_MESSAGE:
--//
--// Determines if a send is valid, in this simple model only
--// terminals at the same level are allowed to pass messages.
--//
procedure PROCESS_SEND_MESSAGE (
                    SOURCE_TERMINAL : in LONG_INTEGER;
                    MESSAGE : in SHORT_STRING) is
DESTINATION_TERMINAL : LONG_INTEGER ;
begin
  DESTINATION_TERMINAL := CHARACTER'POS (MESSAGE (2)) -
                                  CHARACTER'POS ('0') ;
  if (DESTINATION_TERMINAL < MIN_TERMINAL) or
      (DESTINATION_TERMINAL > MAX_TERMINAL) then
    PUT_LINE ( "Server: reads bad destination address" &
        LONG_INTEGER'IMAGE (DESTINATION_TERMINAL) &
        " from terminal" &
        LONG_INTEGER'IMAGE (SOURCE_TERMINAL)) ;
        SEND_BAD_DESTINATION_MESSAGE (SOURCE_TERMINAL) ;
  elsif (TERMINAL_ACCESS (DESTINATION_TERMINAL) =
      TERMINAL_ACCESS (SOURCE_TERMINAL)) and
      (TERMINAL_ACCESS (SOURCE_TERMINAL) > NO_ACCESS) then
    SEND_FROM_HEADER (SOURCE_TERMINAL,
                    DESTINATION_TERMINAL) ;

--// This message is okay to send to the destination
--// terminal. Server puts the message on the wire

    WRITE_WIRE (DESTINATION_TERMINAL, MESSAGE) ;

--// server notifies the destination terminal of the
--// incomming message with the entry point "receive_message".
--// A real system might use handshake lines, or other
--// signals here.

    TERMINALS (DESTINATION_TERMINAL).RECEIVE_MESSAGE ;


  else
    PUT_LINE (
      "Server: Terminal" &
      LONG_INTEGER'IMAGE (SOURCE_TERMINAL) &
      " to terminal" &
      LONG_INTEGER'IMAGE (DESTINATION_TERMINAL) &
      " send message request denied because of access violation."
      ) ;
    SEND_NOT_ALLOWED_MESSAGE (SOURCE_TERMINAL) ;
  end if ;

end PROCESS_SEND_MESSAGE ;
```

FIGURE 5. Ada Source for Procedure PROCESS_SEND_MESSAGE

```
% parse('demo.ada',X).
X = [null,
      procedure_body(demo,null,[],
         [
  constant_decl((min_terminal),name(long_integer),integer(0)),
                                                     .
                                                     .
More?
no
% consult('demo_translations').
yes
% consult('demo_labels').
yes
% propagate(X).
Analyzing flows for: [demo,controller,process_message]
Analyzing flows for: [demo,controller,find_access]
Analyzing flows for: [demo,controller,process_send_message]
Analyzing flows for: [demo,controller,write_wire]
Analyzing flows for: [demo,controller]
Analyzing flows for: [demo,terminal_task]
Analyzing flows for: [demo]
Pass:  1
                                                     .
                                                     .
Get label for: object(message,[demo,controller,write_wire])2
Get label for: object(wire,[demo])1
Comparing: message(2) -> wire(1)
**** Unresolvable conflict **** :
     message, declared in [demo,controller,write_wire], label 2 ->
     wire, declared in [demo], label 1
Get label for: object(kl,[])0
Get label for: object(destination_terminal,
                 [demo,controller,process_send_message])1
Comparing: kl(0) -> destination_terminal(1)
                                                     .
                                                     .
Label changes:  9
Unresolved   :  1
Pass:  2
                                                     .
                                                     .
Label changes:  3
Unresolved   :  1
Pass:  3
                                                     .
                                                     .
Label changes:  0
Unresolved   :  1
X = propagate_results([
        label_change(object(terminal,[demo,_send_bad_destination_message]),1,2),
        label_change(object(destination_terminal,[demo,_send_from_header]),1,2),
        label_change(object(source_terminal,[demo,_send_from_header]),1,2),
        label_change(object(terminal,[demo,_access_level_change_message]),1,2),
                                                     .
                                                     .
        label_change(object(my_terminal_number,[demo,terminal_task]),1,2),
        label_change(object(terminal,[demo,_create_terminal]),1,2),
       label_change(object(console,[]),0,2)],
      [
        object(message,[demo,controller,write_wire]) -> object(wire,[demo])])
More?
no
% logout.
```

FIGURE 6. EXTRACTS from STOP MLS Analysis of SNS

CONCLUSIONS

Secure system development has only recently become a practical, applied science [5]. Two major categories of methodology enhancement need to be pursued: 1) The semantics and logical consequences of certain Ada statements with respect to both MLS and proof-of-correctness properties must be specified. 2) Additional automated verification support tools that shorten the designer/verifier feedback loop must be developed. Achievement of these objectives will result in the means to both quantify and evaluate the level of assurance of operational software development projects requiring higher reliability and quality.

REFERENCES.

[1] Feiertag, Richard J. "A Technique for Proving Specifications Are Multilevel Secure." Computer Science Laboratory Report CSL-109. Menlo Park, CA: SRI International. January 1980.

[2] DoD 5200.28-STD. Trusted Computer Systems Evaluation Criteria. Fort George Meade, MD: National Computer Security Center. December 1985.

[3] Enzmann, A.; Mascherin, B.; Minjarez, F. and Murray, M. "Secure Software Verification Tools Phase I Final Report." Contract Number F19628-86-C-0203. Hanscom AFB, MA: ESD/SYC-2. February 1987.

[4] Enzmann, Alexander. "Examination of Multi-level Security from Data Flow Graphs," in Proceedings of the AFCEA Conference on Electronic and Physical Security. Philadelphia, PA. August 1985.

[5] Griffiths, Georgia; Murray, Margaret; and Norton, William F. "Engineering Systems for Embedded Multilevel Secure Applications: Lessons Learned from the Regency Net Program," presented at the Third Aerospace Security Conference. Orlando, FL. December 1987.

STATIC ANALYSIS TOOLS FOR SOFTWARE SECURITY CERTIFICATION

D. Richard Kuhn

National Bureau of Standards
Gaithersburg, Md. 20899
301/975-3337
kuhn@icst-se.arpa

ABSTRACT

This paper describes a suite of tools used in evaluating software for security certification. The tools are currently being used on software for secure Electronic Funds Transfer, but could be applied to other applications as well.

1. Introduction

One of the valuable services provided by government agencies is certification of commercial products. Familiar examples include the certifications of food and medicine performed by the Food and Drug Administration. To support the need for secure electronic funds transfer (EFT) of both industry and its own bureaus, the U.S. Treasury Department initiated a program for certifying EFT equipment [Ferr87, Trea86]. To assist in this effort, the National Bureau of Standards (NBS) has been developing source code analysis tools to assist in the evaluation of software used in EFT equipment. This paper describes some of the tools that have been developed and work that is currently in progress.

The EFT equipment being certified provides ANSI X9.9 Message Authentication capability [ANSI86] and ANSI X9.17 Key Management functions [ANSI84]. The equipment typically includes a secure microprocessor and a chip to perform encryption using the Data Encryption Standard [NBS77]. Software controls access to the various functions through either password protection or magnetic cards. The software is usually small, approximately 4,000 lines of source code.

Commercial developers supplying EFT equipment to the Treasury Department are required to develop it according to specifications given in [Trea86b]. The specifications mandate security features recommended in [NSA86] and include requirements to aid in verification suggested by [NBS82]. One job of the security analyst is to review the source code to ensure that access control checks are performed properly and that critical data are not accessible to unauthorized users. This review clearly cannot be fully automated; it can only be done by a trained analyst who takes the time to understand the source code. Software tools can improve the process by helping reveal the structure of the system and by performing certain mechanical checks on syntax and control/data flow (e.g. as provided by *lint* [John78]). UNIX* tools such as *lint*, *grep* [BSD80] and *awk* [Aho78] are heavily used in the evaluations, but this paper will present only new tools that have been developed to supplement UNIX utilities and other commercially available tools.

1.1. Prototype Tools

The tools described in this paper are designed to help the security analyst understand the system being evaluated. The information they provide could be gathered without tool support, but to do so would take considerably longer. An additional advantage is that the tools provide a variety of views of the system. Documentation is expected to contain much of the information, but one goal of the tool suite is to assist in checking consistency and correctness of documentation. Many of the functions provided by tools described in this paper are available from other tools.

The paper will point out features of our tools that are not provided by others.

The tools that have been developed are designed for use on C source code [Kern78]. C is an appropriate target language for the prototypes because C is a popular language for micro-processor applications. Most of the source code for the EFT equipment is C, although various assemblers are used as well. The

* UNIX is a registered trademark of AT&T

tools are written in C and are modular so that it is relatively easy to modify them to support other languages. In one case the code for the equipment being evaluated was written entirely in Z80 assembler and the parser was modified in an afternoon to handle the assembly code.

1.2. Tool Design

Most of the tools follow a common design, shown in Figure 1. The pre-processor is a simple parser that recognizes functions or macros defined in the source code but ignores common library routines such as *printf()* or *scanf()*. (An alternate pre-processor can include all functions called.) Function names are stored in *namefile* which is then used by the parser in each tool to recognize routines of interest. This design allows the analyst to remove names of routines that are not security-relevant, such as math library functions. The output of a tool (e.g. a call tree) will then consider only security-related routines, simplifying the analysis. Of course, judgement is required to determine what functions can be safely ignored. *Namefile* can be edited repeatedly to provide different views of the system being studied.

2.1. Operation

Ccalls has two phases. First a file of routine names and the file where they are defined is produced. The second phase reads this file and begins parsing the C source code. An adjacency matrix **D** is built, where $\mathbf{D}_{i,j}$ is 1 if routine i calls routine j and is 0 otherwise. In other words, **D** is simply a matrix representation of the program's call graph. After **D** is completed, *ccalls* builds a matrix $\mathbf{I} = \mathbf{D}^*$, the transitive closure of **D**. **D** is called the *direct* call matrix and **I** is called the *indirect* call matrix, since **I** represents the indirect relationship between routines through calls. In the example discussed above, $\mathbf{D}_{A,B} = 1$, and $\mathbf{D}_{B,C} = 1$, since A calls B and B calls C. Also, $\mathbf{I}_{A,C} = 1$, because of the indirect relationship between A and C.

2.2. Call Tree

After the call matrix is built, a call tree is easily produced. A portion of one is shown Figure 2. *Ccalls* only expands a subtree fully the first time a routine is encountered. Thus in the example in Figure 2, if 'prt_line' were called elsewhere, the tree would not be expanded to show 'get_name' as a subroutine of 'prt_line'. This helps keep the call tree display to a reasonable size.

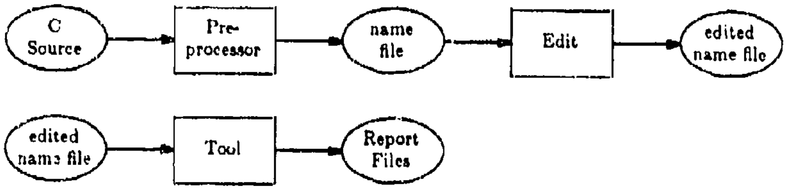


Figure 1. Tool Design

2. Ccalls: Call Tree and Logical Nesting

The first task of the security analyst is to understand the program that is being evaluated. The first step in doing this is to look at the functions of individual routines and the calls they make to other routines. In addition to the calls that occur in the program, the analyst will be interested in the indirect relationships among routines. If routine A calls routine B, and B calls C, then a change to A may affect C, or a change to C may affect A through parameters passed in calls. This relationship can be traced from the call tree, but if a program contains many routines the call tree may be spread across several pages of paper, making it awkward to analyze. The tool presented in this section, called *ccalls* simplifies the analysis of these relationships. Several tools are available to perform some of the functions of *ccalls*, but *ccalls* is apparently the first such tool to identify logical nesting in C source (see Section 2.7). It also appears to be the only tool to extract an indirect calls matrix from C source code, although it is possible to trace indirect calls manually using a call tree.

The UNIX System V utility *cflow* provides a similar call tree but lists calls to library functions such as *printf()* and *getchar()*. This results in extremely large call trees which may make it harder to recognize the structure of the program. *Ccalls* was designed to eliminate this "information overload" by ignoring library routines and printing calls within routines only once. If desired, a different pre-processor can be used with *ccalls* to include all functions, whether they are defined in the system under study or are library functions.

The call tree lines are of the form *routine name:file in which routine is defined* and may be followed by an asterisk if the routine represents the root of a logically nested subsystem (as explained in Section 2.7).

2.3. Call Matrix

Rather than present a large square matrix of 1s and 0s, *ccalls* provides, for each routine in the program, lists of the routines that it calls directly and indirectly and lists of routines that call it directly and indirectly. An example is shown in Figure 3.

2.4. Statistics

Ccalls also provides some statistics that may be useful in estimating the complexity of the program being evaluated. Figure 4 shows an example. The value Functions is simply a count of the C functions or routines in the entire program. Interfaces is a count of the interfaces between routines. For example, if A calls B three times and calls C twice, there are two interfaces: A to B and A to C. The Number of Calls in this example is of course five.

```
main:main.c *
     closure:matrix.c
     copy_array:matrix.c
     init_array:matrix.c
     out_matrix:matrix.c *
          count_interfaces:matrix.c
          do_call_tree:matrix.c
               main_rtn:proc.c
               prt_line:matrix.c
                    get_name:proc.c
               prt_tre:matrix.c
                    add_item:set.c
                    prt_line:matrix.c
                    prt_tre:matrix.c
                    set_mem:set.c
                  .
                  .
                  .
```

Figure 2. Example of Call Tree

The ratios provide some information on the branching factor of the call tree. The main routine, *main* in standard C, is not included in the count, only subroutines. Both ratios thus have minimum values of 1.00, since every routine but *main* must be called at least once. (*ccalls* can identify routines not called.)

2.5. Uncalled Routines

Routines that are never called are easily identified using the direct call matrix **D**. The routine indexed by j is unused if $\mathbf{D}_{i,j} = 0$ for all i. These routines are listed on the analysis report. The UNIX tool *lint* can detect uncalled routines, but some routines may be called yet still be unreachable. For example, routine d in Figure 5 is uncalled and unreachable. Routines e and f are unreachable but not uncalled. *Ccalls* can detect these unreachable routines, as explained in the next section.

```
do_call_tree:      From file: matrix.c

       Calls:
            Directly:
                 main_rtn  prt_line  prt_tre
            Indirectly:
                 add_item  get_name  main_rtn
                 prt_line  prt_tre  set_empty
                 set_mem  set_union  subsys

       Called By:
            Directly:
                 out_matrix
            Indirectly:
                 main  out_matrix
```

Figure 3. Display of Call Matrix

```
-------------- STATISTICS ----------------------

Functions:   34

Interfaces:  36

Calls:       49

Interfaces/Function:      1.06

Calls/Function:           1.44

-------------------------------------------------
```

Figure 4. Example of Statistics

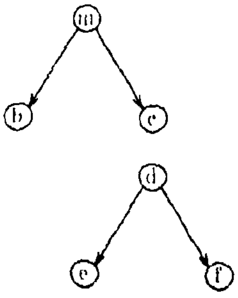


Figure 5. Uncalled and Unreachable Routines

2.6. Unreachable Routines

Some routines that are never called may call other routines. These other routines will not appear on the list of uncalled routines even though there may be no possible execution of the program in which they could be called. Recall that $\mathbf{I} = \mathbf{D}^*$. Therefore $\mathbf{I}_{i,j} = 1$ if and only if there is some call sequence from i to j. Unreachable routines can be identified simply by finding those which cannot be reached by any calling sequence from *main*, i.e., the routine indexed by j is unreachable if $\mathbf{I}_{main,j} = 0$. Note that it is always true that

$$\{\mathit{uncalled\ routines}\} \subseteq \{\mathit{unreachable\ routines}\}.$$

Unreachable routines are listed on the analysis report.

Routines that are called only through function pointers will be listed as uncalled and unreachable. The analyst must check that each such routine is used and that it is possible for a function pointer to be instantiated with the routine's address.

2.7. Logical Nesting

A useful feature of *ccalls* is the identification of logically nested subsystems, marked with an asterisk in Figure 2. Block structured languages such as Pascal and PL/I allow routines to be nested, to explicitly show their hierarchical relationship. The C language does not provide this feature, so hierarchical arrangements of subroutines must be deduced from the calling structure. *Ccalls* saves time by performing this task for the analyst. The value of recognizing a logically nested subsystem is that one can study the subsystem independently from the rest of the code.

We define a logically nested subsystem as a group of routines, headed by a root node, none of which is called by any routine that is not part of the subsystem. Thus leaf nodes, i.e. routines that do not call other routines, are trivially subsystems. The definition is stated formally in Figure 6. The definition indicates if a particular node, r, is the root of a logically nested subsystem. E is a set of routines 'external' to the subsystem (if one exists headed by node r), C is the set of routines called by routines in set E, and r and j are indices of routines in the program. *Ccalls* prints an asterisk adjacent to any routine that is the root of a logically nested subsystem.

A program will typically contain 'library' routines that are used in many places. *Ccalls* allows the user to eliminate these library routines from consideration when checking for logical nesting.

$S = I_{r,\bullet} - \{ r \}$

$E = U - S$

$C = \bigcup_{i \in E} D_{i,\bullet}$

$(S \cap C = \emptyset) \equiv$ r is the root node of a logically nested subsystem

where

$U =$ the set of all reachable routines in the program
$I_{r,\bullet} =$ the set of routines called indirectly by r, corresponding to row r of matrix **I**.
$D_{i,\bullet} =$ the set of routines called directly by i, corresponding to row i of matrix **D**.

Figure 6. Logically Nested Subsystem Identification

2.8. Data Access

An option allows global variables to be included in the analysis. Heavy use of global variables is poor programming practice. However, there are cases where they can be used sensibly. For example, a common way to implement an abstract data type in C is to define a data structure and all functions that access it in a single file. The data structure must be global to all the functions in the file. *Ccalls* allows global variable names to be treated as "functions" and appear in the call tree (as leaf nodes) so the analyst can examine the different ways in which functions accessing a global variable can be reached. The system should perform necessary validations on all calling sequences which can reach critical variables. Including global variables as leaf nodes in the call tree can make it easier to trace what happens on the way to a function that accesses a critical data item. Another tool, described in the next section, unravels the call tree into all possible calling sequences to make it easier to trace events along paths to critical functions and data items.

3. Paths: All Calling Sequences

One check that must be made is to ensure that preconditions for routines are established and/or maintained by routines higher up in the calling sequence. This is particularly important in assembler language where parameters are passed in registers or global variables. To verify that preconditions are met before a routine is called, the analyst must trace upward through all possible paths in the call tree by which a routine can be reached and ensure that registers are set up correctly on all pa.hs. This process is made easier by generating all possible calling sequences in which bottom level routines can be reached. The list of all sequences is restricted to those that contain the routine of interest by passing the tool output through *grep*. The analyst can then check each of the call sequences as far up as necessary to verify preconditions. An example is shown in Figure

7. Lists start with lowest level routines and move up the tree from left to right. For example, the first list shows that *main* calls *csm_errproc* which calls *send_esm* and so on.

```
key_setup<-csm_macproc<-gen_edc<-putedc<-
       send_esm<-csm_errproc<-main

key_setup<-csm_macproc<-gen_edc<-putedc<-
       send_esm<-csm_errproc<-p1_csm_parse<-
       getcsm<-main

key_setup<-csm_macproc<-gen_edc<-putedc<-
       send_esm<-main
```

Figure 7. Calling Sequences

4. Layers: An Alternative View of Program Structure

We have been experimenting with different ways of displaying program structure. One interesting approach is shown in Figure 8*. The matrix was created by dividing the program into layers according to calling structure. The top layer L_0 is defined to be $\{main\}$. Other layers can then be defined as

L_n = {functions called from layer L_{n-1} not assigned to a layer m, m < n }.

More formally,

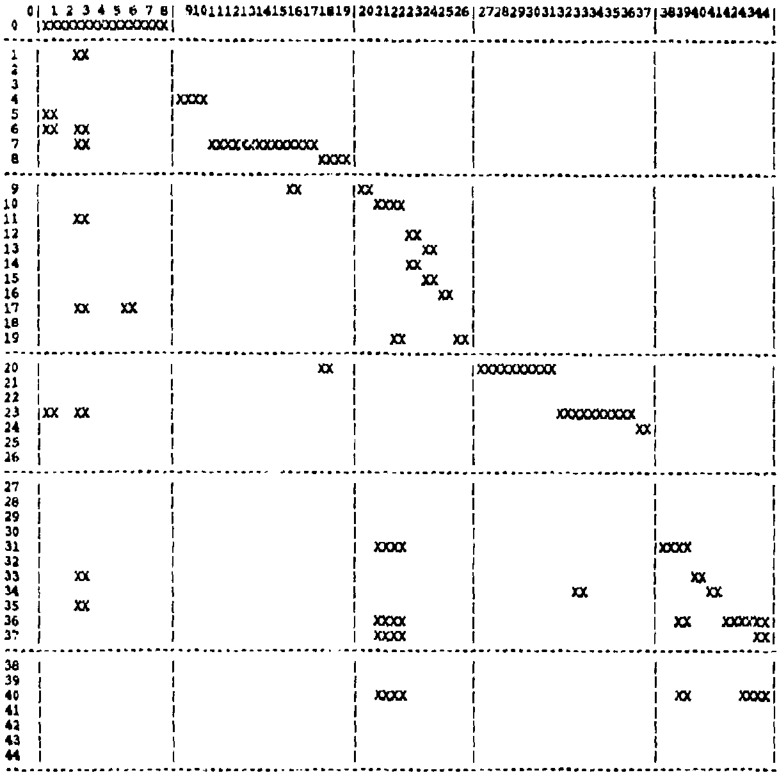
$$L_0 = \{ \ main \ \},$$

$$L_n = \{ \ j : for \quad some \ i \in L_{n-1}, \ i \quad calls \quad j, \ j \notin L_m \ for \ all \ m \ < \ n \ \},$$

where i and j are functions.

A call from routine i to routine j is shown by "XX" at position i, j in the matrix. The index numbers are keyed to function names and are displayed on a separate page or another workstation window.

This type of presentation is useful not only as a way to give a complete cross reference on one page (for the size of systems under evaluation) but also to convey information about the structure (or lack thereof) of the system. In particular it shows the



```
   0| 1 2 3 4 5 6 7 8| 910111213141516171819|20212223242526|2728293031323334353637|38394041424344|
 0  |XXXXXXXXXXXXXXXX|                      |              |                      |              |
--------------------------------------------------------------------------------------------------
 1  |    XX          |                      |              |                      |              |
 2  |                |                      |              |                      |              |
 3  |                |                      |              |                      |              |
 4  |                |XXXX                  |              |                      |              |
 5  |XX              |                      |              |                      |              |
 6  |XX  XX          |                      |              |                      |              |
 7  |    XX          |    XXXXXXXXXXXXXX    |              |                      |              |
 8  |                |                  XXXX|              |                      |              |
--------------------------------------------------------------------------------------------------
 9  |                |              XX      |XX            |                      |              |
10  |                |                      |  XXXX        |                      |              |
11  |    XX          |                      |              |                      |              |
12  |                |                      |      XX      |                      |              |
13  |                |                      |        XX    |                      |              |
14  |                |                      |      XX      |                      |              |
15  |                |                      |        XX    |                      |              |
16  |                |                      |          XX  |                      |              |
17  |    XX    XX    |                      |              |                      |              |
18  |                |                      |              |                      |              |
19  |                |                      |    XX      XX|                      |              |
--------------------------------------------------------------------------------------------------
20  |                |                  XX  |              |XXXXXXXXXX            |              |
21  |                |                      |              |                      |              |
22  |                |                      |              |                      |              |
23  |XX  XX          |                      |              |          XXXXXXXXXX  |              |
24  |                |                      |              |                    XX|              |
25  |                |                      |              |                      |              |
26  |                |                      |              |                      |              |
--------------------------------------------------------------------------------------------------
27  |                |                      |              |                      |              |
28  |                |                      |              |                      |              |
29  |                |                      |              |                      |              |
30  |                |                      |              |                      |              |
31  |                |                      |  XXXX        |                      |XXXX          |
32  |                |                      |              |                      |              |
33  |    XX          |                      |              |                      |    XX        |
34  |                |                      |              |            XX        |      XX      |
35  |    XX          |                      |              |                      |              |
36  |                |                      |  XXXX        |                      |  XX    XXXXXX|
37  |                |                      |  XXXX        |                      |            XX|
--------------------------------------------------------------------------------------------------
38  |                |                      |              |                      |              |
39  |                |                      |              |                      |              |
40  |                |                      |  XXXX        |                      |  XX      XXXX|
41  |                |                      |              |                      |              |
42  |                |                      |              |                      |              |
43  |                |                      |              |                      |              |
44  |                |                      |              |                      |              |
--------------------------------------------------------------------------------------------------
```

Figure 8. Example of Layers

```
   0| 1 2 3 4 5 6 7 8 910|111213141516171819202122232425262728293031323334353637383940414243|444546474849505152535455|56575859|
 0  |XXXXXXXXXXXXXXXXXXXX|                                                                  |                        |        |
-------------------------------------------------------------------------------------------------------------------------------
 1  |                    |                                                                  |                        |        |
 2  |    XXXX    XX      |XXXXXXXX                                                          |                        |        |
 3  |XX  XXXX    XX      |        XXXX                                                      |                        |        |
 4  |                    |                                                                  |                        |        |
 5  |XX    XX            |            XXXXXX                                                |                        |        |
 6  |            XX      |                  XXXXXXXXXXXX                                    |                        |        |
 7  |XX    XX            |                                                                  |                        |        |
 8  |XX  XX          XXXX|  XX        XX                XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  |                        |        |
 9  |                    |                    XX                                      XX    |                        |        |
10  |XX    XX    XX      |    XXXX                                                      XXXX|                        |        |
-------------------------------------------------------------------------------------------------------------------------------
11  |                    |                                                                  |                        |        |
12  |                    |                                                                  |                        |        |
13  |                    |                                                                  |                        |        |
14  |                    |                                                                  |                        |        |
15  |                    |                                                                  |                        |        |
16  |                    |                                                                  |                        |        |
17  |XX    XX            |                XX                                    XX          |XXXXXX                  |        |
18  |            XX  XX  |                                                      XX  XX      |                        |        |
19  |XX    XX    XX      |  XXXXXX                                                          |                        |        |
20  |                    |                                                                  |                        |        |
21  |                    |                                                                  |      XXXX              |        |
22  |XX                  |                                                                  |  XX                    |        |
23  |                    |                                                                  |                        |        |
24  |                    |                                                                  |                        |        |
25  |                    |                                                                  |                        |        |
26  |    XX      XX      |                                                      XX  XX    XX|XXXXXX                  |        |
27  |XX    XX            |                                                      XX          |  XX                    |        |
28  |XX                  |                                                                  |          XX            |        |
29  |            XX      |                                                      XX  XX      |  XX                    |        |
30  |      XX            |    XX                                                          XX|                        |        |
31  |      XX            |    XX                                                            |                        |        |
32  |                    |                                                                  |                        |        |
33  |                    |                                                      XX          |                        |        |
34  |                    |                                                                  |                        |        |
35  |      XX            |                                              XX                  |                        |        |
36  |      XX            |                                                                  |                        |        |
37  |      XX            |      XX                                  XX                      |            XX          |        |
38  |XX                  |      XX                                                          |                        |        |
39  |                    |                                          XX  XX                  |                        |        |
40  |XX    XX    XX      |                                                                  |              XXXXXX    |        |
41  |                    |                                                                  |  XX            XXXX    |        |
42  |      XX    XX      |                                                                  |                        |        |
43  |XX    XX            |                                                                  |                    XXXX|        |
-------------------------------------------------------------------------------------------------------------------------------
44  |                    |                                                                  |                        |        |
45  |                    |                                                                  |              XXXXXX    |        |
46  |                    |                                                                  |              XXXXXX    |        |
47  |                    |                                                                  |                        |        |
48  |                    |                                                                  |                        |        |
49  |XX    XX          XX|    XX                                      XX        XX          |XX                  XXXX|XXXX    |
50  |                    |                                                                  |                        |        |
51  |                    |                                                                  |                        |        |
52  |XX    XX            |                                                                  |                        |    XXXX|
53  |XX    XX            |                                                                  |                        |        |
54  |XX    XX    XX      |                                                                  |                        |        |
55  |                    |                                                                  |                        |        |
-------------------------------------------------------------------------------------------------------------------------------
56  |                    |                                                                  |                        |        |
57  |XX    XX            |                                                                  |                      XX|        |
58  |                    |                                                                  |                        |        |
59  |                    |                                                                  |                        |        |
-------------------------------------------------------------------------------------------------------------------------------
```

Figure 9. Example of Layers

dependence relation among routines. The relation is very similar, but not equivalent, to the *USES* relation defined by Parnas [Parn74, NRL80, Neum86] because it is derived purely from static analysis of source code which can only recognize syntactic uses of functions. Nevertheless, we have found it helpful in recognizing levels of abstraction that may not be immediately apparent from source code packaging, a problem that occurs in many evaluations, particularly in secure UNIX implementations [Sibe87].

An important principle of software engineering holds that reliability is enhanced by organizing a system into a hierarchy of layers [Dijk68]. Each layer of subroutines provides a virtual machine whose functions are used to implement the next higher layer. More recently it has been argued that the same arrangement supports security concerns, especially verification [Neum86].

Well designed systems are often organized into layers, and as a result it is often helpful in understanding a system to categorize subroutines by functional layer. Of course, many routines may be used in several places (e.g. math subroutines). These can be considered general-purpose library routines rather than members of any particular layer. From running the tool on many different systems, it appears that most tend to show a layered arrangement even when this was not an explicit design requirement.

The layers in Figure 8 correspond to nesting levels at which routines first appear in the call tree, i.e. *main* which first appears at nesting level 0 is layer 0, functions that first appear at nesting level 1 are in

* Figures 8 and 9 are small utilities used for illustration.

layer 1 and so on. Calls from functions at lower levels to higher levels (level *i* +1 considered "lower" than level *i*) appear in the lower left corner of the diagram. Calls of this type may be an indication of poor structuring.

Figures 8 and 9 show an interesting contrast. The system in Figure 8 is nicely layered. To gain an understanding of this system, the analyst can proceed by studying routines in increments of reasonable size. In the system in Figure 9, 44 of the 60 functions are called within the first two layers. The analyst must study almost 75% of the functions at once to see how the system works. In addition, this system contains many routines that are called from several layers, where the one in Figure 8 has only a few such routines.

5. Assert: Assertion Recognizer

Treasury Department requirements [Trea86b] specify critical events that must be performed by the EFT equipment, e.g. "Inhibit interrupts for crypto processing", "Perform RAM test." To make it easier to check that these functions are being performed properly, source code is required to contain numbered assertions that help the analyst recognize important points in the code. The following assertions are required:

1. module name,
2. global data items,
3. local data items,
4. module housekeeping prior to return,
5. requirements performed prior to module entry,
6. requirements performed during module processing,
7. other modules accessed during processing of given module,
8. other modules that can access the given module.

For example, the assertion format for item 5 above is

ASSERT 4 5 <category of event> <event>.

The categories and events are specified and numbered in the requirements, so, for example, "MID generation" would be indicated by

ASSERT 4 5 1 1.

The evaluation suite includes a tool to recognize assertions and write out the assertion, line number on which it starts, and the file in which it occurs. The same function may also be performed with the UNIX tool *grep* if the assertions are not broken across two or more lines. Using additional UNIX tools such as *awk*, *sort*, and *uniq* simple scripts check that all required events have been asserted in the code. The analyst must then verify that they are being performed correctly and at the proper times.

6. Ccomments: Condensed Source Listing

Often, the best way to get an idea of what a program does is to proceed in a top-down fashion, determining the function of the top-level routine, then looking for calls to other routines and determining their functions. The important points to consider about the routines are the parameters, and comments in the source code telling what the routine does.

A 'condensed source listing' is provided that abstracts this information from the source code to provide a summary of the program. The condensed source listing is designed to make it easier to study the functions in a large program. It gives comment header blocks and the call interface to each routine. The call tree provided by *ccalls* provides a 'road map' of the program's structure and serves as a guide to using the condensed source listing. An example is shown in Figure 10.

7. Metrics: "Quality Metrics"

One important consideration in evaluations is the degree to which good programming practices have been followed. Interesting characteristics include lengths of routines, number of comment lines, number of declaration lines, ratio of comments to executable code, and "complexity metrics" such as McCabe's cyclomatic number metric [McCa76]. Another tool in the evaluation suite checks and reports on these surface characteristics. It also gives warnings at user determined thresholds for lengthy routines, insufficient comments, or cyclomatic number greater than a user-specified value (typically 10).

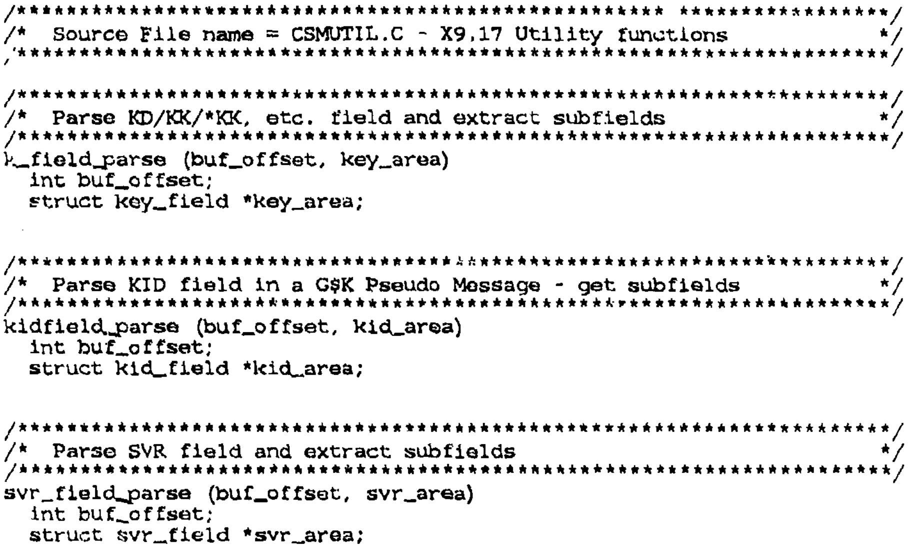
8. Trace: Function Call Trace

It is important that asserted events be performed in the proper order. The tool set also includes one dynamic analysis tool that instruments C source code to print the name of a routine whenever it is executed. This makes it possible to test a system and check that critical functions are being performed in the correct order by examining the trace of function calls.

9. Future Work

9.1. Windowing Environment for Certification Tools

We will be developing a prototype of an integrated windowing environment in which the security analyst can operate the previously developed tools or any others that are available from other sources (e.g. standard UNIX utilities) or may be developed in the future.



```
/****************************************************** *****************/
/*  Source File name = CSMUTIL.C - X9.17 Utility functions              */
/************************************************************************/

/************************************************************************/
/*  Parse KD/KK/*KK, etc. field and extract subfields                   */
/************************************************************************/
k_field_parse (buf_offset, key_area)
  int buf_offset;
  struct key_field *key_area;


/************************************************************************/
/*  Parse KID field in a G$K Pseudo Message - get subfields             */
/************************************************************************/
kidfield_parse (buf_offset, kid_area)
  int buf_offset;
  struct kid_field *kid_area;


/************************************************************************/
/*  Parse SVR field and extract subfields                               */
/************************************************************************/
svr_field_parse (buf_offset, svr_area)
  int buf_offset;
  struct svr_field *svr_area;
```

Figure 10. Condensed Source Listing

The environment will

1. Provide up to four windows simultaneously, with each window able to display or edit a file generated by the tools.
2. Execute a tool from any window.
3. Provide a menu of functions to perform and tools to execute.
4. Allow escape to the UNIX shell without leaving the environment.
5. Be able to reconfigure the windows from one to four and change their sizes as desired by the security analyst.
6. Create files and directories to store results of a session.

9.2. Sequence Analysis Tool

Among the requirements for the EFT software are requirements that certain security critical operations are performed in a specified sequence, in addition to the operations that perform the encryption and decryption [Trea86b]. Examples include the checking of critical routines before they are executed and the clearing of sensitive data from temporary storage after use. The critical operations must be performed in proper sequence on all paths through the program.

By using data and control flow analysis techniques developed for optimizing compilers, it is possible to determine if many sequencing constraints are met [Olen86],[McLe84]. In addition to use in optimizing compilers, analysis of this type has been used in evaluation of software reliability [Fosd76]. It may also be of value in evaluating software security.

To assist the security analyst in evaluating EFT software, a tool is being developed that will accept a specification of event sequences from the analyst and determine through static evaluation if the sequence constraint is met. The following types of event sequences can be checked for:

1 - Does A ever occur before B?
2 - Does A always occur before B?
3 - Does A ever occur after B?
4 - Does A always occur after B?
5 - Does A ever occur immediately before B?
6 - Does A always occur immediately before B?
7 - Does A ever occur immediately after B?
8 - Does A always occur immediately after B?

To make the analysis practical, the events should correspond to functions (or subroutines) in the source code. This should be a reasonable constraint, since the development requirements [Trea86b] specify that a subroutine should perform only a single function and also require assertions to be placed in the source code to assist the analyst in determining where critical functions are performed. It should then be possible for the tool to answer questions given above based on syntactically possible sequences of subroutine calls.

10. Conclusions

The tools described in this paper are experimental. All appear to be useful, but more experience is required to determine those that are most effective in aiding evaluations, and to determine features that are missing. We would like to obtain good data flow analysis tools for C, to supplement the fairly basic capabilities of *lint* and our own tools. Although the tools are now being used on relatively small systems, the Department of Treasury has very large systems that process sensitive data. Another goal of our current work is to determine how the tools can be applied to large systems.

11. References

[Aho78] Aho, A.V., B.W. Kernighan, and P.J. Weinberger, "Awk - A Pattern Scanning and Processing Language (Second Edition)", AT&T Bell Laboratories, 1978.

[Aho86] Aho, A.V., R. Sethi, J.D. Ullman. *Compilers: Principles, Techniques and Tools*, Addison Wesley, 1986.

[ANSI84] Financial Institution Key Management (Wholesale), ANSI X9.17-1984, American National Standards Institute.

[ANSI86] Financial Institution Message Authentication (Wholesale), ANSI X9.9-1986, American National Standards Institute.

[BSD80] BSD 4.2 *UNIX Programmers Manual,* Univ. of California at Berkeley, 1980.

[Chen86] Chen, Y-F. and C.V. Ramamoorthy, "The C Information Abstractor", Proceedings, COMPSAC 86, pp. 291-298, IEEE, 1986.

[Dijk68] Dijkstra, E.W., "The Structure of the THE Multiprogramming System," Communications of the ACM, Vol. 11, No.5, May, 1968.

[Ferr87] Ferris, M. and A. Cerulli, "Certification, A Risky Business," *Proceedings, 10th National Computer Security Conference,* 1987.

[Fosd76] Fosdick, L.D., L.J. Osterweil, "Data Flow Analysis in Software Reliability", *Computing Surveys* Vol. 8, Nr. 3, September 1976.

[John78] Johnson, S.C., "Lint - A Source Program Checker", AT&T Bell Laboratories, 1978.

[Kern78] Kernighan, B.W. and D.M. Ritchie, *The C Programming Language,* Prentice-Hall, Englewood Cliffs, N.J., 1978.

[McCa76] McCabe, T. "A Complexity Measure," *IEEE Transactions on Software Engineering,* Vol. Se-1, No. 3.

[McLe84] McLean, J., "A Formal Method for the Abstract Specification of Software", *Journal of the ACM*, Vol. 31, Nr. 3, July 1984.

[NBS77] National Bureau of Standards, *Data Encryption Standard,* Federal Information Processing Standard, Publication 46, National Bureau of Standards, Gaithersburg, Md., 1977.

[NBS82] National Bureau of Standards, "Software Validation, Verification, and Testing Technique and Tool Reference Guide," *National Bureau of Standards Special Publication 500-93,* P.B. Powell, editor, National Bureau of Standards, Gaithersburg, Md., 1982.

[Neum86] Neumann, P.G., "On Hierarchical Design of Computer Systems for Critical Applications," *IEEE Transactions on Software Engineering,* Vol. SE-12, No. 9, Sept., 1986.

[NSA86] National Security Agency, X12, INFOSEC Standards and Evaluations Group, "Functional Security Requirements Specifications - NSA Specification 86-16". National Security Agency, Fort Meade, Md.

[NRL80] Naval Research Laboratory, Communications Sciences Division, *Software Engineering Principles,* Naval Research Laboratory, No. AD A087997, Washington D.C.

[Olen86] Olender, K.M., and L.J. Osterweil, "Specification and Static Evaluation of Sequencing Constraints in Software", *Proceedings, Workshop on Software Testing*, IEEE 1986.

[Parnas74] Parnas, D.L., "On a 'buzzword': Hierarchical Structure," *Proceedings, IFIP Congress, 1974).*

[Sibe87] Sibert, W.O., H.M. Traxler, D.D. Downs, K.B. Elliott, J.J. Glass, "UNIX and B2: Are They Compatible?" *Proceedings, 10th National Computer Security Conference,* National Bureau of Standards/National Computer Security Center, 1987.

[Trea86] U.S. Department of the Treasury Directive 16-02, "Electronic Funds and Securities Transfer Policy -- Message Authentication and Enhanced Security," October 3, 1986.

[Trea86b] U.S. Department of the Treasury, "Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Devices for Federal E.F.T. Use", Sept. 1, 1986.

PROGRAM CONTAINMENT IN A SOFTWARE-BASED SECURITY ARCHITECTURE

Larry R. Ketcham

Entry/Medium Systems Group
Unisys Corporation
25725 Jeronimo Road MV320
Mission Viejo, California 92691
(714) 380-5948

Abstract

This paper describes how a software-based security architecture can protect itself against programs that attempt to compromise system security. Methods of program containment are explained, using an example of a software-based security architecture: the Unisys A Series with the MCP/AS operating system and InfoGuard security enhancements. The presentation focuses on issues involving creation and protection of program code and the extent to which compilers are included in the Trusted Computing Base (TCB).

Introduction

System security is usually considered "stronger" when based upon a hardware architecture that enforces TCB constraints. Therefore, techniques for building a software architecture that enforces TCB constraints are less widely discussed. A security architecture that relies in large part on a software TCB requires that novel methods of program containment be developed. The threat of system security being compromised by user-written programs must be analyzed carefully in an environment where the hardware is supportive of, rather than primarily responsible for, enforcement of security.

The next section of this paper surveys the central role played by program code in threats to a software-based security architecture. The subsequent section introduces the Unisys A Series architecture as an example of a software-enforced TCB. Then the largest section of the paper is devoted to an examination of the A Series protection methods that provide program containment; the issues covered include the nature and extent of trust in compilers, as well as the controls that must be placed on both compilers and the programs they create. The term "code file" is used throughout in the A Series sense, referring to a file that contains compiler-generated, machine-executable code.

This introductory section concludes with a very brief overview of the Unisys A Series. The basic architecture was introduced in the late 1960s with the Burroughs B6500 (which was itself based on the earlier B5500) and evolved through the other B6000 and B7000 systems to the A Series line. Although each new step in the evolutionary process provides object-code compatibility between new and predecessor systems, the hardware and software architecture does indeed change over time.

At a high level of abstraction, the Unisys A Series operating system, Master Control Program / Advanced System (MCP/AS), is primarily responsible for enforcing the A Series security policy with respect to users, files, programs, and processes (some security enhancements are enabled by InfoGuard, a Unisys software product that is integrated with MCP/AS): high-level protection incorporates the principles of Discretionary Access Control (DAC), Identification and Authentication, Audit, Object Reuse, and Least Privilege as required for Class C2 of the Trusted Computer System Evaluation Criteria [DoD85]. At a low level of abstraction, central-memory objects are protected by a combination of hardware, microcode, the MCP/AS security "kernel", and compilers; low-level protection is accomplished by structural containment (i.e., access is limited by the structure of the environment).

Although A Series processors do not define a privileged execution state, they do provide a number of protection features: a 4-bit tag is associated with each 48-bit memory word; tag values discriminate data from code and various processor control words. Code and critical control words have odd tags and are thus protected from access or modification by the instructions normally used to manipulate data. Memory is managed in segments defined by special control words called descriptors. All accesses to data segments (e.g., arrays) are automatically bounds-checked. A sophisticated stack-based addressing mechanism allows each item declared in a block-structured program to be assigned a static address at compile time; the addressed location may contain a simple variable (tag 0 or 2) or a tag-differentiated item such as an array descriptor (tag 5), a pointer to a character (tag 5), an entry point to a procedure (tag 7), or a reference to another item (tag 1).

The environment of any process consists of its own code and data plus any other items that are provided to it for communication with other processes or the operating system. Because each item can be separately described, the architecture permits controlled sharing of information at an arbitrarily fine level of detail. Thus each instance of a process (actually, each procedure invocation) has its own execution domain in an A Series system; hardware changes context automatically on procedure invocations across process domains. Indeed, it is just this flexibility of domain structure that justifies involving software in protection enforcement, rather than relying exclusively upon simple isolation mechanisms in hardware. (For an illustration of process domains, refer to the Appendix.)

Ancestral versions of the architecture are described by Hauck and Dent [Hau68], Creech [Cre69], Organick [Org73], and Doran [Dor79]. The most significant architectural departure in the A Series Advanced System Architecture is the introduction of Actual Segment Descriptors (ASDs) to extend the virtual and physical addressing space: the virtual segment descriptors that define data and code segments refer to an ASD rather than directly to a memory address [Mem87].

Potential Problems with Code Files

In hardware architectures that support two execution states, privileged and non-privileged, the processor enforces containment of non-privileged code by permitting dangerous operations only in privileged state. However, if the hardware supports only one execution state, the system software must be responsible for supporting privileged and non-privileged operations, while protecting the boundaries of the TCB. Those parts of the system software that create and maintain a self-protecting domain of execution must therefore be trusted, increasing the size of the TCB.

If the hardware has a single execution state, arbitrarily constructed code sequences are capable of subverting system security. In such an architecture, one of the most critical aspects of TCB protection is the control of executable program code. Several vulnerabilities relating to program code must be considered and neutralized in order to preserve the integrity of a software-enforced TCB.

Access to Assembly Language

The most obvious vulnerability of a software-enforced TCB is the ease with which assembly language programs are able to subvert system security. Without hardware controls, any system is inherently insecure if an assembler is available that can create arbitrary, executable, machine-language programs.

Coercion of Valid Compilers

Limiting programmers to use of high-level languages is not a sufficient means of guaranteeing the integrity of a software-enforced TCB. As Gligor points out [Gli83],

> "Attempts to force programmers to use only high-level languages, ... which would obscure the processor instruction set, are counterproductive because arbitrary addressing patterns and instruction sequences can still be constructed through seemingly valid programs; i.e., programs that compile correctly."

If a compiler provides a user with an overt capability to generate arbitrary code sequences, perhaps via specific language constructs, the TCB is once again vulnerable to subversion.

However, a language specification that does not provide an overt means for generation of arbitrary code sequences may be implemented by a compiler with less obvious vulnerabilities. According to Landwehr [Lan87],

> "In practice, it is difficult to prevent users from generating, via a certified compiler, programs that violate security, because compilers can often be subtly coerced into generating and initiating execution of arbitrary bit strings."

From the above arguments, it becomes more evident that trust in a compiler is a critical aspect of a software-enforced TCB. It follows that creation and installation of a compiler are points of vulnerability that require serious consideration.

User-Created Compilers

If a user is allowed to create his own compiler, he has the ability to generate any code sequence desired. Likewise, if the user is able to introduce a compiler to a system from off-line storage or from another host, he may go to a less secure system, alter an otherwise valid compiler or hand-craft his own compiler, and then import it into the system, perhaps as a Trojan Horse.

Landwehr [Lan87] points out that Burroughs systems have relied on software controls that allow users to program only in higher order languages compiled by certified compilers. Landwehr cites an example [Wil81] of how the software controls failed to prevent users from creating their own compilers to generate insecure assembly language programs.

Code Modification

Modification of program code after compilation presents another vulnerability (see Figure 1). Changing program code in memory creates a temporary (or dynamic) capability to circumvent system security. If the modification can be made to code stored on tape, disk, or any other storage medium, security can be compromised more permanently. In the example cited by Landwehr, the key to user creation of a compiler was actually the ability to modify a code file stored on magnetic tape.

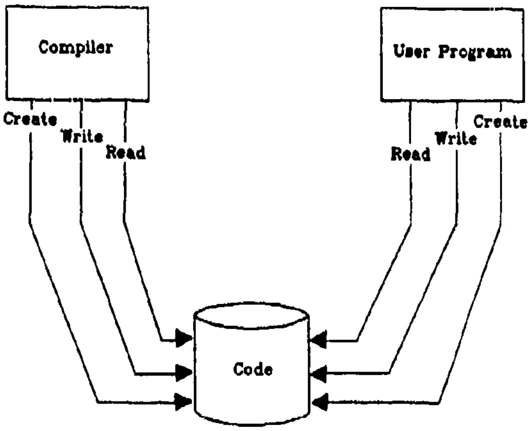


Figure 1. Code is vulnerable to modification.

Memory Vulnerability

Constraints on the construction (compilation) of code would be of no avail if the code were subject to modification while being executed in memory. Code-modifying programs provide a more serious threat to software-protected systems than to other systems. A program might appear benign on disk, waiting until it is actually executed to modify its own code in a dangerous way. Such a program might also modify the code of another process that is resident in memory, thus infecting other processes with dangerous code. A way to stop this type of virus is needed.

Tape Label Vulnerability

A system that depends upon tape label records is vulnerable to any mechanism that permits the label records to be read and written as ordinary data. Such abuse of "unlabelled" tape access was essential to the penetration of a Burroughs B6700 system described by Wilkinson [Wil81]:

> "An ordinary unprivileged user with sufficient knowledge of the system needs only the ability to be able to modify machine-code in order to penetrate the system completely. This ability is provided because the system allows code files to be loaded from storage on magnetic tape. Tape is a standard medium for transferring data between computer systems but has no security structures to protect it."

> "Although the Burroughs software which transfers files between tape and disk storage does use complex protective structures, there is nothing to prevent a knowledgeable user from imitating these structures and creating arbitrary code files which the Burroughs system will load and execute."

Wilkinson presented a step-by-step method for penetrating the system. An important step was the validation of an illegal compiler, which was accomplished with a program that took advantage of unlabelled tape access; the program read a tape, altered critical records to validate the intended program as a compiler, and then wrote the altered information to a second tape. Another critical step involved copying the illegal compiler from tape to disk.

Imported File Vulnerability

Even if a system tightly controls the structure of locally created magnetic tapes, the threat remains that an attacker with access to another system can create a tape with any desired contents. Other off-line media, such as removable disk packs, offer much the same threat, as does any network or other data communications facility that permits files to be transferred into the system.

Hardware Vulnerability

Given the vulnerabilities described above, one may ask whether any options remain once an attempt is made to exploit a vulnerability. Is there a final line of defense in the hardware? Can the hardware protect against faulty or suspect code generated by a compiler or programmer?

Single-state hardware can be designed to help maintain proper separation of user domains and help protect the TCB domain (see Figure 2). However, such hardware enforcement relies on registers, interrupt vectors, or other control information being properly initialized. If an executable code sequence (whether compiler-generated or assembly-coded) fails to supply correct values for base-bound registers, memory descriptors, or code pointers, the hardware cannot provide meaningful enforcement. In fact, when aberrant control information is intentionally supplied, not only is the integrity of the TCB violated, but system security is easily subverted.

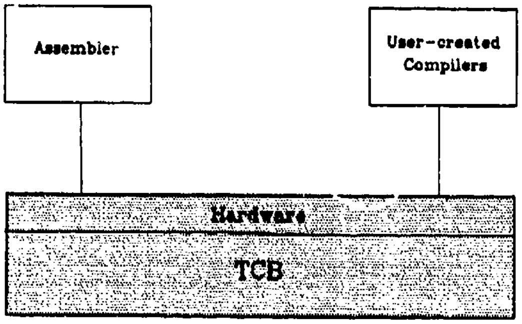


Figure 2. Traditional hardware-enforced TCB.

The problem of hardware enforcement being dependent upon software-supplied control information is not unique to single-state hardware architectures. Dual-state hardware is also vulnerable in this way. In an official interpretation [Arc87] regarding the B1 System Architecture requirement of the Trusted Computer System Evaluation Criteria [DoD85], the National Computer Security Center made the following general observation:

> "Hardware receives information from software, and based on that input it determines what actions are necessary. Therefore hardware is as trusted as the software providing the input."

Having considered several vulnerabilities concerning execution of code, it appears that software controls, particularly with regard to executable code, must play an important part in a general-purpose, secure system.

A Software-Enforced TCB

The Unisys A Series is a worked example of a software-enforced TCB. The hardware architecture supports only one execution state, making the system software responsible for supporting privileged and non-privileged modes of execution, while protecting the boundaries of the TCB. Those parts of the system software that create and maintain a self-protecting domain of execution include the MCP/AS operating system, compilers, Message Control Systems, system libraries, and privileged programs.

The Unisys A Series MCP/AS with InfoGuard security enhancements has been evaluated by the National Computer Security Center [Fin87] as meeting the requirements for Class C2 of the Trusted Computer System Evaluation Criteria [DoD85]. In the process of evaluating the A Series security architecture, the viability of a software-based, two-state architecture was formally addressed for the first time. As a result, NCSC issued the following official interpretation [Arc87]:

> "Software-based architectures are able to provide process separation and a two-state architecture with sufficient assurance to meet the B1 level requirements for System Architecture. Simply because a two-state architecture is provided and maintained primarily by software should not lead to the assumption of its being less secure than hardware in implementing security features.'

For a discussion of how A Series meets the specific Class C2 and B1 system architecture requirements, refer to the Final Evaluation Report [Fin87]. (Although there has been no attempt to demonstrate compliance with additional architectural requirements at Class B2 and above, the size and complexity of a software-enforced TCB would increase the difficulty of that task.)

Unisys A Series is a viable, software-based, multi-domain architecture because cooperation between compilers and the operating system makes extensive software controls possible. One of the more critical aspects of TCB protection is the control of executable program code: only authorized compilers are trusted to generate executable code files.

The nature of the software controls employed by A Series and the integration of compilers, operating system, and hardware to protect against potential threats to the TCB are explained below.

Protection Methods

Language design and compiler implementation are both critical to the protection of a software-enforced TCB. Languages generally available to users must constrain data manipulation and program flow control to a level of abstraction that cannot threaten TCB integrity. Manipulation and control at the potentially threatening lower level must be purposely designed out of user languages.

However, in order to create, maintain, and enhance a system, we recognize the need for systems programming capability. Because languages for systems programmers do provide access to potentially subversive constructs, access to the compilers for such languages must be carefully controlled. More importantly, the programs generated by such compilers must be subject to controls.

Because TCB integrity is at stake, we cannot depend upon a user who writes his own compiler to adhere to safe principles of language design and compiler implementation. For that reason, the ability to install a valid compiler on the system must be carefully controlled.

Controls on Compilers and Code Files

As stated in the NCSC evaluation of Unisys A Series [Fin87]:

> "Compilers in A Series are expected to perform the same functions as compilers on any other system. Compilers are expected to accurately implement the constructs of their language. It is the constructs within these languages that provide capabilities to users."

Therefore, a key protection objective is to limit the availability of compilers that implement constructs capable of subverting security. Unisys A Series uses several methods to accomplish this objective.

Unisys-supplied compilers for A Series are of two types: user-language compilers and systems-language compilers. User-language compilers (not to be confused with user-*created* compilers) implement languages designed with no constructs capable of subverting security. Systems-language compilers implement languages extended for the purpose of system software development; some of the extended constructs are considered "unsafe" because they could be used to subvert security.

Due to the implications of this language design strategy, the controls necessary for the two types of compilers, and for their generated code files, differ markedly.

The Unisys compilers are designed to make the most advantageous use of the hardware enforcement mechanisms present in the A Series architecture. The A Series stack mechanism is well-suited for support of block-structured languages, but the tagged architecture of A Series especially enhances the software's ability to provide TCB protection; the hardware supports the use of tag values to strictly separate code and control structures from data [Fin87].

No Assembly Language

The most dangerous language capability that could be offered to programmers is assembly language, with its unlimited capacity for controlling and subverting a system. Unisys A Series avoids the dilemma of policing assembly language by not providing an assembler, thus continuing a philosophy established for the predecessor Burroughs Large Systems as far back as the B5000. Working under the assumption that software-based systems allowing assembly language programming can never be truly secure, Unisys A Series provides no way to escape to low-level code generation, requiring that programming be done exclusively in high-level languages.

As a result of this high-level approach, compilers are trusted to properly structure accesses to TCB-protected objects (see Figure 3). The compilers are responsible for building correct references to data items, ensuring use of valid descriptors for the various kinds of files, and invoking the necessary interfaces for access to database items. Although compilers are not responsible for access checks on objects, they are responsible for emitting code that employs the defined MCP/AS interfaces to protect objects [Fin87].

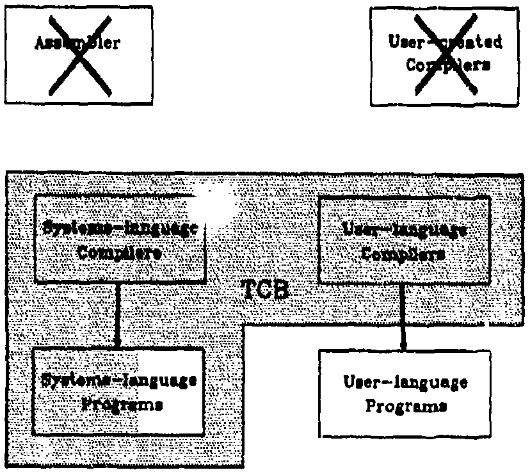


Figure 3. A Series software-enforced TCB.

Benign User Languages

By design, the Unisys-supplied user languages (e.g., COBOL, ALGOL, FORTRAN, Pascal, RPG) do not contain constructs that can be exploited to subvert security. In these languages, the user is allowed unlimited access to only the 48 data bits in some even-tagged words. The tags themselves and the data bits of other words can be accessed "only with compiler-determined code sequences that precisely support the high-level language semantics" [Fin87].

Not only are the user's code and data strictly separated, but the compiler provides access to only those MCP/AS interfaces intended for use by ordinary users [Fin87]. Compilers control the calling sequence and parameter evaluation code, thus ensuring that a user is never given unconstrained access to MCP/AS procedures. Therefore, software restrictions are not required for the use of user-language compilers or the code files they generate. Because the entire A Series system is geared toward high-level language programming, much of the system software, including the compilers, is written in user languages (primarily ALGOL).

Controlled Systems Programming Languages

The systems programming languages (i.e., DCALGOL, DMALGOL, NEWP) on A Series are extended dialects of ALGOL. DCALGOL includes some system control and data communications interfaces. DMALGOL is further extended to include special constructs for database-management and transaction-processing software. NEWP includes low-level constructs for I/O control, memory and processor management, and other operating system functions. These systems programming languages may also be used to write ordinary executable programs composed only of safe constructs.

Using unsafe NEWP constructs, a programmer can manipulate all 52 bits of a word, including the tag. Nevertheless, these systems programming languages do not facilitate generation of completely arbitrary code sequences. Even at the level of unsafe constructs, the compiler imposes conformance with appropriate abstractions (e.g., descriptor semantics).

Conferral of Code File Privileges

Because the systems-language compilers for DCALGOL, DMALGOL, and NEWP offer these additional interfaces for system software implementation, it is possible to write programs that subvert security. However, not all the code files generated by systems-language compilers are dangerous. To avoid placing overly stringent controls on such code files, three mechanisms are used to identify, authorize, and control potentially dangerous programs.

1. DCALGOL code fiies are dangerous only in the sense that they may attempt to access system control interfaces in MCP/AS. However, those MCP/AS entry points protect themselves by requiring that the calling process be privileged. A process is privileged only if executed by a privileged user or if its associated code file was marked privileged via the system command PP (Privileged Program) or by virtue of being a properly defined Message Control System (MCS). A compiler cannot create a privileged code file, and only a trusted user can perform the PP command. Creation of an MCS also requires trusted user intervention.

2. NEWP or DMALGOL code files that use unsafe constructs are marked non-executable when created by the compiler. To enable execution of such a code file, a trusted user must perform the system command XP (eXecutable Program) or SL (System Library).

3. Certain unsafe NEWP code files can be executed only via the system command CM (Change MCP), which is available only to trusted users.

In each of the three cases described above, potentially dangerous code files cannot be executed without an extension of trust to the programs using those interfaces. However, the greater the number of users trusted to perform such enabling actions, the greater the vulnerability to human error or malfeasance.

InfoGuard on an A Series system can erect multiple security barriers to control the introduction of unsafe code to the system. Even though a single barrier would normally suffice, multiple barriers provide additional protection in the event that trust (of programs or people) is misplaced or violated.

On a non-InfoGuard A Series system, there are two classes of users: privileged and non-privileged. Normal DAC mechanisms can make systems-language compilers inaccessible to unauthorized users. However, a privileged user is trusted to bypass file security checks and to exercise system control. Therefore, privileged users can access systems-language compilers, compile unsafe code files, and install (or make executable) such code files.

On a system using InfoGuard, a security administrator role can be defined, thus removing security-critical control functions (e.g., PP, XP, SL) from the privileged users. Even though a privileged user can still access systems-language compilers to create unsafe code files, he is unable to execute or otherwise install those code files because the necessary control functions can only be performed by a security administrator [SAG87].

FILEKIND Controls

In the control of compilers and code files, MCP/AS uses the FILEKIND attribute as an important discriminator. FILEKIND is an attribute of a disk file that denotes its purpose and, to some extent, its internal structure. Subranges of the FILEKIND values are reserved for special purposes: system files (such as disk directories), code files, program source files, and data files.

Unique FILEKINDs are provided for programming languages and dialects, as well as textual data, so that the editing format and appropriate compiler can be inferred. For example, the value ALGOLSYMBOL indicates program source written in ALGOL, while the value PASCALCODE indicates a code file created by the Pascal compiler. COMPILERCODEFILE is a special value in the system-file subrange that indicates an authorized compiler.

The assignment of a FILEKIND value to a file is controlled by MCP/AS. Only authorized system software may assign values in the system-file subrange, including the COMPILERCODEFILE value. Only compilers may assign values in the code-file subrange (thereby creating code). Users are allowed to assign values only in the program-source or data-file subranges. A user can change the FILEKIND of a code file to an unprotected (e.g., data-file) value only if the file is not currently being executed by any process.

Trust and Authorization of Compilers

To prevent the user from generating arbitrary code sequences, the ability to create code is reserved to authorized compilers on A Series. An authorized compiler is recognized by MCP/AS according to its FILEKIND. Although this approach eliminates concerns about the code integrity of ordinary programs, it requires that a great deal of trust be placed in the compilers. For this reason, the ability to authorize a compiler must be carefully controlled.

Trusted Compilers Enforce Protection Rules

In their role as emitters of code, compilers function at a low level of abstraction -- next to the hardware. Because the programmer has no access to the machine language of the A Series processor, the compilers can play an important role in ensuring the integrity of the system. However, for software controls to be adequately enforced on single-state hardware, compilers must complement the hardware reliably.

When a compiler is granted the privilege to create executable code files, it is trusted to rigorously enforce a number of protection rules. A few examples of those rules are summarized below:

Consistent semantics for data abstractions:
Emit valid object addresses; emit code sequences appropriate for manipulating the type of object at each address; enforce type matching on all procedure parameters and results.

Benign interfaces:
Employ defined interfaces for construction and destruction of memory segments; use only benign code sequences in creating or operating upon references. (Safe, properly constructed, A Series code is characterized not just by the absence of particular instructions but also by the use of some potentially harmful instructions only in valid ways.)

Referential integrity:
Avoid dangling references by refusing to store reference words in places where they might outlast their referents and by storing references only where they can be found systematically if necessary.

Controlled branching:

Emit valid branch addresses; properly limit the range of dynamic code selections (e.g., CASE or SWITCH statements).

Structural containment:

Limit the potential impact of one process on another by allowing a process to refer only to other processes that are structurally related to it, either ancestors (including self) or declared task variables within its own addressing scope; constrain references to objects according to the structure of the environment.

In general, a compiler is expected to conform to its language specification and to adhere to architectural principles that enable the hardware to preserve TCB integrity.

Compiler Authorization

In order to authorize a compiler, the system command MC (Make Compiler) must be executed to request that MCP/AS change the FILEKIND of an existing code file to COMPILERCODEFILE. The MC command may only be executed by a trusted user. In a non-InfoGuard system, that trust is extended to operators and privileged users. However, in an InfoGuard system, the circle of trust can be considerably smaller; when the security administrator role is defined, the MC command may be executed only by a security administrator [SAG87].

Due to the FILEKIND controls enforced by MCP/AS, an ordinary user program is prevented from creating a code file. Furthermore, a user program residing on the system cannot become a compiler without the approval or cooperation of an appropriately authorized person. However, to guard against Trojan Horse attacks, an authorized person must be cautious about executing programs that might attempt to use a programmatic interface to authorize a compiler. Only DCALGOL and NEWP provide access to such a programmatic interface, but successful use of that interface requires that trust be extended to the program either explicitly via the PP system command or implicitly when a properly privileged person (e.g., the security administrator) executes the program.

No Code Modification

By preventing a user from creating or importing a compiler, we are assured that the user cannot directly create a code file. However, we must also be sure that a user cannot arbitrarily modify code whether it resides in memory or in an existing code file on disk. The Unisys A Series systems rely upon a combination of hardware and software mechanisms to prevent a user from modifying code.

The fact that existing code files cannot be modified on an A Series system provides additional protection against viruses and Trojan Horses. When compiler validation is properly controlled, viruses cannot be injected into *existing* code files, and Trojan Horses cannot be added to *existing* code files. In short, virus propagation is prevented.

Separation of Code and Data

The tagged architecture of A Series makes strict separation of code and data possible (see Figure 4). Code words and code pointers in memory are protected by odd tag values, meaning they cannot be fetched or stored by the hardware instructions that normally operate upon data. The user is thereby prevented from reading or writing code words in memory. Because the hardware executes only words with odd tags (specifically, tag-3 words) and the user is not able to manipulate code pointers (tag-3 and tag-7 words), there is no way to execute data as code.

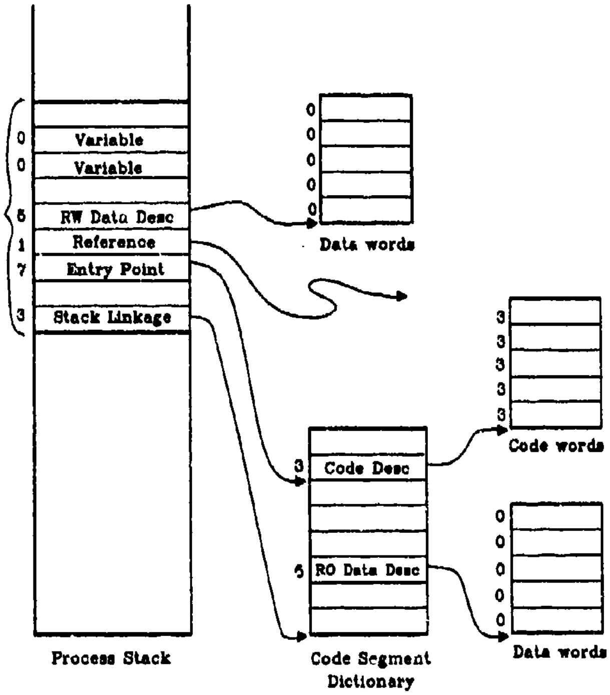


Figure 4. Tags separate code from data.

Software mechanisms prevent modification of code stored on disk in a code file. Tags are rarely stored with information on disk, except for the system-controlled overlay file used by MCP/AS during memory management operations. There is no opportunity for a user to tag data in a file so that it would appear to be code. For each code file being executed, MCP/AS creates a special stack, the Code Segment Dictionary, that contains descriptors to the code file's code segments and read-only data segments (see Figure 4); multiple processes executing the same code file can be linked to the same Code Segment Dictionary. When information is read from a disk file into memory, MCP/AS relies upon the FILEKIND and segment descriptor to assign appropriate tags to the information during the read operation [Fin87].

The FILEKIND controls enforced by MCP/AS are also effective in preventing a user from updating or rewriting an existing code file on disk. MCP/AS ensures that only programs with a FILEKIND of COMPILERCODEFILE are allowed to create *or modify* a code file (see Figure 5). Any attempt by a user program to assign a FILEKIND value of code is rejected. A user program is allowed to read a code file, but if it attempts to modify an existing code file, the write operation is aborted with an error.

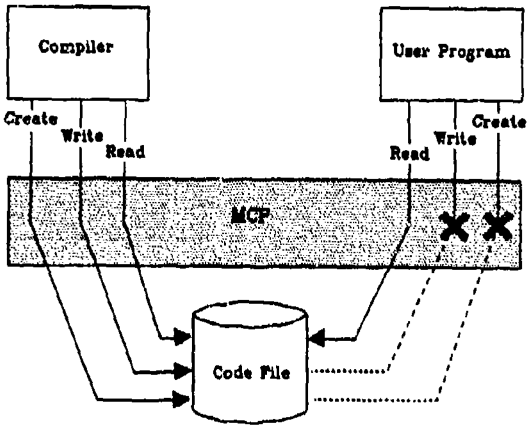


Figure 5. Code file protection on A Series.

Disk Files on Tape

To preserve the contents and attributes of disk files being stored or transported on magnetic tape, MCP/AS uses a special tape format called a "Library Maintenance" tape, which is distinguished by a field in the volume label; these tapes contain images of disk files including their defining header records. Only the library-maintenance utility of MCP/AS can create such tapes or transfer disk files to or from tapes while preserving such attributes as FILEKIND; that utility rejects attempts to read from ordinary data tapes.

In ordinary use, all tapes are written with standard labels, which are used for automatic assignment of tape files to processes. (The tape label convention also prevents reading any information past the nominal end of tape, preventing access to residual data that might follow the labelled contents of the tape.)

A program that reads and writes "unlabelled" tapes could both bypass and forge tape labels and disk-file header images. Indeed, the Wilkinson penetration used just this technique, achieving validation of an illegal compiler by changing the FILEKIND in the tape file header to COMPILERCODEFILE via unlabelled tape access [Wil81]. An InfoGuard system can be configured to meet this threat by requiring operator intervention to assign any tape to a process for unlabelled access [SAG87].

Code File Importation Restrictions

While we have explained how compiler validation for programs *residing* on an A Series system can be carefully controlled, we must still address the issue of illegal compilers (or other code files) introduced from off-line storage or from another host system. Rather than detecting which programs actually threaten system integrity, A Series software provides the means to *control* imported programs, thus helping the security administrator enforce security policy; the security administrator then uses his own judgment to decide which imported programs to release from controls.

Importation from Storage Media

Wilkinson [Wil81] introduced an illegal compiler to the system by copying it from an altered tape to disk. On an A Series InfoGuard system, there are two barriers to the introduction of dangerous code files from tape [SAG87]:

1. The security administrator can set the system security option TAPECHECK to the value AUTOMATIC, requiring that tape labels exactly match an existing entry in the tape volume directory, a central database maintained on disk by MCP/AS, before a copy request involving that tape can proceed. If there is no matching entry in the tape volume directory, the copy request is blocked, requiring operator action.

2. Furthermore, the security administrator can restrict tape units so that any code files copied from those units are marked non-executable, no matter how privileged the user or process performing the copy.

In fact, the second barrier, the ability to designate untrusted file sources and automatically restrict any files from those sources, is a basic feature of the Mark 3.7 release of the A Series operating system and does not require InfoGuard. (Concentrating the restrictive capabilities under security administrator control, rather than trusted user control, does require InfoGuard.) The restricted designation can be applied to tape units, disk units, tape volumes (reels), removable disk packs, remote (network) hosts, and even to individual files. In effect, these restrictions define the logical security perimeter of a system (see Figure 6).

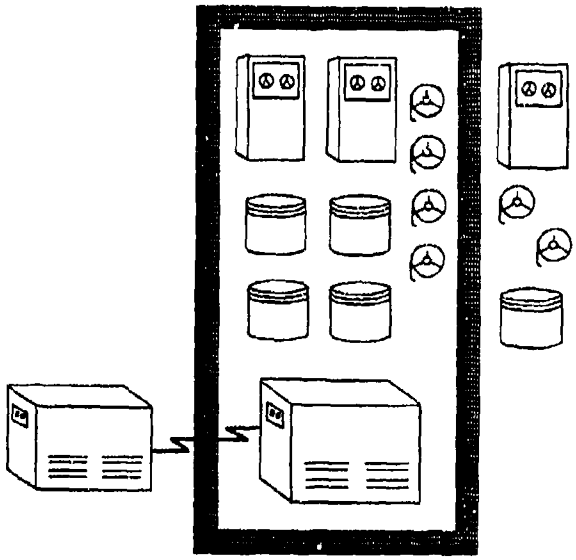


Figure 6. Code file importation restrictions.

Once a code file has been marked non-executable because it originated from an untrusted source, only a system administrator can remove the restriction from that program and make it executable again. Copying the restricted file to another unit or to another storage medium does not remove the restriction.

Any comprehensive security policy for tape files requires physical security on the tape volumes. InfoGuard provides tools that can be combined with sound operational procedures to effect substantial barriers against abuse via tapes or other media.

After attacking a Burroughs system on several fronts, Wilkinson and his colleagues concluded that "a B6700 system which disallowed the loading of code files from demountable tape or disk pack would be very difficult to penetrate." The Unisys A Series systems that evolved from the Burroughs B6700 have responded to this challenge -- the ability to restrict code file importation greatly increases A Series resistance to penetration. The robustness of the Unisys A Series security architecture is further enhanced by the InfoGuard features that support a tape security subsystem and a security administrator role.

Importation across Network

By prudent use of code file importation restrictions, a system administrator can erect a barrier against Trojan Horse programs that might be transported via tape or removable pack. In a similar manner, restrictions can be placed on code files imported across a network.

When the system security option HOSTSRESTRICTED is set, any code file copied onto an A Series system from a remote host in the network is automatically marked as non-executable [SAG87]. Again, only a system administrator can remove that restriction from a program to make it executable.

If a user were to breach security on one host in a network, somehow obtain a valid usercode/password for each remote host, and copy his Trojan Horse program to each remote host, then those hosts that had the HOSTSRESTRICTED option set would be protected by MCP/AS. On the protected hosts, MCP/AS would mark the code file as non-executable, effectively containing the Trojan Horse until the system administrator had the opportunity to scrutinize the program and take appropriate action.

Thus, the HOSTSRESTRICTED option enables a system administrator to erect a barrier against proliferation of Trojan Horse and virus programs across a network. This option can severely limit the ability of a virus to propagate beyond its original host.

Protection of Installed Software

The integrity of the TCB code is protected by the same mechanisms that prevent unauthorized modification or introduction of other code files. The TCB domain is further protected by the fact that installing or changing any part of the TCB software requires execution of privileged commands by trusted users. On an InfoGuard system with the security administrator role defined, those privileged commands are available only to a security administrator.

The A Series TCB software includes MCP/AS, system libraries, compilers, Message Control Systems, and privileged programs. The privileged methods for installing or changing these software components are summarized below [SAG87]:

1. MCP/AS cannot be installed or changed except by a system administrator with physical access to the system. A new MCP must first be compiled by the NEWP compiler, which automatically marks the newly compiled code file with a unique, non-executable FILEKIND. Then, the system command CM (Change MCP) must be executed to install the code file, requiring total interruption (HALT) and restart (LOAD) of the system.

2. System libraries must be installed with the system command SL (System Library), a command that can be restricted to the security administrator under InfoGuard.

3. Compilers must be validated with the system command MC (Make Compiler), a command that can be restricted to the security administrator under InfoGuard.

4. Each Message Control System (MCS) must be named in the data communications netw rk definition (known as Datacominfo) for that system. A new Datacominfo must be installed with the system command ID (Initialize Datacom). The ID command and the privilege to update the active Datacominfo can both be restricted to the security administrator under InfoGuard.

5. Portions of the data-management and transaction-processing software that use unsafe DMALGOL constructs must be made executable with the system command XP (eXecutable Program), a command that can be restricted to the security administrator under InfoGuard.

6. Privileged programs must be marked as privileged with the system command PP (Privileged Program), a command that can be restricted to the security administrator under InfoGuard.

Enforcement of the many software controls on A Series enables greater assurance than mere procedural controls could provide. By virtue of the fact that the software controls are well-integrated in the system and mutually reinforcing, they provide multiple barriers to penetration or compromise.

Summary and Conclusions

In exploring the concept of a software-based security architecture, several vulnerabilities relating to program code were considered, and methods of program containment that protect against those vulnerabilities were presented in the context of the Unisys A Series as a worked example. The threat of assembly language programming is avoided by providing no assembler and no escape to low-level code generation. Coercion of valid compilers is prevented by language design that provides benign user languages, while relying upon compiler implementation to identify "unsafe" programs written in systems programming languages so the operating system can enforce controls on those programs.

The software architecture of A Series requires and provides for careful controls on compilers and code files. Through FILEKIND controls, the operating system ensures that code files may be created or modified only by authorized compilers; only a system administrator can authorize a compiler. Through use of hardware-enforced tags and segments, code in memory is protected from modification, and execution of data as code is prohibited. Code files exist on tape only in special, protected formats. Importation of dangerous code files or illegal compilers from untrusted sources such as tape or remote hosts can be restricted, with enforcement by the operating system. In combination, these controls provide an unusually strong defense against the introduction of viruses or Trojan Horses into existing code files, raising multiple barriers against virus propagation.

Cooperation between compilers and the operating system in adhering to architectural principles can significantly enhance the protection afforded by single-state hardware. With system software supporting privileged and non-privileged operations, a multi-domain security architecture is achieved. Together, the operating system, compilers, and hardware are able to protect the integrity of the TCB.

Acknowledgements

The architecture described in this paper has resulted from the work of many people in Unisys Corporation and Burroughs Corporation through the years. However, Darrell High of Unisys deserves special recognition for his efforts to describe and formalize the A Series architecture; his constructive criticism during the preparation of this paper was much appreciated by the author.

References

[Arc87] "System Architecture (B1) Criterion Interpretation", National Computer Security Center, Report C1-CI-04-85, March 11, 1987.

[Cre69] "Architecture of the B6500" by B. A. Creech, Proceedings COINS-69 Third International Symposium, 1969.

[DoD85] Department of Defense Trusted Computer System Evaluation Criteria, Department of Defense Standard DOD 5200.28-STD, December 1985.

[Dor79] Computer Architecture: A Structured Approach by R. W. Doran, Academic Press, London, 1979.

[Fin87] Final Evaluation Report of Unisys Corporation A Series MCP/AS, National Computer Security Center, Report CSC-EPL-87/003, August 5, 1987.

[Gli83] "The Verification of the Protection Mechanisms of High-Level Language Machines" by Virgil D. Gligor, International Journal of Computer and Information Sciences, Vol. 12, No. 4, August 1983, pp. 211-246.

[Hau68] "Burroughs B6500/B7500 Stack Mechanism" by E. A. Hauck and B. A. Dent, Proceedings 1968 Spring Joint Computer Conference, Thompson Book Company, Inc., Washington, D.C., 1968.

[Lan87] "A Framework for Evaluating Computer Architectures to Support Systems with Security Requirements, with Applications" by Carl E. Landwehr, Brian Tretick, John M. Carroll, and Paul Anderson, Naval Research Laboratory, NRL Report 9088, November 5, 1987.

[Mem87] A Series Memory Subsystem Administration and Operations Guide, Unisys Corporation, Form 1169836, July 1987.

[Crg73] Computer System Organization: The B5700/B6700 Series by Elliott I. Organick, Academic Press, New York, 1973.

[SAG87] A Series Security Administration Guide, Unisys Corporation, Form 1195195, July 1987.

[SFG87] A Series Security Features Operations and Programming Guide, Unisys Corporation, Form 1195203, July 1987.

[Wil81] "A Penetration Analysis of a Burroughs Large System" by A. L. Wilkinson, et al, ACM Operating System Review, Vol. 15, No. 1, January 1981, pp. 14-25.

Appendix

To illustrate the A Series concept of process domains, we consider the situation where a process invokes an entry point of an external program. The A Series system allows binding of external program references to be deferred until execution-time through use of the "library" mechanism; in this context, the term "library" refers to a process that exports entry points (procedures) for dynamic linking by MCP/AS to client processes [Fin87]. Figure 7 shows how the different process domains are structured for the following ALGOL program text:

```
Client program:              Library program:

BEGIN   % outer block        BEGIN
  . . .                      . . .
PROCEDURE Q (K, F);          ARRAY D [0:4];
  VALUE K;                   . . .
  INTEGER K;                 PROCEDURE Q (K, F);
  ARRAY F [0];                 VALUE K;
  LIBRARY L;                   INTEGER K;
. . .                          ARRAY F [0];
ARRAY A [0:4];               BEGIN
. . .                          . . .
PROCEDURE P;                   F[K] := D[K];
BEGIN                        END Q;
  . . .                      . . .
  INTEGER I;                 EXPORT Q;
  . . .                      . . .
  Q (7, A);                  FRE"ZE...;   % as library
END P;                       . . .
. . .                        END.
P;
END.
```

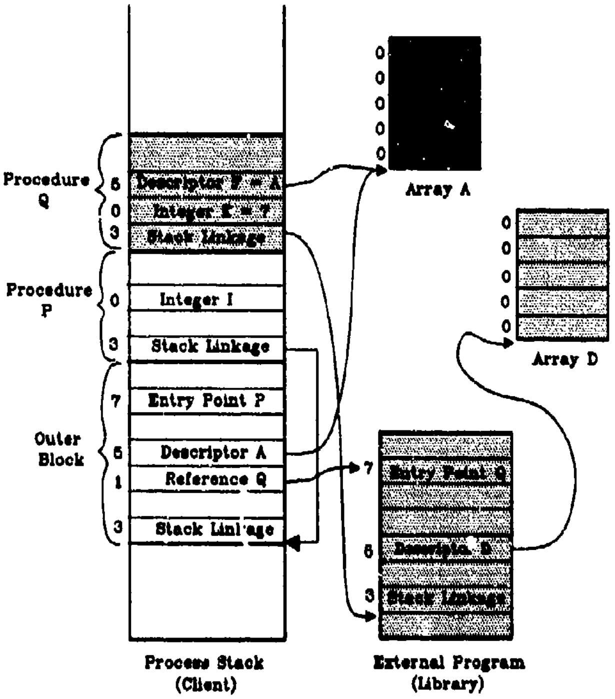


Figure 7. Process domains and controlled sharing.

As each block or procedure of a program is invoked, a new "activation record" is built on the stack; an activation record contains the control words and data that define the state of a procedure once it has been activated (invoked). For each declaration in a procedure, an appropriate stack item is allocated in its activation record. In Figure 7, the activation records for the client process are delineated with brackets and annotated "Outer Block", "Procedure P", and "Procedure Q".

The Outer Block of the client program declares Q, A, and P. Because Q refers to a procedural entry point declared in an external library, the client stack item for Q is a tag-1 reference to the actual tag-7 entry point Q in the library stack. Array A of the client is allocated a descriptor to the data segment for that array in memory. The client's local Procedure P is allocated a tag-7 entry point word.

When Procedure P is invoked from the Outer Block of the client program, the activation record for P includes a tag-0 data word for its local Integer I. P then invokes Q, the library entry point, passing the value 7 to Q's formal parameter K and passing the Array A to Q's formal parameter F.

To show the domain relationships between the two stacks, the execution domain of the library entry point is depicted by shading the relevant activation records (Procedure Q on the client stack plus the outer block of the library stack) and the data segment (Array D) belonging to the library. Even though Procedure Q executes on the client stack, its domain does not directly include items in the client's other activation records. However, Q can reference Array A of the client indirectly through the formal parameter Array F because A was passed as the actual parameter to F. Array A is shaded differently to indicate that it is shared by the client process and the library entry point Q.

To summarize, the execution domain of a procedure is defined by its addressing environment, which includes the procedure's own activation record, all activation records global to the procedure in its *declared* scope, plus any items passed to the proce lure as parameters. In addition, the domain is extended to include any data segments (arrays) referenced by items in this addressing environment.

In a similar manner, a process can invoke entry points of the operating system. An MCP procedure can execute on top of a user's process stack and access the user domain through parameters, but the MCP domain is protected from any direct access by the user's procedures. While executing on the user's process stack, the MCP procedure itself still has access to data in the operating system domain.

ACCESS MEDIATION IN SERVER-ORIENTED SYSTEMS: AN EXAMINATION OF TWO SYSTEMS

Martha Branstad
Frank L. Mayer

Trusted Information Systems, Inc.
3060 Washington Road (P.O. Box 45)
Glenwood, Maryland 21738

ABSTRACT

This paper discusses access mediation approaches for a class of computer architectures termed *server-oriented systems*. The focus is on the architectural issues involved in designing a server-based system that remains faithful to the concept of a *reference monitor*. In addition to general concepts and concerns, two specific architectures are examined.

1. INTRODUCTION

Over the past several years Trusted Information Systems, Inc. (TIS) has been involved in analyzing a number of computer systems to determine, in each case, the feasibility of evolving the system to a B2 or B3 trusted system as defined by the *DoD Trusted Computer System Evaluation Criteria* (TCSEC) [1]. All of these systems are designed to operate on a collection of multiprocessors, usually a small set of tightly coupled processors with shared memory. Although diverse in size, complexity, and targeted application arena, each operating system exhibits a similar set of organizational characteristics that strongly influenced its potential to evolve to a trusted system. These characteristics center around an object-oriented design philosophy where system resources are presented as a set of abstract typed objects that may be manipulated via a set of predefined operations. The set of operations for a particular object type are collected together into a single, independent object-type manager or *server*. Hence, these systems are characterized as *server-oriented* systems.

Generally, the server-oriented systems that TIS has examined are partitioned into three progressively more abstract layers that typically have a direct correspondence to privilege layers provided by the hardware/software. The most privileged layer is the *kernel*, which directly manages the physical machine providing a limited view of the system resources. The kemel encapsulates the physical machine providing for resource utilization via a well-defined set of primitive operations. The second layer is the collection of system *servers*, each managing a particular type of object constructed from the kemel-provided primitives. Examples of typical servers include file, device, and mail servers. The consumers of a server's services, i.e., the *clients*, constitute the third layer of server-oriented systems. Servers can also be clients when the services of another server are required.

We have found that many server-oriented systems, though not initially designed as trusted systems, have potential for evolving to B2 or B3 systems. The primary rationale for this conclusion is that the inherently layered structure of a server-oriented design philosophy results in a system architecture that incorporates many of the fundamental architectural traits required for a highly trusted system (e.g., modularity, least privilege, abstraction and data hiding). The primary task in evolving these systems to trusted systems has proven to be the inclusion of a complete and comprehensive reference validation mechanism that meets the criteria of a *reference monitor* (see [2]) while maintaining the systems' strong architectural traits.

In this paper, experiences with developing trusted versions of two server-oriented systems are discussed. The first system, Aspen, is a prototype system developed by Amdahl Corporation to run on Amdahl 470s, 580s, and other IBM 370-compatible system architectures [3]. Aspen's design is strongly influenced by the server-oriented philosophy with most system resources only accessible via object-type servers. The second system examined is Mach, an operating system kernel being developed at Camegie-Mellon University (CMU) [4]. Mach is designed to be transportable across a broad range of computer architectures, from monolithic processors to highly parallel architectures. Though Mach is currently only a kemel, TIS is developing a prototype trusted version (called TMach) based upon the server-oriented design of Mach's precursor system, Accent[5].

The experienced gained from examining Aspen, Mach, and other server-oriented systems has led to the identification of several general approaches for access mediation within server-oriented systems. The remainder of this paper will discuss these general approaches and the specific solutions used in the trusted designs of both Aspen and Mach.

2. ESSENTIAL SYSTEM CHARACTERISTICS

Several architectural characteristics, which server-oriented systems tend to exhibit, play a central role in determining access mediation approaches for such systems. These characteristics result in an environment in which servers and clients can operate in a controlled and unambiguous fashion. The architectural characteristics that are most central in the design of a trusted server-oriented system include:

(a) Protected and Isolated Execution Domains

Servers and clients can expect to execute in an environment free from interference by other servers and clients. Typically, this is provided through address space isolation and memory protection. For example, servers and clients can both execute as distinct processes within the same hardware privilege state. However, we have seen other methods for providing isolated execution domains that are also adequate. In Aspen, for example, software engineering principles (e.g., modularity) are used to provide separation among trusted server domains while hardware protection mechanisms are used to separate untrusted server and client domains from other untrusted and trusted server and client domains.

(b) Resource Isolation

A fundamental concept of server-oriented systems is that all manipulation of a particular object is controlled via the server responsible for that object-type. As such, primitive resources used by a server to create more abstract objects must be isolated from other entities in the system. In a strict server-oriented paradigm, the resources managed by one server should only be accessible by clients and other servers (either trusted or untrusted) via the managing server.

(c) Controlled Inter-Domain Communication

In order for clients to request the services of a server, a communication mechanism must exist among clients and servers. In order to maintain separation and isolation between server and client domains, this communication mechanism must be controlled in some manner (usually directly by the kernel). Depending upon the level of access mediation incorporated within the individual servers, this communication mechanism must also unambiguously provide the servers with identity and privilege information of all clients requesting services. As will be seen below, both Aspen and Mach provide sophisticated message-passing mechanisms to facilitate this type of inter-domain communication.

3. ACCESS MEDIATION

As a result of our efforts to develop trusted versions of several different systems, we have noted a small number of access mediation approaches for server-oriented systems. All of these approaches are based upon the locality of access control mechanisms within the system's architecture. In the first approach, access mediation is performed within the kernel. This *kernelized* approach, which is most like the classic "security kernel" [6], treats both clients and servers as subjects external to the reference validation mechanism. As such, the servers can augment and extend the kernel's access control mechanisms with a finer granularity of access mediation and the addition of supporting policies (e.g., audit), but the fundamental access control enforcement is provided by the kernel. A kernelized approach to access mediation is not particular to server-oriented systems and has been the common approach for implementing a reference validation mechanism in the past. For example, SCOMP [7], KVM [8], and KSOS [9] all have a kernelized reference validation mechanism.

In contrast to a kernelized approach, the inherent structure of server-oriented systems suggests the possibility of a *distributed* approach to access mediation. Given that servers constitute independent object-type managers, it follows that individual servers can be responsible for mediating access to the objects they manage. This approach is consistent with the object-oriented concept of object-type managers that are responsible for all actions related to their objects. Unlike the kernel, which for the most part is only aware of the primitive resources it offers, a server is aware of the unique nature of its more abstract objects and can provide a tailored access control policy for that object-type. This observation is especially true for discretionary access control policies where, for example, the access control policy for files may differ greatly from that for mailboxes though both are essentially derived from the same primitive resource (e.g., disk storage).

The distributed approach results in an reference validation mechanism that is distributed among the various servers and not centralized as with the traditional security kernel approach. However, we have found that such an approach can be used to design a reference validation mechanism that still meets the criteria of a reference monitor, namely tamperproof, always invoked, and analyzable.

There are two variants to the server-oriented approach for access mediation that strike a compromise between kernelized and completely distributed access mediation. These approaches provide server-based access mediation, but in a centralized fashion. The first of these approaches involves the notion of a *front-end* or *name server*. This approach assumes that all client accesses to server-based objects is via a common, global naming sphere, managed by a name server. The name server logically resides between the clients and the other servers, or at the "front-end" of the servers. Requests to access objects by name are routed directly through the name server, which performs access validation and, if the request is allowed, passes the request onto to the appropriate object-type server. Thus, to establish communication with a server and access the objects that that server manages, clients must first pass the scrutiny of the name server. The name server approach provides centralized access mediation while exploiting the nature of servers. The name server, which operates at a higher level of abstraction than the kernel, can address many of the idiosyncracies associated with the abstract objects offered by other servers. Conversely, the name server may require considerable richness to address the mediation concerns of all object-types. Hence, information that would otherwise be maintained solely by the object-type server must be available to the name server.

The second centralized server-based approach involves the notion of a *back-end* or *catalog server*. Unlike a name server, a catalog server logically resides between the servers and the kernel, or at the "back-end" of the servers. This approach assumes that in order for individual servers to access the kernel resources that they manage, they must reference a "card catalog" managed by a catalog server. Thus, the catalog server intercepts all access attempts and, if properly designed, can perform access mediation based upon a combined server/client identity. A back-end, catalog server approach to access mediation has many of the same pros and cons as a name server approach.

In the systems that we have examined, the above approaches were all considered. Most often, the idiosyncracies of the particular system strongly suggested the selection or rejection of a specific approach. In Aspen for example, the kernelized approach was rejected because it would have placed access mediation at too low a level to be meaningful with respect to server objects. Performance issues were also of concern. By contrast, in Mach, placing some mediation in the kernel appears to be the approach of choice. In most cases the realities of the system architecture suggests a combination of approaches. For example, in Mach, a combination of mediation in the kernel and in a name server is currently being targeted. In the following sections, the approaches devised for both Aspen and Mach are discussed in more detail.

4. ASPEN

Amdahl developed Aspen to provide a lower complexity, higher reliability alternative to existing 370 operating systems that would be compatible with most existing application software. Though Aspen was never marketed many of its design concepts were innovative, especially for large, mainframe operating systems.

4.1 ASPEN ARCHITECTURE

Using the 370 architecture's features of execution states (supervisor and problem), protection keys, fetch-protect bit, and virtual addressing [10], Aspen is organized around three hierarchical privilege levels. The most privileged level is the *Monitor*, which is Aspen's equivalent to the kernel in the server-oriented paradigm. The Monitor executes in supervisor-state in a portion of real memory below that allocable for virtual memory and is responsible for all low-level machine management functions. The remaining two layers, *Supervisor* and *Native*, both execute in the problem-state in virtual addressing mode. Virtual address spaces in Aspen are multi-threaded, that is, a virtual address space, called a *session*, may contain one or more threads of execution, called *processes*. Each session is a distinct address space with the exception of a portion of high memory which is common to all sessions and contains the Supervisor layer. Hence the Supervisor layer, though running in virtual addressing mode, is present in all sessions simultaneously. Supervisor code and data structures are protected from modification and observation by Native layer code via the 370 architecture's storage protection keys. Supervisor layer memory and processes have a key=0 and native layer memory and processes have a key>0. Processes with key=0 may access any page addressable while processes with key>0 may only access those pages with the same key. Thus, the Supervisor layer is able to coexist within the same virtual address space as Native layer processes without interference from them.

The Aspen Supervisor layer is organized primarily as a collection of interacting servers that together provide most of the traditional operating system functionality (e.g., files, devices). Additional system services are provided by Native layer services and routines.

4.1.1 Aspen Concept of Servers

Servers are a well-defined and fundamental concept in Aspen. The Transport Manager, which runs in the Supervisor layer, provides the features and mechanisms that allow servers and clients to interact. Servers are essentially processes that make a collection of abstract "object-identifiers" available to clients via an *offer*. Servers may be executing in either the Supervisor or the Native layer. All servers are accessed via a single global naming space of object-identifiers managed by the Transport Manager. An object-identifier consists of five 8-character qualifiers, which are referred to as `store`, `owner`, `group`, `type`, and `name`. A fully qualified object-identifier is of the form:

```
store.owner.group.type.name .
```

An offer is an object-identifier which is usually only partially qualified. For example, an offer can be of the form:

```
A.B.*.*.* .
```

Such an offer means that the server is offering to handle requests for all object-identifiers where the first two qualifiers are A and B. An offer also has associated with it an "extent", which determines its scope of visibility. Extents may either be local or global. A local-extent offer is only visible to processes within the same session as the offering server. Conversely, a global-extent offer is visible to processes in all sessions.

A client process communicates with a server by issuing a request* with an associated fully qualified object-identifier. The Transport Manager supports two types of client requests: *independent* and *related*. Independent requests establish a one-time communication path between a client and a server for the issuance of a single request and the reception of the results of that request. Such requests are unrelated to any other requests sent to the server. Typically, independent requests include DELETE, DEFINE, and RENAME operations on an object. The one-time communication path established for an independent request never outlives the

*A request is actually a software generated interrupt (MC or SVC instruction) that is trapped by the Monitor and forwarded to the Transport Manager in the Supervisor layer.

life of the request. The exception is the CONNECT independent request, which establishes a *connection* to a server. A connection is a long-term communication path to a server that exists until explicitly destroyed (either by the client or the server). Connections are used for related requests and allow multiple requests to be associated in an ordered fashion. For example, a connection to a file object can be used to do multiple READ requests such that each successive request reads the next record (relative to the record read in the previous request). The Transport Manager manages all current connections between clients and servers.

When the Transport Manager receives an independent request with an associated object-identifier, it determines which (if any) server has an offer that matches that object-identifier. If several offers match, the Transport Manager uses the following precedence rules to select the appropriate server:

(a) The most fully qualified local offer first; then
(b) The most fully qualified global offer.

Thus for example, a local offer `A.B.*.*.*` would take precedence over a global offer `A.B.C.*.*` even though the global offer is more fully qualified.

When the Transport Manager determines the appropriate server, it forwards the request to that server for processing. Based upon its own internal logic, the server may (1) reject the request, (2) fail the request, or (3) accept the request. If a server rejects a request (1), the Transport Manager forwards the request to the next server which has a matching offer. If no other server has a matching offer, the Transport Manager fails the request as "object not found." If a server fails the request (2), then the Transport Manager passes the failure back to the client. Finally, if the server accepts the request (3), the Transport Manager returns the results of the request to the client. If a server accepts a CONNECT request, then the Transport Manager establishes and manages a connection between the client and the server.

4.1.2 Supervisor Object-Type Servers

Object-identifiers offered by a server have no intrinsic meaning, nor do they necessarily represent some actual resource or resource abstraction. Actual mapping of system resources to an object-identifier is accomplished via servers and their internal logic. The Supervisor layer consists primarily of privileged object-type servers. The major servers include File Server, Scheduler, Terminal Server, Volume Server, and Device Server. Most of the Supervisor layer servers are designed around the conventions implemented by the File Server, which is discussed in detail below.

The File Server manages disk storage as files, which are organized into *stores* and *catalogs*. A File Server store is roughly equivalent to a logical disk volume. The first qualifier of a file object-identifier determines the store on which the file is maintained. A catalog, which is represented by the second qualifier in a file object-identifier, is a collection of files all with the same "owner." The owner's userID and the name of the catalog in which the owner's files are stored are synonymous. All files in a catalog are owned by the userID that is reflected by the catalog name.

The File Server is actually a collection of servers and associated offers, each server executing the same Supervisor layer reentrant code within different contexts [11]. There is potentially one of these servers for every store and catalog combination on the system. Each of these servers, which make the offer:

"store_name" . "catalog_name" . * . * . * ,

are dynamically activated and deactivated as references to files in a catalog are made. Since all file servers execute the same reentrant code, it is convenient to consider them as a single File Server with multiple offers. The remaining three qualifiers of a file object-identifier (group, type, and name) have no particular meaning to the File Server other than to uniquely identify a file and may be arbitrarily specified by the client. All files are grouped by the File Server based solely upon their store and catalog qualifiers.

The remaining Supervisor servers offer other system resources in a similar fashion. The Device Server directly provides all primitive functions for physical devices (e.g., disks, tape drives, printers, terminals). The Volume Server presents tape volumes as logical extensions of the file system using the Device Server to access the tape drives. The Terminal Server accesses terminal devices (via the Device Server) and re-offers them with value-added functionality. The Scheduler Server accesses terminal devices (via the Terminal Server) to initiate the logon process on all interactive terminals.

4.1.3 Catalog Manager

Supervisor servers must be able to maintain descriptive information about their objects across system initialization. To facilitate this ability, Supervisor servers use *object information blocks* (OIB), which are objects offered by the Catalog Manager. Typical information stored in OIBs include location, name, type, size, owner, and creation, modification, and reference dates for objects. Though the Catalog Manager is a server, it only services Supervisor Layer clients. Native layer clients access the catalog information associated with an object through the server that manages that object-type. For example, a file's OIBs are made available to client sessions via the File Server as part of the its offered abstractions.

As its name implies, the Catalog Manager manages OIBs in groups called "catalogs." For example, a File Server catalog described above has a one-to-one correspondence to a Catalog Manager catalog. The File Server maintains the necessary information to describe all the files in a file system catalog in one Catalog Manager catalog. Thus, when a client makes a request to the File Server to access a file, the File Server makes a request to the Catalog Manager to access the catalog of file OIBs in order to determine the existence of the file and locate the file on the appropriate store. A similar interaction occurs between the other Supervisor servers and the Catalog Manager.

The Catalog Manager has a special relationship with the File Server that is different than its relationship with the other Supervisor servers. OIBs must be stored in some fashion. Since the File Server manages all disk stores, it is necessary for the Catalog Manager to use the File Server's services to store and maintain its catalogs. This results in a recursive relationship between the Catalog Manager and the File Server. Thus, the Catalog Manager and File Server are required to cooperate in an environment of mutual dependency.

4.1.4 Native Layer Servers

Aspen's server concept is fully exported to the Native layer. Thus, Native layer servers can be created that act and behave exactly the same as Supervisor layer servers. However, because Native layer servers are significantly less privileged than their Supervisor layer counterpart, the Transport Manager levies additional constraints on the offers that these servers may make as discussed below.

Offers may become overloaded with several offers matching the same fully qualified object-identifier. For example, the File Server may offer object-identifiers `A.B.*.*.*` while a Native level server may offer object-identifiers `A.B.C.*.*`. In this case, if the Native layer server's offer is global, it would intercept all requests to access file names with `A.B.C` as their first three qualifiers. In general, the Transport Manager only allows Native layer servers to make global offers that overload Supervisor layer servers if the owner (second) qualifier in the offer is the same as the Native layer server's userID. Thus, Native layer servers may only intercept requests to objects for which they are the owner, essentially a form of discretionary access control. Generally, there are no restrictions on any local offers or those global offers that do not overload a Supervisor server.

4.2 ACCESS CONTROL IN ASPEN

Aspen incorporates many of the server-oriented characteristics discussed earlier. Separate server and client domains exist through the provision of distinct address spaces (sessions). Aspen further extends the concept of domains by providing the Supervisor and Native layers, allowing for servers and clients at different hardware privilege levels. Inter-domain communication is facilitated via the Transport Manager-provided concepts of offers, requests, and connections. While the Transport Manager does not maintain descriptive information about object-identifiers, it does arbitrate all traffic between clients and servers, making it very similar to a front-end name server.

Aspen was originally designed with a strong concept of ownership and related discretionary access control mechanisms as its primary means of access control. Incorporating mandatory access controls was a major issue in developing the design for trusted Aspen. Aspen's architecture requires that processes in the Supervisor layer be trusted, hence Aspen's TCB roughly includes all of the Kernel and Supervisor layers, plus selected processes running in the Native layer [12]. Sessions in Aspen are the basic unit of resource allocation and as such, a session with its associated Native layer processes are treated as a single subject. All sessions have an associated userID and (for the trusted design) a security level. The major objects of the system are the abstractions offered by Supervisor layer servers (e.g., files, tape volumes, devices, scheduling queues).

4.2.1 Ownership and Discretionary Access Control

Aspen was originally designed such that Supervisor servers determined whether a request to access their objects should be honored. The Transport Manager, which controls all client-server communication, determines which server should receive a request, but the individual server determines whether the request is accepted. As before, most Supervisor servers imitate the conventions used by the File Server, which includes a rigid concept of ownership in which the file's owner implicitly has all access to the file. A file owner, reflected by the second identifier in the file's object-identifier, may either be a user or an *account*. Accounts are groups of users and members of an account have implicit access to files owned by that account.

Associated with each file is an *access control list* (ACL) that provides the ability for the file owner to specify lists of individuals (userIDs) and group of individuals (accountIDs) with specific access modes (e.g., read, write, delete, execute). ACLs are kept as file OIBs managed by the Catalog Manager. Servers enforce discretionary access control decisions in two manners. Independent requests are interpreted separately and accepted or denied based on the client's userID. Related requests, however, are mediated in an entirely different manner. When a connection is established, it is created with an associated access mode, which is either read or write. When a client issues a CONNECT request, it includes the desired access mode. The File Server validates whether the client has access to the file in the requested mode and if so, accepts the request. Once the connection is accepted, the Transport Manager maintains the connection's access mode and makes it available to the connected server. Thus, once the File Server allows a connection in a particular mode, it need only ensure that all subsequent related requests associated with that connection are appropriate for the connection's access mode.

From the above description, it would appear that Aspen incorporates a distributed approach to discretionary access mediation. However, the details of how Supervisor servers implement this mediation is more like a back-end catalog server approach. As previously described, the File Server interacts with the Catalog Manager to access catalogs of file OIBs. This interaction is actually via a connection between the File Server and the Catalog Server to the appropriate catalog. It is via this connection that the File Server accesses the file OIBs associated with a particular catalog and determines whether the file exists. For example, if a client requests access to a file `A.B.C.D.E`, the File Server first establishes a connection (if not already established) to the catalog `A.B` managed by the Catalog Manager. The File Server then requests (via this connection) the OIBs for file `C.D.E`. If the file does not exist, the Catalog Manager informs the File Server which informs the client. For independent requests, the File Server includes the client's userID in the request to access the file's OIBs. Using this information, the Catalog Manager makes the decision whether the client can access the file. The File Server only enforces the Catalog Manager's decision by assuring that related requests received across an established connection are appropriate for the access mode associated with the connection.

The major exception to this approach for discretionary access mediation is the Scheduler server, which does not use OIBs to maintain information about its scheduling queues called *transaction tables*. Rather, the Scheduler stores this information directly in private files (of course, via the File Server). Thus, the Scheduler enforces discretionary access control on transaction tables** directly, without direct interaction of the Catalog Manager.

4.2.2 Mandatory Access Control

Since Aspen relies heavily on servers for providing access to system resources, the major task for incorporating mandatory access control into Aspen was to control client-server interaction. In this effort, all four of the server-oriented access mediation approaches (kernelized, front-end, back-end, and distributed) were examined. A kernelized approach was ruled out quite early. The main reason for this decision is that the Aspen Monitor was nearly completed and Amdahl preferred not to significantly modify it. Even so, Aspen was designed such that access mediation for server-based objects was not meaningful in the kernel. The architecture of Aspen strongly suggested a server-based approach for access mediation.

The initial approach attempted to have server-related access mediation performed in a central location by the Transport Manager (i.e., front-end mediation). It was apparent that the Transport Manager had to perform some access mediation. Since servers and clients can both be untrusted Native layer subjects, the Transport Manager's server-client abstractions provide a major mechanism for inter-subject communication. Additionally, offers made by a server may contain up to 40 characters of information which, if global, would be visible to all sessions. Thus, the Transport Manager had to be modified to associate security levels with all offers made by Native layer servers that could be used to restrict communication with such servers. A global offer made by a Native server would only be visible to Native layer sessions at the same security level. Thus, since a client may only send requests to offers that are visible to it, untrusted server-client communication is controlled by the Transport Manager.

However, extending this notion to Supervisor layer servers proved more difficult. The nature of the Supervisor servers demonstrated that they each must be fully trusted servers, servicing requests from clients at all security levels in addition to other trusted Supervisor layer servers. Further, the problem with Supervisor layer servers is not necessarily controlling access to the servers, but rather controlling access to the resources the servers manage. Thus, mandatory access control for Supervisor servers has to be based upon the resources represented by an object-identifier associated with a request and not the offer made by the server.

In order for the Transport Manager to perform mediation based on individual object-identifiers, it must possess the appropriate information (e.g., security level) about the object. Further, this information differs depending on the object-type (e.g., multilevel devices versus single-level devices). In addition, the nature of a request is also dependent upon the object-type and requires an understanding of how the managing server implements the request. Thus, to facilitate the Transport Manager's ability to perform access mediation for Supervisor servers' objects, it must maintain (or access) this information about the objects. Unfortunately, the Transport Manager was not originally designed as a true name server, rather it was designed to simply resolve overloaded offers and to facilitate client-server communication. Its design did not encourage the addition of this added information, which would essentially be redundant with the logic already provided by the individual server. An alternative was to have the individual servers perform mediation themselves.

The final design proposed for Aspen incorporated mandatory access controls for Supervisor server objects in a fashion as was originally designed for discretionary access controls, i.e., via individual servers interaction with the Catalog Manager. In addition to passing the client's userID when requesting a file's OIBs, the Supervisor servers would also pass the client's security level allowing the Catalog Manager to determine whether the client may access the file in the requested mode. For related requests, once the Catalog Manager validated the creation of a connection, the individual server would need only ensure that the connection is used to access only the associated object in the associated mode.

4.3 ASPEN SUMMARY

Though the upgrade for Aspen was only partially implemented at the time the effort was discontinued, a multilevel secure design for Aspen had been nearly completed. Discretionary access controls were included in Aspen's original design and were implemented as a cooperative effort between the individual Supervisor servers and the back-end Catalog Manager. The multilevel design of Aspen split mandatory access controls between the cooperative relation of the individual servers and the Catalog Manager for trusted server-based objects, and the front-end Transport Manager for untrusted server-client interactions. Though this approach to mandatory access control resulted in a distributed reference validation mechanism, its highly-structured architecture appeared to allowed the mediation mechanisms to be a faithful implementation of the reference monitor concept.

5.0 MACH

Mach is currently an operating system kernel that is designed to be transportable over a range of differing hardware from microprocessors to large parallel machines. Because of this design goal, dependence upon specific hardware features is minimal. Although most of the features of the Mach kernel, as specified in the Mach Kernel Interface Manual [13], have been implemented,

**The nature of this policy is out of the scope of this paper.

the nonkernel servers that will provide the bulk of the system functionality are currently being designed by CMU. It is expected that the CMU design will follow the general pattern of Accent, the precursor of Mach. Mach is still a prototype system in a state of rapid evolution. Under DARPA contract, TIS is developing a Trusted Mach (TMach) prototype that will provide a proof-of-concept for the transformation of a fully kernelized, server-oriented Mach system into a trusted operating system.

5.1 TMACH ARCHITECTURE

TMach is structured as a kernel which provides a small set of basic objects and services, and a collection of system servers that provide the bulk of the operating system functionality. Since Mach is designed to be transportable, little is specified about the mapping to any one hardware configuration; however some assumptions are made about the hardware base. It is tacitly expected that any hardware will have at least two execution states. The Mach kernel executes in the most privileged state and the system servers (along with untrusted user applications) execute in the unprivileged state. Address space separation provides system servers protection from each other and other system entities. Additional hardware protection features are used if they exist, but are not assumed by the CMU design.

5.1.1 The Kernel

The Mach kernel handles process management, interprocess communication, and memory management. In the fully kernelized system it will also handle low level I/O and device management. Currently the Mach kernel supports four basic abstractions. A *task* is an execution environment and the basic unit of resource allocation. Each task is a separate virtual address space. A *thread* is the basic unit of execution. A task may have several threads executing within its environment, all having access to the same set of resources. A *port*, the most central abstraction in Mach, is a message queue protected by the kernel and also act as the basic object reference mechanism. In addition, ports are also used as the primary mechanism for tasks to communicate with the kernel. A *message* is a typed collection of data used in communication via ports. A message, which has a fixed size header and a variable size body, can include the transfer of access rights to ports.

Ports have three types of access rights associated with them: *send*, *receive*, and *own*. A port may only have a single receiver and single owner but any number of tasks may possess send rights to the same port. Receive and own rights to a port also imply send rights to that port. Threads and tasks are represented by special ports called *tasks_ports* and *thread_ports*. The kernel holds both receive and own rights to all task_ports and thread_ports. Any task*** that holds send rights to a task_port or thread_port can issue commands to the kernel for which the results of the command will affect the associated task or thread.

Message passing is the primary means of communication both among tasks and between tasks and the Mach kernel itself. Thus information flows in the system is the result (directly or indirectly) of the kernel processing message send or receive requests. The only commands implemented by system traps are those directly concerned with message communication (`msg_send`, `msg_receive`, and `msg_rpc`) and a few others (`task_self`, `task_data`, `task_notify`, and `thread_self`). The rest are implemented by sending messages to a task_port or thread_port, all of which have the kernel as the receiver.

5.1.2 System Servers

The bulk of TMach functionality is provided by a set of system servers. The most central of these servers is the Name Server which provides (with support from the kernel) the mechanisms by which servers and clients can interact. The Name Server essentially manages a hierarchical directory structure of server-managed abstractions called *items*. The Name Server maintains certain descriptive information about each defined item including its relationship within the directory tree, its name, and its *item type*. All defined items are an instantiation of an item type (e.g., directory, file). Each item type is managed by a particular server which is identified by a port, called a *server-port*, to which the Name Server possesses send rights. The Name Server itself directly manages all directory item types. Client tasks access a particular item by sending a request to the Name Server which in turn routes the request to the server-port for items of that type. Other system servers that will be included in our prototype include File Server (mass storage management), Verification Server (user identification and authentication), SysAdmin Server (support for system administration), and Audit Server (collection and query of audit data).

Architecturally, TMach's trusted servers avoid the mutual dependency problems that Aspen possessed. The Name Server is logically the most primitive of all the trusted servers. Therefore, the Name Server will only depend upon the kernel for its correct operation. The kernel provides direct support for storage and retrieval of the Name Server's internal database ensuring that the Name Server is not dependent upon the File Server. The remaining servers may depend upon the Name Server for storage of private information (e.g., the File Server may store disk addressing information with each file item record maintained by the Name Server). Our current plans are for trusted servers other than the Name Server to depend upon the File Server for mass storage requirements.

5.2 ACCESS CONTROL IN TMACH

In the current TMach design, access control is provided by the kernel for basic system objects (i.e., ports, tasks) and by the Name Server, in conjunction with the kernel, for server-managed objects.

5.2.1 Access Control in the Kernel

In a very real sense, ports are similar to *capabilities* [14]. The kernel maintains the equivalent of a *capability-list* for each task that defines which ports the task can name and in which mode (e.g., send, receive).

***Although a task is simply an execution environment, for simplicity of discussion, "task" is being used as an active entity in this paper. The more correct wording is "thread within a task."

However, Mach's implementation of ports avoids the inherent access control inabilities commonly associated with capability-based systems [15]. Specifically, tasks can not pass port-rights directly to other tasks. The "capability-list" for each task is implemented as a global, kernel-protected hash table called the *T-P table*. Port-rights for a task may be added to this table in only a few well-defined ways — primarily via the message send and receive kernel primitives (i.e., `msg_send`, `msg_receive`, `msg_rpc`). The kernel recognizes when a message contains a transfer of a port-right and updates the T-P table accordingly.

In TMach, the kernel will mediate all transfers of port-rights to ensure that no task possesses a right in violation of the system security policy. This is accomplished by mediating all messages that contain a transfer of a port-right. Once a task obtains a port-right, it may use the port without further mediation by the kernel. In this manner, access to ports and therefore the flow of message traffic is guaranteed to be consistent with the system security policy. It is expected that the addition of this mediation on port-right transfers will have a very small impact on the performance of the Mach kernel.

The kernel performs two forms of access mediation. The first is a mandatory label-based security policy that is an interpretation of the Bell-LaPadula model [16]. The kernel maintains security levels for all tasks and ports. Tasks have two levels, *maximum* and *minimum*, that define a range over which a task may possess port-rights. The actual security policy enforced is a direct derivative the Bell-LaPadula model's simple-security condition and *-property [17]. The majority of tasks (e.g., all untrusted user tasks) will have their maximum and minimum security levels set exactly the same (i.e., single-level subjects).

The second form of access mediation within the kernel is an identity-based security policy. The kernel maintains a userID for all tasks and can determine which task currently holds receive rights for all ports. Using this information, the kernel enforces the following additional constraints on the transfer of port-rights:

(1) Receive and own rights for a port *P* cannot be transferred unless both the sending and receiving tasks have the same userID ; and

(2) Send rights for a port *P* cannot be transferred unless both the sending and receiving tasks have the same userID, or the sending task and the task which currently possess receive rights for port *P* have the same userID.

The first constraint essentially disallows the transfer of any receive right except among tasks of the same userID. The second constraint allows a task to transfer send rights to another task only if it, or another task with the same userID, has control of the port. TMach also provides two privileges that may be associated with tasks. Transfer-receive-rights privilege allows a task to violate constraint (1) and transfer-send-rights privilege allows a task to violate constraint (2). These identity-based controls are not *discretionary* as normally associated with identity-based policies, but rather are *mandatorily* enforced by the kernel. They, along with the associated privileges, provide the "hooks" that can be used to enforce a more traditional discretionary access control policy on server-based objects or perhaps a more interesting identity-based policy.

5.2.2 Access Control in the Servers

For TMach, the locus of access mediation for server-based objects will be within the Name Server, which will be a multilevel task which possesses the transfer-send-rights privilege. The Name Server will enforce both mandatory and discretionary access control policies with the assistance of the kernel provided controls discussed above. Since all interaction between clients and servers must be initiated via the Name Server, it provides a central location in which the kernel's security policy can be extended to more abstract objects. The controls enforced by the Name Server are of two forms, those placed upon the server tasks and those placed upon the client tasks.

Servers offer items by registering themselves with the Name Server. This registration consists of two phases. The first is to create an item type by defining an *type-record* to the Name Server. The Name Server maintains these records along with the userID and security levels of the creating task and a specified access control list (ACL). The second phase, which can occur concurrently with the creation of a type-record or separately (i.e., for permanent item types such as files), is to provide the Name Server with a port over which the server will receive requests for items of that type. In order to register this port, called a *server-port*, the server's task must have the same userID of the type-record's creator and have the exact same maximum and minimum security levels as stored in the type-record.

Client tasks access all server-managed objects via the naming conventions implemented in the directory tree structure maintained by the Name Server. Client tasks cannot directly access (and therefore directly name) items maintained by servers. All references to server-managed items, including their creation and deletion, is directly managed and mediated by the Name Server. When an item is created, the Name Server ensures that the client is both on the ACL for that item type and has maximum and minimum security levels within the range stored for that item type (this prevents illicit communication between clients and servers). When an item is created, the Name Server associates with the item the security level of the creating task (or for multilevel creating tasks, a specified security level within the task's maximum and minimum security levels) and an ACL specified by the client. To subsequently access an item, a client task sends a request to the Name Server specifying the name of an item and a port over which a response is expected (called a *reply-port*). The Name Server will mediate the request and, if the request passes access mediation, will forward the request along with the reply-port (hence the need for transfer-send-rights privilege) to the server-port for items of that type. In the case of single-level servers, this mediation (along with the kernel's controls on the transfer of port-rights) is sufficient to ensure that communication is in accordance with the system security policy. In the case of trusted multilevel servers, the servers are expected to function correctly but are not expected to enforce any additional constraints. For example, the File Server, after receiving a request to open a file for "read", must ensure that it only allows the client to "read" the file since the Name Server mediated the request for "read" access. One manner in which this may be accomplished is for the File Server to create a port, associate the port with the target file, transfer send rights for the port to the client via the client's reply-port, and ensure that subsequent requests received via this port are all "read" in nature.

5.3 MACH SUMMARY

The current TMach effort is to develop a prototype system that explores the feasibility of developing a B3 system based upon CMU's Mach kernel. The current Mach kernel is not complete and relies upon UNIX™ to perform all I/O functions. However, we expect that a complete kernel (i.e., the inclusion of I/O and device control) can support a highly trusted system. The kernel will eventually provide only task/thread management, most memory management, message-passing capabilities, and primitive I/O and device management. We expect that a general-purpose server-based system built upon this kernel can meet the B3 requirements. In the case of the TMach prototype, the Name Server provides a locus for additional access mediation required for the more abstract server-managed objects while the kernel provides resource isolation, controlled message-passing, and access mediation for primitive objects. This allows a server layer to be constructed that exhibits strong modularity and independence among the trusted servers. Our initial proof-of-concept prototype is expected to be operational by the end of this year.

6.0 CONCLUSIONS

We have found that server-based mediation can be used to implement a reference validation mechanism that is entirely faithful to the Reference Monitor Concept. This conclusion is supported by the fact that the inherent structure of such systems allow access controls to be included in trusted servers such that the completeness and correctness of the mediation mechanisms can be assured. The trusted designs of both Aspen and Mach incorporate access controls within servers. Aspen's mediation is designed entirely in the Supervisor layer while TMach has a kernel which provides resource isolation and access mediation on primitive objects and a Name Server which (in conjunction with the kernel) provides access mediation for more abstract objects.

It is apparent from Aspen, Mach, and other systems we have examined, that the server-oriented paradigm is becoming a popular design philosophy in new operating system development efforts. We feel that in many cases, reference validation contained entirely within a security kernel is not an optimal solution to access control concerns for this type of system. A server-based approach for access mediation is a viable alternative.

REFERENCES

[1] *DoD Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, National Computer Security Center, Ft. Meade, MD, December 1985.

[2] J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-75-51, Vol. I, AD-758 206, Hanscom AFB, Bedford, MA, October 1972.

[3] J. F. Hogg, "Aspen System Overview," Amdahl Corporation, Santa Clara, CA, October 1984, *unpublished*.

[4] Richard Rashid, "From RIG to Accent to Mach: The Evolution of a Network Operating System," Technical Report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, May 1986.

[5] Martha A. Branstad, Pamela S. Cochrane, et al., "Trusted Mach Design Issues," *Proceedings Third Aerospace Computer Security Conference*, Orlando, FL, December 1987.

[6] Roger R. Schell, "Security Kernels: A Methodical Design of System Security," Technical Papers, Use Inc. Spring Conference, Blandensburg, MD, 1979.

[7] Lester J. Fraim, "Scomp: A Solution to the Multilevel Security Problem," *IEEE Computer*, Vol. 16, No. 7, July 1983.

[8] B.D. Gold, R.R. Linde, and P.F. Cudney, "KVM/370 in Retrospect," *Proceedings of 1984 IEEE Symposium on Security and Privacy*, Oakland, CA, April-May 1984.

[9] E.J. McCauley and P.J. Drongowski, "KSOS — The Design of a Secure Operating System," *AFPIS Conference Proceedings*, Vol. 48, 1979 NCC, AFIPS Press, Montvale, NJ, 1979.

[10] *IBM System/370 Extended Architecture — Principles of Operation*, First Edition, International Business Machines, Inc., Poughkeepsie, NY, March 1983.

[11] D. Elliott Bell and Frank L. Mayer, "A Model Interpretation of Aspen," TIS Report 124, Trusted Information Systems, Inc., Glenwood, MD, September 1987.

[12] "Aspen Security Architecture," Ver. 2.1, Trusted Information Systems, Inc., Glenwood, MD, June 1987, *unpublished internal draft*.

[13] Robert Baron, et al., "MACH Kernel Interface Manual," Technical Report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, April 1987.

[14] Henry M. Levy, *Capability-Based Computer Systems*, Digital Corporation, Bedford, MA, 1984.

[15] W.E. Boebert, "On the Inability of an Unmodified Capability Machine to Enforce the *-property,

Proceedings of the Seventh DoD/NBS Computer Security Conference, Gaithersburg, MD, September 1984.

[16] D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2998, The Mitre Corporation, Bedford, MA, July 1975.

[17] Frank L. Mayer, "Formal Security Model Considerations for Trusted Mach," TIS Report 139, Trusted Information Systems, Inc., Glenwood, MD, January 1988.

LOCK/ix: On Implementing Unix on the LOCK TCB

Mark A. Schaffer
Honeywell Secure Computing Technology Center
2855 Anthony Lane South - Suite 130
Minneapolis, MN 55418
(612) 782-7134
APARNET: Schaffer@DOCKMASTER

Geoff Walsh
R&D Associates
4640 Admiralty Way
Marina del Rey, CA 90295
(213) 822-1715

ABSTRACT

The LOgical Coprocessing Kernel (LOCK) is a Trusted Computing Base (TCB) that is designed to meet and exceed the requirements for a Class A1 secure system. This paper describes the results of a study that determined how to port the Unix* System V Operating System to the LOCK TCB, while maintaining maximum compatibility with the System V Interface Definition (SVID) [SVID86].

1.0 Background of the Problem

Over the years, Unix has gained widespread acceptance as the de facto standard Operating System (OS) within the U.S. Government and private industry. During the same time that Unix has gained in popularity, a demand for secure computing systems has developed. Recently, the demands for these two technologies have created a demand for secure Unix systems within the user community.

To help meet the demand for secure Unix systems, we decided to port Unix to LOCK rather than develop a new OS. This is very appealing from both a developer and user point of view because of the large amount of portable Unix applications that already exists.

1.1 Background of the Solution

Traditional approaches to providing Multi-Level Secure (MLS) computing systems have emphasized implementing software security kernels that run when the target processor is operating in privileged mode. In some cases, security has been provided by redesigning the OS. These purely software approaches to providing multi-level security have four primary disadvantages:

1. DECREASED ASSURANCE since a software malfunction could cause total security failure
2. DECREASED PERFORMANCE to usually unacceptable levels because of the high overhead incurred by performing the security access checks in software
3. LOSS OF EXISTING APPLICATION SOFTWARE because of the extensive redesign of the operating system, and
4. INABILITY TO FUNCTIONALLY ENHANCE the OS without requiring expensive and time-consuming re-verification and revaluation [SAYD87].

The LOCK TCB is a MLS computing system currently being prototyped at Honeywell Secure Computing Technology Center. It has been designed to meet and exceed the requirements for a Class A1 system as defined in the DoD Trusted Computer System Evaluation Criteria (the Orange Book) [TCSEC85].

LOCK is the third phase of a continuing project previously called the Secure Ada Target (SAT), which was started by Honeywell in 1982. The first phase of the SAT program (SAT-0) resulted in a high-level requirements specification [HONE83]. The second phase (SAT-I) resulted in an intermediate specification [HONE86]. The third phase (SAT-II), renamed LOCK, will result in a detailed design specification and MLS minicomputer prototype in 1990 [SAYD87].

1.1.1 The LOCK Solution to Multi-Level Security

The LOCK system takes a hardware-oriented approach to providing a MLS computing system.

This approach should enable the system to overcome the disadvantages associated with purely software approaches.

The security policy of the system is enforced by a physically separate, multi-processor, coprocessing unit called the System-Independent, Domain-Enforcing, Assured, Reference Monitor (SIDEARM). The SIDEARM has its own processors, memory, and mass storage. All security-related data is stored on the SIDEARM mass storage unit. All security policy decisions and access computations are performed by the SIDEARM.

The physical separation of the protection-critical from the non protection-critical elements in the LOCK system makes it physically impossible for a user process to access or tamper with the SIDEARM firmware or its data, giving the LOCK system a high degree of assurance.

The LOCK host processor provides TCB-mediated resource management and computing power for user applications. Since it performs no security access checks, the performance degradation imposed on the system by the security mechanisms should be minimal.

*Unix is a trademark of AT&T.

The results reported in sections 1.0 through 4.0 were supported by National Computer Security Center contract MDA904-87-C-6011.

The security functionality provided by the SIDEARM is generic in nature, and largely independent of the characteristics of the host processor. The security policy that the SIDEARM enforces is configured through the use of special administrative tools at system generation time. Virtually any security policy can be defined to meet the needs of an installation.

Other than the physical hardware connection between the SIDEARM and the bus of the host processor, the SIDEARM is also mechanically independent of the host architecture. A fundamental design goal of the SIDEEARM was to design it in such fashion as to allow it to be ported to other non-LOCK systems.

The LOCK OS will not be responsible for enforcing the security policy of the system, and therefore, it will not be part of the TCB and not have to be verified or evaluated when it is updated. Further, we make the assumption that the OS, and other non-TCB software for that matter, are hostile programs that will attempt to violate the security policy of the system. As a consequence of this assumption, we follow a least-privilege design philosophy for all LOCK software and rely heavily upon Type Enforcement (see subsection 2.2.2) to limit the objects application may access.

Since Unix is not part of the TCB, we will not have to modify it to provide security policy enforcement mechanisms. These capabilities are provided by the underlying TCB.

We do plan to extend the Unix interface in a non-intrusive manner to make the MLS features (e.g. ACLs) of the LOCK TCB available to users and applications. With the implementation approach we have developed, we should be able to maintain a great deal of compatibility with the SVID and, hence, with the existing base of Unix applications.

1.2 The Study Goals and Results

During 1987, we performed a study of the Unix kernel to determine if it could be (relatively easily) ported to the LOCK system, and if so, determine what the effect on the interfaces would be. To enable us to determine if it would be worthwhile to port Unix, we established the following research goals:

- The number of modifications to the Unix kernel should not be extensive.
- The TCB could not be modified to "tailor" it to running Unix.
- Unix had to be able to service many concurrent users running at different security levels without becoming part of the TCB.
- The file system had to be able to manage data at different security levels requiring trusted servers and without introducing covert channels.
- The resultant system must maintain a maximum compatibility with the SVID.

The results of our study indicate that these goals can be met. The application visible interface to the LOCK implementation of Unix (LOCK/ix) is nearly identical to that of a standard implementation of Unix System V. The security policy enforced by the underlying LOCK TCB should have little, if any, impact on the majority of existing Unix applications.

We feel one major result of the study is our approach for implementing an untrusted file system (see section 4.0) that will manage the multi-level data. Internally, our file system implementation will be quite different than in a standard Unix kernel. However, users and applications should not notice the differences.

Our file system design was originally intended to support LOCK/ix. However, we feel that it is general enough that it can be used to support other (non-Unix) LOCK applications, or be applied to other non-LOCK TCBs as well.

1.3 Overview of the LOCK Architecture

The LOCK system consists of two computing units: the SIDEARM and the host processor. The majority of the TCB functionality resides in the SIDEARM, whose firmware coordinates with a small (TCB) software kernel (the Supervisor) that runs on the host processor.

The resultant LOCK TCB provides low-level services for subject, object, and device management. The TCB is restricted, for reasons of verifiability, to minimum functionality. It is intended to support, not replace, traditional OS services, such as a hierarchical file system.

The Supervisor (see Figure 1) functions as a low-level resource manager, and provides an application visible interface to the TCB's capabilities. The Kernel Extensions are a set of verified, security-relevant utilities whose capabilities cannot be provided by the SIDEARM in a generic fashion.

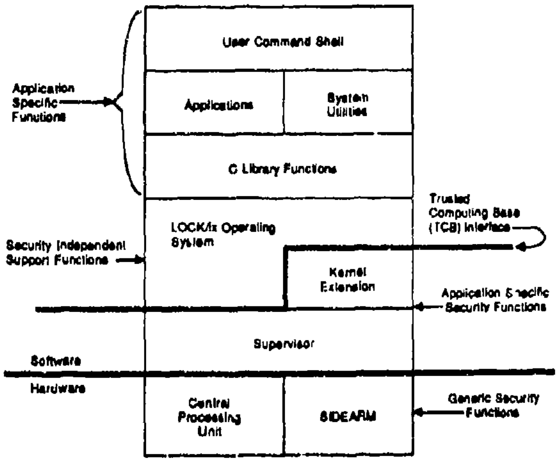

User Command Shell
Applications
System Utilities
C Library Functions
Application Specific Funutions
LOCK/ix Operating System
Trusted Computing Base (TCB) Interface
Security Independent Suppon Functions
Kernel Extension
Application Specific Security Functions
Supervisor
Software
Hardware
Central Processing Unit
SIDEARM
Generic Security Functions

Figure 1

1.3.1 The SIDEARM

The SIDEARM implements what is called the Reference Monitor (RM) concept (see Figure 2). In general, an RM can be thought of as a guard between people, and the information they would like to access. There are three important criteria for an RM:

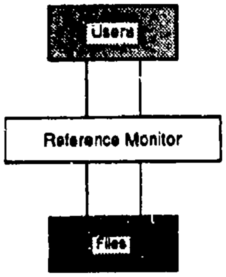
1. It must always be invoked.
2. It must be verified to be correct (i.e., properly enforce the security policy of the system).

3. It must be unbypassable.

4. It must be tamperproof.

The LOCK hardware-oriented approach (see Figure 3) provides a good match to the RM model [SAYD87].

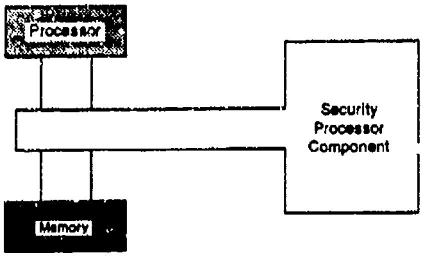
Reference Monitor Concept



A Reference Monitor Must Be
1. Always Invoked
2. Verified Correct
3. Unbypassable
4. Tamperproff

Figure 2

Security Coprocessor Component Approach



How A Security Coprocessor Meets Reference Monitor Criteria
1. Always Invoked - No Way To Bypass
2. Verified Correct - Simpler, Machine Independent
3. Tamperproof - No Way To Attack Security Coprocessor

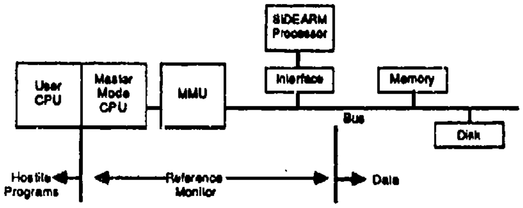
Figure 3

When the system is booted, the SIDEARM is booted and initialized before the host processor begins to run and continues to run until the system is shut down. All security-related data and most of the security functionality is implemented in the SIDEARM, thus making it possible to verify that it is correct. And finally, since the SIDEARM is physically separate (see Figure 4) and maintains its own memory and mass storage, there is no (physical) way for a user process to tamper with its firmware or data. It is unbypassable since it is the SIDEARM, and not the Host processor, that has exclusive control over the Memory Management Unit (MMU).

1.3.2 The Host Processor

As mentioned previously, a small software kernel (which is part of the TCB) runs on the host processor. This software kernel is responsible for preserving, and not enforcing, the security policy of the system by performing correct, low-level resource management. This software kernel, called the Supervisor, consists of code that runs in both privileged and user mode of the host processor. The portion of the Supervisor that runs in the privileged mode is only that code which is forced there by the hardware, such as the interrupt handlers. Other code, such as the subject scheduler, runs in user mode of the host processor.

Reference Monitor in LOCK



Hardware Advantages
1. High Assurance
2. Reasonable Performance
3. Application Portability
4. Functionality

Figure 4

All code that runs in privileged mode will be placed in Read-Only Memory (ROM) that is addressable only when the processor is running in privileged mode, thereby making it tamperproof. Other software, such as the OS, will run in user mode on the host processor.

2.0 Overview of the LOCK Security Model.

The LOCK TCB enforces a MLS policy. The policy is enforced by mediating access between subjects, the active entities of the system, and objects, the inactive entities of the system.

To enforce this policy, the SIDEARM maintains a large database called the Global Object Table (GOT). Each time a subject or object is created, it is assigned a unique identifier (UID). A GOT entry is then created for the new entity where the UID is used as the primary key. A GOT entry will contain additional information such, as the level and the creator.

The LOCK TCB provides Discretionary Access Control (DAC) and Mandatory Access Control (MAC) mechanisms to enforce the system's security policy. In order for a subject to be granted access to an object, the request must be allowed by both the DAC and MAC mechanisms of the system.

2.1 Discretionary Access Control Policy

A DAC policy is discretionary because its administration is up to the discretion of the system users. The LOCK TCB provides Access Control Lists (ACLs) as the mechanisms for providing DAC.

ACLs allow a user to specify, for each named object he is authorized to control, a list of named individuals and a list of groups of named individuals and their respective modes of access to the object. Additionally, for each named object, the authorized user may specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given. The currently supported modes of discretionary access are read (r), write (w), execute (x), and null (n).

One aspect of the LOCK DAC policy that should be noted is the fact that there is no concept of object ownership, or any special privileges associated with object ownership. Rather, the security policy the system enforces is configured to indicate who can control (change the ACL of) objects of a specific type. This typically will be, but is not limited to, the creator of an object.

2.2 Mandatory Access Control Policy

A MAC policy is mandatory because it is always enforced by the system. Unlike a DAC policy, the system users have no say in how the policy is administered. The LOCK MAC policy is enforced by Labeled Security Protection and Type Enforcement mechanisms.

2.2.1 Labeled Security Protection

The LOCK TCB enforces Labeled Security Protection as required by the Orange Book. The policy is enforced over all system resources (e.g., subjects, objects, and I/O devices) that are directly or indirectly accessible by subjects external to the TCB.

The LOCK TCB maintains a SIDEARM-resident data structure that is a partially ordered set (POSet) of all security levels (combination of one hierarchical level and a set of non-hierarchical categories) defined in the system.

The LOCK POSet is a generalization of the Orange Book concept of a security lattice. The difference them is the fact that the POSet has no lowest or highest bound. For example, two levels may be equal in classification but have a different (and incompatible) category set (e.g. TOP_SECRET.A and TOP_SECRET.B). Hence, one can not be said to be "higher" than, or dominate the other.

When a subject or object is created, it is assigned one of the levels (nodes) from the POSet. Access is then computed using the level of the subject requesting access and the level of the object being accessed in the following manner:

- To read an object, the level of the subject must dominate the level of the object (the Simple Security Property).
- To write an object, the level of the subject must be dominated by the level of the object (the *-Property).

As used in the rules above, the term dominate means greater than or equal to. For each required access computation, the POSet is consulted to determine if one level dominates another.

2.2.2 Type Enforcement

Type enforcement is a mechanism that is unique to the LOCK TCB. Not required by the Orange Book, it is this mechanism that will (in part) allow the LOCK TCB to exceed the Orange Book Class A1 requirements. Type enforcement is based on two attributes:

- The domain of execution of a subject.
- The type of the object a subject is attempting to access.

A domain is similar in concept to rings in ringed architecture machines. Unlike rings, though, there is no hierarchical relationship between domains. Moving from one domain to another does not necessarily imply an accumulation of increasing system privilege. Rather, each domain has a set of privileges different from other domains.

To represent the domains and the privileges allowed in them, the TCB maintains a SIDEARM-resident data structure called the Domain Table. It contains the following information:

- The UID of the domain.
- The human-readable name of the domain.
- A list of special privileges.
- A list of domains to which other subjects in the named domain have access to, and the modes of access (create a subject, destroy a subject, signal a subject, etc.) permitted.

The special permissions that are allowed in domains are the ability for a subject to take exception to the DAC and/or the Labeled Security Protection mechanisms of the system. Since it is the type enforcement mechanism that allows a subject to have these special privileges, a subject may never take exception to the type enforcement rules of the system.

The domain of execution is an attribute of a subject that remains constant throughout its lifetime. In other words, a subject can only execute in one domain.

All objects have a type associated with them. The concept of type is similar in nature to types in high level programming languages. The TCB restricts operations on objects of specific types based on the domain of execution of the subject attempting the access.

To represent object types and the operations allowed on them, the TCB maintains a SIDEARM-resident data structure called the Type Table. It contains the following information:

- The UID of the type.
- The human-readable type name.
- Allowable object sizes (minimum and maximum).
- List of domains from which subjects have access to objects of the named type, and the modes of access (read, write, etc.) permitted.
- Default ACL.

When a subject requests access to (or attempts to create) an object, the TCB consults the Domain and Type Tables to determine if the access, based on the domain of execution and the object type, is allowed.

Both the Domain Table and Type Table are initialized at system generation time by the System Security Officer (SSO) and are inaccessible to user processes. It is this ability to configure the Domain and Type Tables that enables the LOCK to support virtually any security policy an installation desires.

As mentioned earlier, type enforcement can be used to grant special privileges. For example, it may be necessary to implement an application that is allowed to downgrade files. The list of special privileges in the Domain Table is used to grant such privileges. The Type Table is used to restrict which object types can be read and written in the downgrade process.

Type enforcement is also useful for integrity reasons. For example, the system may grant subjects running in a system administration domain read and write access to objects of type username_file. Subjects running in the OS domain may be granted only read access to objects of type username_file. With the Domain and Type Tables established in this fashion, the system will prevent unauthorized modification (integrity) of objects of the type password file.

The type enforcement mechanism can be used to support a variety of integrity models such as the Clark-Wilson [CLARK87] model, and as described in [BOEB85], the Biba [BIBA75] model as well.

2.3 Subjects

The basic execution (active) entity in LOCK is the subject. A subject is a process that executes in a particular security context. The security context comprises the level of the subject, the domain of execution, and the user on whose behalf the subject is executing. In many ways, a subject is like a Unix process: it shares the processor with other subjects through timeslicing, it has access to a "file system" that other subjects also have access to, it can open and operate on "files," and it has limited capabilities for communicating with other subjects.

There are some notable differences between subjects and Unix processes. There are no hierarchical parent/child relationships; each subject is independent of the subject that created it. For Unix processes with the user ID of superuser, the entire system is accessible; there is no corresponding notion of superuser in LOCK/ix. Under Unix, multiple processes can be writing to the process control terminal simultaneously; LOCK allows only one subject to perform terminal I/O to a given (process control) terminal at a time.

All subjects have associated with them a Subject Translation Table (STT). The STT contains an entry for each object that the subject has opened (see Figure 5). In LOCK/ix, objects are used to represent Unix file system objects (file, directories, etc.), text segments and data segments, process stacks, and kernel level data structures. The STT is similar in nature to the Unix per-user open file table. Each entry in the STT identifies an object and the current access that the subject has to it. The STT is resident in the host processor's memory and provides the first level of address translation for the MMU.

Subjects within the LOCK system are characterized by the following:

- Each subject is uniquely identified within the SIDEARM's security database (the GOT) by the UID the SIDEARM assigned to it when it was created. The GOT entry represents the security context of the subject.
- The subject manager (within the host resident portion of the TCB) maintains a data structure called the Active Subject Table. Each active subject within the system is uniquely identified by subject manager by its entry in the Active Subject Table.
- A user subject may execute instructions if and only if the host processor is operating in user mode. (Only TCB subjects may operate when the Host processor is in privileged mode.)

User subjects are created as a result of a TCB Create Subject request. They come into existence as the result of a user action, perform their function, and are terminated by a TCB Destroy Subject request at some later time. When a subject is destroyed, the subject ceases to exist within the system. All objects allocated by the subject (contained within its STT) are closed, and all resources (e.g., GOT entry, memory, etc.) previously allocated to the subject are released.

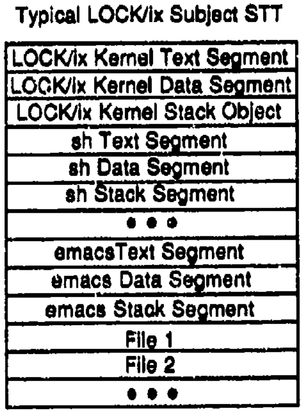


Figure 5

2.3.1 Relation to Unix Virtual Machines/Unix Processes

The differences between Unix processes and LOCK subjects strongly influenced the way Unix processes are represented in LOCK/ix. To cleanly support Unix process mangement functionality, each subject represents the equivalent of an abstract Unix virtual machine.

To provide support for operating systems built on top of the LOCK TCB, as well as multitasking applications such as an Ada run-time environment, a subject has periodic software interrupt, similar to a timeslice interrupt, available to it. A "beginning timeslice" signal is sent to all LOCK subjects from the TCB when they begin to execute in a new timeslice.

To take advantage of this feature, a subject must enable a signal handler, in much the same way as is done for Unix signal handlers. If a subject does not wish to take advantage of this signal and does not define a handler for it, the signal is ignored and the subject is allowed to run without the knowledge of receipt of the signal.

Unlike the Unix signal handling mechanism, the LOCK signal handling mechanism provides to the subject its context (register, stack pointer, and program counter values) when the signal occurred. There is no way for a subject to control the frequency of this signal. The frequency of occurrence of this signal is unpredictable.

The use of this feature allows a subject to perform its own process multiplexing. Each subject can run its own process (or task) multiplexing algorithm to provide multiprocessing support within the subject.

2.4 Objects

One of the most unusual features of LOCK (at least for those accustomed to Unix) is that there is no notion of external files of or a file system; instead, there are objects. Objects are containers for data that reside in the virtual memory and can be (physically) stored on disk, tape, or other media such as optical disk.

All objects have associated with them a set of attributes that are similar in nature to Unix file attributes. These attributes include the object type, size, security level, creator, ACL, permanent location, and the present location (in main memory).

Objects are a generalization of a segmented memory system. The TCB Open Object operation maps an object into the virtual address space of a subject and returns a pointer to a memory address. Datum with an open object can be accessed by referencing offsets into the object's memory range. I/O is performed on objects by modifying the contents of memory addresses in the open object. One object can be open by multiple subjects simultaneously, with the object mapped to different virtual addresses in each subject's address space.

2.4.1 Relation to Unix Virtual Memory

All memory that a subject references, even the subject itself, consists of open objects. The virtual memory space of a subject is the union of open object virtual addresses. LOCK imposes a limit on the number of open objects a subject is allowed, which is currently 256. The maximum size of on object is 16Mbytes.

Disk I/O is performed by LOCK without explicitly doing I/O (i.e., issuing a command to a device driver). The MMU provides the mapping between memory references and modifications and physical I/O. If a piece of an open object that has been paged out to disk is referenced, the MMU causes the appropriate piece of the object to be brought into memory. To an application, the entire contents of an object appear to be in memory when an object is opened, and the contents disappear when the object is closed.

Referencing a memory address that is not mapped to an open object generates a bus error. A bus error will be interpreted by the TCB as an attempt to violate the security policy of the system and cause the termination of the offending subject. Although termination of the subject seems to be a bit harsh, under similar circumstances Unix would terminate the offending process and generate a "core dump".

LOCK object operations are analogous to Unix memory management functions in many ways. Opening an object is similar to allocating a region, in order to obtain memory for a process. Objects can expand and shrink, as can regions. Open objects are memory regions associated with each process.

2.4.2 Relation to Unix File System and Files

Objects provide the foundation for building a file system that will appear to operate similar to Unix. However, from a programming standpoint, object operations are quite different than file I/O operations.

The LOCK/ix kernel is responsible for providing the functional bridge between the LOCK TCB and Unix applications. It provides the functionality necessary to support a Unix file system built on top of LOCK objects.

File creation requires that an object be created and cataloged into the file system in the correct directory, with the inode table providing the linkage between physical storage and external appearance. Open and close operations logically perform the same function in both LOCK and Unix, making the objects "known" and "unknown" to an executing process.

LOCK/ix will map the Unix-style access operations into their LOCK counterparts. Unix-style I/O operations will be mapped into open object references and updates. File deletion removes a reference to an object from the file system, and if there are no references remaining, the object will be deleted from the file system.

3.0 Process Management in LOCK/ix

The Unix process management services provide process creation and deletion, program execution, and synchronization between related processes. The Unix model of process creation, using the fork() operation, enforces parent-child relationships between processes and ensures that a child process is initially created to be an exact copy of its parent. The Unix model of program execution, using the exec() operation, provides for the inheritance by the new program of part of the environment of the process that executes the program.

In contrast to Unix in which all user processes are managed and coordinated by a single kernel entity, the LOCK/ix implementation encapsulates the management of processes for each LOCK/ix login session within a single LOCK subject (see Figure 6). Each LOCK/ix subject contains an (virtual) instance of the Unix kernel that manages only the user processes associated with its login session. The LOCK/ix kernel is in reality a shared text segment that is used by all LOCK/ix subjects. However, at any point in time, the kernel only "knows" about the single LOCK/ix subject that it is currently servicing.

Although this approach to process management is a bit unusual, it does provide an implementation of fork() and exec() that is, from the viewpoint of an executing process, compatible with Unix. The set-user-ID modes of file execution are partially supported. They effect only the user-IDs of an individual process, and the the user-ID of the containing subject.

LOCK/ix Address Space Organization

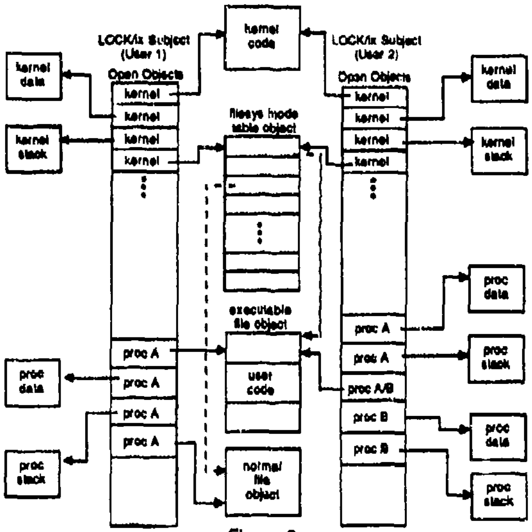


Figure 6

There is a problem that is encountered when trying to support Unix set-user-ID applications in a MLS environment. Unix maintains both a real-user-ID that indicates who logged on, and an effective-user-ID that indicates on whose behalf the process is executing, for each active process. Under normal circumstances, these user-IDs represent the same individual. When the process is an instance of a set-uid application, the effective-user-id of the process is set equal to the owner-id of the set-uid application.

Unix always uses the effective-user-id of a process for computing access rights. Running a set-user-ID program has the effect of temporarily granting the owner's access rights to the user of the set-user-ID application. In contrast to this, the LOCK TCB maintains only one user-ID for each active subject. A LOCK subject user-ID and a process real-user-ID are the same individual (the person who is actually logged on).

The LOCK TCB does not support this notion of granting temporary access. To compute access, the LOCK TCB will always use (the equivalent of) the real-user-id of a process. As a consequence of this, our current design does not provide fully compliant support for set-user-ID applications.

We are currently investigating several alternative approaches to solving this problem (see section 5.0). Since the Unix user community appears to have a strong desire to continue to run set-user-ID applications, our current design will most likely be criticized as providing unacceptable support for set-user-id applications.

There are some within the computer security community who argue that a capability such as set-user-ID should not be provided by A1 systems. This is a debate that is likely to continue for many years to come. Our feeling is that is the existing Unix file protection mechanisms, and in particular the set-user-id capability, are really integrity mechanisms. If a method can be devised to support a fully functional set-user-id capability in a secure manner, we see no disadvantage to doing so.

3.1 Memory Management

Unix kernel memory management functions provide user processes with an expandable data area that can be used for dynamic heap allocation. The heap allocation algorithms are supplied by the run-time library and provide a generalized memory block allocation scheme to user programs. They call on the kernel to expand the data segment of a user process as needed to increase the pool of memory that is available for allocation.

Although not explicitly visible at the user program interface, Unix memory management also normally enforces protection over the address space of a user process from access by other processes. The address space of the kernel is also protected from access by user processes.

These capabilities are dependent on the characteristics of the MMU. In LOCK it is the TCB, and not LOCK/ix, that has explicit control over the MMU. As a consequence of this, such protection cannot be provided by LOCK/ix.

The LOCK/ix kernel manages memory via calls to the TCB storage manager. The physical allocation of memory is replaced by requests to create, delete, open, close, and expand the LOCK objects that compose the address space of user processes. Each user process within the subject is assigned a text segment, data segment, and stack object by the kernel which are open as long as the process is executing. When a process terminates, these objects are closed and deleted by the kernel.

Expansion of the data segment is implemented by calling the TCB storage manager request Expand Object. This TCB request automatically handles physical relocation of the object if needed and zeros the newly allocated space for LOCK/ix.

Expansion of the stack object is implemented by a special LOCK/ix system call, which is used to request that the stack of the calling process be extended. This requires that the compiler generate a stack overflow check as part of every C function's entry preamble code. Although relatively inefficient, this method of automatic stack growth is used on many standard Unix machines that have no hardware support for stack overflow detection in the MMU.

4.0 The LOCK/ix File System Design

The file and I/O system is one of the principal components of Unix. Conceptually, it provides a uniform method for performing I/O, by mapping all I/O into file I/O. Applications can operate in the same manner whether I/O is being done to a terminal, a file, a interprocess communication pipe, or a physical device.

Due to the nature of the Unix file system, many applications tend to be highly I/O intensive. If the file system does not work as expected, the effect on applications ported to LOCK/ix could outweigh the effort to develop the software from scratch. A fundamental design goal of LOCK/ix was to make the file system appear as much like the Unix file system as possible.

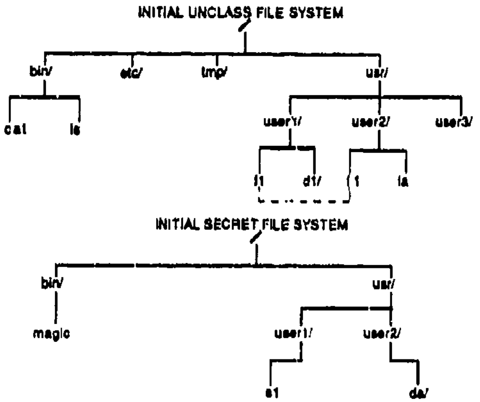
In our design of the file system structure, a separate file system at each security level is supported. An inode table for each file system is stored in an object that is equal in level to the file system it represents. Directory entries will map inodes to files as is done in existing Unix implementations.

The major difference between the LOCK/ix file system structure and the standard Unix file system structure is that directories with the same name may exist at multiple levels. The directories at all observable levels are logically overlaid to produce a virtual directory. To a user process, it will not be apparent that an observable directory has been constructed from multiple file systems. The level at which the user process is executing will determine which files are visible in its view of the file system.

To avoid the need for trusted code, a directory is restricted to containing only entries that are at the same level as the directory. However, if the same directory exists at several levels, each containing files, applications are given the illusion that the (virtual) directory contains files at multiple levels. To support this capability, only the directory paths are duplicated, not the files.

Figure 7 presents a simple two-level file system that contains a UNCLASS file system and a SECRET file system. A LOCK/ix process running at UNCLASS would see the files cat and ls in the /bin directory. A SECRET process would see those two files (as read-only), and in addition, the file magic.

Initial Two Level File System Example



THE CORRESPONDING UNCLASS DIRECTORIES ARE:

/	bin/	usr/	user1/	user2/	user3/
1 .	2 .	5 .	6 .	7 .	14 .
1 ..	1 ..	1 ..	5 ..	5 ..	5 ..
2 bin/	8 cat	6 user1/	10 f1	12 fa	
3 etc/	9 ls	7 user2/	11 d1/	10 f1	
4 tmp/		14 user3/			
5 usr/					

THE CORRESPONDING SECRET DIRECTORIES ARE:

/	bin/	usr/	user1/	user2/
1 .	2 .	3 .	4 .	6 .
1 ..	1 ..	1 ..	3 ..	3 ..
2 bin/	14 magic	4 user1/	5 s1	7 da/
3 usr/		6 user2/		

Figure 7

LOCK/ix will perform the logical combination of the file systems from multiple levels to present the illusion that there is a single, multi-level file system. The files contained in the UNCLASS file system that the process running at SECRET can observe are of course not writable. The view of the file system presented to a SECRET process will contain the SECRET file system overlaid on top of the UNCLASS file system.

An process running at UNCLASS will have no way of determining the existence of anything in SECRET (or anything above UNCLASS for that matter). The security policy enforced by LOCK deny access to any file system objects above the level of UNCLASS.

It should be noted that the LOCK TCB enforces the protection of objects so that even malicious programs can not affect the data contents of an object unless the subject has write access to the object.

The LOCK/ix design provides what appears to be the best compromise between Unix compatibility and the required MLS functionality. We have developed a design that should have the "look and feel" of Unix. Each file system is completely isolated from the file systems at other levels.

The concept of virtual directories could also be applicable to a conventional, unsecure networked Unix environment. If file systems resided on multiple machines, some with the same directory path names, virtual directories could be created. The issue of which network machine a file resided on would no longer be significant.

4.1 Path Inheritance

In order to support virtual directories, a method is needed whereby files can be created at the current level if the directory path exists only at a lower level. It would be incompatible with Unix to allow a process to create a directory path that already exists. LOCK/ix will automatically a create directory path at the subject's level, when required, to fulfill a create request. We call this operation path inheritance.

As in Unix, an attempt to create a file in a directory that does not exist, or is not observable, will fail. If the directory exists at the subject's current level, no special processing is required to fulfill a create request. Only when components of the required path do not exist at the subject's current level (but exist at a lower, observable level) does LOCK/ix need to create them. The creation of path components at the current level is handled in a manner that is transparent to user processes.

A good analogy to this operation is a virtual memory system. The working set is at first very small, containing only the top-level path components. As files are created, as with pages not in memory, a "fault" occurs and the appropriate pages are brought in from disk, or in this case, path components are created. Eventually, a stable working set is established that handles most references, for either virtual memory or the LOCK/ix file system. The difference is that the directory paths created are permanent and will continue to exist in the file system. There is no way a process can determine that a directory path was inherited.

If directory entries are created with names that (unknowingly and unintentionally) match those at a higher level, the higher level virtual directory view will contain all the files, including multiple files with the same name. The lower level view will consist of only those files and subdirectories that exist at the lower level.

A design issue that is currently open is how to handle this potential name collision. If identically named files exist at multiple levels in a directory, the higher level processes will need a way to determine, or specify, which file gets accessed. An extension to the namei routine, which performs name to inode mapping, is planned for the future. This extension will allow a process to specify the level of a file to be opened. Naturally, this capability will be restricted by the Labelled Security Protection mechanism (see subsection 2.2.1) of the TCB.

The main changes in the internal logic of Unix required to support file systems at different levels, that are combined to appear as one composite file system, has been limited to the namei module. In standard Unix, it is the namei module that performs pathname parsing to retrieve the inode that represents a file.

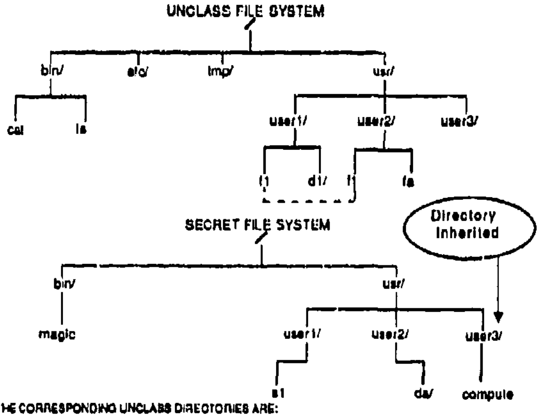
In addition to the enhancements to namei that are required to support the LOCK/ix file system, we have chosen to encapsulate its functionality in a file server subject (see subsection 4.3).

4.2 File System Examples

To help illustrate how the files system will work and appear to users, we present several simple examples. The examples are built on the file system shown in Figure 7.

Figure 8 shows how the file system would appear if an application running at the SECRET level reated a file named compute in the directory usr/user3. Before creating the file compute, the file server subject was required to create the full directory path at the SECRET level so that the file could be placed in the correct the file system.

File System After File /usr/user3/compute
Created by SECRET Process



THE CORRESPONDING UNCLASS DIRECTORIES ARE:

/	bin/	usr/	user1/	user2/	user3/
1 .	2 .	5 .	6 .	7 .	14 .
1 ..	1 ..	1 ..	5 ..	5 ..	5 ..
2 bin/	8 cat	6 user1/	10 f1	12 fa	
3 etc/	9 ls	7 user2/	11 d1/	10 f1	
4 tmp/		14 user3/			
5 usr/					

THE CORRESPONDING SECRET DIRECTORIES ARE:

/	bin/	usr/	user2/	user2/	user3/
1 .	2 .	3 .	4 .	6 .	8 .
1 ..	1 ..	1 ..	3 ..	3 ..	3 ..
2 bin/	14 magic	4 user1/	5 s1	7 da/	9 compute
3 usr/		6 user2/			
		8 user3/			

Figure 8

In this case, only the path component /user3 would have to be created the directory /usr already existed at the SECRET level. The inode numbers for directories with the same name that exist in multiple file systems can be, and typically will be, different in each file system.

The application running at the SECRET level that created the file compute is unaware that the directory /user3 was inherited. There will still appear to be only one /usr/user3 directory to both the UNCLASS and SECRET applications. However, to the application running at SECRET, /usr/user3 will (potentially) contain more accessible files than it will for the application running at UNCLASS.

If a filo named /usr/user3/compute already existed at UNCLASS, the application running at SECRET would not have been able to create the file since a file of that name would have already existed, as a read only file. If a file named /usr/user3/compute existed at the SECRET level, the process running at UNCLASS could still create the file, since the existence of the SECRET /usr/user3/compute would not be known at the UNCLASS level.

Suppose that the file /usr/user3/compute exists at both the UNCLASS and SECRET level. An application running at SECRET could delete the file. The file /usr/user3/compute at UNCLASS would remain intact, and the application running at UNCLASS would not be aware of the fact that the (higher level) file was deleted.

Figure 9 illustrates the file system after the directory /usr/user2/da is created at the UNCLASS level, and the file display is created in that directory. The directory /usr/user2/da existed at the SECRET level. However, this was not know not at the UNCLASS level so the operation succeeds. At the SECRET level, there will still appear to be only one /usr/user2/da directory, but it now contains the file display, which is read-only to processes running at the SECRET level.

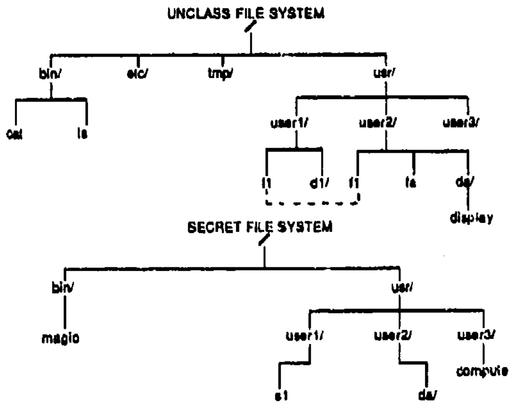
Figure 10 shows the results of the removal of the directory /usr/user3 by a process running at the UNCLASS level. To the UNCLASS process, the directory appeared to be empty, so it was permissible to unlink (delete) the directory. The directory path /usr/user3 at the SECRET level is left undisturbed by this operation.

Maintaining a separate file system per level, and providing the path inheritance capability allow the actions described in the examples to be performed without trusted code. Additionally, since file systems at higher levels are not known to processes at lower level, file system object create and delete operations can be performed at the higher levels without introducing covert channels.

4.3 Integrity Considerations

The LOCK/ix kernel runs in the same virtual address space as user processes. LOCK does not allow a single subject to execute in multiple domains. A process within a given LOCK/ix subject could gain access to any kernel file system structures at its level and modify them. It is for this reason the file system update operations are removed from the LOCK/ix kernel and implemented in separate file system server subject per (active) level. Type enforcement is used to limit write access to critical file system objects (directories, inodes, etc.) to the file server subjects.

File System After Directory /usr/user2/da and
File /usr/user2/da/display Created by UNCLASS Process



THE CORRESPONDING UNCLASS DIRECTORIES ARE:

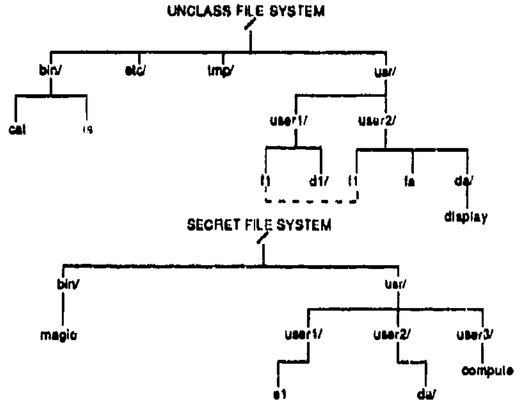
/	bin/	usr/	user1/	user2	user3/	da
1 .	2 .	5 .	6 .	7 .	14 .	13 .
1 ..	1 ..	1 ..	5 ..	5 ..	5 ..	7 ..
2 bin/	8 cat	6 user1/	10 f1	12 fa		15 display
3 etc/	9 ls	7 user2/	11 d1/	10 f1		
4 tmp/		14 user3/		13 da/		
5 usr/						

THE CORRESPONDING SECRET DIRECTORIES ARE:

/	bin/	usr/	user1/	user2/	user3/
1 .	2 .	3 .	4 .	6 .	8 .
1 ..	1 ..	1 ..	3 ..	3 ..	3 ..
2 bin/	14 magic	4 user1/	5 s1	7 da/	9 compute
3 usr/		6 user2/			
		8 user3/			

Figure 9

File System After Directory /usr/user3
Deleted by UNCLASS Process



THE CORRESPONDING UNCLASS DIRECTORIFS ARE:

/	bin/	usr/	user1/	user2	da/
1 .	2 .	5 .	6 .	7 .	13
1 ..	1 ..	1 ..	5 ..	5 ..	7 ..
2 bin/	8 cat	6 user1/	10 f1	12 fa	15 display
3 etc/	9 ls	7 user2/	11 d1/	10 f1	
4 tmp/				13 da/	
5 usr/					

THE CORRESPONDING SECRET DIRECTORIES ARE:

/	bin/	usr/	user1/	user2/	user3/
1 .	2 .	3 .	4 .	6 .	8 .
1 ..	1 ..	1 ..	3 ..	3 ..	3 ..
2 bin/	14 magic	4 user1/	5 s1	7 da/	9 compute
3 usr/		6 user2/			
		8 user3/			

Figure 10

The file system inode table is broken out into two parts: non critical components, such as time modified, and critical components, such as object UIDs. The non critical components consist of fields that can be updated by the LOCK/ix kernel and that would not cause any security problems if they were updated incorrectly. The critical components will be in an object type that can be read by, but not written by, the LOCK/ix kernel. The file system server subject will run in a domain that is different than that of the LOCK/ix kernel and user processes, and will have both read and write access to the critical inode table.

The file system server will only perform file system update operations; it will not run processes. No malicious programs that could cause unexpected and undesirable consequences will be allowed to run in the file server domain. This particular instance shows how Type Enforcement can be used to support an integrity policy that will achieve the desired end result.

5.0 Supporting set-user-ID Applications on LOCK/ix.

One of the most critical factors that will ultimately decide user acceptance of a secure Unix system is whether or not the system will support set-user-id/set-group-id applications. Our work to date in developing the model for LOCK/ix has not specifically addressed this issue. However, we have investigated several potential models and identified one that appears to be very promising from both an implementation and security point of view. We call this the new subject model.

To support this model, the Unix kernel will need to be modified to support the creation and start up of a new subject that will contain a set-user-id or set-group-id application. For simplicity, we will discuss this process in terms of set-user-id applications only. Set-group-id applications would be handled in the same manner.

5.1 Create New Subject

When the Unix kernel encounters an exec() system call, it determines if the new program is a set-user-id application. If it is, it makes sure that the new process will inherit the parent's environment which includes things such as open files. In LOCK/ix, the same actions would be required. However, the method by which the new process inherits this environment is somewhat different than is done in Unix.

When a new subject in LOCK is created, it receives what is called a new subject parameter object. This parameter object contains all the information that the new subject will need to execute in its environment. The format and contents of the parameter object will vary, depending upon the application.

When a new subject is to be created in response to an exec() request, the LOCK/ix kernel will create and initialize a new subject parameter object for the new subject. It will place the following information in the parameter object:

- The UID of the set-user-id application.
- The UIDs of the parent's open files.
- The UID and process-id of the parent process.

- Environment of the parent process (level, domain, etc.)

- The UID of a signal channel of the parent process.

This information will provide the new subject with everything it needs to inherit the parents attributes, and establish the necessary communication with the parent.

The LOCK/ix kernel will then ask the TCB create a new subject, specifying the UID of itself as the object module to be executed, and the owner UID of the set-user-id application as the user on whose behalf the new subject is to execute. This will have the desired effect (from a Unix point of view) of allowing the user of the set-user-id application to inherit the discretionary access rights of the owner of the set-user-id application.

5.2 Start Up of the New Subject

When the LOCK TCB allows the new subject to execute, it will begin execution in the LOCK/ix kernel. After the the LOCK/ix kernel has created and initialized its data structures, it will open the new subject parameter object. Upon opening the object that contains the set-user-id application, the LOCK/ix kernel will discover that it is a set-user-id application and know additional processing is required to start the new process. The following processing will occur:

- Open the text, data, and stack objects for the new process.

- Open the files of the parent process.

- Retrieve the UID and process-id of the parent process.

- Retrieve the environment data of the parent.

- Establish a signal channel with the parent.

These operations are essentially what is currently done in Unix. The method is different, but the effect is the same.

As stated earlier, our current model of LOCK/ix does not support this model. To support it, there are two primary requirements that are placed on the kernel:

- The kernel must modified so it knows that it must read the environment information out of the new parameter object when starting a set-user-id application.

- The Unix signalling mechanism must be enhanced to allow inter-processing signalling between processes contained within different subjects.

The first requirement is fairly straight-forward to implement. The things that must be done to start a set-user-id application in LOCK/ix are not much different than what is currently done in standard Unix.

The later is a somewhat more difficult feature to implement The difficulty in implementation is not really how to do it, but how to do it in a manner that is compatible (or invisible) at the programmer interface to the kernel.

Overall, the model appears to be promising and will be investigated further in the future.

6.0 Current Status

The results of our study were encouraging. We were surprised to find how much of the Unix kernel code can be retained and unmodified.

A continuation of the design began in April 1988. The initial implementation of LOCK/ix will be completed by April 1989. This version of the system will essentially be a port of Unix to LOCK.

Upon completion of the initial implementation, we will enter an enhancements phase. During this phase we will work on extending the functionality to incorporate such things as support for set-user-id/set-group-id applications. We will also be extending the standard Unix interface to incorporate some of the functionality provided by the LOCK TCB, such as ACLs. The enhancements should be completed by July 1990.

7.0 Acknowledgements

We would like to thank Bob Hartman, Jim Papke, Glen Swonk (ComputerBase), and Mike Carty (Sendona, formerly I.C.E.S. Ltd.) who also participated in the study.

8.0 References

[BIBA75] K.J. Biba, "Integrity Considerations for Secure Computer Systems", The MITRE Corporation, Bedford MA, MTR-3153, 30 June 1975.

[BOEB85] W.E. Boebert, "A Practical Alternative to Hierarchical Integrity Policies", Proceedings, 8th National Computer Security Conference, 1985.

[CLARK87] D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Security Policies", Proceedings, IEEE Symposium on Security and Privacy, 1987.

[HONE83] Honeywell, SCTC, A-Specification, Contract MDA 904-82-C-0444, April 1983.

[HONE86] Honeywell, SCTC, B-Specification, Contract MDA 904-84-C-6011, March 1986.

[SVID86] AT&T, "System V Interface Definition", 1986

[SYAD87] O. Sami Saydjari, et al, "LOCKing Computer Securely", Proceedings, 10th National Computer Security Conference, 1987.

[TCSEC85] "Department of Defense Trusted Computer System Evaluation Criteria", DoD 5200.28.STD, December 1985.

INTERDEPENDENCE OF EVALUATED SUBSYSTEMS

R. Leonard Brown, Ph. D.
Computer Security Office
Mail Stop M1/055
The Aerospace Corporation
P.O. Box 92957
Los Angeles, CA

1 Introduction

The *Department of Defense Trusted Computer System Evaluation Criteria* (TCSEC)[4] was published in 1983 to establish a uniform DoD policy for acquisition of trusted, commercially available, automatic data processing systems. The agency that originally produced the TCSEC is now called the National Computer Security Center (NCSC). It was established to assist all sectors of the Federal government in acquiring such systems by evaluating prospective trusted systems with respect to the Criteria in the current version of the TCSEC, which was revised and re-published as DOD 5200.28-STD [5].

Since the publication of the TCSEC, the NCSC has performed two major tasks relative to evaluation of trusted computer systems. It has evaluated a number of commercially available operating systems and added entries describing these systems to the Evaluated Products List (EPL). It has also published interpretations and guidelines intended to assist its evaluation staff, the vendors of trusted systems, and prospective trusted system users. An example of a guideline to assist the vendor is *A Guide to Understanding Audit in Trusted Systems* [3]. An example of a guideline intended to assist the user is *Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments* [2].

The formal interpretation process has resulted in publication of explanatory material about individual requirements in the TCSEC. The most common type of interpretation is that published to answer a question concerning how the evaluation staff of the NCSC intended to apply the TCSEC to a particular technical implementation. Unlike guidelines, which are a general explanation of the TCSEC requirements for a specific audience, an interpretation is an authoritative clarification to the TCSEC, and has the same authority as a TCSEC requirement. By making an interpretation formal, the NCSC evaluation teams can apply the TCSEC criteria in the same way when a similar implementation is encountered. By reading the interpretations, vendors are able to avoid designs that had previously caused problems during the evaluation process and to see acceptable implementations. As an outgrowth of the interpretation process, and as a result of vendor requests to evaluate products that do not exactly fit the mold of a "trusted, commercially available, automatic data processing system" the NCSC has published the *Trusted Network Interpretation* (TNI) [6] and has proposed the publication of "Computer Security Subsystem Interpretation of DoD Trusted Computer Evaluation Criteria" [1].

1.1 Purpose

The purpose of this paper is to provide guidance to those who must certify that a proposed computer system may be used to process sensitive information. In the military realm, this is a Designated Approving Authority; in the civil and commercial realm there are equivalent entities who must assure proper separation of data from system users. Such separation is required by law or best business practice. Similar guidance is given for the Department of Defense in [2], which provides guidance to DoD personnel concerning which level of TCSEC protection is required for particular mixtures of authorized users and DoD classified or sensitive data. This guidance is specific to entire trusted computer systems, and does not address the topic of trusted subsystems running on otherwise untrusted computer systems. This paper will provide guidance to any user who plans to use such trusted subsystems by pointing out how sensitive each such subsystem is to other parts of the computer system in which it is installed. When such a dependency can be met with an appropriate evaluated subsystem, then it will be recommended that the dependent evaluated subsystem, and the evaluated subsystem on which it depends, both be installed.

It is not the purpose of this paper to provide a guideline on the use of unevaluated computer systems to process classified information. However, if no computer system on the Evaluated Products List will suffice to meet procurement requirements, the recommendations here should help military, civil and commercial procurement personnel in choosing a viable subset of evaluated protection subsystems to install on their system.

1.2 Organization

In addition to the background material presented in this section, the paper contains three further sections. Section 2 provides definitions of the five types of subsystems that are mentioned in the proposed Subsystem Interpretation [1]. The possible subsystem security levels, corresponding to TCSEC levels, are given for each type of subsystem.

Section 3 displays a dependency graph for each subsystem level. An explanation is given for each arrow of the graph. Finally, in subsection 3.4 a chart is used to show all recommended sets of evaluated subsystems that may occur for at least one level. For each recommended set, the most dependent subsystem within that set has all the subsystems it needs to depend on also included in the set.

2 Subsystem Definitions

Several trusted subsystems have already been added to the Evaluated Products List, and the expertise gained during those evaluations has been used to propose a general interpretation of the TCSEC to the task of evaluating subsystems. In particular, it has described which types of subsystems can have requirements within the TCSEC isolated to the extent that it is possible to evaluate them separately, and has determined what levels of trust from the TCSEC (C1 through A1) may be applied to each such subsystem. The evaluatable subsystems are:

- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)
- Object Reuse
- Identification and Authentication (I&A)
- Audit

2.1 Details of Subsystem Types

In the following sections, each type of subsystem is briefly discussed, and allowable level of trust values given. The level of trust values are from the Subsystem Interpretation, and correspond to the equivalent levels of trust in the TCSEC; they are designated as subsystem levels of trust by preceding TCSEC levels with S-. For example, S-C2 in the Subsystem Interpretation corresponds to C2 in the TCSEC.

Discretionary Access Control DAC provides user-specified access control. This control is established from security policies which define, given identified subjects and objects, the set of rules that are used for the system to determine whether a given subject can be permitted to gain access to a specific object. The type of access, such as read, write, append, is also determined by this set of rules.

To be evaluated as an S-C1 feature, the DAC subsystem must provide mediation between objects and system users. For S-C2, the mediation must be done at the granularity of a single user, and objects must, as the default case, be protected from the time of their creation.

Mandatory Access Control MAC provides access control for classified or other specifically categorized sensitive information. This control is established from security policies which define rules for controlling and limiting access based on identifying individuals who have been determined to have both the proper authorization and need-to-know for the information. A MAC subsystem may be evaluated only at the S-B1 level, corresponding to the lowest TCSEC level that has a MAC requirement.

Object Reuse Object reuse subsystems clear storage objects to prevent subjects from scavenging data from storage objects which have previously been used. Object Reuse subsystems may be evaluated only at the S-C2 level, corresponding to the lowest TCSEC level with this requirement.

Identification and Authentication I&A subsystems provide the authenticated identification of a user seeking to gain access to the protected system. The most typical I&A system consists of a data base of identifiers which are valid on the system, and of authentication data such as encrypted passwords. However, other types of pertinent physical or procedural data may also be used in the authentication process. This type of subsystem provides fertile ground for innovative technological solutions to one of the main problems of secure computing. However, in order to meet the appropriate TCSEC requirement, the computer system itself must have access to at least some parts of the data base so that it can identify the valid users of the system.

I&A subsystems may be evaluated at the S-C1 level, or at S-C2 if they provide granularity to the level of a single user. If the subsystem is able to determine the clearance and allowable authorization levels of the user, then the subsystem may be evaluated at the S-B1 level.

Audit The audit subsystem helps achieve accountability for access to the system objects by authorized subjects through logging data from security-relevant events. This allows a system administrator to search for possible security breaches, or attempted penetrations, and to trace the breach to the responsible party.

TCSEC levels of S-C2 and S-B1 are possible. The minimum level is S-C2 since the audit log data must include the identity of the person for whom the security relevant event was attempted; S-B1 is possible if the authorization level of the individual and the security level of the object are both recorded. An additional requirement is that events be auditable based on the security level of the object. In the event that all events are logged, one must be able to generate reports that extract only events involving objects at the specified security. The final S-B1 requirement is that the system be able to audit attempts to override the printing of human readable output labels.

2.2 Other Requirements

In addition to requirements which describe specific features that each subsystem must have, the Subsystem Interpretation imposes additional assurance requirements and documentation requirements similar to those in the TCSEC. These requirements are in the areas of system architecture, system integrity, security testing, design specification and documentation, and test documentation. Furthermore, a description of how to use the subsystem in a secure manner must be included in the Security Features User's Guide and the Trusted Facility Manual that must accompany any evaluated subsystem.

3 Interdependence of Subsystems

Inspection of the various features and assurance requirements indicates that secure operation of certain of the evaluated subsystems depends on the proper working of other subsystems. The draft Subsystem Interpretation does not specify that the subsystem that is depended on must also have been evaluated under the Subsystem Interpretation of the TCSEC. However, it only seems logical to make such a recommendation. A typical example of the interdependence of evaluated subsystems occurs with the audit subsystem. This subsystem must be able to log the identity of a user who causes a security relevant event to occur, so an I&A subsystem is required; however, unless the I&A subsystem has been evaluated and found to meet the TCSEC criteria, it is possible that the I&A subsystem can be spoofed causing the audit system to record the wrong username in its log record.

Also, since the TCSEC may impose an additional requirement at each increasing level for each requirement section, so the evaluated subsystems should only depend on other subsystems that have met their additional requirements. For example, an S-C2 audit subsystem should not depend on an S-C1 I&A subsystem since it is possible for I&A to be evaluated at S-C2.

The intent of this paper is to provide recommendations as to which combinations of evaluated subsystems will assure that the most dependent subsystem in each combination is relying only on appropriately evaluated subsystems. This effort is made in the same spirit as the *Guidance for Applying the TCSEC in Specific Environments* [2]. In the sections that follow, for each criteria level a dependency tree will be derived showing which subsystems are dependent on which other subsystems.

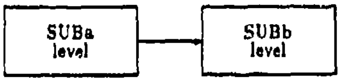


Figure 1: Sample Dependency Tree

This paper will derive a dependency graph for each subsystem level of trust. In such a tree, an arrow from a subsystem to another subsystem means that the first subsystem relies on the correct functioning of the second subsystem for its own correct functioning. All recommended combinations of subsystems may then be derived by listing for each subsystem in each graph all the subsystems in the subgraph for which it is the root. Thus, the notation example in figure 1 illustrates the notation that is used in this paper to show that subsystem SUBa depends for its correct working on subsystem SUBb also working correctly. So the two recommended combinations are {SUBb} and {SUBa, SUBb}. Note that {SUBa} is not recommended since it depends on an evaluated subsystem SUBb for its correct functioning.

Exceptional cases may exist where all subsystems in a dependency subgraph are not to be evaluated. The official certifying a computer system for the processing of sensitive data must decide whether to follow the recommendations in this paper or to entrust the sensitive data to a partially secured system. This is always the case, since the EPL is only one input to the certification process. This paper is an attempt to provide guidance on the trustedness of systems that are secured only by the addition of evaluated subsystems.

Before getting to the recommendations and their rationale, it is worth noting that an object reuse subsystem does not have any dependence on any other of the four types of subsystems described in the Subsystem Interpretation, and will not be discussed further.

3.1 S-C1 Level

At the S-C1 level, the following graph in Figure 2 can be derived.

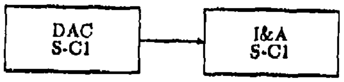


Figure 2: Dependency Graph at S-C1

which yields the sets {I&A}, {DAC,I&A}.

The DAC mechanism must decide whether an authorized user or user group may access a particular subject. The I&A mechanism at this level has granted the user access to the system and has at least identified that user as a member of a particular group. The DAC mechanism requires the knowledge of which group of users the particular user belongs to in order to make the access control decision. A non-evaluated I&A system might allow a user to login as another user, without authorization, or to masquerade as the member of any group and thus gain access to any object.

3.2 S-C2 Level

At the S-C2 level, the dependencies shown in Figure 3 can be derived:

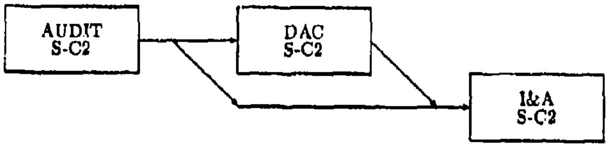


Figure 3: Dependency Graph at S-C2

which yields the sets {I&A}, {DAC,I&A}, {Audit,I&A}, and {Audit,DAC,I&A}.

DAC depends on I&A for the same reasons as those previously discussed for S-C1, only more so since the I&A system must identify the subject down to the granularity of a single user.

If there is a DAC subsystem present, Audit depends on it for two reasons. The first reason derives from the fact that most security relevant events at the C2 level are due to attempted access to objects by identified subjects. To depend on a non-evaluated DAC subsystem to report all such events accurately would mean that the audit system was not getting information that is accurate at the C2 level since the non-evaluated DAC subsystem may not fulfill all the TCSEC requirements. The second reason is the implicit requirement to protect the audit log data with the best resources available to the computer system. If an S-C2 DAC system exists on the system, the audit log data should be stored in an object which is restricted by DAC to the appropriate access by a limited set of administrative personnel.

Alternatively, the system could protect its audit logs in an appropriately configured Write Once Read Many (WORM). This could be used even if an evaluated DAC subsystem were present. Such devices already exist in the form of writable optical disks. The optical disks must be unloaded and processed on a separate audit machine.

3.3 S-B1 Level

At the S-B1 level, the dependency graph in Figure 4 can be derived:

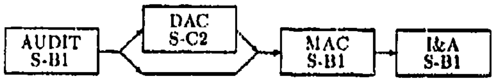


Figure 4: Dependency Graph at S-B1

This yields the recommended sets {I&A}, {I&A,MAC}, {I&A,DAC,MAC}, {Audit,MAC,I&A}, and {Audit,DAC,MAC,I&A}.

S-B1 MAC depends on S-B1 I&A to obtain the clearance and allowed range of authorization levels when the user logs in. S-C2 DAC depends on at least an S-C2 I&A system, but in an S-B1 system it would depend on an S-B1 I&A because it is required for MAC and has already met all the S-C2 requirements on the way to fulfilling the S-B1 requirements.

If the S-C2 DAC subsystem is present, then it must depend on the MAC subsystem to provide the separation of subjects and objects according to authorization levels and need-to-know categories. Then the DAC subsystem provides discretionary access to objects within each level and category. Although the TCSEC [5] requires any B level or higher system to have both DAC and MAC, there is no requirement in the Subsystem Interpretation that both exist. A B1 level system requires MAC to enforce the need-to-know and authorization requirement. The S-B1 I&A system is still needed by the Audit subsystem since it is required to record the authorization of a user when the Audit subsystem logs a security relevant event. Similarly, the S-B1 level audit system is required since it must record the label of any object and the authorization of any subject involved in any security relevant event. This is not a requirement at S-C2.

The Audit subsystem must depend on MAC to protect the audit logs in a S-B1 system since the logs themselves may contain classified information, such as the names of categories. The most effective way to do this is to create a system high sensitivity level and a special category just for the audit logs. As with S-C2 DAC, the audit subsystem depends on the DAC subsystem, if present, and the MAC subsystem to truly report all security relevant events which a non-evaluated access control subsystem might not report correctly. It also needs the MAC system to truly report the sensitivity level of each object involved in a report of a security relevant event. The audit subsystem depends on an S-B1 I&A system to provide it the clearance and authorization level of each user which it needs to report in the audit log.

3.4 Summary

The information in the above dependency graphs can be summarized in Table 1, which appears without the levels of trust from the Subsystem Interpretation. The chart was compiled by aggregating all recommended sets of subsystems that appeared for each of the levels in the previous section. These are listed in the "Recommended Combinations" side of the chart. All other combinations of evaluated subsystems, whether they depend on an unevaluated subsystem or are some kind of standalone system, are not recommended for use in a trusted computer system to process sensitive data. Also, the Subsystem Interpretation warns that the subsystem rating is no guarantee that the system into which the evaluated subsystem is integrated then becomes equivalent to a similarly rated integrated system, such as the systems that are evaluated against the entire TCSEC. This paper points out that even evaluated subsystems may have their value in the security of an overall system overstated if they are allowed to rely on other unevaluated subsystems.

Table 1: Possible Combinations of Evaluated Subsystems

RECOMMENDED	NOT RECOMMENDED
I&A	DAC
I&A, audit	MAC
I&A, DAC	audit
I&A, DAC, audit	MAC, audit
I&A, MAC	DAC, audit
I&A, DAC, MAC	DAC, MAC
I&A, MAC, audit	DAC, MAC, audit
I&A, DAC, MAC, audit	

References

[1] National Computer Security Center. *Computer Security Subsystem Interpretation*. Technical Report Draft version 1.0, United States Department of Defense, September 1987.

[2] National Computer Security Center. *Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*. Technical Report CSC-STD-003-85, United States Department of Defense, June 1985.

[3] National Computer Security Center. *A Guide to Understanding Audit in Trusted Systems*. Technical Report NCSC-TG-001, United States Department of Defense, July 1987.

[4] National Computer Security Center. *Trusted Computer System Evaluation Criteria*. Technical Report CSC-STD-001-83, United States Department of Defense, December 1983.

[5] National Computer Security Center. *Trusted Computer System Evaluation Criteria*. Technical Report DoD 5200.28-STD, United States Department of Defense, December 1985.

[6] National Computer Security Center. *Trusted Network Interpretation*. Technical Report NCSC-TG-005, United States Department of Defense, July 1987.

ALTERNATE AUTHENTICATION MECHANISMS

Authors

Stephen F. Carlton
John W. Taylor
John L. Wyszynski

National Computer Security Center
9800 Savage Road
Fort George G. Meade, Maryland 20755-6000

ABSTRACT

This paper will present three classes of authentication mechanisms in use today. In addition, it will also show the need to change the Trusted Computer System Evaluation Criteria (TCSEC) at its upper levels such that new means of authentication will be encouraged in the next generation of products evaluated by the National Computer Security Center.

HISTORY

Background

On January 2, 1981, the Director of the National Security Agency was assigned the responsibility for increasing the use of trusted computer security products within the Department of Defense. As a result, the DoD Computer Security Center was established at the National Security Agency. Its official charter is contained in DoD Directive 5215.1. The Center became known as the National Computer Security Center (NCSC) in August 1985.

The primary goal of the NCSC is to encourage the widespread availability of trusted computer systems; that is, systems that employ sufficient hardware and software integrity measures for use in the simultaneous processing of a range of sensitive or classified information. Such encouragement is brought about by evaluating the technical protection capabilities of industry- and government-developed systems, advising system developers and managers of their systems' suitability for use in processing sensitive information, and assisting in the incorporation of computer security requirements in the systems acquisition process.

The NCSC Computer Security Sub-System Evaluation Program

While the NCSC devotes much of its resources to encouraging the production and use of large-scale, multi-purpose trusted computer systems, there is a recognized need for guidance on, and evaluation of, computer security products that do not meet all of the feature, architecture, or assurance requirements of any one security class or level of the TCSEC. The NCSC has, therefore, established a Computer Security Sub-system Evaluation Program.

The goal of the NCSC's Computer Security Sub-system Evaluation Program is to provide computer installation managers with information on sub-systems that would be helpful in providing immediate computer security improvements to existing installations.

Sub-systems considered in the program are special-purpose products that can be added to existing computer systems to increase some aspect of security and have the potential of meeting the needs of both civilian and government departments and agencies. For the most part, the scope of a computer security sub-system evaluation is limited to consideration of the sub-system itself, and does not address or attempt to rate the overall security of the processing environment. To promote consistency in evaluations an attempt is made, where appropriate, to assess a sub-system's security-relevant performance in light of applicable standards and features outlined in the TCSEC. Additionally, the evaluation team reviews the vendor's claim. and documentation for obvious flaws which would violate the product's security features, and verifies, through functional testing, that the product performs as advertised. Upon completion, a summary of the evaluation report is placed on the Evaluated Products List.

Many of the sub-systems evaluated in this program have been Identification and Authentication sub-systems. These systems, although deemed useful, do not tend to be incorporated in larger TCSEC evaluated systems because they would constitute a change to the Trusted Computing Base (TCB). Such a change to any evaluated system renders the rating assigned void. Although the weakest Identification and Authentication mechanism, a password mechanism is currently acceptable for all levels of the TCSEC. Because of this and the added cost of incorporating another type of I&A mechanism, vendors currently in or considering evaluation by the NCSC for a TCSEC rating have chosen not to incorporate other I&A mechanisms.

INTRODUCTION

The Department of Defense Trusted Computer System Evaluation Criteria is built around three basic control objectives, the security policy, accountability, and assurance [1]. Much emphasis is placed on better meeting the security policy and assurance objectives, while accountability is often taken for granted. In fact, the requirements stated in the TCSEC are more centered around security policy (Mandatory and Discretionary Access Controls) and assurance (Documentation and Verification) than accountability (Identification, Authentication, and Audit). New technologies are being developed which will better meet this requirement and the computer security community needs to take a look at them, encouraging their use where appropriate.

Briefly, Identification and Authentication (I&A) is the process by which users log onto a computer system. The identification step simply identifies who the user is, by account name. Authentication is the step which proves that the user (person) is indeed the owner of that account. As these are so closely associated, usually they're considered as the same thing. In reality, identification is easily implemented while authentication needs more consideration.

The importance of this issue can be demonstrated by citing the Department of Defense Trusted Computer System Evaluation Criteria's Control Objectives [1]:

> Identification is functionally dependent on authentication. Without authentication, user identification has no credibility. Without a credible identity, neither mandatory nor discretionary security policies can be properly invoked because there is no assurance that proper authorizations can be made.

The conclusion from the above paragraph is that the credibility of any security policy is directly dependent on the credibility of the authentication mechanism backing it.

The strength of an I&A mechanism can be measured in terms of the certainty that the person requesting access is indeed who he claims to be. Quantifying this certainty is difficult, however, a feel for the relative strength can be gained. For example, if the authentication of a user could be duplicated by any user, the mechanism would have certainty of 0%. While a method which would distinguish between every possible person in the world would have a certainty of 100%. Since the latter does not exist, something which closely approximates it is the most desirable mechanism.

There are several areas which have been recognized as legitimate tests of user uniqueness. These are things that the user knows, things which the user has, and things that the user is. While there is no quantitative way to measure the certainty of any of these, the strengths and weaknesses of each area can be examined and compared to each other.

THINGS YOU KNOW

Overview

The objects which are categorized as being "things you know" includes passwords and all password like mechanisms (such as pass-phrases). A password is known by the user and only the user. Virtually anyone who has ever used a computer system is familiar with the concept of passwords as being special words which must be entered before access to the computer is allowed.

Passwords work on principle of being a secret between the computer and the user. The computer can only be sure of the user's identity if they know the same secret. The better this secret is kept, the better the password scheme.

Strengths

The strength of a password scheme is very dependent on its implementation. Passwords become stronger as difficulty in forging them increases. The Department of Defense Password Management Guideline (CSC-STD-002-85) [2] provides specific guidance on how to make a strong password system.

Passwords guessing should be like hitting a moving target. Good schemes allow the users to change their passwords and require it be done on a regular basis.

Machine generated passwords are also important. This reduces the biases (such as the user's first or last name, wife's/husband's name, pet's name, ad infinitum) which are present in user chosen passwords, resulting in a much better mix. This results in a much larger set of targets which must be examined.

Weaknesses

The strengthening of passwords on any system tends to have consequences which at the same time weakens some of the basic principles of passwords.

Making passwords more difficult to guess also makes them harder to remember. This increases the chance that a less than cautious or well informed user will write them down. In a large computing environment this becomes very likely. Also, this is likely to happen if the user is only logged onto a system occasionally.

If passwords are difficult to remember, as machine generated passwords can be, they tend to get changed less often by the users. Users are just not willing to learn a new jumble of letters every few days. This causes the target to move less frequently. If the system has a password expiration capability, a system security administrator is be able to force users to change their passwords more frequently, but this increases the likelihood that the passwords will be written.

Passwords on a network make for interesting security problems. Imagine a user who has accounts on 30 nodes of a network. If able to choose his own password, the user will most likely use the same password on each node for ease of use. If this is the case, hacking the password on one node compromises each node the user has access to. On the other hand, if password generation in enforced on each node of the network, it is likely that the user will have scripts containing the node and password because it is simply too difficult to remember 30 "pronounceable" passwords.

The most serious weakness is that compromise of passwords is not usually detected until well after any damage has taken place. Even when users regularly check the time that they last logged in, it may be several days between sessions by the legitimate user of the system. This is more than enough time for the damage to be done.

There are a number of other similar methods which have been thought up which are inherently weak. Instead of passwords, users are asked a series of questions about their background, typically their mother's maiden name, first car, or their pet's name. These methods fail basic password philosophies in that while this information is not common knowledge, it is far from being secret.

THINGS YOU HAVE

Overview

Those things that fall under the category "Things You Have" can be thought of as identifiers similar to badges worn for entrance to buildings. Ownership of the badge authenticates that person as belonging to a particular company or holding a particular position.

Likewise, things that fall under this category are physical devices that provide authentication via possession of the device. Such devices include, but are not limited to: smart cards, tokens, data keys, encryption/decryption keys, magnetic strip cards, calculator-type devices, random number generators, laser cards, code decryptors, and the like. These devices are normally incorporated into systems through the use of special dedicated hardware and software. They can include front-end or back-end processors, vendor-written machine specific software, and buyer-written machine specific software. Depending on the amount of storage available, these devices may also contain audit data, biographic data, and account balances.

Most devices in this category also incorporate mechanisms that fall under the other two categories covered in this paper. Take, for example, a money machine card. Although one needs this card to access the machine, one must also know the Personal Identification Number associated with that card in order to conduct business at any automatic teller machine. This example incorporates a mechanism from the "Things You Know" category. Likewise, with a credit card, one must physically posses the card to be able to transact business, but one must also be able to forge the signature on the card. This example incorporates a mechanism from the "Things You Are" category.

To use a specific example of such a device, Sytek's PFX A2000 [6] has been chosen. This device has been evaluated by the National Computer Security Center Sub-system evaluation program. The PFX A2000 system contains a back-end processor that interacts with the host machine through the use of buyer-written machine specific software. From the users perspective, a typical login scenario would be the following. The user begins by entering his login identifier to the host system. The host system passes this information to the back-end processor which returns a challenge/response combination. The host displays the challenge information on the user's terminal and prompts the user for a response. The user enters his PIN into a calculator-type device, followed by the challenge displayed on the terminal screen. The calculator processes this number to produce the correct response which the user then enters to the host machine, thus granting him access.

Strengths

Manufacturers have been placing a great deal of emphasis on the usefulness of "Things You Have" in the areas of Identification and Authentication. Since they are portable devices, they can be carried by the user and used to replace or enhance password systems which are commonly used today. Data about the holder can be stored on the device such that either the device passes data to the host system to be used in querying the user or the device queries the user. Vendors state that it is nearly impossible for a person, after acquiring another's device, to unlock the memory stored there. This is mainly due to the use of Personal Identification Numbers to authenticate the user to the device and custom built chips which require the destruction of the chip to glean any useful information.

Another advantage to having a device for an authentication mechanism is that it is easy to determine when a compromise may have occurred. A user may only logon to the system if he possesses the device. The loss of the device would certainly signal that compromise may have occurred.

If each system in a network uses the same type of device for authentication, a user would only need the one device to access each node of the network. Thus, a user would not be required to remember a different password for each node or use the same password for each node. A device that changes with each login attempt, such as a random number generator or challenge/response device, strengthens the schema even more.

Weaknesses

All things categorized under "Things You Have" have the same basic weakness due to their main strength: they are physical devices and are thus subject to physical protection issues. Possession of a card implies that the user is who he says he is. The host system, in effect, is authorizing the device rather than the user.

Although a device is useless as an authenticator without the proper identifier, many of the devices now marketed have the associated identifier embossed on them. They also tend to be carried with other personal possessions, such as in a wallet, where the identifier can often be quickly surmised.

With the proper technology, these devices can be easily forged. For example, if the device is simply a smart card with a number contained on it, anyone that can write to a similar smart card can gain access to the system. Depending on the level of technology involved, this may be a trivial task, thus providing no authentication assurance whatsoever.

Lastly, although these devices are less expensive than their biometric counterparts, they tend to be more expensive than the development of password mechanisms and are therefore not very cost effective in the development of general purpose machines.

THINGS YOU ARE

Overview

In recent years a new type of authentication mechanism that works based on some characteristic of a user has appeared in the marketplace. Devices which use this type of authentication mechanism are known as biometric devices. Biometric devices work by attempting to match some unique characteristic of a user with a known version of the characteristic. Common characteristics currently being used include fingerprints, eye retina patterns, voice characteristics, and signatures. For biometric devices to work as an effective authentication mechanism in an ADP environment, the known version of the characteristic must be protected from modification by users of the system. This requires that the known version be treated as an object that is protected by the TCB.

One example of a biometric device is the IDX-50 from Identix Incorporated. The IDX-50 has been evaluated by the NCSC sub-system evaluation program [3]. The IDX-50 provides authentication data to a host based on a comparison between a user's fingerprint and a pattern (representing the user's fingerprint) stored on a smart card. The result of the comparison made (either confirmed or denied) is sent to the host system.

Strengths

Unlike other authentication mechanisms, biometrics makes it almost impossible for a user to pass his means of authentication to another user. For example, a user can tell someone his password or give someone his smart card, but giving someone his fingerprint in a form that will be accepted is extremely difficult. By linking a characteristic of the user to the authentication process, biometrics eliminates the possibility that the user's means of authenticating himself to the system is easily obtained by another user.

One of the most important features of an authentication mechanism is how difficult it is to spoof. Biometric authenticators are extremely difficult tc spoof because of the complexity that most biological characteristics have.

Weaknesses

Biometric devices use some unique characteristic of a user as an authentication data point. Unfortunately, it is impossible for the user to present the characteristic in the exact same form as the known version. For example, a finger may have more oil on it on a particular day than that which was on it at the time the known version was obtained. This could cause the newly scanned image to vary slightly. To account for the variations it is necessary to allow some tolerance when comparing the authentication data point to the known version. The larger the tolerance, the greater the likelihood that bad data will be accepted (Type I errors). In contrast, the smaller the tolerance, the greater the chance that good data will not be accepted (Type II errors).

There is a yet unexplored ethical side to biometrics concerning user acceptance. Cats and other animals can mark and identify territories by urine traces around the perimeter. Suppose someone devises a method of performing foolproof authentication via a device that performs instant urinalysis. Each terminal could be equipped with one of these devices and a sign reading: "For login, please deposit sample here." Likewise, a device able to perform a spectrcgraphic analysis where the user must supply his own blood sample would probably have a fairly serious user acceptance problems [9]. Case studies of users concerning fingerprint readers and retinal eye scanners have shown that users are greatly reluctant to use these devices. Very few people are willing to place their finger in a hole in the side of their terminal as an authentication mechanism for fear of bodily harm. Many people are also reluctant to have their fingerprint/eyeprint stored on line.

Although the cost of biometric devices has been steadily decreasing, they are still the most costly of the authentication mechanisms available. The unit price of these products can vary from about $3500 to $10000 [8]. Prices of this nature make them too costly for many general purpose applications.

CONCLUSIONS

The TCSEC requires that the TCB obtain some data that will authenticate a user before granting access to system resources. Authentication data falls into three general categories:

1) Something the user knows.

2) Something the user has.

3) Something the user is.

"Something the user knows" is some piece of knowledge which a user must memorize and present to the TCB at authentication time. A password is an example of this type of authentication data. "Something the user has" is a physical device which a user must present to the TCB at authentication time. A smart card is an example of this type of authentication data. "Something the user is" is a characteristic that a user must present to the TCB at authentication time. A fingerprint is an example of this type of authentication data.

The TCSEC does not require that more than one type of authentication data be presented to the TCB. In fact, all computer systems that have been evaluated by the NCSC at this time have used password based mechanisms (i.e. mechanisms based on "something the user knows"). If an evaluated product were to change its authentication mechanism such that two or more pieces of authentication data were required before granting access to system resources (thereby strengthening the mechanism) the rating would be invalidated due to a change of the TCB.

As has been shown, each type of authentication data discussed in this paper has some weaknesses however, these weaknesses seldom overlap. Therefore, a combination of these mechanisms would most likely be a stronger authentication mechanism. One mechanisms strengths could compensate for another mechanisms weaknesses. Since the TCSEC defines what is minimally acceptable at a given level of trust, it should be modified at its upper levels to require that developers of trusted ADP systems require at least two types of authentication data (Have/Know/Are) before gaining access to system resources. Current technology has created other mechanisms, that can be used to strengthen authentication than were readily available at the time the TCSEC was written. The NCSC should encourage the use of these mechanisms.

REFERENCES

[1] Department of Defense Trusted Computer System Evaluation Criteria, DoD-5200.28-STD, December 1985.

[2] Department of Defense Password Management Guideline, CSC-STD-002-85, 12 April 1985.

[3] Final Evaluation Report of Identix, Inc. IDX-50, CSC-EPL-88/001, 1 February 1988.

[4] Final Evaluation Report of Security Dynamics Access Control Encryption System, CSC-EPL-87/001, 31 March 1987.

[5] Final Evaluation Report of Gordian Systems Access Key, CSC-EPL-85/001, 7 April 1986.

[6] Final Evaluation Report of Sytek PFX A2000 and PFX A2100, CSC-EPL-86/006, 7 November 1986. (Note: These systems are now owned by RACAL-GUARDATA.)

[7] Final Evaluation Report of Codercard CPP-300 Port Protector, CSC-EPL-86/002, 7 April 1986.

[8] A Performance Evaluation of Personnel Identity Verifiers, Russell L. Maxwell & Larry J. Wright, Sandia National Laboratories, July 1987.

[9] Operating Systems: Design and Implementation, Andrew S. Tanenbaum, Pentice-Hall, 1987.

SECURITY AWARENESS: MAKING IT HAPPEN

Dennis F. Poindexter

Department of Defense Security Institute
c/o Defense General Supply Center
Richmond, Virginia 23297-5091

Most people who attend conferences are immersed for a certain length of time in a subject they are interested in. They share ideas, develop new ones, improve their people networks, and occasionally have a good time besides. When they go back to work they will be "pumped" and some, especially those who are relative newcomers to the business, will be ready to help out all their office companions by giving them newfound wisdom and maybe even "changing a few things" to make them work better. They are frequently discouraged and somewhat disappointed by the responses they will most often get. People are not ready to respond and may even be downright negative. Having a conference about security will not make this any easier for two reasons. First, security is not one of the favorite topics of most office personnel, unless they happen to work in the business all the time. Second, changing people's attitudes towards a subject doesn't happen quickly; it happens a little bit at a time, over a long period of time [1]. This is what makes security so interesting and so difficult.

What security awareness does is sell security procedures and performance consistent with those procedures. In the computer environment, security means restraint on an activity that is far more efficient unrestrained, so some people are not just going to refuse to buy it, but will challenge the right to sell it at all. A small minority of people will actively fight a security program; a small minority will actively support it [2]; the rest of the population is the principle target audience for a security awareness program.

Program Changes Attitudes Towards Security

What a security awareness program does is change people's attitudes towards security procedures and policy. The selling aspect of advertising recognizes much of what a security person needs to know about attitude change.

While there is a large body of research surrounding the activity and an equally large amount of money being spent on it, its success is seldom measured in terms of days or weeks and is far from guaranteed. The best minds in the business fail as often, sometimes more often, than they succeed. Few people in security education like to admit it, but they don't do much better. One obvious problem they have is a measure of success to check performance against. Unlike advertising where there is a market feedback for the successful, security seems to have no such product to measure.

The measure of success is how many people follow the procedures they are trained to follow. Knowing how many don't is a matter of auditing and testing of the system, something that reinforces whatever education and training preceded it. It is always best to train first, test second, or there is little way to measure the success of the training. To get performance one must train to a standard and measure the performance. Some people honestly believe that education alone will result in action, when they should realize that it won't. If advertising campaigns worked with the same logic, the goal would be to get the audience to know a trade name, but not really care if anyone bought the product. To be effective, they have to do both.

Awareness is not easy, but it is easier than getting performance. To be effective it must follow principles of organizational and interpersonal communication. The first is that people don't always get messages that are directed to them [3]. As a trainer, I have occasionally been reminded of this when students ask questions about subjects that have just been covered in class. It shouldn't be surprising. Nobody ever sold anything without overcoming this problem.

People are "tuned" to different types of media to get the majority of their information from particular types; some are print oriented; some are video oriented; some may be oriented to things they listen to [4]. In order to sell products, advertisers use a variety of media over a length of time so that sometimes it seems, unless a person is near death, they will get the message. Still, they don't always get it or perform by buying the product.

In order to assure a person gets the message, advertising has to be memorable -- good or bad. Mediocre is death. We are so bombarded by media that we don't have time for average commercials and the same goes for security. Many active programs fail because they are neither good nor bad. Bad is not recommended, but it is more memorable than mediocre. Overall, messages have to appear in a variety of media over an extended period of time and most of those have to be good, i.e. interesting enough to be seen, read, or heard. Usually this also means short.

An audience has an attention span of about 20 minutes, a long time when compared to a reader of print [5], yet we have many people who think it is a great idea to get everyone together and do all the security training at once. A midwestern contractor used to do this once a year for 5 hours, meeting every required security briefing for all of its employees; it looks good on paper and is well documented, but it doesn't work. In the same way, some people write long articles for house organs or publications and believe these are read by large numbers of people. Long articles about security are only read by security people, and not very many of those. Some people make monumental, epic films which are occasionally viewed by somebody, but not often remembered by very many. Steven Spielberg makes memorable films, but if everyone could do this, they wouldn't pay him nearly as well for what he does.

Posters are the best illustration of a print media that gives a short message. Posters are for people who haven't the time to read very much about security. A popular misconception is that this means people who are undereducated or low on intelligence. Along the mahogany rows are just as many people who are overwhelmed by the amount of text coming across their desks. They don't read well either, when it come to security.

In order to be noticed posters, like advertising, have to be good or bad. The federal government has frequently made a series of mediocre posters which are not memorable. Lockheed Space and Missile Co. in Sunnyvale, CA has made many of the best : A teddy bear with badge around its neck and the inscription "We can't bear it, if you don't wear it"; three otters standing side by side, with the inscription "You otter not talk classified when you"re out with the gang"; and a series of cave dwelling comic characters representing computer security items of concern and containing the telephone number of the computer security office. They are done by commercial artists and well thought out. They have to lock these up or they are regularly stolen by employees and visitors.

Crude drawings can be just as effective. A quick (and cheap) source of these is a local grade school. Children are perceptive and uncomplicated in the way they express their ideas, though not always artistic. The art sometimes falls into the "bad" category of advertising, but is frequently memorable. It also helps, as I saw Sheraton Hotels do once, to put the originators name on the bottom along with a ribbon. Everyone had a red or blue ribbon with absolutely no legend to explain what these meant. Parents intently locked at them, at least until they found the one they were looking for, but they get the message represented.

There are many media to choose from and not enough activities make use of all of them. Audio tape, in the land of the Walkman, could be an effective media for foreign travel or special access briefings, changes in operating procedures and certain training where the user has the equipment in front of them. Video cameras make it possible for anyone to demonstrate how to do various tasks, including something as simple as logon procedures. Interactive video does this better, but it is much more expensive to develop. Off-line computer aided training can be done in house relatively inexpensively. Various forms of print from job aids to comic books will work. The trick is to not try to do everything with any single media. Break it up into small pieces and use several forms to get the message across. Don't be afraid to repeat points from one aspect to another.

Honing a Message to Present

Perhaps a harder task than selecting a medium is selecting the message to be put across. The government makes this somewhat easier by having certain required subjects that have to be covered. DODSI deals with the protection of classified information, both in and out of computers, but the guidance is applicable to any environment where information is to be protected.

Everyone in government, and in industry where classified work is done, has to have an initial security briefing outlining:

1) The importance of the information

2) The obligation of every person to protect it

3) Procedures which govern the protection

4) Reporting requirements for certain status changes (e.g. foreign travel, establishing relationships with persons from designated countries, bankruptcy and the like)

5) Laws and statutes governing espionage

Initial briefings should be short because a new employee is not very receptive to much of anything except their boss and payroll procedures. The acknowledgment of the briefing can, however, be very important, particularly where there are obligations for protecting trade secrets, process patents, and other proprietary information. Borland and Microsoft recently went to court over a similar business. The point here is that the employee know that some types of information will be safeguarded and there are procedures to tell them how this is to be done. Of course, there has to be a corporate policy and procedures or this type of briefing will be ceremonial.

Actual briefings on procedures should be in work centers. Rarely are these done, or done well when they are. Work center briefings are more current and credible than any other an employees will receive. They will actually use this information often and should be geared to training i.e. performance. It has to be more than "Here is your password." Work center briefings in classified information settings cover very generally these types of things:

1. Where classified is stored and how access is to be gained to it

2. How it is made available to the employee

3. How it is to be protected while in use

4. What unauthorized acts are reportable

5. What actions to take if an unauthorized act is observed

6. Machine specific (AIS) procedures are to be followed:

   a. upgrading a system to process
   b. in use controls of media, hardcopy, and visuals
   c. marking output products
   d. when to downgrade or declassify a system
   e. audit trail records activity
   f. emergency procedures

7. Internal labeling

8. How access is governed during processing

9. Who is to be escorted and escort responsibilities

The media used to present these ideas should vary depending upon the length. Some of these can be done with CAE since computers can best duplicate the environment the employee will operate in. Others can be written and used when the task has to be performed. Unfortunately this is not always any better than some software documentation and has to be done as well as the best of it. Some of it is a one-on-one supervisors briefing, and some can be written summaries of laws and regulations, covering computers in the work place. A combination is more effective than a single medium.

The government also requires security deficiency briefings which are statements to employees of conditions identified during inspections or internal audits. This is just a good business practice, but it is not always done. In order to get across to other employees, the problems of a few doesn't mean that problems have to be stated exactly, by naming names, times, and dates. A general overview would be adequate through a memo, a short internal house organ article, or a letter to supervisors. Nothing irritates inspectors or auditors more than identifying a type of problem in one shop or building and having it come up again somewhere else during the next audit.

Perhaps the most difficult area of briefings are those that require action by the employee to report suspicious behavior or actions of another employee. The government places considerable emphasis on the individual in a managerial or supervisory position to report factors which may influence a person's position of trust. Similar security procedures are frequently stated or implied in other requirements outside of the government. People see indicators that can signal internal abuses and fraud, the most difficult kind to detect. Getting them to report these may require more than a written or verbal statement that it should be done. For several reasons employees don't want to "rat" on their friends or coworkers. For one thing, there is the matter of reciprocity, i.e. if I tell on you, you may tell on me. For another, Willis Ware of RAND pointed out several years ago that it would be possible to stop computer crime by planting informants in every DP shop, only the consequences would not be worth achieving the result. Balance is really the goal. Gossip has to be discouraged, but the tendency is to do otherwise. Security people tend to think that lots of information, however incredible it may be, is better than very little. A little quality information about certain auditable activities is far more beneficial than a hundred rumors that will create more dissention, mistrust, and internal squabbling than they are worth. Management has to agree on what is going to be said about a policy and how such reports are to be investigated.

Making a Security Program Visible

In many ways security will not survive without management support. No organization can have more security than management wants or will tolerate. Management is also, then, a target audience for security awareness. I have seen a few Security Officers do this well. They never miss a chance to keep security in the management mind. Particularly in computer security, this means keeping them informed on incidents inside and outside of the company, something that can be done with newspaper and periodical summaries of computer crimes and abuses, and memos. It means including them in the general security awareness program on other issues and soliciting their support on policy, discipline, and money for the security program. Another recommendation is to be personally involved with top level management. If they are new to the organization, go over security procedures with them one-on-one. Be well prepared and brief, but cover the essentials. Leave a telephone number in case they have difficulties and handle their inquiries personally. If an audit shows some problem they are having, help correct it, and keep notes on what was done. Visit now and again and show the flag.

There is another time to keep accurate notes when dealing with management. Some security officers think that their role is tantamount to a mission from God and their approach to management parallels this. It is possible to treat every security incident as the only one there has ever been and the worst there ever will be. Every policy decision can be a life or death struggle. A dispute over money can be another Persian Gulf crisis. This doesn't mean that a security person has to be meek, but there is a limit to pushing a point. A security officer is responsible for doing three things:

1. Make management aware of its responsibilities

2. Advise on the realistic consequences of not meeting those responsibilities

3. CYA, a short term which translates into Take Good Notes

Personally dealing with management carries an aspect of being visible to the public of the work place. Security people who bottle themselves up in an office with a terminal and 100 different reference books, put a large sign outside that says Computer Security, and announce to all that they are "available" anytime, rarely are. I used to go around with security people who had to introduce themselves to everybody we met. Part of security awareness is being out where the people are answering questions, and finding out what's going on. Most of our security audiences are very uncomfortable sitting for a week in a class because they are used to being out of their offices. They should be.

The danger in being out where people are is that they ask questions that are difficult to answer. This sometimes means saying, "I don't know, but I'll find out." Finding out means both people learned something and the next time the same question comes up, the answer is ready. This is what makes experts. People move hardware all the time, add equipment, change offices, move walls, build buildings and a host of other things that affect security. They don't always call and tell the one responsible that it is being done. This is sometimes called "liaison" with personnel, maintenance, engineering, physical/administrative security, the DP staff itself, and a few others. It is just a specialized form of security awareness. Making people who are the "changers" aware that what they do may affect what we do. They will usually "make conversation" if you come around in areas that are of security interest. It always helps to keep a security program proactive instead of reactive; remembering that security is more than just paper and audit records, will help to do that.

While a security person is out and about, there are two special kinds of people to look for. The first is that small, but important group, who will do their security functions well, in spite of often getting nothing for it and, rarely if ever, having any mention of security in their job description. Security people, as a group, are the most persecuted, downtrodden, neglected, and underpaid people in the world, at least to hear them talk. On the underpaid part, they are right. A DP staff will not be very sympathetic because they also fit into the same category. This is all the more reason to go out and find the people who are doing a good job and say "thank you" once in awhile. Every one of these people does a job that the security person would have to do if they didn't. They deserve some thanks for this, and it will be rewarding for both parties.

The other kind of person lies at the other extreme of the cooperation spectrum. This is the kind of person who won't cooperate, hasn't got time for security measures, and doesn't like anyone coming around telling them how to act. This person needs help -- a specialized kind of security awareness. The type of approach is one that comes from The Godfather ; make them an offer they can't refuse. Reason with them in a way they can understand. This doesn't mean threaten, or even give the appearance of doing it. Tone and inflection have quite a bit to do with how a message is perceived [6]. Start with the policy of the agency or company e.g. this real policy of a large defense contractor:

> The first violation results in a letter being placed in the person's personnel file. This assumes the action was inadvertent or accidental.
>
> The second violation results in a 5 day suspension without pay. This may occur for a willful or intentional violation even if it is the first occurrence.
>
> The third violation results in dismissal.

There is no threat made or implied in the expression of the policy. This is help for a person who would otherwise be surprised by the action. Apologizing for the policy or the action is equally poor procedure. The person needs the facts in as straight-forward a manner as possible. Every good security person knows who the people are who need this kind of attention. It is partially an instinct, and partially being visible to the various audiences the security person must serve.

Media, Messages, and the Program Objective

Any person who has a security responsibility has an obligation to inform and educate the audiences they serve. When they get together, at a conference, a symposium, or at a chapter meeting of a security group in their area, they learn things they want to do when they get back. Usually, they try to do too much, too fast. Security education doesn't work quickly, even when spurred on by a management generated crises. It has to work slowly and in small, "chewable" pieces. Map out what is to be done and make a list of priorities. Divide the list into obtainable objectives and select media of several types to carry the messages to the target audiences. Keep the same messages going out on a regular basis to get them to people who missed them the first and second time or are new to your activity.

Second, test to see that the education results in performance. The same office may not be responsible for education and testing so the education program has to mesh with the testing objectives. Coordinate the education program with auditors and inspectors to see that it meets their program direction. Ask for, and expect, feedback that will support or change the thrust of the security education effort and give feedback where it works (people get the message) but doesn't result in performance. This may be a function of poor, or unenforceable policy that needs to be changed.

Third, get out of the office and actually do the job. Reinforce the good and mitigate the poor performance where ever possible.

Security awareness is more than a program; it is a way of doing the things that make up a security officer's responsibilities. It doesn't come quickly or easily and deserves the same amount of attention that other aspects of the security program receive. It means prevention more than correction, though some of each is required. It requires planning, coordination, and a lot of hard work to implement. In the end, it means making the security person's job easier by having employees perform at an acceptable level, not because someone tells them they have to, but because everybody else does it too.

References:

[1] Martin Fishbein, Readings in Attitude Theory and Measurement. Attitude and the Prediction of Behavior. New York: Wiley, 1962, pp. 477-492.

See Also:

C.E. Osgood and P.H. Tannenbaum, "The Principle of Congruity in the Prediction of Attitude Change,"
Psychological Review, vol. 62, pp. 42-56.

[2] John Carroll, Confidential Information Sources: Public and Private. Boston: Butterworth Publishers, 1975, p. 2.

[3] Robert K. Avery, Communication and the Media. New York: Random House, Inc., 1977, pp. 279-293.

[4] Frank Dance and Carl Larson, Speech Communication: Concepts and Behavior. New York: Holt, Rinehart and Winston, 1972, pp. 315-322.

[5] Cal Hylton, "Intra-Audience Effects: Observable Audience Response,: Journal of Communication, pp. 253-265, September 1971.

[6] Wayne N. Thompson, Three Nonverbal Forms Compared with Verbal Code: The Process of Persuasion. New York: Harper and Row, 1975, pp. 149-154.

RETROFITTING AND DEVELOPING APPLICATIONS FOR A TRUSTED COMPUTING BASE

D. Gambel   S. Walter
Grumman Data Systems, Washington Operations
6862 Elm Street
McLean, Virginia 22101

Abstract

This paper discusses the concept of a software analysis procedure to aid in the conversion of existing applications and in the development of new applications for use with a Trusted Computing Base. The use of this analysis within two separate projects, one involving conversion of existing software and one involving development of new software, is discussed to demonstrate the process and to provide background for our conclusions.

Introduction

Recent developments in the field of computer security have brought about a great need for provable secure systems that operate in accordance with DoD 5200.28-STD, known as the Orange Book, published by the National Computer Security Center (NCSC) [1]. With the emergence of several trusted computing bases (TCBs) in the last year, meeting the goal of fielding secure systems has become a possibility. There is also significant effort on the part of several vendors to develop B level generic networks and data base management systems (DBMSs). This leaves a neglected area in secure systems -- integration of the "rescue" of the investment in an installed software base when secure systems are implemented. For these reasons, Grumman Data Systems (GDS) has developed a manual software analysis procedure designed to aid in the conversion of untrusted applications software for use with a generic TCB.

Software Analysis

The software analysis consists of a step-by-step process to determine which pieces of the application software need to be trusted and which do not. Figure 1 gives a graphic view of this process.

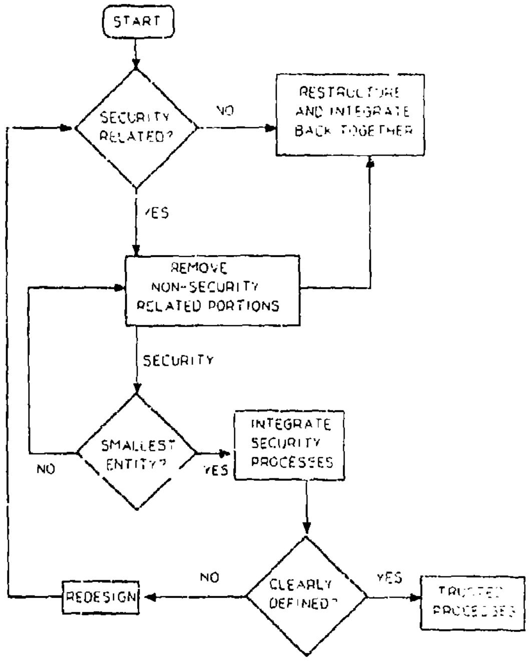

START
SECURITY RELATED?
NO
RESTRUCTURE AND INTEGRATE BACK TOGETHER
YES
REMOVE NON-SECURITY RELATED PORTIONS
SECURITY
SMALLEST ENTITY?
NO
YES
INTEGRATE SECURITY PROCESSES
REDESIGN
NO
CLEARLY DEFINED?
YES
TRUSTED PROCESSES

FIGURE I. SOFTWARE ANALYSIS

The first step is to break the software down into modules (routines and/or subroutines) and then identify which modules perform a security-related function. Security-related implies that the module function relates to enforcement of the security policy and/or accountability criteria detailed in the Orange Book [1]. These modules must be part of the trusted software.

Modules identified with security-related functions must then be further analyzed. Because of the burden of proof placed on trusted software during certification, the use of new trusted software must be minimized. Any of these security-related functions which will now be supplied by the TCB can be eliminated from the applications at this point. All remaining security-related modules must be further analyzed and broken down into the smallest entities possible until they eventually consist of only those functions that absolutely must be trusted. When these software entities have been reduced to the absolute minimum, they are then further analyzed to eliminate any duplicate functions.

These minimum security-related entities are then isolated as trusted processes under the control of the TCB. This isolation requires well structured software with a clearly defined and strictly enforced TCB interface. If the software is not well structured, the requirements should be reviewed, the software redesigned, and then analyzed again.

The next step is to correlate all routines that do not contain any security relevancy and all software entities that do not need to be trusted; these form the bulk of the untrusted software of the system and should be restructured and integrated together.

Converting Existing Applications

Initially, GDS developed this project to explore the possibility of applying retrofitting procedures to applications running on an untrusted multilevel system. We assumed that a TCB would soon be available to rehost existing applications, and we targeted on determining what would be necessary to convert existing applications for use with that TCB.

The untrusted multilevel system in use performs basic C3I functions. Data is entered into the system from either an external communications system or manually from a single keyboard. After entry and labeling, the data is transformed into internal format and maintained in the DBMS where it can be manipulated by the user on site. The system also produces intelligence data to be output to various offsite users. The security clearances of the off-site users were used to determine the protection levels required for internally generated data. The current system uses three protection levels: high (Top Secret with compartments), medium high (U.S. Secret), and medium (Secret Releasable).

Two alternative system designs, multilayered (kernel based [2]) and multilevel (totally trusted [2]), were originally considered to offer the best solutions to converting this system to a trusted multilevel capability. We considered only these two architectures because our research showed that that only these approaches are likely to reach the higher (B3-A1) certification levels.[2] Each design requires a TCB but the multilevel approach also requires a trusted DBMS and trusted applications. We chose the multilayered approach because of our interest in utilizing our existing application base and because no trusted DBMS exists.

The multilayered approach uses existing (therefore untrusted) software supported by a TCB. With modifications as needed according to the results of the software analysis discussed earlier, these applications programs can be used to perform man-machine interface functions, message processing, communications support, correlation, graphics, data base management, and other capabilities as needed. The use of a single untrusted DBMS would require a cryptographic seal to protect resident data from corruption, while multiple single level DBMSs would not.[2,3]

Developing New Applications

In May 1987, a large-scale system was needed within the Department of Defense to support resource tracking for planning, programming and budgeting, tactical and strategic wargaming models; manpower models; force structuring; congressional inquiries; and various program administration and management tasks. The existing system consisted of a B2 Top Secret/Secret subsystem and a separate Unclassified subsystem. The replacement system specified an initial C2 host environment for each processing level that would later be upgraded into a B3/A1 environment by use of trusted communications processors to connect the hosts in accordance with Table 5 of CSC-STD-004-85 [4]. The initial architecture of separate C2 "System High" mainframes for each classification level could be satisfied by using a C2 certified, discretionary access control package for each of the hosts.

The system specification required a single communication interface permitting any user, under mandatory access rules, access from any terminal, any operating system, and/or any data base at any of the three classification levels, subject to the clearance level of the terminal and user. To meet this requirement, GDS designed a Secure Communications Processor Environment (SCPE) based on the Gemini Secure Operating System (GEMSOS). GEMSOS is currently under evaluation by the NCSC and is targeted at the A1 level. The SCPE consists of a cluster of Gemini communications processors (using GEMSOS) connecting the hosts in a network fashion. The communications processors serve as Communications Control Modules (CCMs), providing direct communications from the users to the hosts. One communication processor is reserved to serve as the Access Control Module (ACM), providing access control and audit and file maintenance functions. Figure 2 illustrates this architecture.

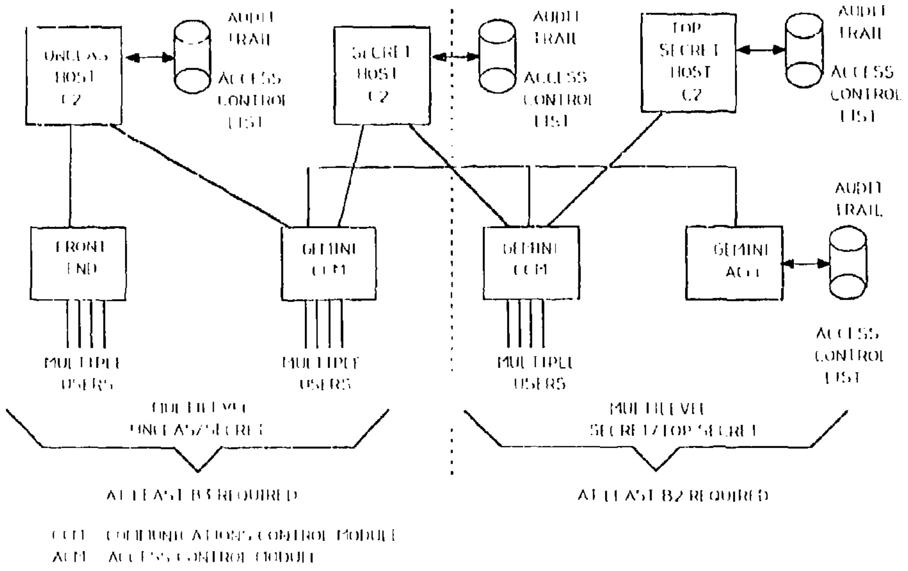


FIGURE 2 SECURE COMMUNICATIONS PROCESSOR ENVIRONMENT

This system required the development of applications programs to perform man-machine interface as well as to perform CCM and ACM functions. The first step was to detail the requirements of the applications to the finest granularity possible. At this point, we were faced with the same question concerning trusted code: What must be trusted and what can be untrusted? While our software analysis procedure was developed for use with existing software, it also served just as well in the case of designing new software. We applied the procedure to each requirement to provide insight into the functions that are security relevant. Experience with both old and new applications demonstrated that the same minimization technique applies to both situations.

Discussion

During system analysis, we considered several questions. First, how to determine which portions of the software must be trusted? We began by separating the requirements into security related and non-security related portions as a base for analysis. We originally assumed that all or most of the security related software would need to be trusted because of the security environment of the system. However, as the requirements analysis continued and as we analyzed trustability in each case, it became apparent that this assumption was wrong. Only the security-related software which does not conform to the model must be trusted.

Consider the case of an incoming message. When the message first enters the system, it must be labeled with the high water mark of the incoming communications line because all data within the system must be labeled immediately upon entry. As the header is parsed, the classification line of the message is read and compared to a table. When the correct label has been identified, it replaces the initial high water label. At first glance, it appears that all of this software needs to be trusted because it deals with security data. Further analysis shows that this is not true, however. The classification line parsing function can be reduced into three subfunctions: reading the classification line, comparing it to a table, and writing the new label to the record. The first two subfunctions, reading the classification line and comparing it to a table, conform to the Bell-Lapadula model [5]. Since these subfunctions conform to the model, they do not alter the secure state of the system and therefore do not need to be trusted. The third subfunction, writing the new label, does not conform to the model because in most cases the new classification label is lower than the old, which requires a write down. Since the writing subfunction does not conform to the model, it must be trusted.

The next question is how to allow the remaining untrusted applications to operate in a multilevel environment and still eliminate the usual performance problems which plague systems of this type. Performance problems are caused by the necessary TCB mediation of all accesses between subjects and objects in the untrusted software. With existing applications, the accesses between subject and object are so numerous that performance is significantly degraded while the TCB performs its required management activities. The solution is to minimize TCB mediation as much as possible without sacrificing the security integrity of the system. Our solution is to limit mediation by implementing more complex, larger object level entities (such as entire applications processes instead of modules). We accomplish this by using several layers of each untrusted application, providing one layer for each classification level of data within the system. For example, our workstation needs three layers, one Top Secret with compartments, one U.S. Secret, and one Secret Releasable. These larger level objects require less import and export of data and therefore less TCB mediation. The final design structure is shown in Figure 3.

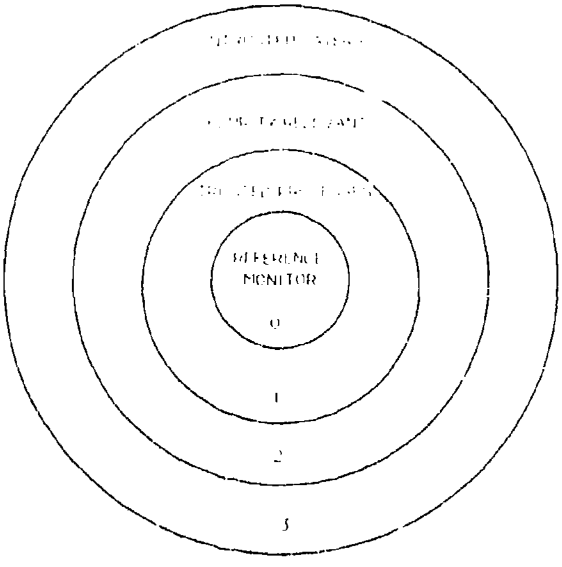


FIGURE 3. DESIGN STRUCTURE

Summary

In conclusion, we believe that the analysis procedure presented provides an organized approach to secure software development which can be applied to existing applications and to new design requirements. In this procedure, the system processes are broken down into small entities that permit detailed analysis to ensure that the trusted processes will be at the absolute minimum. We have developed two conclusions from these experiences. First, the processes which we identified as needing to be trusted were those which violated the security model. All other security related processes are supplied by the TCB. Second, performance problems caused by TCB mediation brought about by security requirements can be somewhat alleviated by implementing larger object level entities in a layered fashion. While this implementation will become less efficient with larger numbers of layers, it is adequate for many existing requirements today.

References

[1] DoD 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria, December 1985 .

[2] Hinke, T., "Secure Database Management System Architectural Analysis," in Proceedings of the National Computer Security Center Invitational Workshop on Database Security, Baltimore, MD, June 17-20, 1986.

[3] Troxell, P.J., "Trusted Database Design," in Proceedings of the National Computer Security Conference, September 15-18, 1986, pp. 37-40.

[4] CSC-STD-004-85, Technical Rationale Behind CSC-STD-003-85, Computer Security Requirements, Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, June 25, 1985.

[5] Bell, D.E. and Lapadula, L.J., Secure Computer Systems: Mathematical Foundations and Model, Technical Report M74-244, The Mitre Corporation, Bedford, MA, May 1973.

CONFERENCE REFEREES

Alfred Arsenault, *National Computer Security Center*
Marshall Abrams, *The MITRE Corporation*
James P. Anderson, *James P. Anderson Company*
Blaine Burnham, *National Computer Security Center*
David Balenson, *National Bureau of Standards*
D. Elliott Bell, *Trusted Information Systems, Inc.*
Dennis K. Branstad, *National Bureau of Standards*
Earl Boebert, *Honeywell Corporation*
R. Leonard Brown, *The Aerospace Corporation*
Thomas Berson, *Anagram Laboratories*
John Campbell, *National Computer Security Center*
Deborah Cooper, *Unisys Corporation*
Donald Crossman, *National Computer Security Center*
Paul Cudney, *System Development Corporation*
Deborah Downs, *The Aerospace Corporation*
Dorothy E. Denning, *Digital Equipment Corporation*
Gregory Elkman, *Department of Defense*
Martin Ferris, *Department of the Treasury*
Morrie Gasser, *Digital Equipment Corporation*
Harriet Goldman, *The MITRE Corporation*
Joshua Guttman, *The MITRE Corporation*
Douglas B. Hardie, *National Computer Security Center*
Brett Hartman, *Research Triangle Institute*
Jack Holleran, *National Computer Security Center*
Jim Houser, *National Computer Security Center*
Brian Hubbard, *National Computer Security Center*
Douglas B. Hunt, *NASA*
Bill Hunteman, *Los Alamos National Laboratories*
Irene Isaac, *National Bureau of Standards*
Dale M. Johnson, *The MITRE Corporation*
Stuart Katzke, *National Bureau of Standards*
Richard Kemmerer, *University of California*
Stan Kurzban, *IBM*
Steve LaFountain, *National Computer Security Center*
Sue Landauer, *Trusted Information Systems, Inc.*
Carl Landwehr, *U. S. Navy Research Laboratory*
Steven Lipner, *Digital Equipment Corporation*
Teresa Lunt, *SRI International*
Barbara A. Mayer, *National Computer Security Center*
Frank Mayer, *Trusted Information Systems, Inc.*

Terry Mayfield, *Institute for Defense Analysis*
John McLean, *U. S. Navy Research Laboratory*
Lynn McNulty, *U. S. Department of State*
Jonathan Millen, *The MITRE Corporation*
Robert Morris, *National Computer Security Center*
Jack Moskowitz, *National Computer Security Center*
William H. Murray, *Ernst & Whinney*
Peter Neumann, *SRI International*
Tom Parenty, *Trusted Information Systems, Inc.*
Donn Parker, *SRI International*
Mario Pozzo, *University of California at Los Angeles*
Sami Saydjari, *National Computer Security Center*
Marvin Schaefer, *Trusted Information Systems, Inc.*
Roger R. Schell, *Gemini Computers, Inc.*
Daniel D. Schnackenberg, *Boeing Military Airplane Company*
Suzanne T. Smith, *Los Alamos National Laboratories*
Bill Shockley, *Genini Computers, Inc.*
Dennis Steinauer, *National Bureau of Standards*
Tad Taylor, *Research Triangle Institute*
Mario Tinto, *National Computer Security Center*
Hal Tipton, *Rockwell International*
Geoffrey Turner, *Bank of America*
Holly M. Traxler, *National Computer Security Center*
Grant Wagner, *National Computer Security Center*
Wayne Weingaertner, *National Computer Security Center*
Stephen Walker, *Trusted Information Systems, Inc.*
David Wilson, *Ernst & Whinney*
Kimberly Wilson, *Booz, Allen & Hamilton, Inc.*