

272 FILE COPY

UNLIMITED

BR108825

2

AD-A201 604



**RSRE
MEMORANDUM No. 4222**

**ROYAL SIGNALS & RADAR
ESTABLISHMENT**

A SECURITY MODEL AND ITS IMPLEMENTATION

Author: S R Wiseman & C L Harrold

RSRE MEMORANDUM No. 4222

**PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.**

**DTIC
ELECTE
DEC 27 1988**
S D
cb **H**

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4222

Title: A Security Model and its Implementation

Author: S R Wiseman & C L Harrold

Date: September 1988

SUMMARY

A new security model is proposed which allows the notions of confidentiality and integrity to be expressed in one coherent framework. Confidentiality is taken to be solely concerned with the observation of classified information, while separation of duty is employed as the technique for assuring the integrity of the security labels which are the basis of confidentiality. Discretionary access control is modelled by access control lists whose integrity is also provided by separation of duty controls. The specification of a simple system is given as an example of the techniques and methods for implementation are discussed.

Copyright
©
Controller HMSO London
1988

Contents

1. Introduction
 2. The Model
 3. μ SERCUS: An Example System
 4. Implementation & Assurance
 5. Specification of μ SERCUS
 6. Conclusions
- References
- Z Notation



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

1. Introduction

This paper describes how the model of security described in [Terry&Wiseman88] can be used to implement flexible systems with high assurance using the SMITE [Wiseman86] secure system. The model was developed because SMITE appeared to offer more flexible protection than is required to implement other security models. In particular SMITE seemed to provide a way of tackling the wider issues of computer security, such as integrity. However, even though the model was developed specifically for SMITE it is believed that it is more generally applicable and, as is shown later, can be implemented on conventional systems.

The modelling approach is basically that of the original Bell-LaPadula paper [Bell&LaPadula73a], however the notion of confidentiality, the axioms and techniques used are profoundly different from their later papers ([Bell&LaPadula73b], [Bell74] & [Bell&LaPadula76]) and as enshrined in the "orange book".

The SMITE approach has been strongly influenced by the concept of separation of duty presented by [Clark&Wilson87]. Clark and Wilson argued that maintaining consistency between the internal system state and the external environment was the fundamental mechanism for upholding a system's integrity, and that this is achieved by employing separation of duty. This paper argues that separation of duty is more fundamental and, along with confidentiality and discretionary access control, plays an indispensable part in security.

The approach proposed in this paper aims to cover the whole problem of security, but with no loss of rigour in its approach to individual aspects, such as confidentiality or assurance. The approach may therefore seem somewhat radical, so to avoid any misunderstandings the notions of security will be developed from scratch. It must be emphasised that if an "Orange Book reserved word" is used in this paper, its true English meaning is intended.

1.1 What is Security?

The dictionary defines "security" as "assured freedom from danger, damage etc.". The security of a system can therefore be defined in these terms by specifying:

1. which properties of which elements of the system are important,
2. what protection from which attacks is required,
3. how much assurance is required that such defences are successful.

The general English sense of this definition is not at odds with either military or commercial security practice. However in the military and government arena, attempts to produce a more definitive definition of security have emphasised a particular property, confidentiality, as being of paramount importance.

The dictionary definition of "confidential" is "given in confidence; secret; private" and "confidence" is defined as a "feeling of trust in a person or thing" and "secret" as "kept hidden or separate from the knowledge of others". Thus confidentiality is about the knowledge of things and the distribution of that knowledge amongst individuals.

The aspect of confidentiality which says that things are kept secret and private is easily stated and can be implemented by ensuring that things are discrete and attributed to just one individual. The assurance of such an isolation policy can be very high because it simply says that if you want something done in such a way that no one else knows about it, you must do it yourself.

Obviously such an isolation policy is not practical, because it must be possible for one to delegate tasks, thus it must be possible to distribute knowledge amongst individuals "in confidence". This can be implemented by extending the isolation policy in the following way. If things are isolated and are secret or private, that is known to just one individual, and that individual can establish a basis of trust in some other individual then the private thing may be passed on.

A general confidentiality policy therefore requires that things are isolated and attributable to individuals, and states the means by which an individual can come to trust that others do not pass on or leak the secrets given to them.

While military computer security is oriented towards confidentiality, much current commercial security is oriented towards another particular property: integrity. The dictionary defines integrity to be "free from damage, injury etc; valid, logical or justifiable". This adequately describes the facet of security which requires the system to have "desirable" properties, such as that things are undamaged and valid.

The integrity requirement can be easily fulfilled, with high assurance, by a policy which states "if you want it done right then do it yourself". This is similar to the isolation approach to confidentiality and is similarly impractical. Thus a general integrity policy requires that things are isolated and attributed to an individual and lays down a basis for establishing trust in others that they do not damage or invalidate things given to them.

1.2 Establishing Trust in Individuals

While high assurance confidentiality and integrity are provided by a policy of "doing it yourself", in practice it is necessary to delegate tasks and share things and knowledge. For this it is necessary to establish trust in individuals. This is often done by "investigating" individuals to ensure that they have no inclination to engage in activities which are prejudicial to the tasks and information given to them. If the individuals prove to be acceptable they are given a certificate (perhaps in the form of a clearance or pass) and are appraised of the procedures and properties that they must follow and uphold.

For confidentiality individuals are required to label data correctly and only pass information to those individuals who are cleared and have a need to know. For integrity individuals must not defraud or sabotage the company or allow others to do so.

This approach depends on the cooperation of all individuals to uphold the common aim of the policy, which assumes they all share the same objectives. However the assurance that this is true is only as good as the assurance that the investigation of individuals was accurate and remains valid. This is unlikely to be good enough because such checks are fallible and an individual's motivation may change. In any case mistakes and oversights are bound to occur when individuals cooperate to carry out complex tasks. The system must therefore be arranged to give additional assurance by making it as difficult as possible to violate the policy. That is the actions of the individuals must be controlled.

Obviously if every action of the individuals is controlled the system is secure because the job is effectively no longer delegated, but this is impractical. The system must therefore leave individuals a degree of discretion in their actions, but seek to limit the damage caused by an errant or subverted individual. This loose control of individuals, along with a means of labelling the degree to which individuals share objectives for the uncontrolled behaviour, forms the basis of trust which allows tasks to be delegated with a high assurance that the system remains secure.

1.3 Individual's Motivation

In this discussion we take the owners of the system to be a government or company who delegate tasks to individual citizens or employees, yet require that the interests of the corporate body are protected. Those whose interests are being protected and those who may subvert them may be drawn from the same population, eg. the citizens of a country, or from disjoint populations, eg the shareholders and employees of a company.

For a system involving sensitive information, but which does not cover official secrets, background checks to assess an individual's loyalty may be considered unacceptable because of concerns for individual privacy, civil rights issues and, not least, expense. However, Clark and Wilson have shown that the same goal is achieved in the commercial world, without background checks, by exploiting differences in the motivation of individuals.

For example, the highly paid manager of a bank will not collude with lowly paid counter clerks in a petty fraud. Thus it suffices to ensure that transactions made by clerks require a final counter signature from the manager. Major fraud by the manager alone is prevented because the bank does not allow the manager to carry out basic transactions. The manager and a clerk are unlikely to collude to perpetrate a major fraud because of the paradoxical motivation of the clerks not to cooperate, perhaps because "its always the little guy who carries the can".

This notion of separation of duties amongst individuals gives the added assurance that a task will be carried out without any collusion to violate the policy. However it is only capable of ensuring the integrity of tasks because it only takes one subversive individual, or a mistake, to compromise confidentiality. Thus the more individuals who possess a secret the greater the likelihood that confidentiality will be compromised, but conversely for integrity the more individuals who must be involved in completing a task the lower the chances of unnoticed mistakes or collusion.

This divergence of properties between confidentiality and integrity does not in practice present a problem as, when total security is considered, the two aspects are played off against each other.

1.4 Total Security

In a system which includes confidentiality controls, it is necessary to assume the worst of the individuals who produce and label information. That is new information must be labelled with the clearance of the individual who produced it. This, of course, causes data to be over classified, which leads to a requirement for altering the classification of data. This need also arises because, if the system is viewed as part of an organisation, classifications need to be changed to reflect changes in the environment. Effectively the system must allow individuals to alter the controls that limit the activities of individuals. There is obviously a requirement for maintaining the integrity of these controls is the sense that an inappropriate or fraudulent change is not made. In other words integrity control is a prerequisite for implementing confidentiality control in practice.

To establish a suitable basis of trust for confidentiality and integrity, the system must ensure that each entity manipulated by individuals is attributed with a classification and that each request indicates the identities of the users responsible for it. Confidentiality insists that in order to observe the contents of an entity in the system, the clearances of the individuals making the request must dominate the classification of the entity. Integrity says a modification may only occur if enough individuals wish it to occur.

A third area of security, which can be used to enforce a finer granularity of protection than straight separation of duty, can be provided. The use of this is left to the individuals' discretion to use as they see fit, and so is usually called discretionary access control, though this does not imply that it is not enforced by the system.

Discretionary access control is simply the notion of attaching to each entity a list of those individuals who are allowed to access them, with an indication of what operations those individuals may perform. When an entity is created the list contains only the creator's identity, which prevents anyone else from accessing it. Those who wish to risk the threats of sharing access to their entities may do so by extending the access control list, but they are at least safe in the knowledge that the system's confidentiality and integrity constraints will limit their risk.

For example, separation of duty in the bank example ensures that transactions require two different signatures. However to ensure that it is the manager who provides the counter signature, and not just another clerk, access control lists can be used. This is, therefore, an application specific constraint placed above the system policy of separation of duty.

1.5 Summary

The notion of security that will be modelled therefore comprises three parts: confidentiality, integrity and discretionary access control. However these aspects do not stand alone. In practice users must be able to manipulate security labels and so confidentiality relies upon the label's integrity. The principle of separation of duty is used to provide such integrity, but in practice it must be relaxed. It is then necessary to utilise discretionary access control or confidentiality to provide assurance that integrity is not lost.

Fortunately this recursive spiral does not go on forever, because ultimately some controls are the responsibility of the system owner. For example a government is responsible for the procedures which concern giving clearances to individuals and will incorporate in them the necessary separation of duty controls. These, by definition, are sufficient.

2. The Model

2.1 The Model of Execution

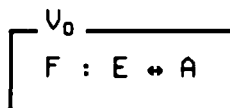
The model is based on a simple state machine and thus consists of a definition of the state and a definition of the transition rules. The state is modelled as a relation between abstract sets of entities and attributes. Entities represent the active and passive components of the computer system which are visible to each other and can be detectably changed by active entities. Attributes represent immutable components which are only visible to some subset of the entities.

For example, in a bare machine process shells and memory segments would be active and passive entities and integer values would be attributes. The relation between entities and attributes would model integers being stored in memory locations and registers. Note that integers do not change. Assignment simply relates a location to a different integer, it does not alter the integer initially stored in the location.

A more security oriented example is where text and classifications are attributes, because they do not change, while classified documents are entities, because their classification may be changed or their text may be replaced with an amended version.

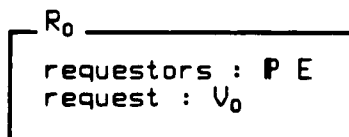
The formal specification, which is written in Z [Spivey87], begins by introducing two sets, E and A , to represent entities and attributes. A schema, V_0 , which represents the state of a simple machine is then introduced. This comprises a relation, F , between entities and attributes.

[E, A]



The domain of the relation F may not include all entities, modelling the fact that some entities have not yet been created or have already been destroyed. Note that this implies that an entity which contains nothing does not exist. Similarly the range of F may not include all attributes, and this may be viewed as modelling the fact that not all attributes currently exist in the state.

A request to change the state of the machine, R_0 , is considered to originate from a set of entities, the requestors. The request itself comprises some state which is essentially the parameters of the request. At this stage nothing is said about the validity of the request.



The valid state transitions that the machine can make is given by W_0 . This insists that the entities making the request exist and that the request is a subset of the current state. The transition will take the machine to a new state, v' , and will output a decision, $d!$, as to whether the request is allowed, though the value these take are as yet unspecified.

$D \in \{ \text{"yes"}, \text{"no"} \}$

W_0
$r? : R_0$ $d! : D$ $v, v' : V_0$
$r?.requestors \subseteq \text{dom } v.F$ $r?.requestors \neq \{\}$ $r?.request.F \subseteq v.F$

The request does not restrict which entities may be altered, because the model assumes that all entities are universally accessible, however it is assumed that any entity mentioned in the request is "observed", and that no others are. That is all attributes, not just those mentioned in the request, that are contained in the entities mentioned in the request are either moved, copied or used to influence the creation of new attributes. Note that it is possible to "create" attributes which already exist somewhere else in the state. For example the integer attribute "42" may already exist in some entity, but it can be created by multiplying 6 and 9[†] without observing that entity. However if a created value could also have been observed the model must assume that it has been observed.

This view allows the interesting situation where an unclassified individual is able to create attributes which are exactly the same as the contents of a secret entity though, because the system is secure, without observing the entity. This is effectively guessing the secret and of course poses no security risk if the source of the information is properly attributed.

The effect of the request, that is the choice of new state, is governed by state transition rules of the form p_0 . These specify the final state, v' , after applying a request, $r?$, in a state v . In a particular system design, there will be many such rules, each differing in the constraints it applies to the parameters and the effect it has. In effect p_0 is the most general rule because it places no conditions on the parameters and does not specify the result. Particular rules will be subsets of p_0 .

p_0
$r? : R_0$ $v, v' : V_0$

A particular system therefore comprises a collection of rules. This collection does not necessarily form a set of p_0 because more than one rule may allow the same state change with a particular request. This is represented by a bag, which is effectively a function that gives the number of times an element occurs in a collection. So a system design is represented by a bag of rules, w .

However for this model of a computer to be of use in modelling secure systems it is important that the system's behavior is predictable. To this end the set of rules must be constrained so that only one rule can cause the system to enter a particular state for each combination of request and initial state.

The schema Transition_0 defines the permitted transitions, that is those valid transitions which are allowed by the bag of rules. It also implies that at most one rule can apply to each permitted transition. Effectively Transition_0 is the set of valid transitions where exactly one rule applies and some state change occurs or no rules apply and nothing changes.

[†]arithmetic courtesy of The Hitch Hikers Guide to the Galaxy

Transition ₀
W_0 $\omega : \text{bag } P_0$
$\omega(\text{rule}) = 1 \wedge d! = \text{"yes"}$ $\vee \omega(\text{rule}) = 0 \wedge d! = \text{"no"} \wedge \text{rule.v} = \text{rule.v'}$ where $\text{rule} : P_0 \mid \text{rule.r?} = r? \wedge \text{rule.v} = v \wedge \text{rule.v'} = v'$

The behaviour of a particular run of a system is defined by its initial state, z_0 , the sequence of inputs (requests) and the set of permitted transitions. A record of the resulting outputs (decisions) and intermediate states is called an appearance of the system.

Appearance ₀
$\text{permitted} : P \text{ Transition}_0$ $r : \text{seq } R_0$ $d : \text{seq } D$ $v : \text{seq } V_0$ $z_0 : V_0$
$\#r = \#d = \#v$ $\forall n : \mathbb{N} \mid 2 \leq n \leq \#r .$ $\exists t : \text{permitted} .$ $t.r? = r(n) \wedge t.d! = d(n) \wedge t.v = v(n-1) \wedge t.v' = v(n)$ $\exists t : \text{permitted} .$ $t.r? = r(1) \wedge t.d! = d(1) \wedge t.v = z_0 \wedge t.v' = v(1)$

2.2 Modelling a Secure System

The description given in the previous section is a model of computer systems in general. To model the approach to security described in the introduction it is necessary to refine the components of the state relation, F . It is necessary to distinguish certain classes of attributes, such as classifications and identities, and to add structure to the state relation, for example to identify those parts which represent an entity's classification.

The set CLASS is a subset of the attributes which represent both classifications and clearances. These form the basis of trust which allows the system's owners to rely on confidentiality controls to restrict the behaviour of the individuals they delegate to operate the system.

CLASS : P A

The set TRUST is a subset of the attributes which designate various degrees of trust that can be ascribed to active entities, that is those that make transition requests. The amount of trust placed in an entity depends on the trust in the individual it acts for and also whether it is a faithful proxy of that individual. To be a faithful proxy the entity must be directly commanded by the individual or by other faithful proxies. This is the notion of the trusted path [Wiseman et al. 1988]. One particular degree of trust is distinguished, untrusted, which indicates the individual cannot be relied on to uphold the security of the system or is not using the trusted path.

TRUST : P A
untrusted : TRUST

The set ID is a subset of the attributes which act as unique identifiers for individuals. This is used to control separation of duty and for discretionary access control.

ID : P A

Note that the specification does not insist that these are disjoint sets. It may be possible for a particular system to use the same attribute, an integer for example, to implement CLASSES and IDs. These would be distinguished by context arising from the structure imposed on F.

Other attributes, not included in CLASS, TRUST and ID, will be used by particular implementations as information attributes required to carry out the application specific tasks.

Parts of the state relation F are distinguished, giving it structure and allowing components of the state to be examined. This gives a new version of the state schema specifically for modelling secure systems, V_s .

V_s	
F	: E $\leftrightarrow$ A
Class	: E $\leftrightarrow$ CLASS
Id	: E $\leftrightarrow$ ID
Trust	: E $\leftrightarrow$ TRUST
Acl	: E $\leftrightarrow$ ID
Class $\cup$ Id $\cup$ Trust $\cup$ Acl $\subseteq$ F	

Class is a partial function giving the classification or clearance of an entity. Id is a partial function giving the identity of the individual who "owns" the entity. Active entities are proxies for their "owner". Trust is a partial function that defines the degree of an entity's trustworthiness. Acl is a relation giving the identities of individuals who are allowed to observe or modify an entity.

Note that the specification does not insist that these relations and functions are disjoint, merely that they are each subsets of the total state relation F. Also, other components of F will be used to represent application specific information.

From this new version of V new versions of R, W, p, Transition and Appearance are defined, which are just those using V_s instead of V_0 . Strictly this is a refinement [Morgan+1.1.88] but for conciseness this is left as an intuitive step.

Classifications must form a lattice in the usual manner, with $\geq$ representing "dominates" and LUB and GLB giving the least upper bound and greatest lower bound of a set of classifications. The function HIGHER takes a function from entities to classifications and a classification and returns that set of entities which the function maps to higher (or equal) classifications. LOWER is similar and gives the entities dominated by the classification. LUB is also defined to merge together two entity to classification functions, where the result maps elements which are in both functions to the least upper bound of their two classifications and entities in just one are passed through. For conciseness, the exact specification of these is omitted.

$\geq$	: lattice(CLASS)
LUB	: P CLASS $\rightarrow$ CLASS
GLB	: P CLASS $\rightarrow$ CLASS
HIGHER	: (E $\leftrightarrow$ CLASS) $\times$ CLASS $\rightarrow$ P E
LOWER	: (E $\leftrightarrow$ CLASS) $\times$ CLASS $\rightarrow$ P E
LUB	: (E $\leftrightarrow$ CLASS) $\times$ (E $\leftrightarrow$ CLASS) $\rightarrow$ (E $\leftrightarrow$ CLASS)

A number of auxilliary definitions are required to make the specification more readable, though again their exact specifications are omitted. The generic function RELATES produces the relation which maps each member of the first set to each member of the second set. The domain restriction operator, δ , is defined to give the sub-state which is only concerned with a particular set of entities and the domain subtraction operator, $\ominus$, gives the sub-state without those entities. The subset relation, $\subseteq$, between V_s holds when the components are subsets. The subtraction operation, $\setminus$, between V_s gives those elements of the state which are in the first state but not in the second.

$$\text{RELATES} : (P X \times P Y) \rightarrow (X \leftrightarrow Y)$$

$$\begin{aligned} \delta & : (P E \times V_s) \rightarrow V_s \\ \ominus & : (P E \times V_s) \rightarrow V_s \\ \subseteq & : V_s \leftrightarrow V_s \\ \setminus & : (V_s \times V_s) \rightarrow (E \leftrightarrow A) \end{aligned}$$

An additional schema will now be given that defines a number of useful sets of entities regarding a transition. This will be incorporated into following schemas making the specification more concise.

$$\begin{aligned} & \delta \text{Transitions} \\ & t : \text{Transition}_s \\ & \text{observed, trustworthy, new, changed, accessed, source} : P E \\ \\ & \text{observed} = \text{dom } t.r?.\text{request}.F \\ & \text{trustworthy} = \text{dom}(t.v.\text{Trust} \triangleright \{\text{untrusted}\}) \\ & \text{new} = \text{dom}(t.v'.F) \setminus \text{dom}(t.v.F) \\ & \text{changed} = \text{dom}(t.v' \setminus t.v) \cup \text{dom}(t.v \setminus t.v') \\ & \text{accessed} = \text{changed} \cup \text{observed} \\ & \text{source} = \text{observed} \cup t.r?.\text{requestors} \end{aligned}$$

Observed is the set of entities which are observed during a transition. Those entities who are acting on behalf of individuals who are not untrusted and are using the trusted path are given by trustworthy. Entities which are created are given by new. Entities which either gain or lose attributes, including those entities that are created or destroyed, are given by changed. Entities which have been changed or observed are deemed to have been accessed. The source of influence over the transition is the observed entities and the entities making the request.

2.3 The Axioms

Axioms for the three aspects of security, that is confidentiality, separation of duty and discretionary access control, will now be given separately. It should be noted that these do not stand alone, and that in isolation they are likely to run counter to one's intuitive notion of security.

2.3.1 Confidentiality

Confidentiality is concerned solely with the Class subset of F , which attributes classifications to observed entities and clearances to the requestors. It is assumed that all entities in the domain of that subset of the state which forms the parameters of the request are observed and no others are.

Thus the axiom for confidentiality insists that each requesting entity and each observed entity has a classification, and that the classification of each requestor dominates the classification of all the observed entities.

Confidentiality _____

δ Transitions

source $\in$ dom t.v.Class

$\text{GLB } t.v.\text{Class}(t.r?.requestors) \geq \text{LUB } t.v.\text{Class}(\text{observed})$

This axiom does not preclude a system that contains rules which allow individuals to either enter secrets and then downgrade them to unclassified or to copy them to unclassified entities. At first sight this may appear to be blatantly insecure. However both these examples concern the integrity of the classification labels attached to entities. This aspect is therefore properly the concern of separation of duty and not confidentiality.

2.3.2 Separation of Duty

Separation of duty is concerned with the Trust and Id subsets of the state relation. Requestors act on behalf of individuals and have ascribed to them a degree of trust which reflects the degree of trust placed in the individuals and whether they are on the trusted path. The strictest form of separation of duty insists that a transition cannot occur unless all trustworthy individuals agree that it may. Untrusted requestors acting for trusted individuals are essentially those who are not being commanded from the trusted path and hence their opinion is of no value.

The strict separation of duty transition axiom ensures that all individuals are represented by at least one requestor which is not untrusted.

Strict Separation Of Duty _____

δ Transitions

$t.v.\text{Id}(t.r?.requestors \cap \text{trustworthy}) = \text{ID}$

It can be seen that this axiom is very strong and in practice it will be necessary to allow some relaxations. For example, if the only concern is the integrity of security labels then separation of duty need only be applied to those transitions which alter the classification of some material.

In practice transition requests will only be made by a single requestor. It will therefore be necessary to ensure that changes occur only after a suitable collection of requests have been made. For example one transition may request a change and another may action it. As long as two different individuals were responsible for these requests, separation of duty will be upheld. In general a sequence of requests made by single individuals will be needed to provide n-way separation of duty.

This corresponds to the history recording mechanisms proposed by [Karger88] and an earlier formulation of this model [Terry&Wiseman88]. These, however, relied on examining the entire history of the system and it is unlikely that this could be done efficiently. The approach proposed here allows separation of duty to be upheld on the basis of very small histories which can be incorporated into the objects themselves.

The definition of confidentiality did not specify that all entities which are accessed must have appropriate classifications, only that they have some classification. Separation of duty is offered as the means of ensuring that classifications are appropriate. Similarly separation of duty depends upon the integrity of the Id and Trust labels. If the task of changing these controls is delegated, they must themselves be subject to separation of duty.

Separation of duty does not say anything about who may carry out changes to an entity, other than sufficient individuals must agree that it can happen. Discretionary access control may be used to ensure that only appropriate individuals can make certain changes.

2.3.3 Discretionary Access Control

Discretionary access control is the notion that an individual is associated with each requestor and that a set of individuals are associated with each entity. An entity may only be accessed if the requestors' identities are in its set of "permitted" identities. This is most commonly implemented as an access control list (ACL). However, if the security of a large system is based entirely on ACLs which can be modified at the discretion of the individual users, its behaviour cannot be predicted. This is the reason for introducing the broad controls of confidentiality and separation of duty. Note that altering an ACL is itself typically controlled by some ACL.

Discretionary access control is therefore concerned with the Id and Acl subsets of the state relation. The access control list modelled here is the simplest possible, though more grandiose schemes are readily incorporated into the model. This simple version allows any individual mentioned in the ACL equal rights to alter or observe the entity. In particular altering the ACL is not limited to the entity's owner and "observe access" cannot be granted without granting "modify access". However, altering an ACL is like altering anything else and is subject to separation of duty.

The axiom states that each requestor's identity must be included in the ACLs of all entities which are observed or changed (excluding new entities). Note that entities with no entries in their ACL can never be accessed and that all requestors must be acting on behalf of an individual.

Discretionary

δ Transitions

$t.r?.requestors \text{ RELATES } (accessed \backslash new) \subseteq t.v.Id ; t.v.Acl^{-1}$

2.4 Summary

The model presented here has dispensed with notions introduced by later examples of Bell-LaPadula modelling, like *-property, Hierarchy, Compatability and Tranquility, and has returned to the simple elegance of the basic approach. Systems based on an "Orange Book" Bell-LaPadula model employ Trusted Processes to allow transitions that do not meet the axioms yet still uphold the system's policy. While these do in practice employ mechanisms like separation of duty to preserve the integrity of labels, the model proposed by this paper explicitly incorporates separation of duty and so makes the design decisions and their effect more open.

The "Orange Book" Bell-LaPadula model describes security as a property of a state in terms of the kinds of access that can potentially occur in the future. This is reasonable if the system is being modelled at a low level, in that checks can only be made efficiently when files are mapped into a process' address space, as performing checks on every access is very costly. Defining confidentiality in these terms is quite straightforward, the Simple Security Property, however ensuring the integrity of labelling requires the use of the much less obvious and very restrictive *-Property.

The model proposed by this paper describes security as a property of a transition. This allows confidentiality and integrity of labelling to be defined directly in terms of what actually happens, rather than what may happen in the future, giving a more obvious and flexible definition. The reason this can be implemented efficiently is that the system is being modelled at a high level. It is intended that systems are modelled in terms of objects at the user level, such as documents and mail messages, rather than operating system objects like memory segments and processes. Accesses are therefore actions such as "obtain a copy of the (unalterable) contents of this document", for which a single check is affordable.

3. μ SERCUS: An Example System

To illustrate the use of the modelling approach in practice, the specification of a simple system, μ SERCUS, will be developed. This is a cut down version of SERCUS which is a document handling system that has been used to demonstrate various aspects of SMITE [Harrold88]. This section presents a set of transition rules which are employed in section 5 where a complete specification of the example system is given. This specification combines the rules, which describe the security aspects of the system, with a description of the system's functionality.

This set of rules is just one of the many that are possible. They are tailored towards the example system and have been deliberately made as simple as possible. This should increase the likelihood that they can be shown to be consistent and to uphold the axioms, which will be an important consideration for systems of significant complexity. Therefore the rules each perform a simple operation, such as create/destroy an entity and create/lose an attribute.

3.1 The Requirements of μ SERCUS

The primary concern of this system is maintaining confidentiality. The owners of the system are absolutely trusted in that they, by definition, cannot violate confidentiality. Certain software, such as the login program, is certified to behave as their faithful proxy at all times. Such entities are ascribed the highest form of TRUST which will be denoted by "trusted".

The system's owners delegate work to the users of the system, however they do not trust the users to maintain the integrity of the labels, upon which confidentiality is based, under all circumstances. They therefore ascribe to them a weakened form of trust, denoted by "loyal".

It is assumed that loyal users are generally reasonably honest and that on the whole they will not knowingly cause a breach of security. However they may do so through mistakes and oversights. Therefore, when documents are to be downgraded below a certain level, separation of duty is to be applied as a safeguard. However it is not considered feasible that loyal users will mistakenly send information through the signalling channels which, for example, arise through creating or regading entities.

For flexible operation, the system must allow loyal users to create documents which are classified lower than their clearance. In effect this is a downgrade without the involvement of another user. To prevent misuse of this feature it is necessary to introduce a system of "dual labels" [Woodward87]. A high water mark is maintained for draft documents which records the highest classification of information that may have been included in them. A loyal user, acting alone, may downgrade a draft document as long as the new classification dominates the high water mark. To downgrade lower than this the consent of another loyal user is required.

3.2 Application Specific Extensions

The relaxed separation of duty axiom which is to be used by μ SERCUS insists that downgrade requests can only be made by more than one requestor. In practice some individual first requests that a document be downgraded and some time later another will approve the request and the document will be downgraded. Thus two transitions, each made by just one requestor, are made. To ensure that the axiom is upheld it is necessary to record the requested classification and the identity of the requestor and ensure that it is a different individual who actually carries out the downgrade.

Thus the state must be extended to include high water marks and downgrade requests. At the same time the part of the state which is not refined into Class, Id, etc. is formed into the relation Other which allows the specification to assert that "everything else stays the same". However, note that the specification still does not insist that these structures partition the state.

V_A	
V_S	
Hwm	: E $\leftrightarrow$ CLASS
ReqClass	: E $\leftrightarrow$ CLASS
ReqId	: E $\leftrightarrow$ ID
Other	: E $\leftrightarrow$ A
Class U Id U Acl U Trust U Hwm U ReqClass U ReqId U Other = F	

The schemas R_A , W_A etc. are defined in exactly the same way as before, except that V_A is used instead of V_S . Similarly for operators like Δ and $\subseteq$.

The trusted and loyal attributes must be introduced and for convenience an ordering, $\geq$, is given. The functions LEAST and MOST give the least and most trusted member of a set, though these are not fully specified.

trusted, loyal : TRUST $\geq$: TRUST $\leftrightarrow$ TRUST LEAST _ : P TRUST $\rightarrow$ TRUST MOST _ : P TRUST $\rightarrow$ TRUST

trusted $\geq$ loyal $\geq$ untrusted

The Hwm function records information about the history of modifications made to an entity. It is therefore necessary to give an axiom which ensures that this history is correctly updated. Essentially, when an entity is altered its high water mark is raised so that it dominates the classification of all the observed entities and that of the requestors. Another example of a history mechanism would be the recording of journalling information.

HighWaterMarks
$\delta\text{Transition}_A$
$t.v'.Hwm^{-1} ; t.v.Hwm \subseteq (\geq)$ $GLB\ t.v'.Hwm(\text{changed}) \geq \text{level}$ where $\text{level} \triangleq LUB\ (t.v.Class \bullet t.v.Hwm)(\text{source})$

The first predicate states that high water marks can never be lowered, though they may be removed, and the second that the resulting high water mark, if any, on all changed entities must dominate the highest classification of information that could have been placed in it. When entities have no high water mark their classification is used to calculate this.

Separation of duty has been proposed as the means of ensuring the integrity of Class, Id, Trust and Acl. The axiom given so far is too strong because it requires that all users must request a change before it can happen. In μSERCUS separation of duty can be relaxed, because, for example, it is considered sufficient to have only two individuals agree that a classification can change, and under some circumstances only one is required. Thus an application specific separation of duty axiom is required. This will be given in several parts, each governing changes to different parts of the state.

51. If a new entity is given an Id or an existing entity changes its Id, the requestors must all be trusted. Note that only one requestor is required, because Trusted individuals may perform any action.

SeparationOfDuty _{Id}
t : Transition _A
$t.v'.Id \setminus t.v.Id \neq \{\}$ $\rightarrow t.r?.requestors \subseteq \text{dom } t.v.Trust$ $\wedge \text{LEAST } t.v.Trust(t.r?.requestors) = \text{trusted}$

52. All entities whose degree of trust is changed and those who are created and given a degree of trust, cannot become more trusted than the least trusted requestor. Again, it is not necessary to have more than one requestor for such changes.

SeparationOfDuty _{Trust}
t : Transition _A
$t.r?.requestors \subseteq \text{dom } t.v.Trust$ $\text{LEAST } t.v.Trust(t.r?.requestors) \geq \text{MOST } t.v'.Trust(\text{altered})$ where $\text{altered} \triangleq \text{dom}(t.v'.Trust \setminus t.v.Trust)$

53. Single requestors that are at least loyal may directly alter the classification of an entity with a high water mark, though the new classification must dominate the entity's high water mark. Two or more requestors, who are all at least loyal, have complete freedom to directly alter the classification of any entity. Classifications may only be altered indirectly, that is copying or moving attributes to lower classified entities, if all the requestors are trusted.

SeparationOfDuty _{Class}
$\delta \text{Transition}_A$
$t.r?.requestors \subseteq \text{dom } t.v.Trust$ $\text{LEAST } t.v.Trust(t.r?.requestors) = \text{untrusted}$ $\rightarrow 1 \triangleleft t.v' = 1 \triangleleft t.v$ $\text{dom } t.v'.F = \text{dom } t.v.F$ $t.v'.Class = t.v.Class$ $\text{LEAST } t.v.Trust(t.r?.requestors) = \text{loyal}$ $\wedge \# t.r?.requestors \geq 2$ $\rightarrow 1 \triangleleft t.v' = 1 \triangleleft t.v$ $\text{LEAST } t.v.Trust(t.r?.requestors) = \text{loyal}$ $\wedge \# t.r?.requestors = 1$ $\rightarrow 1 \triangleleft t.v' = 1 \triangleleft t.v$ $t.v'.Class^{-1} ; (t.v.Class \bullet t.v.Hwm) \subseteq (_ _)$ $\text{GLB } t.v'.Class(\text{new}) \geq \text{highest}$ where $\text{highest} \triangleq \text{LUB } (t.v.Class \bullet t.v.Hwm)(\text{source})$ $1 \triangleq t.v.Class \text{ LOWER LUB } t.v.Class(t.r?.requestors)$

The axiom states that all of the requestors must be attributed with some degree of Trust. If any one of them is untrusted then no entity classified lower than the highest requestor may be modified, no entities may be created or destroyed and no classifications may be changed.

If none of the requestors are untrusted, but at least one is loyal, then classifications may be altered, though no entities classified lower than the highest requestor may be modified. If there is only one requestor, an entity's new classification must not be lower than its high water mark and the classification of new entities must dominate the highest high water mark of the observed data.

The axiom places no constraints on a request if all the requestors are trusted. Therefore trusted requestors may make arbitrary changes to classifications, by direct or indirect means.

Note that this version of separation of duty allows untrusted requestors to "write up" and a single loyal requestor to raise the classification of an entity.

54. If a new entity is given an ACL or an existing ACL is changed, none of the requestors can be untrusted. Note that just one loyal requestor is sufficient to alter an ACL, but such changes are themselves subject to an ACL.

SeparationOfDuty _{ACL}
t : Transition _A
$t.v'.ACL \setminus t.v.ACL \neq \{\}$ $\rightarrow t.r?.requestors \subseteq \text{dom } t.v.Trust$ $\wedge \text{LEAST } t.v.Trust(t.r?.requestors) \geq \text{loyal}$

The complete separation of duty axiom is given by the conjunction of the constraints on changing Id, Trust, Class and ACL. No axiom is given governing changes to Hwm because this records part of the history and will be fully specified. Changes to Other parts of the state are unconstrained. Changes to ReqClass and ReqId are not constrained because these have been introduced to implement the separation of duty constraints for changing classifications.

$$\text{SeparationOfDuty} \triangleq \text{SeparationOfDuty}_{Id} \wedge \text{SeparationOfDuty}_{Class} \\ \wedge \text{SeparationOfDuty}_{Trust} \wedge \text{SeparationOfDuty}_{ACL}$$

3.3 The Example Rules

The following set of rules will be used in section 5 to describe the security aspects of the μ SERCUS document handling system. The proof that a system based on these rules is secure has not been completed, but it is hoped that this will be relatively straightforward.

In this example three rules are given for creating entities, two for gaining attributes and three for downgrading entities. These will be combined with the specification of functionality in section five to give an overall specification of the system.

The following schema defines various sets of entities regarding a rule. This will be used in following schemas to make the specification more concise. Those entities observed by the request are given by the set observed. The set new gives those entities created by the request. Those entities which have gained or lost attributes, including entities which are created or destroyed, are given by the set changed. The set source gives those entities which are the source of influence over the request, that is those observed and the requestors.

The schema also insists that there is just one requesting entity and that the request is valid. The requestor's classification must dominate the classification of all observed entities and the requestor must be mentioned in the access control lists of all accessed entities.

ΔV_A

$r? : R_A$
 $v, v' : V_A$
observed, new, changed, source : P E
requestor : E

observed = dom $r?.request.F$
new = dom($v'.F$) \ dom($v.F$)
changed = dom($v' \setminus v$) U dom($v \setminus v'$)
source = observed U $r?.requestors$

 $r?.requestors = \{ requestor \}$
 $requestor \in \text{dom } v.F$
 $r?.request \subseteq v.F$

 $v.Class(requestor) \geq \text{LUB } v.Class(observed)$
 $observed \cup changed \subseteq v.Acl^{-1}(\{ v.Id(requestor) \})$

The next schema will be used when high water marks are altered. This specifies that any entity which is changed, but not destroyed, has its high water mark raised to the level of the source of influence. Note that new entities are given the appropriate high water mark.

ΔHWM_A

ΔV_A

$v'.Hwm = \text{dom } v'.F \triangleleft (v.Hwm \text{ LUB } (\text{changed RELATES } \{ \text{level} \}))$
where
 $\text{level} \triangleq \text{LUB } (v.Class \bullet v.Hwm)(\text{source})$

The next schema will be used when high water marks do not change, unless the entity is destroyed, and new entities are not given a high water mark.

$\exists HWM_A$

ΔV_A

$v'.Hwm = (\text{dom } v'.F) \triangleleft v.Hwm$

Create Entities

Creation of entities can be detected by other active entities. However the separation of duty requirements state that the system is secure if the creation is performed by individuals, or rather their faithful proxies, who are loyal or trusted. Three forms of CreateEntities are given because of the differing constraints imposed by separation of duty on classifying new entities and slightly different requirements.

The first rule states that the requestor must be trusted and no existing entities are changed. The new entities may only be given attributes taken from the entities mentioned in the request, because this rule is concerned with creating entities only and excludes creating attributes. The new entities are not given a high water mark, but must be given some identity, classification, access control list entries and a degree of trust. However, these are not fully specified because a trusted requestor is free to chose any values for them.

CreateEntities_{TRUSTED}

ΔU_A
 $\exists HWM_A$

```
v.Trust( requestor ) = trusted  
  
dom( v.F )  $\Join$  v' = v  
rng( dom( v.F )  $\Join$  v'.F )  $\subseteq$  v.F( observed )  
  
new  $\subseteq$  dom v'.Id  
new  $\subseteq$  dom v'.Class  
new  $\subseteq$  dom v'.Acl  
new  $\subseteq$  dom v'.Trust
```

The create entities rule for loyal requestors is basically the same as for trusted requestors, except that the new entities are ascribed neither a degree of trust nor an identity. They are however given the classification of the requestor and the requestor is included in their access control lists. They are also given a suitable high water mark.

CreateEntities_{LOYAL}

ΔU_A
 ΔHWM_A

```
v.Trust( requestor ) = loyal  
rng( dom( v.F )  $\Join$  v'.F )  $\subseteq$  v.F( observed )  
  
v'.Trust = v.Trust  
v'.Id = v.Id  
v'.Class = v.Class  $\cup$  new RELATES { v.Class(requestor) }  
v'.Acl = v.Acl  $\cup$  new RELATES { v.Id(requestor) }  
v'.ReqClass = v.ReqClass  
v'.ReqId = v.ReqId  
{new}  $\Join$  v'.Other = v.Other
```

The final create entities rule is similar to CreateEntities_{LOYAL} except that the new entities may gain attributes which did not exist anywhere in the state beforehand and they are not given a high water mark. This is called CreateEntities_{CAP} because, as explained in the next section, it is used when new capabilities are created for the new entities.

CreateEntities_{CAP}

ΔU_A
 $\exists HWM_A$

```
v.Trust( requestor ) = loyal  
rng( dom( v.F )  $\Join$  v'.F )  $\cap$  (rng v.F \ v.F( observed )) = {}  
  
v'.Trust = v.Trust  
v'.Id = v.Id  
v'.Class = v.Class  $\cup$  new RELATES { v.Class(requestor) }  
v'.Acl = v.Acl  $\cup$  new RELATES { v.Id(requestor) }  
v'.ReqClass = v.ReqClass  
v'.ReqId = v.ReqId  
{new}  $\Join$  v'.Other = v.Other
```

Gain Attributes

There are two versions of this rule, one for loyal requestors and one for untrusted requestors. The difference is that loyal entities have no high water mark.

For a loyal requestor the structure of the initial state is preserved, in that the only change possible is for the requestor to gain attributes from the observed entities.

GainAttributes _{LOYAL}
ΔV_A ΔHWM_A
<pre> v.Trust(requestor) = loyal requestor $\in$ dom v.Hwm changed = {requestor} rng(v'.F \ v.F) $\subseteq$ v.F(observed) v $\subseteq$ v' v'.Acl = v.Acl v'.Id = v.Id v'.Trust = v.Trust v'.Class = v.Class v'.ReqId = v.ReqId v'.ReqClass = v.ReqClass </pre>

The same is true for an untrusted requestor, except that its high water mark is updated.

GainAttributes _{UNTRUSTED}
ΔV_A ΔHWM_A
<pre> v.Trust(requestor) = untrusted requestor $\in$ dom v.Hwm changed = {requestor} rng(v'.F \ v.F) $\subseteq$ v.F(observed) v'.Acl = v.Acl v'.Id = v.Id v'.Trust = v.Trust v'.Class = v.Class v'.ReqId = v.ReqId v'.ReqClass = v.ReqClass </pre>

Downgrade

Downgrading usually requires a request made by two requestors, but this is supported by two rules each made by one loyal requestor.

This rule allows a loyal requestor to request that an entity be downgraded. The only changes to the state are that the requested classification and identity of the requestor are added to some entities. These entities cannot already have a downgrade request outstanding.

RequestDowngrade

ΔU_A
 $\exists HWM_A$

```
v.Trust( requestor ) = loyal

dom v'.F = dom v.F
v'  $\subseteq$  v'
v'.Class = v.Class
v'.Acl = v.Acl
v'.Id = v.Id
v'.Trust = v.Trust
v'.Other = v.Other

changed = dom v'.ReqClass \ dom v.ReqClass
v.Class-1 ; (v'.ReqClass \ v.ReqClass)  $\subseteq$  ( _≥_ )
v'.ReqId = v.ReqId  $\cup$  changed RELATES { v.Id( requestor ) }
```

A downgrade may occur only if it has already been requested. Separation of duty is enforced because the requestor performing the downgrade cannot also have requested it. Once the entities have been downgraded, the request to downgrade them is removed. Other elements of the state remain unchanged.

PerformDowngrade

ΔU_A
 $\exists HWM_A$

```
v.Trust( requestor ) = loyal

dom v'.F = dom v.F
v'  $\subseteq$  v
v'.Id = v.Id
v'.Trust = v.Trust
v'.Acl = v.Acl
v'.Other = v.Other

changed = dom v.ReqClass \ dom v'.ReqClass
changed  $\subseteq$  dom v.ReqId
v'.Reqid = changed  $\Join$  v.ReqId

v.Id( requestor )  $\notin$  v.ReqId( changed )

(changed  $\Join$  v'.Class)-1 ; (changed  $\Join$  v.ReqClass) = id CLASS
```

A third form of downgrade allows entities with high water marks to be downgraded by a single loyal requestor, as long as the new classification dominates the high water mark.

DowngradeHwm

ΔV_A
 $\exists HWM_A$

$v.Trust(requestor) = loyal$

$dom\ v'.F = dom\ v.F$

$changed\ \&\ v'.Class = changed\ \&\ v.Class$

$v'.Hwm = v.Hwm$

$v'.Id = v.Id$

$v'.Trust = v.Trust$

$v'.Acl = v.Acl$

$v'.ReqClass = v.ReqClass$

$v'.ReqId = v.ReqId$

$v'.Other = v.Other$

$(changed\ \&\ v.Class)^{-1} ; (changed\ \&\ v'.Class) \subseteq (_ \geq _)$

$(changed\ \&\ v'.Class)^{-1} ; (changed\ \&\ v.Hwm) \subseteq (_ \geq _)$

4. Implementation & Assurance

The security policy model is described in terms of entities and attributes, the relationship between them and a set of state transition rules. An entity is an abstraction of a discrete object that can be altered, while an attribute is one that is unchanging. The state relation between entities and attributes, F , is an abstraction of entities "containing" or "addressing" attributes. The transition rules are an abstraction of programs or procedures which observe and alter this relationship. A request is an abstraction of the parameter passing and address translation mechanisms used to invoke these programs.

4.1 Implementation on a Conventional Architecture

The model assumes that all entities are universally visible, that is an entity can know of the existence of all others, but they can only be observed and altered by invoking a transition rule. Thus the concrete realisation of an entity must be able to hide and protect its contents from other entities, but not from transition rules. This can be implemented using a simple two state machine.

The state transition rules would be implemented as supervisor calls and the mapping between entities and attributes would be implemented as data structures accessible only in supervisor state. An active entity would be implemented by a process running in user state. It would be able to "name" entities and attributes using memory addresses or indices, but is prevented from observing or modifying them by the memory management system.

For example, a secure filing system could be implemented by storing passive entities on disc. Such "files" may only be observed and altered by transition rules implemented as device drivers running in supervisor state. Attributes such as file names may be stored directly in the memory of the active entity. Transition rules which allow files to be read, written, created and deleted would be implemented as device driver functions which take a file name as a parameter. However transition rules that manipulate file names would simply be implemented by the instructions of the machine.

To model the notion of "opening" a file to gain a "file control block", which keeps context information that controls sequential file accesses, a class of entities could be introduced to represent open files. These would contain attributes, protected by supervisor state, which keep the file's address on disc and describe the current position in it. Note that although open files are entities, and hence are accessible to all entities in the system, the protection facilities of the system will probably only allow their "owner" to use them.

While it is obviously possible to build a secure system, modelled using this approach, on a two state machine this is not the most sympathetic architecture. Many advantages are to be gained by using a ring based protection system, in particular this allows some structure to be imposed on the large amount of supervisor state code. However, the use of certain capability protection schemes allows a much finer structuring to be imposed on both the system and user software, to the point of removing the distinction between the two.

4.2 Implementation using Capability Addressing

The most fundamental property of capability addressing is that capabilities are the only means of addressing an object and they cannot be forged. That is scalar data cannot be treated as capabilities and a capability cannot be altered so it refers to a different object. Capabilities provide a uniform, system wide method of addressing[†], even extending to distributed systems [Foster&Currie86] and backing store [Wiseman88], which greatly eases the task of software construction and reuse [Stanley85]. In addition, the use of capabilities make it possible to rely on low level automatic "garbage collection" [Wiseman85] to recover unused objects, which relieves all software of the burden of memory management.

[†]with the notable exception of the Cambridge CAP Computer which has a unique hierarchy of address spaces.

Capabilities can also be used as the means of addressing objects by the users of the system, as has been demonstrated with the Flex system [Foster et al. 82]. The traditional approach of using names for files is often inappropriate, as seen by the rise in popularity of icon based systems, and as an implementation technique it is prone to attack by viruses [Wiseman 88].

In using such a system to implement the security policy model, entities would be represented by objects addressed by capabilities. Attributes would either be scalar data, such as integers, or further objects addressed by capabilities. However entities and attributes require some protection, and simple capability based addressing offers none.

There are two requirements of this protection. First it must be possible to distinguish between capabilities for entities and attributes and those for arbitrary data structures which are not being modelled. Second it must be possible to restrict access to entities and attributes to software that implements transition rules. Both requirements can be satisfied by utilising sealed objects [Redell 74].

In the simplest form of this mechanism two kinds of capability are distinguished, sealed and unsealed, and a seal is attached to each object. Using an unsealed capability for an object gives full access to it, but a sealed capability gives no access. An unsealed capability may be converted into a sealed version at any time, but a sealed capability may only be converted into an unsealed version if the object's seal can be presented. Note that the seal is unforgeable if it is a capability which is kept secret.

Entities and attributes will therefore be implemented as sealed capabilities for objects whose seal is known only to the transition rule's code. It is possible to distinguish between them and arbitrary capabilities by attempting an unseal operation. This will fail to unseal arbitrary capabilities because the seal is bound to be wrong.

A prerequisite for protecting data with sealed capabilities is that the seals themselves must be protected from theft and misuse. The requirement is in fact that the seals used for entities and attributes are only available to the transition rules when they are invoked. In particular, software which invokes the rules must not be able to extract the seals from the rule. Protection of this form is offered by closures [Landin 64]. These are bindings of code with data that is only available when the code is executed and are in effect first class procedures [Currie 82].

The machine must distinguish closures from other objects, so that it can ensure that they may only be called and not read or written. This could be implemented using a seal which is known only to the hardware, though it is more convenient to distinguish between the different kinds of object directly by giving each object a type.

It will be convenient if active entities can be implemented directly as processes, however they must be prevented from making arbitrary alterations to their attribute relationships. These relationships must, however, be accessible to the transition rules, for example to allow the clearance to be inspected. The transition rules cannot rely on the process to pass on such information, and in practice this would be far too inconvenient anyway. The requirement is for context information to be attached to processes, which is only accessible to transition rules. This control is provided by associating a seal with an item of the context information, so that it can only be retrieved or altered if the seal is known.

It would be possible to use just one seal for all entities and attributes, though there is no reason why different seals should not be used for the different types of entities and attributes found in an application. Such a practice would allow the development of the transition rules concerned with each type to proceed independently and would limit damage if mistakes did occur in the implementation.

The model's requirement for protection therefore leads to a machine with capability addressing that distinguishes between four types of object; data, sealable, closure and process. Possession of a capability for a data object allows it to be read and written. A capability for a sealable object either gives no access or full access depending on whether the capability is sealed or unsealed. Capabilities for closures only allow the object to be called. Capabilities for process objects allow other processes to have some control over their offspring but are not used for communication.

The SMITE capability computer provides these protection mechanisms within a high level language oriented instruction set [Wiseman88], though it is equally possible to incorporate them into a reduced instruction set architecture [Wiseman89]. To see how it is possible to provide these mechanisms using computer architectures with conventional addressing their relationship with strongly typed programming languages must be considered.

4.3 Implementation using Compile Time Protection

Programming languages provide protection through their scope rules, that is variables which are not in scope cannot be accessed. This is directly equivalent to the use of capability addressing, where objects cannot be accessed unless a suitable capability is possessed. The type rules of a language prevent inappropriate operations being applied to data, for example procedures may only be called. The typing provided by the capability computer hardware is much more limited, there being only four types of object and two kinds of data, but it too prevents inappropriate usage.

Thus if a programming language has a strong type system which pervades the entire system and provides the necessary protection features, it will be possible to implement the security model using it. If the language can be compiled to run efficiently on conventional computer architectures then a highly portable secure system is the result.

The compiler solution to protection has been used quite successfully before, eg. the Burroughs B6700, but it has suffered from assurance problems, which are discussed further in the next section. Also it is necessary to ensure that all software is ruled by the strong type system, which means the type system must be sufficiently powerful to allow all operating system and application software to be implemented.

A system with these properties is called Ten15 [Foster89]. This is an algebraically defined abstraction of a computer which acts very much like an intermediate language. However it has a powerful type system, including polymorphism, abstract types, universal union of all types and types which describe persistent and remote objects, that is those on disc and in other machines. Compilers output Ten15 'code' which is then translated, by applying a homomorphism, into the target machine code for execution. The resulting program is not interpreted and most checks, such as array bounds checking, can be made at compile time. Compilers for Algol68, Pascal and C exist and Ada is under development. Translators for VAX and Flex, on which SMITE is based, are currently available and others are under development.

The environment in which Ten15 executes may be implemented on a bare machine or may exist on top of an existing operating system, allowing peripheral driving software to be reused. In the latter case the correctness of the overall system obviously depends upon the correctness of the underlying operating system. For example the type system may be circumvented if an error in the disc driver causes the wrong data file to be read, which is fatal because even a small flaw in the type system will allow the security mechanisms to be bypassed.

The security of the type system therefore depends upon the correctness of the underlying operating system, not its ability to uphold partial properties like confidentiality. Therefore the use of a security enhanced version is of no benefit. However most operating systems are reasonably correct and can be relied upon to access the right file. Therefore if only moderately low assurance of overall security is required, an off the shelf processor and operating system can be used as the basis for the Ten15 abstract machine.

The implementation of a Ten15 system on bare hardware requires some operating system software to be written. However this would be relatively small and tailored to the task. It should therefore be possible to produce a system with much higher assurance that the type system is correct, though at extra expense.

The problem with this approach is that signalling channels and denial of service threats arise through the heap management system on which the abstract machine is based. These problems can be fixed [Wiseman88], but the solution requires hardware support to run efficiently. Using a capability architecture as the vehicle for Ten15 allows hardware based garbage collection to be implemented and also provides an orthogonal layer of checks which guard against errors in the Ten15 translator. There are also tradeoffs between translator and hardware complexity. The use of a high level language architecture like SMITE means that relatively simple translators are required at the expense of complex microcode, whereas a RISC architecture would require much more complex translators but no microcode.

4.4 Formal Verification

Having modelled security as axioms regarding confidentiality, discretionary access control and integrity, it is necessary to prove properties such as data confinement and that the transition rules chosen as the basis of implementation uphold these axioms. Note, there is no inductive proof akin to that of the "Orange Book" Bell-LaPadula model, because the axioms all concern state transitions and no definition of a secure state is given.

Data confinement properties, for example that the contents of a secret message will never be seen except by those individuals cleared to secret, may hopefully be established using a proof of non interference. For practical systems this proof will fail, but in doing so it will reveal the channels responsible for the failure. These will include legitimate channels, such as properly authorised downgrading, and acceptable covert channels, such as loyal users signalling by creating entities.

The proof that the rules uphold the axioms is relatively straightforward, though it will involve showing that the separation of duty requirements are met by the appropriate sequence of single requestor transitions.

In addition to these proofs it is necessary to establish that the initial state is appropriate. Note that this is not the notion of a secure initial state, since in this model it is transitions that are secure, not states. What is of concern is the integrity of a state. That is the integrity of the security labels, identity attributes, trustworthiness ascribed to entities, etc. must be shown for the initial state. This of course is not something that can be proven.

For the highest assurance, it is necessary to prove the correspondence between a design and its implementation, which occurs at several levels. The underlying assumptions made by the model about execution must be shown to be valid, the implementation of the rules must be correct and the correctness of the system's external interfaces must be established.

Firstly, it must be shown that entities and attributes are discrete, in that altering one entity does not have a side effect on another. For example, this would be the case if two entities were implemented using some common storage which was updated. It must also be shown that the relation between entities and attributes can only be observed and altered by the transition rules. In terms of a capability based implementation this involves showing that the microcode provides unforgeable capabilities, closures that can only be called and a safe sealing mechanism. Then it must be shown that the rules do not compromise the seals.

It appears that much can be achieved in this area using program analysis, as both the microcode and the Ten15 abstract machine are defined algebraically. Since capabilities cannot be forged they can only be passed around by copying them, it is not possible to transmit capabilities through signalling channels. Thus information flow analysis of capabilities will not be as pessimistic as for scalar data, and hence it is expected that meaningful results can be obtained.

Secondly the rules must be correctly implemented, in that they must perform the appropriate checks. This is the main reference monitor function of rejecting illegal requests. The technique of program refinement is of use here.

Thirdly the external interfaces must be correct so that the world modelled inside the computer faithfully represents the real world outside. This covers correct authentication of users and the correct implementation of the trusted path as well as appropriate labelling of output. Again, the technique applicable is program refinement.

At present, none of the proofs, refinements or analyses have been carried out. The next stage of the SMITE research programme is to carry these out for the SERCUS demonstration.

5. Specification of μ SERCUS

Section 3 refined the state components of F to distinguish certain security related attributes and defined some rules which ensure that the security of other data is always maintained. F is now refined further to introduce the attributes and entities required to carry out the particular application, in this case μ SERCUS.

5.1 Application Specifics

In μ SERCUS users may create, open and downgrade classified documents. The contents of a document can never be altered once it is created. This mimics a real world system in which writing in the margins and correcting fluids are banned. Documents are created from other objects, called variables, whose contents can alter.

High water marks will only be used for variables, allowing users to enter information below their clearance. A variable would be downgraded to the correct classification before producing a document from it. Documents can only be downgraded with the consent of two individuals.

The users of the system interact with it through display terminals. The display presents an environment in which documents and variables may be stored and manipulated. In effect displays are proxies of the users.

Documents, variables and displays must be represented by entities, since their classification or contents can be altered.

Document, Variable, Display : P E

Before users can use the system an agent of the system's owners must check that they are authorised. This is essentially a "login process" which checks the user's name and password and creates a suitable Display. This entity, called welcome because the system is user friendly, is a proxy of the system's owners and hence is trusted.

welcome : E

The documents and variables of μ SERCUS are named by simple capabilities. These are attributes because they never change. Other unalterable objects, such as characters which make up the text of documents and procedures taken from a non-overwriting backing store, are also attributes which are effectively just constants.

DocCap, VarCap, Constant : P A

The state, F, can now be refined to define how these application oriented entities and attributes are related.

U	
U _A	
Var	: Variable $\leftrightarrow$ VarCap
Doc	: Document $\leftrightarrow$ DocCap
Contents	: E $\leftrightarrow$ A
Doc U Var U Contents $\subseteq$ F	
dom Var U dom Doc U dom(Display $\triangleleft$ F) = dom Contents	
Contents(Document) $\subseteq$ Constant U DocCap	

The relation Var gives the set of capabilities that can be used to access a Variable. Variables represent complex objects made up of arbitrary graphs of capabilities for primitive objects. Since such a graph may have more than one way in it is possible to have more than one capability refering to the same Variable, and hence a set is required. If untrusted software is given access to two Variables it may arrange for their two graphs to be joined. Therefore, assuming the worst, it is necessary to consider the two Variables as one in future.

Doc is a function that uniquely defines the capability for each document that currently exists. The relation Contents gives the contents of all the variables, displays and documents. Documents may only contain constants and capabilities for documents. Variables and Displays may contain anything.

As before it is assumed that all operators and schemas have been refined to use this new definition of the state.

5.2 The Specification

In the initial state only welcome exists. It is trusted and has a suitable ACL. It also has an Id and Class though these are undefined.

InitialState	
U	
dom F = {welcome} dom Class = {welcome} dom Id = {welcome} Trust(welcome) = trusted Acl(welcome) = {welcome} Hwm = {} ReqId = {} ReqClass = {} Other = {}	

5.2.1 Login

The first operation to be specified logs a user onto the system. This is instigated by the welcome entity once it has validated the login request. The operation creates a new display entity which will act for the user.

Login₀

r? : R
v, v' : V

id? : ID
class? : CLASS

display! : display

r?.requestors = {welcome}
r?.request.F = { welcome } Δ v.F

display! $\in$ dom v.F
dom v'.F = dom v.F $\cup$ { display! }
id? $\in$ rng Id

v'.Id(display!) = id?
v'.Class(display!) = class?
v'.Acl({display!}) = { id? }
v'.Trust(display!) = loyal
display! $\in$ dom(v'.ReqClass $\cup$ v'.ReqId $\cup$ v'.Hwm $\cup$ v'.Other)
{display!} Δ v' = v

So Login can be fully specified as

Login $\triangleq$ CreateEntities_{TRUSTED} $\wedge$ Login₀

5.2.1 Open Existing Document

The operation to open a document returns the contents of the document to the requestor's display.

Open_document₀

r? : R
v, v' : V

doc? : Document
display? : Display

display? $\in$ dom v.F
r?.requestors = {display?}
r?.request.F = { doc? $\mapsto$ Doc(doc?) }
Doc(doc?) $\in$ v.Contents({display?})

v'.Contents = v.Contents
 $\cup$ {display?} RELATES v.Contents({doc?})

v'.Class = v.Class
v'.Id = v.Id
v'.Trust = v.Trust
v'.Acl = v.Acl
v'.ReqClass = v.ReqClass
v'.ReqId = v.ReqId
v'.Doc = v.Doc
v'.Var = v.Var

OpenDocument $\triangleq$ GainAttributes_{LOYAL} $\wedge$ Open_document₀
 $\vee$ GainAttributes_{UNTRUSTED} $\wedge$ Open_document₀

5.2.3 Create New Document

Documents are created from variables by copying their contents, though they must not include any capabilities for variables.

Create_document₀

$r? : R$
 $v, v' : V$

$display? : Display$
 $var? : Variable$
 $class? : CLASS$

$doc! : Document$
 $cap! : DocCap$

$display? \in \text{dom } v.F$
 $r?.requestors = \{display?\}$
 $r?.request.F = \{var? \mapsto v.Var(var?) \}$

$v.Var(var?) \in v.Contents(\{display?\})$
 $v.Contents(\{var?\}) \mapsto VarCap = \{ \}$

$doc! \notin \text{dom } v.F$
 $\text{dom } v'.F = \text{dom } v.F \cup \{doc!\}$
 $cap! \notin \text{rng } v.F$

$v'.Class = v.Class \cup \{ doc! \mapsto class? \}$
 $doc! \notin \text{dom } v'.Hwm$
 $v'.Id = v.Id$
 $v'.Trust = v.Trust$
 $v'.Acl = v.Acl \cup \{ doc! \mapsto v.Id(display?) \}$
 $v'.ReqClass = v.ReqClass$
 $v'.ReqId = v.ReqId$
 $v'.Doc = v.Doc \cup \{ doc! \mapsto cap! \}$
 $v'.Var = v.Var$
 $v'.Contents = v.Contents \cup \{doc!\} \text{ RELATES } v.Contents(\{var?\})$

CreateDocument $\triangleq$ CreateEntities_{cap} $\wedge$ Create_document₀

5.2.4 Downgrading

There are three operations concerning downgrading. The first requests that a document be downgraded, the second carries out the downgrade and the third downgrades draft documents, ie. variables.

RequestDowngrade₀

$r? : R$
 $v, v' : V$

$display? : Display$
 $doc? : Document$
 $class? : CLASS$

$display? \in \text{dom } v.F$
 $r?.requestors = \{display?\}$
 $v.Doc(doc?) \in v.Contents(\{display?\})$
 $r?.request.F = \{ display? \mapsto v.Doc(doc?), display? \mapsto class? \}$

$doc? \in \text{dom } v.ReqId$
 $doc? \in \text{dom } v.ReqClass$
 $v'.ReqId = v.ReqId \cup \{ doc? \mapsto v.Id(display?) \}$
 $v'.ReqClass = v.ReqClass \cup \{ doc? \mapsto class? \}$

$v'.Class = v.Class$
 $v'.Id = v.Id$
 $v'.Trust = v.Trust$
 $v'.Acl = v.Acl$
 $v'.Doc = v.Doc$
 $v'.Var = v.Var$
 $v'.Contents = v.Contents$

$\text{RequestDowngrade} \triangleq \text{RequestDowngrade}_{\text{LOYAL}} \wedge \text{RequestDowngrade}_0$

PerformDowngrade₀

$r? : R$
 $v, v' : V$

$display? : Display$
 $doc? : Document$

$display? \in \text{dom } v.F$
 $r?.requestors = \{display?\}$
 $v.Doc(doc?) \in v.Contents(\{display?\})$
 $r?.request.F = \{ doc? \mapsto v.Doc(doc?) \}$

$doc? \in \text{dom } v.ReqId$
 $doc? \in \text{dom } v.ReqClass$
 $v'.ReqId = \{doc?\} \uplus v.ReqId$
 $v'.ReqClass = \{doc?\} \uplus v.ReqClass$
 $v'.Class = v.Class \bullet \{ doc? \mapsto v.ReqClass(doc?) \}$

$v'.Id = v.Id$
 $v'.Trust = v.Trust$
 $v'.Acl = v.Acl$
 $v'.Doc = v.Doc$
 $v'.Var = v.Var$
 $v'.Contents = v.Contents$

$\text{PerformDowngrade} \triangleq \text{PerformDowngrade}_{\text{LOYAL}} \wedge \text{PerformDowngrade}_0$

RegradeVariable₀

r? : R
v, v' : V

display? : Display
var? : Variable
class? : CLASS

display? ∈ dom v.F
r?.requestors = {display?}
v.Var(var?) ∈ v.Contents({display?})
r?.request.F = { var? ↦ v.Var(var?) }
class? ≥ v.Hwm(var?)
v'.Class = v.Class • { var? ↦ class? }

v'.Id = v.Id
v'.Trust = v.Trust
v'.Acl = v.Acl
v'.ReqClass = v.ReqClass
v'.ReqId = v.ReqId
v'.Doc = v.Doc
v'.Var = v.Var
v'.Contents = v.Contents

RegradeVariable • DowngradeHwm ∧ RegradeVariable₀

5.2.5 Other Operations

Other operations have been omitted but are similar to those which have been given. An operation that allows untrusted displays to be created and destroyed is required. This effectively models running untrusted software in another window, controlled from the trusted path. Obviously logout is also required, as are operations for creating, editing and destroying variables.

The complete SERCUS demonstration includes files of documents, full journalling facilities, mail messages and private storage facilities (the equivalent of cupboards in the real world). The existing SERCUS specification is written as a conglomeration of security and functionality requirements [Harrold88] and this is currently being recast in terms of this model.

5.2.6 Summary

This section has specified the functionality of the μSERCUS application independently of the security aspects. The total specification was given by using the Schema Calculus of Z to combine this with a specification of security given as transition rules.

6. Conclusions

In this paper a new security model has been presented which suggests that confidentiality is solely about the observation of classified material and that the integrity of the labelling upon which it is based is a separate concern. Inspired by [Clark&Wilson87], separation of duty is proposed as the mechanism for ensuring that the integrity of labels are preserved. Similar proposals are made for the access control lists upon which discretionary access control is based.

Strict separation of duty is expressed as the requirement that a change can only happen if all individuals want it to occur, though this is obviously too strong in practice. Also requests for change are made by software proxies acting on behalf of individuals and varying degrees of trust are placed in them. This reflects the degree of trust the system's owners have in the individual and whether the proxy is being commanded via the trusted path. A practical separation of duty requirement will therefore state the number of individuals and their trustworthiness needed to allow a change.

The model is based upon a state machine with security defined by axioms on transitions which define confidentiality, separation of duty and discretionary access control. No axioms are given for states, though the system must start in a state which is appropriate, that is one in which labels etc. have integrity.

To demonstrate the modelling approach the specification of a simple document handling system has been given. In this system the downgrading of documents is subject to separation of duty in that one individual must request a downgrade and a different individual must authorise it before the change occurs.

It is shown that the model can be implemented using a two state or ring based protection architecture, but a system utilising strong typing with underlying capability addressing is proposed as the ideal solution.

References

- D.E.Bell
Secure Computer Systems: A Refinement of the Mathematical Model
Mitre Corp. MTR-2547, Vol 3
April 1974
- D.E.Bell & L.J.LaPadula
Secure Computer Systems: Mathematical Foundations
Mitre Corp. MTR-2547, Vol 1
November 1973
- D.E.Bell & L.J.LaPadula
Secure Computer Systems: A Mathematical Model
Mitre Corp. MTR-2547, Vol 2
November 1973
- D.E.Bell & L.J.LaPadula
Secure Computer Systems: Unified Exposition and Multics Interpretation
Mitre Corp. MTR-2997
March 1976
- D.D.Clark & D.R.Wilson
A Comparison of Commercial and Military Computer Security Policies
Procs. 1987 IEEE Symp on Security and Privacy
Oakland CA., April 1987, pp184..194
- I.F.Currie
In Praise of Procedures
RSRE Memorandum 3499
July 1982
- J.M.Foster
The Algebraic Specification of a Target machine: Ten15
from "High Integrity Software", C.T.Sennett (Ed.), Pitman Press
to appear 1989
- J.M.Foster & I.F.Currie
Remote Capabilities in Computer Networks
RSRE Memorandum 3947
March 1986
- J.M.Foster, I.F.Currie & P.W.Edwards
Flex: A Working Computer with an Architecture Based on Procedure Values
RSRE Memorandum 3500
July 1982
- C.L.Harrold
Formal Specifcation of a Secure Document Control System for SMITE
RSRE Report 88002
February 1988
- P.Karger
Implementing Commercial Data Integrity with Secure Capabilities
Procs. 1988 IEEE Symp on Security and Privacy
Oakland CA., April 1988, pp130..139
- P.J.Landin
The Mechanical Evaluation of Expressions
Computer Journal, Vol 6, Num 4
January 1964
- C.Morgan, K.Robinson & P.Gardiner
On the Refinement Calculus
Oxford Univ. Programming Research Group Draft Report
April 1988
- D.D.Redell
Naming and Protection in Extendible Operating Systems
MIT Technical Report MAC-TR-140
November 1974

- J.M. Spivey
The Z Notation: A Reference Manual
Programming Research Group, Oxford Univ., Draft JMS-87-12a
1987
- M. Stanley
The Use of Values without Names in a Programming Support Environment
RSRE Memorandum 3901
November 1985
- P.F. Terry & S.R. Wiseman
On the Design and Implementation of a Secure Computer System
RSRE Memorandum 4188
June 1988
- S.R. Wiseman
On the Garbage Collection of Block Structured Memories
RSRE Report 85006
May 1985
- S.R. Wiseman
A Secure Capability Computer System
Procs. 1986 IEEE Symp on Security and Privacy
Oakland CA., April 1986, pp86..94
- S.R. Wiseman
Protection and Security Mechanisms in the SMITE Capability Computer
RSRE Memorandum 4117
January 1988
- S.R. Wiseman
The SMITE Object Oriented Backing Store
RSRE Memorandum 4147
March 1988
- S.R. Wiseman
Garbage Collection in Distributed Systems
PhD Thesis, Univ. Newcastle upon Tyne
RSRE Report to appear 1989
- S.R. Wiseman
SMITE on a Chip
RSRE Report to appear 1989
- S.R. Wiseman
Causing and Preventing Viruses in Computer Systems
RSRE Report to appear 1989
- S.R. Wiseman, P.F. Terry, A.W. Wood & C.L. Harrold
The Trusted Path between SMITE and the User
Procs. 1988 IEEE Symp on Security and Privacy
Oakland CA., April 1988, pp147..155
- J.P.L. Woodward
Exploiting the Dual Nature of Sensitivity Labels
Procs. 1987 IEEE Symp on Security and Privacy
Oakland CA., April 1987, pp23..30

Appendix A: The Z Notation

In the following, x is an identifier, T is a type, P and Q are predicates, S is a set and R is a relation.

$LHS \triangleq RHS$	LHS is syntactically equivalent to RHS
$x : T$	declare x as type T
$P \wedge Q$	P and Q
$P \vee Q$	P or Q
$P \rightarrow Q$	P implies Q
$x \in S$	x is an element of set S
$S_1 \subseteq S_2$	set S_1 is included in set S_2
$\{\}$	the empty set
$\{x_1, x_2, \dots, x_n\}$	the set containing $x_1, x_2, \dots, x_n$
$P S$	powerset: the set of all subsets of S
$S_1 \cap S_2$	set intersection
$S_1 \cup S_2$	set union
$S_1 \setminus S_2$	set difference
$\#S$	size of finite set
$T_1 \leftrightarrow T_2$	the set of relations from T_1 to T_2
$T_1 \rightarrow T_2$	the set of total functions from T_1 to T_2
$\text{dom } R$	the domain of a relation R
$\text{rng } R$	the range of a relation R
$R_1 ; R_2$	forward relational composition
R^{-1}	inverse of a relation R
$\{a \mapsto b, c \mapsto d, \dots\}$	the relation mapping a to b , c to d ,
$R(S)$	relational image of set S through relation R
$S \triangleleft R$	domain restriction of relation R to set S
$S \dashv R$	domain subtraction
$S \triangleright R$	range restriction
$S \triangleright R$	range subtraction

The schema notation is a way of grouping together some variable declarations and a predicate that relates them.

EG	This schema is called EG. It declares a variable x which is drawn from the set S and a function f , from S to S . The predicate states that x and f must be such that f maps x to itself.
$\begin{array}{l} x : S \\ f : S \rightarrow S \end{array}$	
$f(x) = x$	

A schema may be included in the declarations of another, in which case the declarations of the two schemas are merged together and the predicates are conjoined.

OP	Identifiers may be decorated. By convention a dashed variable indicates the state of a variable after an operation. Thus in the schema OP f' is the function resulting from changing f .
$\begin{array}{l} f, f' : S \rightarrow S \\ x, y : S \end{array}$	
$f' = f \bullet \{x \mapsto y\}$	

DOCUMENT CONTROL SHEET

Overall security classification of sheet UNCLASSIFIED

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference Memorandum 4222	3. Agency Reference	4. Report Security Classification Unclassified	
5. Originator's Code (if known) 7784000	6. Originator (Corporate Author) Name and Location Royal Signals and Radar Establishment St Andrews Road, Malvern, Worcestershire WR14 3PS			
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location			
7. Title A SECURITY MODEL AND ITS IMPLEMENTATION				
7a. Title in Foreign Language (in the case of translations)				
7b. Presented at (for conference papers) Title, place and date of conference				
8. Author 1 Surname, initials Wiseman S R	9(a) Author 2 Harrold C L	9(b) Authors 3,4...	10. Date 1988.9	pp. ref. 35
11. Contract Number	12. Period	13. Project	14. Other Reference	
15. Distribution statement Unlimited				
Descriptors (or keywords)				
continue on separate piece of paper				
Abstract A new security model is proposed which allows the notions of confidentiality and integrity to be expressed in one coherent framework. Confidentiality is taken to be solely concerned with the observation of classified information, while separation of duty is employed as the technique for assuring the integrity of the security labels which are the basis of confidentiality. Discretionary access control is modelled by access control lists whose integrity is also provided by separation of duty controls. The specification of a simple system is given as an example of the techniques and methods for implementation are discussed.				