

AD-A059 728

MITRE CORP BEDFORD MASS
CAUSAL SYSTEM SECURITY.(U)

F/G 9/2

UNCLASSIFIED

OCT 78 J K MILLEN

F19628-78-C-0001

MTR-3614

ESD-TR-78-171

NL

| OF |
AD
A059728



END
DATE
FILMED
12-78
DDC

AD A059728

DDC FILE COPY

LEVEL

12

18 19
ESD-TR-78-171

14 MTR-3614

6 CAUSAL SYSTEM SECURITY.

10 BY JONATHAN K. MILLEN

11 OCT 6 1978

12 32p.

Prepared for

DEPUTY FOR TECHNICAL OPERATIONS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts

9 Technical rept.

DDC
OCT 11 1978
40



Approved for public release;
distribution unlimited.

Project No. 672B
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-78-C-0001
15

235 050

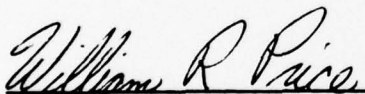
78 10 05 041

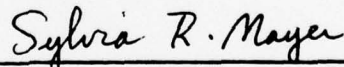
When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

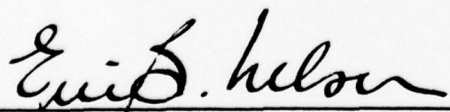
REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.


WILLIAM R. PRICE, Captain, USAF
Technology Applications Division


CHARLES J. GREWE, Jr., Lt Colonel, USAF
Chief, Technology Applications Division

FOR THE COMMANDER


ERIC B. NELSON, Colonel, USAF
Acting Director, Computer Systems Engineering
Deputy for Technical Operations

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-78-171	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CAUSAL SYSTEM SECURITY		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Jonathan K. Millen		6. PERFORMING ORG. REPORT NUMBER MTR-3614
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation P.O. Box 208 Bedford, MA 01730		8. CONTRACT OR GRANT NUMBER(s) F19628-78-C-0001
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Technical Operations Electronic Systems Division, AFSC Hanscom AFB, MA 01731		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 672B
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE OCTOBER 1978
		13. NUMBER OF PAGES 30
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTE		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) FORMAL SPECIFICATION INFORMATION FLOW COMPUTER SECURITY CONSTRAINTS SECURITY KERNELS SECURITY MODELING SECURITY VERIFICATION		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Security properties of discrete systems can be analyzed using prime constraints, which are related to prime implicants in switching theory. Prime constraints can be generated from nonprocedural transition specifications such as those commonly used in security kernel design and verification techniques. A security test similar to the *-property is derived.		

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

ACKNOWLEDGMENTS

This report has been prepared by The MITRE Corporation under Project No. 672B. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts.

The work reported in this paper is the joint effort of the author and Frederick C. Furtek.

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
TOP SECRET	
A	

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
LIST OF ILLUSTRATIONS	4
I INTRODUCTION	5
PROJECT GOALS	8
SUMMARY OF REPORT	9
BACKGROUND	9
II PRELIMINARIES	13
SYSTEM CONCEPTS	13
CONSTRAINTS	14
III DEFINITION OF SECURITY	17
DEDUCTIONS	17
SECURITY CONSIDERATIONS	18
THE DEFINITION	19
IV A SUFFICIENT CONDITION FOR SECURITY	21
COVERS	21
THE MONOTONICITY CONDITION	22
V CONCLUSIONS	23
APPLICATION	23
SUMMARY	25
EXTENSIONS	25
REFERENCES	27

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1	Two-Phase Security Verification	6
2	Verification Gaps	7

SECTION I

INTRODUCTION

This report introduces new concepts in the verification of security in computer systems, as well as reinforcing the approach ESD and MITRE have been using in the past few years. Our security kernel design methodology is a progression from an abstract model of a secure system, through more concrete representations of a kernel, to the binary machine code. This design methodology is associated with a two-phase security verification approach: first, verify the security properties of the design; and second, verify that the design has been implemented correctly in code. The first step is called property verification; the second step, functional verification. See Figure 1.

Rigorous verification, in the sense of mathematical proof, is still a research area. In a few limited applications, however, it is on the verge of entering the state of the art in computer system engineering. The three areas where it is becoming practical in security kernel validation are these:

1. property verification of high level formal specification;
2. functional verification of high-order-language code with respect to a high level formal specification;
3. microprogram verification.

The third area, microprogram verification, is aimed at establishing that the machine language instructions are implemented correctly by the firmware, which uses hardware-implemented register transfer operations. There is a "verification gap" between the high-order-language code and its implementation in machine instructions, and another gap between the register transfer microinstruction level and the integrated circuit packages. See Figure 2. Medium scale integrated circuits are simple enough so that they can be tested exhaustively, but design problems crop up again in large scale integrated circuits, such as microprocessors.

Verification techniques appear to be adequate in principle to cover the spectrum from high level specifications to gate level hardware components. Whether it is practical to do the verification



Figure 1. Two-Phase Security Verification

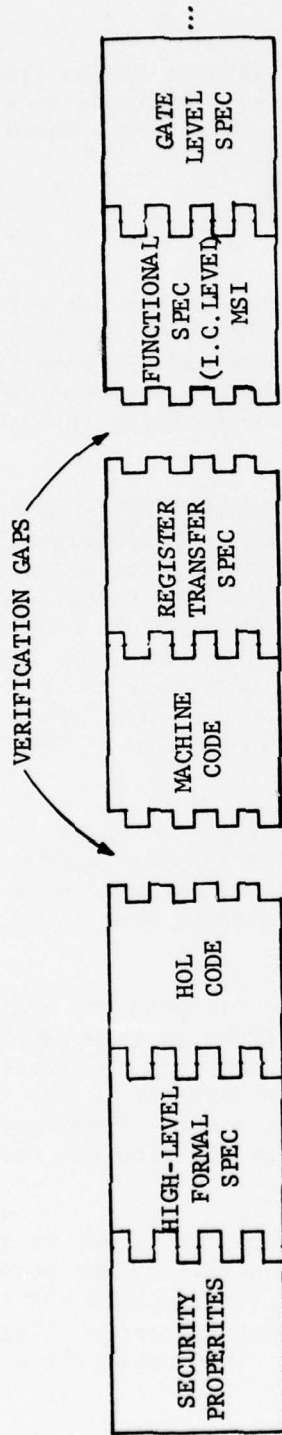


Figure 2. Verification Gaps

at each level depends primarily on two factors: design discipline, and a combination of tools and techniques. The importance of these two factors has been demonstrated in the three areas where verifications have been carried out.

PROJECT GOALS

Our project goals were closely related to the two verification gaps. Verification of machine language code was one goal; the second was to develop techniques for ensuring that machine code is robust, from a security point of view, with respect to hardware design flaws or failures. The state of the art is disappointing in both of these areas.

Rather than attempt to push present technology into jobs for which it was not yet suited, we took a more application-oriented route: by considering the specialized requirements of data security, it appeared possible to develop efficient tools and techniques for answering security questions without performing full machine language program verifications or their equivalents for hardware fault analysis. The new approach hinges on identifying a necessary as well as sufficient condition for security, so that a secure system will not be improperly rejected, regardless of how detailed or low-level a specification we are given to verify.

It was the lack of such a condition that forced us to try to go through functional verification of machine language code in the first place. The *-property, being sufficient but not necessary for security, usually fails when applied directly to security kernel code or detailed, low level specifications.

A necessary and sufficient condition for security against unauthorized disclosure of information is given in this report. It is derived from fundamental considerations about the ability to deduce or infer information about digital system variables. The theory is set out in a forthcoming working paper at a more elementary level. This report describes the initial steps in applying the theory in the context of security testing.

Because this theory identifies information flows, it should be possible to apply it to hardware design specifications to determine how variations in component functions or interconnections result in different communication paths between input and output lines, and hence lead to possible data compromises. This application is still in a conceptual stage.

SUMMARY OF REPORT

As in the initial Bell-Lapadula reports [8], we return in Section II to the notion of a general system as a common ground to begin the study of computer systems. Instead of specializing the system by introducing particular entities like subjects and access sets, however, we remain at a level at which any computer system, secure or otherwise, can be described.

Compromise of a system object is characterized in Section III as the ability to deduce something about a high level input by observing lower level outputs and controlling various inputs. Security is just the absence of compromise.

In order to show how the present theory can provide support for tests of security based on the *-property, it has been proved that a form of the *-property, called the monotonicity condition, stated in Section IV, is sufficient (though not necessary) for security against unauthorized information disclosure.

In order to apply the concepts in this report to computer system validation, current techniques and theoretical results must take steps toward one another to meet in the middle. System specification languages must be provided with "semantics", or precise explanations, so that the systems they describe can be fully analyzed. The theory needs to be made more flexible, and to be extended to eliminate certain assumptions that were made in the first cut to simplify it. A discussion of both activities is given in Section V, Conclusions.

BACKGROUND

In its early days, the theory of information security in computer systems was regarded solely as a matter of access control. Subjects had read or write access to objects. Subjects had a natural interpretation in a manual data-processing environment as people, and objects as documents. When this philosophy was transferred to computer systems, subjects became processes and objects became files. The process/file level of granularity was acceptable for ordinary user programs, but turned out to be too coarse for system programs where efficiency is of great importance. The operating system software that handles access requests, and changes in access authorization, was found to be a prime source of the need to work at a finer-grained level. The subject/object approach is awkward at this level because there is no natural interpretation for subjects.

The reason that processes no longer suffice as subjects can be illustrated with an example. Consider a program with the two assignment statements:

```
U2 := U1;
```

```
S2 := S1;
```

and let us assume that information flow from S1 to U2 is not authorized. The process evidently needs read access to S1 and write access to U2. From a subject-object-access point of view, the situation is insecure. What makes the difference here is the fact that we know what the program is, and we can see that it causes no information flow from S1 to U2. How can we formalize this argument?

One way is to introduce new, more abstract, subjects, and say that the two statements could, in principle, be executed by two distinct subjects. When subjects are reinterpreted, however, access also has to be viewed differently, and there is less intuitive assurance that read and write accesses are being interpreted appropriately in any but the simplest situations.

If the primary objective of the analysis is to detect unauthorized disclosure of information, an appealing alternative is to formalize the notion of information flow from one object or variable to another.

Shannon's theory of communication [11] does not seem to be directly applicable here, primarily because it deals with a single communication channel. In a computer or computer program, there is potentially a channel between any pair of variables, and the usefulness of the channel often depends on the current values of other variables. In this context, also, probability distributions are usually not known.

There have recently been several papers that have taken information flow approaches to computer security. Their common setting is a deterministic abstract machine whose current state is embodied in a set of state variables. Information flow from each state variable to others may result from each transition of the machine.

Jones and Lipton [1] consider a transition as the result of invoking a program. A program is a function from its input--which includes global variables and data structures as well as arguments--to its outputs, which can be stored in global variables or just viewed. If an output can be determined from some proper subset of the inputs, then there is no information flow from the inputs not in that subset to that output.

D. Denning and P. Denning [2] classify program statements according to the information flows that can occur between the variables that participate in the statement. An assignment statement potentially transfers information from its right hand variables to its left hand variable. A conditional statement potentially transfers information from the condition variables to any variables that can be modified in its sequel. The flow characteristics or "certification semantics" of a wide variety of statements are given.

Feiertag [3] has a functional definition like Jones and Lipton, but considers the flow only from the past succession of external inputs to a given external output. This yields the most immediate application to multilevel computer security, since levels are known a priori only for external variables. It is then shown that a per-transition policy based on assigning security levels to internal state variables is sufficient to protect against unauthorized disclosure.

Cohen [4] gives a sufficient as well as necessary condition for information flow, suggested by Shannon's probabilistic theory. A variable B is "strongly dependent" on a variable A over execution of an operation if variety in the value of A beforehand forces variety in the value of B afterward. This definition satisfies the requirement of a functional approach: if an output is determined by a certain set of variables, it is not strongly dependent on any variables not in that set, with the exception of those linked by some relation or invariant to a variable in the set.

Our approach uses a particular static representation of a system in terms of "prime constraints", which are analogous to prime implicants in switching theory. A prime constraint characterization:

1. describes the system as a whole, rather than single operations, programs, or statements;
2. exhibits security compromises transparently;
3. exists for nondeterministic systems.

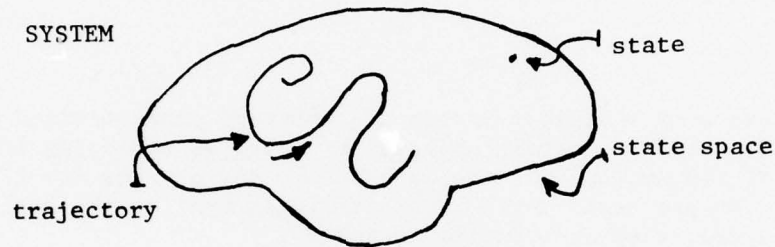
The prime constraints of a system are derivable from non-procedural transition specifications such as those introduced by Parnas [9] and used by MITRE [10] and SRI [3] in security verifications. A way to generate a set of prime constraints sufficient for security analysis will be suggested.

The main result in this paper is the statement that a certain condition on transitions, similar to the *-property, is sufficient to guarantee security against unauthorized disclosure.

SECTION II
PRELIMINARIES

SYSTEM CONCEPTS

The concept of a system is often presented in terms of a state space and a set of possible trajectories through it. The state space can be pictured as a cloud-shaped area within which individual states are represented as dots.



A trajectory is a mapping of a time interval into the state space; it can be pictured as a directed curve embedded in the state space.

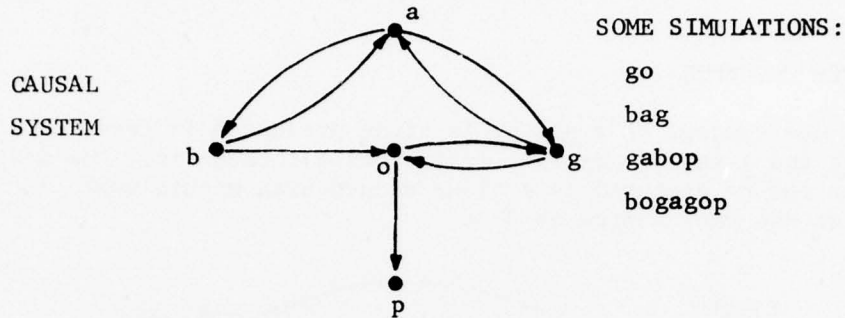
A major dichotomy in the treatment of systems is the continuous/discrete decision. A fully discrete system has a finite or countably infinite space, and time intervals are represented by sequences of consecutive integers. For analysis of algorithms and higher level hardware logic design, a discrete formulation is productive and reasonably safe, because digital computers are designed to foster an illusion of discreteness.

Trajectories in discrete systems are called simulations. A simulation is pictured below. The arrows between states are transitions from state to state.

DISCRETE
SYSTEM



A causal (discrete) system can be described completely by its set of possible transitions. A sequence of states is a simulation if and only if each state is followed by one to which there is a possible transition.



The state of a digital system can often be characterized by the individual states of certain objects or variables embodying the "memory" of the system. At the gate level, the objects are flip-flops; at the processor level, they are registers; at the high-order software level, they are program variables.

Besides holding the state of a system, objects are useful in describing input and output behavior. The output of a system is simply the state of a subset of the objects, called outputs or visible objects. Inputs are objects whose states are "free", or independent of the system. That is, if there is a transition from system state q to q' , there will also be a transition from state q to any state which differs from q' only in the states of the inputs. Since the system does not constrain input states, we may imagine that their states are determined externally by the "user".

CONSTRAINTS

Constraints are built up out of conditions. If a is a variable and v is a possible value for a , then a_v is a condition. The condition a_v is satisfied by a state q if $a = v$ in q .

A term is a conjunction of conditions in which each variable occurs at most once. The symbol 1 is used to denote a term in which no variable occurs. Examples of terms are a_0 , $a_u b_v$, $b_1 c_0 e_3$.

One may form symbolic cross products of terms, of the form

$$t_1 \times \dots \times t_n.$$

Such a product is satisfied by a state sequence q_1, \dots, q_n if, for all i , q_i satisfies every condition in t_i . A symbolic product is a constraint if it is not satisfied by any simulation.

A constraint is prime if deleting any condition results in a symbolic product that is not a constraint. Given any constraint, one can find a prime constraint from it by deleting conditions until the result would no longer be a constraint. A prime constraint obtained this way is said to cover the given constraint.

As an example, consider the system with two binary variables whose transition set is described by the assignment statement

$$b := a$$

with the understanding that a is free. The system has four possible states:

$$\begin{array}{cc} a_0 b_0 & a_1 b_0 \\ a_0 b_1 & a_1 b_1 \end{array}$$

Each of the terms above, because it has a condition from every variable, is satisfied by exactly one state; it is often convenient to use such terms to represent states.

The product

$$a_0 b_1 \times a_1 b_1,$$

which is satisfied by just one state pair, is a constraint because $a_0 b_1 \not\rightarrow a_1 b_1$. It is not prime, however, because

$$a_0 \times b_1$$

is also a constraint. In fact, $a_0 \times b_1$ is prime.

SECTION III

DEFINITION OF SECURITY

DEDUCTIONS

One can use a constraint, plus knowledge of the value of all but one of the variables appearing in it, to deduce something about the value of the remaining variable. For example, the constraint

$$a_0 b_1 \times c_0,$$

together with the knowledge that $b = 1$ in state q and $c = 0$ in the next state q' , permits the deduction that $a \neq 0$ in q . Knowledge about b and c has led to a conclusion about a .

If the constraint is prime, we can guarantee that the events $b = 1$ in q , and $c = 0$ in q' are possible, since

$$b_1 \times c_0$$

cannot be a constraint. Thus, a prime constraint is sufficient to make a deduction about any one of the variables occurring in it, if one can control or observe the others.

Prime constraints are also necessary for a deduction of this type. Suppose that one observes and/or causes a series of events expressed by the product

$$p_1 \times \dots \times p_n.$$

We shall say that one can "deduce something about" a variable a if he can exclude at least one possible value u for a at some time i (relative to the series of events). There is no loss of generality by assuming that $1 \leq i \leq n$, since the product can always be extended by annexing observations of the universal event 1. The conclusion that a cannot have value u at time i implies that

$$p_1 \times \dots \times p_i a_u \times \dots \times p_n$$

is a constraint. This constraint might not be prime, but there is some prime constraint covering it. Furthermore, any prime constraint covering it still contains the occurrence of a_u , since $p_1 \times \dots \times p_n$ is not a constraint -- it expresses events that have occurred in some simulation.

In summary, a necessary and sufficient condition to deduce something about the value of a variable on the basis of access to other variables is the existence of a prime constraint with an occurrence of the variable in question, such that all other variables occurring in it are accessible. (This statement would have to be refined somewhat to take into consideration access capabilities that change in time.)

SECURITY CONSIDERATIONS

In defining security, we consider only those constraints involving solely input and output variables. Any control or observation of system variables by a user must be *managed via inputs and outputs*, and those are the only variables for which security levels are given.

Although a user can directly observe outputs at his own level or lower, he can sometimes control inputs at higher or incomparable levels. One way to do so is by introducing a "Trojan Horse" into the system software. This higher-level control ability can be limited or eliminated in some environments, but only the worst case of unlimited ability to control all inputs is treated below.

Of course, it does not help a penetrator to control all the inputs to a system, since he will learn nothing he did not already know, but we do not exclude the possibility that he will control all but one, or as many as he needs, to learn something about one particular input still controlled by a high level user.

For example, if there is a prime constraint of the form $a_0 b_0 \times c_0$, where a and b are secret inputs and c is an unclassified output, we would identify a possible compromise of a (with a Trojan Horse controlling b) or of b (with a Trojan Horse controlling a).

Security compromises are not limited to deductions involving two consecutive states. A value entered in some input at the "Secret" level should not predictably reappear as the value of an "Unclassified" output at any later time. The definition of security, therefore, involves n-term rather than just two-term constraints.

Security levels are defined for inputs and outputs only, and, in this paper, are assumed constant in time. Security levels do change in real systems, but it is possible to regard a variable as a collection of "virtual" variables of constant security level, and then prove later that the virtual variables are multiplexed correctly into the single real one. The ability to virtualize away certain complexities for purposes of security analysis is one of the advantages of using a high level, formal transition specification [5].

We define an external security level assignment as a function

$$\lambda: XUY \rightarrow L$$

where X is the set of input variables, Y is the set of output variables, and L is a finite lattice of security levels. A lattice is a partially ordered set (some pairs of elements are incomparable) such that each finite subset has a least upper bound and a greatest lower bound in the lattice.

THE DEFINITION

A system is secure against an authorized disclosure in a Trojan Horse environment if no user at level s can deduce something about the value of an input of a higher or incomparable level, on the basis of observations of external variables at level s or lower and/or control of inputs at any level. By the above arguments, a system is secure in this sense if and only if, for

any prime constraint in which only inputs and outputs occur,
the least upper bound of the input levels is greater than or
equal to the least upper bound of the output levels. (Otherwise,
let the least upper bound of the output levels be s ; there must
be an input of level greater than or incomparable to s , about which
a user at level s could deduce something.)

SECTION IV

A SUFFICIENT CONDITION FOR SECURITY

COVERS

In practice, it is desirable to draw conclusions about security by analyzing some small-as-possible presentation of a system, such as a program listing or a formal specification. In more abstract terms, we wish to analyze the transition set of a system without having to generate simulations (or constraints) of greater length than two. Rather than look at the transition set directly, we shall work with a cover. A cover is a set of two-term prime constraints such that each non-transition (state pair) satisfies some constraint in the cover.

A cover consisting of prime constraints is called a prime cover.

Covers, even prime covers, are not unique, but one can always produce a cover simply by listing all state pairs that are not transitions, representing each state by the conjunction of the conditions that hold for it. Then a *prime cover* can be found by replacing each constraint by a prime constraint that covers it.

The results in this paper apply to systems coverable by constraints of a restricted form: those with single conditions on the right. We call a constraint simple if it is of the form

$$p \times a_v.$$

A simple system is one possessing a simple cover (a cover consisting of simple constraints). This category of systems includes all systems that would be considered deterministic. Let us define a system to be structurally deterministic if the value of every non-free variable is determined by the previous state.

The unqualified term "deterministic" should probably be reserved, in a security context, for structurally deterministic systems whose free variables are all inputs.

The practical question of how to produce such a cover depends, for its answer, on the way in which the system is presented or specified. Some discussion on how to generate a simple prime cover from an established specification format is given in the Conclusions Section.

THE MONOTONICITY CONDITION

The Bell-LaPadula *-property [6] requires that if a subject has read access to an object a and write access to an object b in the same state, the security level of a must be dominated by the level of b. The idea is that no information could be transferred from a to b in a single transition without the accesses indicated.

The nearest equivalent in the present context is the following monotonicity condition. Given an external security level assignment λ , an extension $\hat{\lambda}$ of λ to all variables is monotone with respect to a simple cover if, for all variables a and constraints $p \times b_v$ in the cover,

$$\text{if } a \text{ occurs in } p \text{ then } \hat{\lambda}(a) \leq \hat{\lambda}(b).$$

Since simple two-place prime constraints express the intuitive idea of information transfer, the monotonicity condition says that information is never transferred down in security level, but only (monotonically) up or on the same level. It is shown in [12] that a system is secure against unauthorized disclosure in a Trojan horse environment if there exists a monotone extension of the external level assignment.

Although this result is perhaps not surprising, it is difficult to prove. The key fact is that any prime constraint linking inputs to outputs, for example:

$$a_0 \times 1 \times 1 \times b_1,$$

implies the existence of a staircase of two-place constraints, for example:

$$\begin{array}{r} e_1 \times b_1 \\ f_0 \times b_1 \\ d_1 c_0 \times f_1 \\ d_0 c_0 \times e_0 \\ a_0 \times c_1 \end{array}$$

which, together, imply the long constraint (by extended consensus [7], assuming that c , d , e , and f are binary). The monotonicity condition applied to the two-place constraints allows us to conclude easily, then, that the security level of the input variable a must be dominated by the level of the output variable b .

SECTION V

CONCLUSIONS

APPLICATION

To test a system for security using the monotonicity condition, a simple prime cover must be found. While a "constructive" proof of the existence of such a cover was given for structurally deterministic systems, a practical technique for producing them has not yet been implemented. Some ideas on how to do so are presented in this section.

Constraints can be viewed as a means of expressing the semantics of other, more convenient, specification languages. Formal transition specifications, like some of those suggested by Parnas [9], lend themselves to this treatment.

A simple, but not untypical, transition specification for an operation to copy one element of an array into another with a greater or equal index is given below:

```
O-function copy(i,j)
  exception
    i > j
  effect
    m(j) := m(i)
```

The first step in translating this type of specification into a simple prime cover is to identify the variables. First, the arguments i and j are stored in some variables, say a and b . The array m is composed of the variables $m(1), m(2), \dots$

The assignment statement in the effect, considered in isolation, suggests the constraints

$$m(i)_u \times m(j)_v \quad (u \neq v) \quad (1)$$

where $u \in V(m(i))$ and $v \in V(m(j))$ are understood, and it is assumed that $m(k)$ exists for all $k \in V(a) = V(b)$.

The constraints specified by (1) apply only when i and j are the argument values, and the exception condition does not hold. Hence, the function as a whole has constraints

$$a_i b_j m(i)_u \times m(j)_v \quad (u \neq v, i \leq j) \quad (2)$$

When the exception condition holds, $m(j)$ is not modified. Hence, we have also:

$$a_i b_j m(j)_u \times m(j)_v \quad (u \neq v, i > j) \quad (3)$$

Finally, no element of m other than $m(j)$ is ever modified; this gives

$$b_j m(k)_u \times m(k)_v \quad (u \neq v, k \neq j) \quad (4)$$

There are no constraints with a or b on the right because a and b , which hold arguments of the call, are inputs, and could change arbitrarily for the next call.

The fact that a constraint cover is for a whole system, while formal specifications are presented function by function, is not an obstacle. Assume that we have a collection of covers R_1, \dots, R_n , each of which specifies one function. Let us introduce a new variable e with values $1, \dots, n$ to "choose" the function. Replace each constraint $f \times g$ in R_i by $e_i f \times g$. The collection of all of the new constraints is a cover for a system in which any of the n functions may be chosen freely.

The copy example above also provides an easy demonstration of a security validation. Suppose that a and b are at the minimum security level and that the level of $m(k)$ is k . The monotonicity condition can be verified by inspection of (2) - (4).

SUMMARY

A system comprises variables, whose combined values express the system state, and transitions. Systems are not necessarily deterministic or even structurally deterministic. Products have been defined as certain sets of state sequences. A constraint is a product containing no simulations. The transition set of a system can be expressed with a cover, which is a set of two-place constraints. A cover of simple prime constraints, which express strong dependencies, can be found for structurally deterministic systems. A prime constraint of any length is the extended consensus of extensions of elements of a cover.

Security is defined in terms of prime constraints, regardless of length, involving inputs and outputs, relative to a given external security level assignment. A monotonicity condition for any extension of the level assignment, applied to a simple prime cover, is sufficient for security.

A possible way of constructing simple prime covers in practice starts with a formal transition specification.

EXTENTIONS

Three simplifying assumptions were made that could be relaxed to extend the theory in natural directions.

Nothing was assumed known about the initial state of a system. In practice, however, there are typically some "invariants" of the system state that are guaranteed initially and preserved by every transition. As Cohen* points out, this additional knowledge can affect information flow, and hence security, since it makes certain observations unnecessary. Invariants could be expressed as constraints of length one, i.e., Boolean products containing only "illegal" states. Such constraints would have to be included in a cover, and the monotonicity condition would probably have to insist that variables in the same one-place constraint have the same level. Actual validations have not used such invariants to estimate information flow, but have used them in connection with non-constant security level assignments.

* See Reference 4.

Non-constant security level assignments allow the security level of a variable to be a function of the current state. They can be handled by defining security in terms of simulations, and expressing the monotonicity condition in terms of transitions. High level specifications can be used to trade this complication for a proof of correct implementation, but such proofs can be very difficult if the system has not been designed to facilitate them.

The worst-case assumption of complete control by arbitrarily high level inputs by uncleared users can be relaxed by weakening the definition of security. If there were no Trojan Horses, for example, one could admit a compromise of an input variable only when all other variables in a prime constraint, rather than just all non-inputs, are bounded by a lower or incomparable security level. The monotonicity condition would then still be sufficient, but one might look for a weaker condition of comparable simplicity.

REFERENCES

1. A.K. Jones and R.J. Lipton, "The Enforcement of Security Policies for Computation", Proc. of the Fifth Symposium on Operating Systems Principles, November 1975, 197-206.
2. D.E. Denning and P.J. Denning, "Certification of Progress for Secure Information Flow", Comm. ACM, Vol. 20, No. 7 (July 1977) 504-513.
3. P.G. Neumann, R.S. Boyer, R.J. Feiertag, K.N. Levitt, and L. Robinson, A Provably Secure Operating System: The System, Its Applications, and Proofs (Final Report), 11 February 1977, Stanford Research Institute, Menlo Park, California.
4. E.S. Cohen, Strong Dependency: A Formalism for Describing Information Transmission in Computational Systems, Dept. of Computer Science, Carnegie Mellon U., Pittsburg, Pa., August 1976.
5. J.K. Millen, "Formal Specifications for Security", Proc. of Trends and Applications 1977: Computer Security and Integrity, May 1977, IEEE Computer Society.
6. D.E. Bell and J.J. LaPadula, Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306, Electronic Systems Division, AFSC, Hanscom AFB, MA, March 1976 (ADA023588).
7. P. Tison, "Generalization of Consensus Theory and Application to the Minimization of Boolean Function", IEEE Trans. on Electronic Computers, Vol. EC-16, No. 4, August 1967, 446-456.
8. D.E. Bell and L.J. LaPadula, "Secure Computer Systems", ESD-TR-73-278, Volumes I-III, Electronic Systems Division, AFSC, Hanscom AFB, MA, November 1973 - June 1974 (AD770768, AD771543, AD780528).

LIST OF REFERENCES (Concluded)

9. D. L. Parnas, "A Technique for Software Module Specification with Examples", Communications of the ACM, Volume 15, Number 5, May, 1972, pp. 330-336.
10. J. K. Millen, "Security Kernel Validation in Practice", Communications of the ACM, Volume 19, Number 5, May 1976, pp. 243-250.
11. C. E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, Volume 27, p. 379 ff. (July, 1948).
12. J. K. Millen, "Constraints and Multilevel Security", in Foundations of Secure Computation, Academic Press, 1978.