

A Survey of Formal Models for Computer Security

CARL E. LANDWEHR

*Computer Science and Systems Branch
Information Technology Division*

September 30, 1981



NAVAL RESEARCH LABORATORY
Washington, D.C.

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8489	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A SURVEY OF FORMAL MODELS FOR COMPUTER SECURITY		5. TYPE OF REPORT & PERIOD COVERED Final report on a continuing problem.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Carl E. Landwehr		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375 Code 7590		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NRL Problem 75-0113-0-0 61153N, RR0140941
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375		12. REPORT DATE September 30, 1981
		13. NUMBER OF PAGES 39
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		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 NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer: security protection, operating system, data security, access control, access matrix, capabilities, confidentiality, privacy, information flow, security classes, confinement, integrity, aggregation, sanitization, program verification.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Efforts to build "secure" computer systems have now been under way for more than a decade. Many designs have been proposed, some prototypes have been constructed, and a few systems are approaching the production stage. A small number of systems are even operating in what the Department of Defense calls the "multi-level" mode: some information contained in these computer systems may have a classification higher than the clearance of some of the users of those systems.		

20. ABSTRACT (Continued)

This report reviews the need for formal security models, describes the structure and operation of military security controls, considers how automation has affected security problems, surveys models that have been proposed and applied to date, and suggests possible directions for future models.

CONTENTS

1.0 INTRODUCTION	1
2.0 WHY FORMAL MODELS?	1
3.0 STRUCTURE OF MILITARY SECURITY	2
4.0 DYNAMICS OF MILITARY SECURITY	3
5.0 EFFECTS OF AUTOMATION	4
5.1 Old Problems Aggravated	4
5.2 New Problems	5
5.3 Potential Benefits	6
6.0 FORMAL MODELS FOR COMPUTER SECURITY	7
6.1 Basic Concepts and Trends	7
6.2 High Water Mark Model	9
6.3 Access Matrix Model	10
6.4 Models Based on Access Matrices	11
6.5 Bell and LaPadula Model	15
6.6 Information-Flow Models	18
6.7 Extensions and Applications of the Bell and LaPadula Model	20
6.8 Programs as Channels for Information Transmission	24
7.0 DISCUSSION	27
8.0 CONCLUSION	30
9.0 ACKNOWLEDGMENTS	31
10.0 REFERENCES	31

A SURVEY OF FORMAL MODELS FOR COMPUTER SECURITY

1.0 INTRODUCTION

Efforts to build "secure" computer systems have now been under way for more than a decade. Many designs have been proposed, some prototypes have been constructed, and a few systems are approaching the production stage. A small number of systems in the Department of Defense (DoD) are even operating in "multilevel" mode: some information in any of these systems may have a classification higher than the clearance of some users.

Nearly all of the projects to design or construct secure systems for processing classified information have had a formal mathematical model for security as part of the top-level definition of the system. The model functions as a concise and precise description of the behavior desired of the security-relevant portions of the system. These models have been influenced by the DoD regulations for processing classified data, by intuitive notions of security, by the structure of existing computer systems, and by the capabilities of program-verification technology. They have not always been influenced by or even recognized the ways in which security regulations are applied in practice.

The purpose of this report is to review the need for formal security models, to describe briefly the structure and operation of military security controls, to survey models that have been proposed and applied to date, and to suggest possible directions for future models. All of the models described concern access to information within a computer and the flow of information within a computer; they are not concerned with the areas described by the Dennings [1] of user authentication, inference controls, or cryptographic controls.

Our descriptions will, whenever possible, avoid formal notation. The purpose of this paper is to make the basic concepts of each model apparent, not to restate each model in complete detail.

2.0 WHY FORMAL MODELS?

To build a secure system, designers must first decide exactly what "secure" means for their particular needs. In a private company, security may be related to the nondisclosure of confidential accounting data or trade secrets, or to the enforcement of privacy regulations regarding personal medical or credit records. If national security data is involved, security becomes the protection of classified material, as detailed in various DoD instructions and regulations. One might hope these regulations to be clear-cut and directly applicable to information stored in computers: not so. Because most of the regulations were originally constructed for an environment where information was recorded on paper and stored in safes, they have had to be revised as the use and understanding of computers within DoD has increased.

Although the DoD regulations can be said to define the security required for systems processing classified national security data, their form is not very helpful to system designers. Typically, regulations are written in English and are descriptive ("safeguards must permit accomplishment of mission functions while affording an appropriate degree of security" [2]) rather than prescriptive ("the system shall have the following design characteristics:...").

The point here is not that the regulations are poorly phrased—indeed, it would be undesirable for regulations to specify particular approaches when many of the questions involved are still research issues—but that formal models of security are needed for design. Since the system must not only be secure, but must be demonstrably so, designers need formal security models to be able to convince others of the security of the system. By constructing a formal model for security, demonstrating that systems enforcing this model are secure (according to the applicable DoD regulations, privacy laws, or company policy), and then demonstrating that the design to which the implementation corresponds enforces the model, the designers can make a convincing argument that the system is secure.

To date, the need for computer security has been more apparent in military than in commercial applications; consequently, the models discussed below concern military rather than industrial security. As security concerns become more important to the private sector, and to the nonmilitary parts of the government, formal models appropriate to these applications will also be needed.

3.0 STRUCTURE OF MILITARY SECURITY

Because most of the models described below were constructed with military security in mind, it will be helpful to review briefly some of the major aspects of military security for readers unfamiliar with them.

The requirement for military security arises from the existence of information that, if known by an enemy, might damage the national security (by making defenses more easily penetrable, for example). Because there are costs associated with protecting such information, and because not all information is equally sensitive, different *sensitivity levels* of information are distinguished. The recognized sensitivity levels, in increasing order of effect on national security, are unclassified, confidential, secret, and top secret. Information that has been assigned any of the three levels above unclassified is called *classified*. The *classification* of information takes into account its sensitivity level, and, in some cases, additional factors described below.

Since the purpose of the classification system is to prevent the uncontrolled dissemination of sensitive information, mechanisms are required to ensure that those individuals allowed access to classified information will not distribute it improperly. In the military security system, the granting of a *clearance* to an individual indicates that certain formal procedures and investigations have been carried out and that the individual is considered trustworthy with information classified up to a certain sensitivity level. Clearances for higher levels of information correspond to greater degrees of trust, and correspondingly require more extensive background investigations. The discretionary power accorded individuals of increasing clearance levels is enforced by explicit legal penalties for any improper handling of classified information.

The smaller the number of people that know a secret, the easier it is to control further dissemination. In recognition of this fact, and of the fact that few individuals need to be aware of *all* of the information classified at a given sensitivity level, a finer grain of classification has been created on the basis of *need to know*. The general principle is that classified information should not be entrusted to an individual unless he has both the clearance required for it and some specific, job-related need to know that information. Although this principle applies to all classified information, in some cases information relating to specific subject areas is formally designated as a separate *compartment* of information (e.g., all information related to nuclear weapons might be in a compartment called NUCLEAR). Compartment designations are in addition to the sensitivity level designations; information might be designated "CONFIDENTIAL, NUCLEAR" or "SECRET, NUCLEAR," for example. Compartments may overlap, with some information designated as being in two or more compartments. A classification, or *security level*, then, consists of both a sensitivity level and a (possibly empty) set of compartments.

Corresponding to these formally designated need-to-know compartments are additional clearances that are used to control the compartments to which an individual may have access. If information is designated with multiple compartments, an individual must be cleared for all of them before he can view that information.

In addition to compartments, there are restrictions known as *caveats* placed on some documents. Although these serve a function quite similar to compartments, they are usually broader in scope. One caveat, for example, is the "Originator Controlled (ORCON)" caveat, indicating that its originator must approve any further dissemination of the information. There are no specific clearances that correspond to the caveats; instead, specific properties of individuals (such as authorship or citizenship) are referred to.

The dissemination of information of a particular security level (including sensitivity level and any compartments or caveats) to individuals lacking the appropriate clearances for that level is prohibited by law; these statutory restrictions are sometimes referred to as *mandatory* access controls. In distributing information of a given security level to those who possess the necessary clearances, a cleared individual must exercise some discretion in determining whether the recipient has, in addition, a need to know the information. These imprecise but important restrictions are referred to as *discretionary* access controls.

4.0 DYNAMICS OF MILITARY SECURITY

The structure described above is generally adequate to describe a static set of information recorded on paper. Each piece of paper can be appropriately classified and physically protected (e.g., by storage in a safe). The dynamics of information handling under such a system are more difficult to model than are its static aspects. These include such operations as creating a new piece of classified information (perhaps using a collection of existing information), sanitizing information by removing the sensitive parts, declassifying information, copying information, and so on.

Creation of new classified information can cause a number of problems, the first of which is determining whether new information should in fact be classified. In the case of a new document relating to a previously classified system or topic, or using information from classified sources, it will usually be clear to the author that the new document will be classified as well. Generally, a document can be viewed as a sequence of paragraphs, each of which is assigned a classification. Because the document as a whole also has a classification, the document is in this sense a *multilevel object*—it can contain information classified at various levels.

The level of classification of a document as a whole is usually that of the most classified information it contains. In some cases, however, a collection of information, each component of which is by itself unclassified (or classified at a low level) may yield a more highly classified document. For example, a picture of the Statue of Liberty and its caption, "Location of secret Particle Beam Weapon" could, if separated, both be unclassified. Together, they might be top secret. The problem of detecting whether such a collection exists is called the *aggregation problem*. If the new document is created by sanitizing an existing one, the new document may be classified at a lower level than the original. Determination of when the information in a document has been sufficiently "desensitized" is called the *sanitization problem*. Proper identification of aggregated or sanitized information is the obligation of the document creator in cooperation with his security officer. If a document is found to have been more highly classified than required, it may be *downgraded* (given a lower security level without changing its contents).

As long as the principal storage medium for the information is paper, and the principal tools for creating it are manual (e.g., pens, pencils, typewriters), the control of these operations is not too

difficult. When a document is not in a safe, it is in the custody of some individual trusted not to distribute it improperly. A draft document with an as-yet-undetermined classification can be protected by storing it in a safe without declaring a specific classification or entering it into the formal system for control of classified documents. The tools used to create and modify documents are simple and generally passive; they cannot easily alter a classified document or betray its contents to an unauthorized person without the knowing cooperation of the tool user.

5.0 EFFECTS OF AUTOMATION

The use of computers to store and modify information can simplify the composition, editing, distribution, and reading of messages and documents. These benefits are not free, however. Part of the cost is the aggravation of some of the security problems just discussed and the introduction of some new problems as well. Most of the difficulties arise precisely because a computer shared by several users cannot be viewed as a passive object in the same sense that a safe or a pencil is passive.

For example, consider a computer program that displays portions of a document on a terminal. The user of such a program is very likely not its author. It is, in general, possible for the author to have written the program so that it makes a copy of the displayed information accessible to himself (or a third party) without the permission or knowledge of the user who requested the execution of the program. If the author is not cleared to view this information, security has been violated.

Similarly, recording the security level of a document, a straightforward task in a manual system, can be a complex operation for a document stored in a computer. It may require cooperation among several programs (e.g., terminal handler, line editor, file system, disk handler) written by different individuals in different programming languages using different compilers. It is much more difficult to establish that the computer program(s) for recording a classification behaves in accordance with its user's wishes than it is to establish the same criterion for a pen or a pencil.

Information contained in an automated system must be protected from three kinds of threats: (1) the *unauthorized disclosure* of information, (2) the *unauthorized modification* of information, and (3) the *unauthorized withholding* of information (usually called *denial of service*). Each of the problems discussed below reflects one or more of these dangers.

5.1 Old Problems Aggravated

Aggregation

The aggregation problem exists in a computer-based system just as it does in a manual one. Forming aggregate objects may be easier, though, because users may be able to search many documents more quickly and correlate the information in them more easily than could be done manually. Database management systems that include numerous files of information indexed in several different ways, and that can respond to user queries have no direct analog in the world of documents and safes. The response to a single query can aggregate information from a wide variety of sources in ways that would be infeasible in a manual system. A closely related problem is the *inference problem*. Studies have shown that database systems, if they provide almost any statistical information (such as counts of records, average values, etc.) beyond the raw data values stored, are relatively easy to compromise [1,3-6]. By carefully constructing queries and using only small amounts of outside information, a user can often infer the values of data he is unauthorized to obtain directly.

Authentication

In the manual system, keys and safe combinations are entrusted to humans by other humans; it is not generally difficult to recognize the trusted individual. A person opening a safe and examining its

contents is likely to be observed by other people who will know whether that person is authorized to do so. Further, an individual with access to a safe must have a clearance sufficient for him to see every document stored in the safe without violating security. Individuals with different clearance levels may have access to the computer system, so the system must be able to distinguish among its users and restrict information access to qualified users. Since the computer will have access to all of the information it stores, and since it must provide access to those documents only to authorized individuals, the *authentication problem* is aggravated: the computer system must have a reliable way of determining with whom it is conversing.

Browsing

Computers generally maintain directories for files to facilitate searching large bodies of information rapidly: rarely is there a similar catalog of all of the information contained in even a single safe. Unless a computer system implements strict need-to-know access controls, it may be possible for a user to examine secretly all documents stored in the system at or below his clearance level (this is called the *browsing problem*). Browsing through all the documents in a safe would be a much more difficult activity to conceal.

Integrity

Undetected modification of information is much easier to accomplish if the information is stored on electronic media than if it is stored on paper, both because changes are harder to detect and because there is often only a single copy of the information that need be altered. Protecting information against unauthorized modification is called the *integrity problem*.

Copying

Although paper documents may be copied without altering the original, making such a copy entails removing the original from the safe. Undetected copying of files within most computer systems presents no similar barrier and usually can be done much more rapidly.

Denial of Service

In the manual system, the combination for a safe or a cipher lock may be forgotten or misplaced, or the lock may malfunction. In either case, the legitimate users of the information in the safe may be denied access to it for a time. Such occurrences, however, are rare. Denial of service is a much more notorious characteristic of computer systems, which can be vulnerable to power outages (or even fluctuations) and to hardware and software problems.

5.2 New Problems

Confinement

Storage of information in a computer can also cause new kinds of security problems. In a computer system, programs are executed by the active entities in the system, usually called *processes* or *jobs*. Generally, each process is associated with a user, and programs are executed by the process in response to the user's requests. A program that accesses some classified data on behalf of a process may leak that data to other processes or files (and thus to other users). The prevention of such leakage is called the *confinement problem* [7]. Lampson identifies three kinds of channels that can be used to leak information. *Legitimate channels* are those that the program uses to convey the results of its computation (e.g., the printed output from the program or a bill for the services of the program). It is possible, for example by varying line spacing, to hide additional information in these channels. *Storage channels* are those that utilize system storage such as temporary files or shared variables (other than the legitimate

channels) to pass information to another process. *Covert channels* are paths not normally intended for information transfer at all, but which could be used to signal some information. For example, a program might vary its paging rate in response to some sensitive data it observes. Another process may observe the variations in paging rate and "decipher" them to reveal the sensitive data. Because they generally depend on the observation of behavior over time, covert channels are also referred to as *timing channels*.*

Trojan Horses and Trap-doors

A program that masquerades as a useful service but surreptitiously leaks data is called a *Trojan horse*** A *trap-door* is a hidden piece of code that responds to a special input, allowing its user access to resources without passing through the normal security enforcement mechanism. For example, a trap-door in a password checking routine might bypass its checks if called by a user with a specific identification number.

Other Threats

Another class of threats introduced by automation is related to the electrical characteristics of computers. Wiretapping and monitoring of electromagnetic radiation generated by computers fall into this class. The formal models described below do not address this class of threats, nor do they cover problems of authentication, inference, or denial of service.

5.3 Potential Benefits

In compensation for the added complexities automation brings to security, an automated system can, if properly constructed, bestow a number of benefits as well. For example, a computer system can place stricter limits on user discretion. In the paper system, the possessor of a document has complete discretion over its further distribution. An automated system that enforces need-to-know constraints strictly can prevent the recipient of a message or document from passing it to others. Of course, the recipient can always copy the information by hand or repeat it verbally, but the inability to pass it on directly is a significant barrier.

The sanitization of documents can be simplified in an automated system. Removing all uses of a particular word or phrase, for example, can be done more quickly and with fewer errors by a computer than by a person (presuming, of course, that the editing programs work correctly!). Although it is doubtful whether a completely general sanitization program is feasible, automated techniques for sanitizing highly formatted information should be available in a few years.

Automated systems can apply a finer grain of protection. Instead of requiring that an entire document be classified at the level of the most sensitive information it contains, a computer-based system can maintain the document as a multilevel object, enforcing the appropriate controls on each subsection. The aggregation and sanitization problems remain; nevertheless, the opportunity exists for more flexible access controls.

An automated system can also offer new kinds of access control. Permission to execute certain programs can be granted or denied so that specific operations can be restricted to designated users. Controls can be designed so that some users can execute a program but cannot read or modify it directly. Programs protected in this way might be allowed to access information not directly available to the user, sanitize it, and pass the results back to the user. Naturally, great care would be needed in the construction of such a sanitization program and the controls protecting it.

*Although these terms had been in use for some time, Lampson was apparently the first to introduce this nomenclature for kinds of leakage channels into the open literature. We will employ his definitions, using "timing channel" in place of "covert channel." The reader is cautioned that usage in the literature is not uniform.

**This term was introduced by Dan Edwards [8].

Although these benefits are within reach of current technology, they have been difficult to realize in practice. Security is a relative, not an absolute, concept, and gains in security often come only with penalties in performance. To date, most systems designed to include security in the operating system structure have exhibited either slow response times or awkward user interfaces—or both.

6.0 FORMAL MODELS FOR COMPUTER SECURITY

The formal structures described below can be used to model the military security environment. These same structures can also be used as the basis for specifying programs that cause a computer to simulate the security controls of the military environment. Because it is difficult to capture the complexities of the real world in a formal structure, each model deviates from reality in some respects. Generally, the models enforce controls that are more rigid than the controls in the actual environment; any computer operations that obey the structures of the model will be secure according to the conventional definitions, and some operations disallowed by the model would nevertheless be considered secure outside of the formal model. Although this is the "safe" side on which to err, use of overly restrictive models to improve the security of a system can lead to systems that are unacceptable to their intended users [9].

The models presented in this section are diverse in several ways: they have been developed at different times, they treat the problem from different perspectives, and they provide different levels of detail in their specifications. We have tried to consider both chronology and formal similarity in organizing our presentation. Since models with different formal bases sometimes influence each other over time, it is hard to provide an ordering that both respects formal similarity and avoids forward references. Consequently, we include a brief discussion of some useful concepts and historical trends before presenting the individual models.

6.1 Basic Concepts and Trends

The *finite state machine model* for computation views a computer system as a finite set of states, together with a transition function to determine what the next state will be, based on the current state and the current value of the *input*. The transition function may also determine an *output* value. Transitions are viewed as occurring instantaneously in this model; therefore certain potential information channels (e.g., those related to observing the time spent in a certain state) in real systems tend to be hidden by it. Different security models apply different interpretations of this general model, but this structure is the basis for all of those surveyed below.

The *lattice model* for security levels is widely used to describe the structure of military security levels. A lattice is a finite set together with a partial ordering on its elements such that for every pair of elements there is a least upper bound and a greatest lower bound [10]. The simple linear ordering of sensitivity levels has already been defined. Compartment sets can be partially ordered by the subset relation: one compartment set is greater than or equal to another if the latter set is a subset of the former. Classifications, which include a sensitivity level and a (perhaps empty) compartment set, can then be partially ordered as follows: for any sensitivity levels a and b and any compartment sets c and d

$$(a,c) \geq (b,d),$$

if and only if $a \geq b$ and $c \supseteq d$. That each pair of classifications has a greatest lower bound and a least upper bound follows from these definitions and the facts that the classification, "unclassified, no compartments" is a global lower bound and that we can postulate a classification "top secret, all compartments" as a global upper bound. Because the lattice model matches the military classification structure so closely, it is widely used. The *high water mark* model [11], one of the earliest formal models, includes a lattice of security levels, though it is not identified as such.

The *access matrix model*, described in detail below, was developed in the early 1970's as a generalized description of operating system protection mechanisms. It models controls on users' access to information without regard to the semantics of the information in question. A *reference monitor* checks the validity of users' accesses to objects. Models based on access matrices continue to be of interest because of their generality; recent examples include studies of *take-grant* models [12] and the model of data security used by Popek [13].

When classified information is involved, the semantics of the information must be considered: the classification of the information and the clearance of the user must be known before access can be granted. For this purpose, models based on the access matrix were extended to include classifications, clearances, and rules concerning the classifications. The best known such model is the *Bell and LaPadula model* [14], which may be summarized in two axioms:

- (a) No user may read information classified above his clearance level ("No read up")
- (b) No user may lower the classification of information ("No write down")

The full statement of the model includes several more axioms and is quite complex.

In the early 1970's, Roger Schell conceived an approach to computer security based on defining a small subset of a system that would be responsible for its security and assuring that this subset would monitor all accesses (i.e., it would provide *complete* validation of program references), that it would be *correct*, and that it would be *isolated* (so that its behavior could not be tampered with). This mechanism would be called a *security kernel* [8,15]. Similar considerations motivated the work of Price and Parnas [16,17] on virtual memory mechanisms for protection. The Bell and LaPadula model grew out of work on the security kernel concept.

This idea fit well with the notions of operating system kernels and layered abstract machines that were being circulated widely at that time. The security kernel would be the innermost layer of the system and would implement all of the security-relevant operations in the system; for the access-matrix model, the kernel would implement the functions of the reference monitor. Because the security kernel would be of minimal size and functionality, it would be feasible to examine it closely for flaws and perhaps even to verify its correctness (or at least its security properties) formally. In practice, it has been difficult to identify and isolate all of the security-relevant functions of a general-purpose operating system without creating a fairly large, fairly slow "kernel."

Information-flow models, based partly on work by Fenton [18], and first introduced by Denning [19], recognize and exploit the lattice structure of security levels. Instead of requiring a list of axioms governing users' accesses, an information-flow model simply requires that all information transfers obey the flow relation among the security classes. The information-flow properties of each statement type in a programming language can be defined, and proofs can be developed about the flows caused by executing a particular program. By focusing on the flow of information instead of on individual accesses to objects, the models achieve an elegance lacking in the Bell and LaPadula model.

Because of continuing DoD interest, work on developing and applying the Bell and LaPadula model has continued. The original model dealt only with the unauthorized disclosure of data, but an extension of it by Biba [20] added the concept of *integrity* to deal with the unauthorized modification of data. The model was reformulated for use with automated tools for program verification by Feiertag and others [21]. This reformulation actually focuses on the information flow possible in a formally specified set of functions, and in this respect is similar to the information-flow models. Efforts have also been made to model security in database management systems using the Bell and LaPadula model [22,23].

Finally, several authors [24-27] have developed models that, in a variety of ways, view programs as channels for information transfer. These models are generally further from the mainstream of computer security than the others, but they provide some interesting comments on the fundamental question of what it means for a program or a computer system to be secure.

6.2 High Water Mark Model

The ADEPT-50 time-sharing system, constructed at the System Development Corporation in the late 1960's, was one of the first systems that attempted to implement software controls for classified information [11]. Although the system was never certified by the DoD for operation as a multilevel secure system, its controls were based on a formal model of military security.

Four types of objects are defined by the ADEPT-50 security model: users, terminals, jobs, and files. Each object is described by an ordered triple of properties, called Authority (A), Category (C), and Franchise (F). The first two of these correspond to a sensitivity level and a compartment set; the third consists of a set of user designations. The Franchise sets are used to implement discretionary need-to-know controls, but they are formally equivalent to an extension of the compartment set that allows a compartment for each user. The model also defines an ordering on these triplets that corresponds to the lattice model (though the structure is not identified as a lattice). "History functions" are defined for the authority and category properties of an object. These functions record the highest authority assigned to the object and the union of all categories assigned to the object since its creation. These are referred to as the *high water mark* of the object, from which the model takes its name.

The values of the current A, C, and F properties and the history functions are used to control the properties assigned to new objects (e.g., newly created files) and to determine whether requested operations will be allowed. To access the system from a terminal, a user must present a user ID and a password. The system then checks a list stored at system start time to see that this ID is known to the system, that it is in the franchise set for this terminal, and that the password is correct. If the log-in succeeds, the given user ID is assigned to the job servicing the user's terminal. The job is assigned the minimum of the authorities assigned to the user and the terminal, and is assigned a category set corresponding to the intersection of the user and terminal category sets. Permission for this job to access a file is granted if and only if the level of the job in the lattice is at least that of the file. Granting access to a file causes the history functions to be updated according to the authority and category set for that file. New files created by this job are assigned an authority and a category set based on the history functions: the authority is set to that of the highest file accessed by this job since log-in, and the category is the union of the category sets of all files accessed since log-in. The franchise is set to that of the job.

The ADEPT-50 time-sharing system, using the security model just described, was implemented on an IBM /360 model 50 and installed in several locations in the Pentagon. In addition to enforcing this model, a number of other security provisions (e.g., audit trails, clearing of newly acquired storage, etc.) were included in the implementation.

The principal reason the high-water-mark policy is of interest is that it is one of the few policies actually implemented on an operational computer system. The ADEPT-50 system, operating with this model, provided an acceptable interface to its users. The authority, category, and franchise elements of the model are sufficient to describe the static structure of military security. The restriction that a user can only have access to a file at or below his level in the lattice ensures that he cannot directly read information contained in a file classified above his clearance level. It is possible, however, for a Trojan horse to copy classified information to a (pre-existing) file that is unclassified. This copying can be done because the rules of the model allow authorized "downward" flows of information. Consequently,

information can flow out of the system via legitimate, storage, or timing channels. Control of sanitization and aggregation is provided by user vigilance and by audit mechanisms that record the explicit downgrading of information. The controls over the classification of new files are helpful, but can lead to the overclassification of data, since the high-water mark can never decrease during a given run. Routine overclassification is likely to lead to routine downgrading of classified data, which would make errors or intentional violations in downgrading harder to detect.*

6.3 Access Matrix Model

The access matrix model for computer protection is based more on abstraction of operating system structures than on military security concepts. One of the earliest descriptions of this model was provided by Lampson [28]; Denning and Graham [29,30] describe and extend it. Because of its simplicity and generality, and because it allows a variety of implementation techniques, it has been widely used.

There are three principal components in the access matrix model: a set of passive *objects*, a set of active *subjects*, which may manipulate the objects, and a set of rules governing the manipulation of objects by subjects. Objects are typically files, terminals, devices, and other entities implemented by an operating system. A subject is a process and a *domain* (a set of constraints within which the process may access certain objects). It is important to note that every subject is also an object; thus, it may be read or otherwise manipulated by another subject. The *access matrix* is a rectangular array with one row per subject and one column per object. The entry for a particular row and column reflects the mode of access between the corresponding subject and object. The mode of access allowed depends on the type of the object and on the functionality of the system; typical modes are read, write, append, and execute. In addition, flags may be used to record ownership of a particular object.

The access matrix can be viewed as recording the *protection state* of the system. Certain operations invoked by subjects can alter the protection state—for example, if the owner of a file deletes it, the column corresponding to that file is removed from the access matrix. In addition, some modes of access may permit users to alter the contents of particular entries of the matrix. If the owner of a file grants another user permission to read it, for example, the permission must be recorded in the appropriate access matrix entry. Graham and Denning provide an example set of rules—for creating and deleting objects and granting or transferring access permissions—that alter the access matrix. These rules, together with the access matrix, are at the heart of the protection system, since they define the possible future states of the access matrix.

All accesses to objects by subjects are assumed to be mediated by an enforcement mechanism which refers to the data in the access matrix. This mechanism, called a *reference monitor* [8], rejects any accesses (including improper attempts to alter the access matrix data) that are not allowed by the current protection state and rules. Graham and Denning [30] consider each object to be an instance of a particular object type. References to objects of a given type must be validated by the *monitor* for that type. Each type monitor then uses the data in the access matrix to validate the requested operations. In this view, there is a separate monitor that controls requests to change the access matrix. If *all* accesses of the access matrix pass through the access matrix monitor, that monitor is equivalent to the reference monitor.

Because the access matrix model specifies only that there are rules (and subjects and objects and access modes) but not what the rules (or subjects or objects or access modes) are in detail, the model has great flexibility and wide applicability. It is difficult, however, to prove assertions about the protection provided by systems that follow this model without looking in detail at the particular subjects,

*Part of the information in this paragraph (in particular, the assessments of the utility of the user interface and the security model) is derived from conversations with Marv Schaefer and Clark Weissman of SDC.

objects, modes of access, and rules for transforming the access matrix. Harrison, Ruzzo, and Ullman [31] investigated an access matrix model with six rules similar to the examples posed by Graham and Denning and found undecidable the question of whether, given an initial access matrix configuration, an arbitrary access right can later appear at an arbitrary location in the access matrix.

In actual computer systems, the access matrix would be very sparse if it were implemented as a two-dimensional array. Consequently, implementations that maintain protection data tend to store it either row-wise, keeping with each subject a list of the objects and access modes allowed it, or column-wise, storing with each object a list of those subjects that may access it and the access modes allowed each. The former approach is called the *capability list* approach, the latter, the *access control list* approach. These approaches are often used together, as in MULTICS [32] and other virtual memory systems. Virtual memory addresses can act as capabilities; possession of the address (and of the corresponding translation tables) in this sense suffices to authorize access to the corresponding data. And files in the system may have access control lists attached to control which subjects may actually read or alter the data in the file (even though all users may know the name of the file).

The access matrix model, properly interpreted, corresponds very well to a wide variety of actual computer system implementations. Without some additions, however, it does not include mechanisms or rules corresponding to the requirements for military security. In systems based on this model, the protection of a file of information is the responsibility of the file's owner. He can grant access to any user, and, typically, any user granted read-access to the file can copy and distribute the information in any way. Thus, without specializing the model, it would be very difficult to prove any theorems concerning the flow of information. On the other hand, the model neatly separates the mechanisms for enforcement from the policy enforced: the mechanisms of the system are the enforcers, and the current policy is contained in the current state of the access matrix. Note, however, that this interpretation of "policy" implies that any subject with the ability to create or delete objects, or to grant or revoke object-access can alter the policy enforced. The simplicity of the model, its definition of subjects, objects, and access control mechanisms, is very appealing. Consequently, it has served as the basis for a number of other models and development efforts, described below.

6.4 Models Based on Access Matrices

This section presents two models that are based on the concept of an access matrix. Both are intended to represent the behavior of a capability-based operating system. The first was developed as part of an effort to construct a prototype security kernel; the second, developed in terms of graph theory, has had little practical application.

UCLA Data Secure UNIX Model*

The efforts at UCLA to design, implement, specify, and verify a security kernel for UNIX have been described in numerous papers and technical reports [13,33-36]. The approach taken by Popek and his group is based on a concept they call *data security*: direct access to data must be possible only if the recorded protection policy permits it. The kernel is intended to enforce only this notion of security; it does not embody a particular security policy (in contrast to the kernels based directly on the Bell and LaPadula model). In the UCLA implementation, the protection policy is embodied in a separate process called the *policy manager*. A particular request from a user (e.g., to open a file) must be approved by the policy manager before the kernel will honor it. The kernel supports a form of capabilities, and the policy manager informs the security kernel of security policy by issuing the "grant-capability" kernel call.

The specification of the kernel is given in four increasingly abstract levels [36]. The lowest level is the kernel implementation in an extended Pascal; next is a "low-level specification" in the language of

*UNIX is a trademark of Bell Laboratories.

the XIVUS verification system [37], organized as a "data-defined specification." Then comes an "abstract-level specification" formulated as a finite state machine with the effect of each kernel call reflected in the transition function; and finally there is a "top-level specification," also given as a finite state machine. Mapping functions are provided from each lower level to the next higher one, so that a chain exists from the implementation to the top-level specification.

The security model implemented by the UCLA Data Secure Unix (DSU) corresponds to the data security property required of the top-level specification. The simplest description of the top-level model for DSU is given by Walker, *et al.* [36]. It is a finite state machine model, with the state defined by the following four components:

- a) Process objects,
- b) Protection-data objects, with values being sets of capabilities,
- c) General objects (comprising both pages and devices), and
- d) A current-process-name object, whose value is the name of the currently running process.

The security criterion is given in terms of the state: a component of the state is actually modified or referenced only if the protection data for the process named by the current-process-name object allows such access. In Ref. 13 a more formal and detailed definition of data security is given. It has three assertions, stated informally below:

- S1. Protected objects may be modified only by explicit request;
- S2. Protected objects may be read only by explicit request;
- S3. Specific access to protected objects is permitted only when the recorded protection data allows it.

In Refs. 13 and 38, these assertions concern the abstract-level specification; the top-level specification was apparently added later.

The UCLA DSU model is in one sense more general than the Bell and LaPadula model. It includes no mention of classifications, clearances, or the security lattice. All of these could be introduced, presumably, by an appropriately specified policy manager. The policy manager described in Ref. 34, though, is based on "colors." Each user and file has an associated color list, and for a user to access a file, his color list must cover the color list of the file. This access control technique also extends to processes and devices. Formally, this model appears equivalent to the military compartment structure, and it could be used to implement a lattice structure.

The UCLA DSU model was constructed only with the goal of preventing unauthorized direct references to or modification of protected data; it is not concerned with storage or timing channels.

Take-Grant Models

Take-grant models use graphs to model access control. They have been described and studied by several people [12,39-43]. Although couched in the terms of graph theory, these models are fundamentally access matrix models. The graph structure can be represented as an adjacency matrix, and labels on the arcs can be coded as different values in the matrix. Because it is the most recently published, and because it deals with a wider class of security problems than previous versions, the particular model of Ref. 12 will be described here.

In a take-grant model, the protection state of a system is described by a directed graph that represents the same information found in an access matrix. Nodes in the graph are of two types, one corresponding to subjects and the other to objects. An arc directed from a node A to another node B indicates that the subject (or object) A has some access right(s) to subject (or object) B. The arc is labelled with the set of A's rights to B. The possible access rights are read (r), write (w), take (t), and grant (g). Read and write have the obvious meanings. "Take" access implies that node A can take node B's access rights to any other node. For example, if there is an arc labelled (r,g) from node B to node C, and if the arc from A to B includes a "t" in its label, then an arc from A to C labelled (r,g) could be added to the graph (see Fig. 1). Conversely, if the arc from A to B is marked with a "g," B can be granted any access right A possesses. Thus, if A has (w) access to a node D and (g) access to B, an arc from B to D marked (w) can be added to the graph (see Fig. 2).

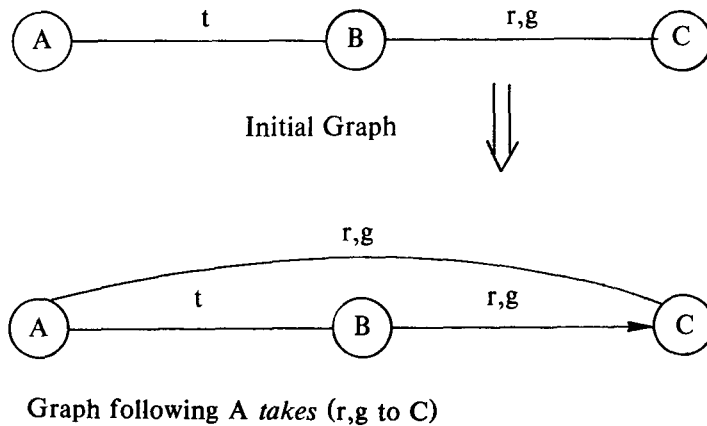


Fig. 1 — Example of take

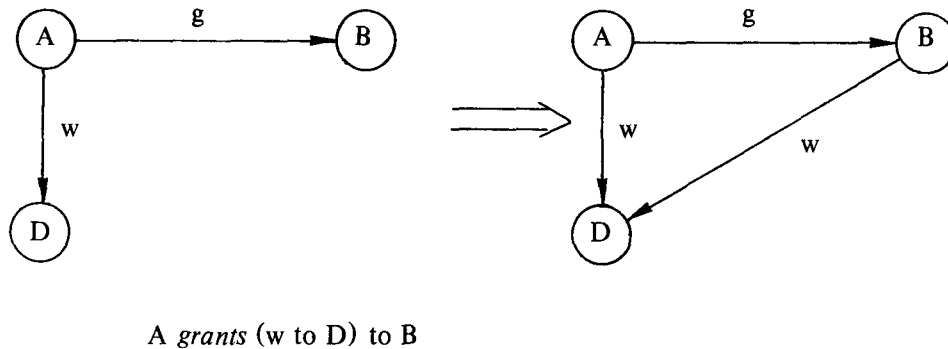


Fig. 2 — Example of grant

Because the graph need only include arcs corresponding to non-null entries in the access matrix, it provides a compact way to present the same information given in a relatively sparse access matrix. Capability systems are thus prime candidates for this modelling technique; each arc would then represent a particular capability.

Together with the protection graph, the model includes a set of rules for adding and deleting both nodes and arcs to the graph. Two of these, corresponding to the exercise of "take" and "grant" access rights, have already been described. A "create" rule allows a new node to be added to the graph. If subject A creates a new node Y, both the node Y and an arc AY are added to the graph. The label on

AY includes any subset of the possible access rights. A "remove" rule allows an access right to be removed from an arc; if all rights are removed from an arc, the arc is removed as well. An early version of the model [40] also included a "call" rule to model invocation of a program as a separate process. Other rules can be added, depending on the properties of the system being modeled, but in the published literature, take, grant, create, and remove are the key operations.

The questions first asked of this model were of the form: "Given an initial protection graph and the set of rules, is it possible for a subject A to gain a particular access right to an object B?" Notice that this is a question about the *possibility* of the initial graph being transformed into one containing a specified arc through some sequence of rule applications. The work of Harrison, Ruzzo, and Ullman [31] showed this problem to be undecidable for an arbitrary set of rules and an initial graph, but decidable for a specific set of rules. The answer is stated as a theorem [41]: A can acquire the right in question if and only if there is some subject or object that already has the right and A and B are connected by a path with a certain structure. For the rules of the take-grant model, this answer can be computed in a time directly proportional to the size of the graph [39].

Bishop and Snyder [12] recognize that information about an object can sometimes be transferred to a subject without the subject's gaining a direct access right for that object. For example, information can be copied from one object to another and access to the copy can be granted to others without ever granting others direct access to the original file. An information transfer of this type is called *de facto*, while the transfer of authority according to the rules discussed earlier is called *de jure*. Four "representative" graph rewriting rules to model *de facto* transfers are described and studied. Edges added to the graph by application of *de facto* rules are called implicit edges to distinguish them from the explicit edges added by the *de jure* rules. Predicates called *can-know* and *can-tell* are defined to characterize the possibility that an edge can be constructed between two nodes by application of the *de facto* rules.

An objection sometimes made to take-grant models is that they are too "weak" to provide useful inferences about protection systems: it is claimed that the result of applying a take-grant model to a "real" system will be a fully connected graph—all subjects can gain access to all objects. Certainly, this will be the case in any system in which a user can create a file and grant access to it to all users. The problem is that the model makes a worst-case assumption about the behavior of users—if a user can grant access rights for an object to some other user, the model assumes that at some time he will do so. In some cases, of course, this may be the appropriate assumption. If the users of the system cannot be trusted, for example, and if the system itself can enforce no finer controls than those on capabilities, this model may yield useful results. It does seem limited with respect to its ability to model controlled sharing of information, though.

Snyder partially addressed this problem [41] by defining a predicate *can-steal* to distinguish cases in which a subject can gain an access right to an object without the collusion of another subject who already has that right. This treatment deals only with *de jure* access. Jones [42], in applying the model to demonstrate a security flaw in MULTICS, extended the model to provide a finer control on user discretion. She introduced the concept of property sets as a restriction on the behavior of subjects and added procedure objects (a new node-type) and rights for creating and invoking them.

Like the UCLA DSU model, the take-grant model does not include security classes. Subjects and objects are not distinguished according to clearance levels or security levels. The levels could be added by labeling subjects and objects and by restricting the graph rewriting rules according to the lattice relations. The likely result, in the case of the military security lattice, would be a graph theoretic formulation of the Bell and LaPadula model.

6.5 Bell and LaPadula Model

As part of its computer security program, the Air Force sponsored the construction of some prototype security kernels and some formal models for computer security. The principal prototype efforts were conducted at MITRE and (sponsored by DARPA) at UCLA, while the research in formal models was performed both at Case Western Reserve University, by Walter, *et al.* [44-46], and at MITRE, by Bell and LaPadula [14,47-50]. These prototype and model developments were seminal; current efforts to build "kernelized" systems are based on the same ideas and use security models similar to the ones developed in the Case Western and MITRE projects. Both of these models are formalizations and specializations of the access matrix model to incorporate military security policy. Because the models developed at Case and at MITRE are so similar, only the latter (Bell and LaPadula) version is described here.

Bell and LaPadula use finite state machines to formalize their model. They define the various components of the finite state machine, define what it means (formally) for a given state to be secure, and then consider the transitions that can be allowed so that a secure state can never lead to an insecure state.

Although the presentations in the original reports carry a heavy burden of notation from systems theory, the model can be understood informally without the notation. In addition to the subjects and objects of the access matrix model, it includes the security levels of the military security system: each subject has a clearance and each object has a classification. Each subject also has a *current security level*, which may not exceed the subject's clearance.

The access matrix is defined as above, and four modes of access are named and specified as follows:

- Read-only: subject can read the object, but not modify it;
- Append: subject can write the object but cannot read it;
- Execute: subject can execute the object but cannot read or write it directly; and
- Read-write: subject can both read and write the object.

A control attribute, which is like an ownership flag, is also defined. It allows a subject to pass to other subjects some or all of the access modes it possesses for the controlled object. The control attribute itself cannot be passed to other subjects; it is granted to the subject that created the object.

Creation of objects is viewed as a two-part operation: (1) addition of a new inactive object to the existing set of objects, and (2) activation of an inactive object. The *tranquility principle* asserts that no operation may change the classification of an active object. Bell and LaPadula state and adopt this principle, although they recognize that it is not required by military security structures.

For a state to be secure, two properties must hold:

- (1) The *simple security property*: no subject has read-access to any object that has a classification greater than the clearance of the subject; and
- (2) The **-property* (pronounced "star-property"): no subject has append-access to an object whose security level is not at least the current security level of the subject; no subject has read-write access to an object whose security level is not equal to the current security level of the subject; and no subject has read access to an object whose security level is not, at most, the current security level of the subject.

A set of rules governing the transition from one state to another is also given. These rules are analogous to the example rules given by Graham and Denning for altering an access matrix, and are required to preserve the two security properties. The particular rules defined by Bell and LaPadula provide the following functions:

- (1) Get (read, append, execute, or read-write) access, to initiate access to an object by a subject in the requested mode;
- (2) Release (read, append, execute, or read-write) access, the inverse of get access;
- (3) Give (read, append, execute, or read-write) access, to allow the controller of an object to extend the designated access to another subject;
- (4) Rescind (read, append, execute, or read-write) access, the inverse of give access;
- (5) Create object, to activate an inactive object;
- (6) Delete object, to deactivate an active object; and
- (7) Change security level, to allow a subject to alter its current security level.

With the formal definition of each rule is given a set of restrictions on the application of the rule to generate a new system state. For example, a subject can only give or rescind access to an object if the subject has the control attribute for that object, and a subject can only get read-access to an object if the security level of the object is at most the current security level of the subject. In Ref. 48, it is demonstrated that each of the specified rules preserves the security property and the *-property. Since none of the rules affects the classifications of active objects, the rules obey the tranquility principle as well.

The definition of the *-property given above is taken from Ref. 48, p. 30 and Ref. 50, p. 83. Bell and LaPadula also develop the notion of *trusted subjects*. A trusted subject is one that can be relied on not to compromise security even if some of its current accesses violate the *-property; the *-property need only be enforced on requests made by untrusted subjects. The definition of this class of subjects recognizes that the *-property is more stringent than military security requires. The version of the *-property given above actually includes the simple security property as well, since the current security level of the subject can never exceed the clearance of the subject. Despite the detailed definition given by Bell and LaPadula, the term "*-property" today is usually identified only with the prohibition of "writing down" (i.e., the restriction on read-write and append modes of access), and the simple security property (or simple security *condition*) is still identified with the restriction on "reading up" (i.e., the restriction on read access).

The description of the Bell and LaPadula model given so far considers only a "flat" set of objects—objects are atomic elements, each with a single classification and containing no distinguishable subelements. In Ref. 48, the model is extended to include hierarchies of objects, so that a MULTICS-like tree-structured directory can be included. The report credits the group at Case Western Reserve as the originators of this aspect of the model. The approach in the extended model is to define a set of objects in a hierarchy to be *compatible* with the model if any path from the root node outward encounters objects with monotonically nondecreasing classification levels. This aspect of the model has rarely been exploited in subsequent kernel development efforts.

Since the Bell and LaPadula model was documented in Refs. 14 and 49, it has been modified and reformulated in some respects as it has been applied to various design and implementation projects. The particular set of rules developed and proven by Bell and LaPadula is not integral to the model and is not generally used, although any system based on an access matrix model will have similar ones. The names used for modes of access are typically read (for read-only) and write or modify (for write-

only), and these modes of access can be granted independently. Execute-access is generally unused (although the model itself specifies no restrictions on its use), perhaps because few systems can effectively enforce execute-only access.

Perhaps the most succinct and widely used restatement of the Bell and LaPadula model is given by Feiertag, *et al.* [21]. They define the model in terms of subjects, objects, modify operations, and read operations. Each subject and object is assigned a security level, and the following five axioms govern the behavior of the system:

Simple security condition: a subject can read an object only if the security level of the subject is at least that of the object;

*-property: a subject can modify an object O_1 in a manner dependent on an object O_2 only if the security level of O_1 is at least that of O_2 .

Tranquility principle: a subject cannot change the security level of an active object;

Nonaccessibility of inactive objects: a subject cannot read the contents of an inactive object;

Rewriting of inactive objects: A newly activated object is given an initial state independent of any previous incarnation of the object.

In addition to their restatement of the model, the authors develop a nearly equivalent model that is more amenable to automated proofs of security. It is this revised model that has been (and is being) used in the automated verification of a number of systems now under development. This somewhat more restrictive model incorporates the notion of information flow described below.

Designs and implementations based on the Bell and LaPadula model, or modifications of it, include the security enhancements to MULTICS for the Air Force Data Services Center [51], the MITRE brassboard kernel [52,53], the SIGMA message system used in the Military Message Experiment [54], the Kernelized Secure Operating System (KSOS) for the PDP-11 [55], the Secure Communications Processor (SCOMP, also known as KSOS-6) for the Honeywell Level 6 [56], and Kernelized VM/370 (KVM/370) [57]. The UCLA kernel [13,35,36] and the Provably Secure Operating System (PSOS) design [58,59] are based on a separation of enforcement mechanisms from security policy. These systems are based on the use of capabilities for referencing objects: proofs about the enforcement mechanisms must demonstrate that the mechanism cannot be subverted or circumvented. Separate proofs would be needed to show that a particular use of the mechanism correctly enforces a particular policy, but in both of these systems, the Bell and LaPadula model seems the intended policy for military applications.

When it was first constructed, the Bell and LaPadula model (and the Walter, *et al.*, model as well) was a significant advance in defining military security concepts formally in a way applicable to computer systems. It has served as the basis for several design, prototype, and implementation efforts. Partly because of these efforts, some problems with it have been discovered. The static representation it provides for military security is restrictive; although hierarchies of objects are provided for [48], the model does not lay out an appropriate set of axioms governing references to multilevel objects. The dynamics of security, reclassification, sanitization, and downgrading can only be handled using the trusted process concept, and the model gives little guidance for determining which processes can be trusted. As originally formulated, some of the rules in the model allowed information to be transmitted improperly through control variables, (storage channels) but the work done by Walter, *et al.*, recognized this problem. In their final form, the rules of the model do not contain storage channels, but timing channels can exist. In sum, the principal problems with the model are not in the things it allows, but in those it disallows: many operations that are in fact secure will be disallowed by the model. Systems based on

the model are then faced with the choice between obeying the model, but imposing severe constraints on functionality, and allowing the desired functions by relying heavily on trusted processes.

6.6 Information-Flow Models

The significance of the concept of information flow is that it focuses on the actual operations that transfer information between objects. Access control models (such as the original Bell and LaPadula model) represent instead the transfer or exercise by subjects of access rights to objects. Information-flow models can be applied to the variables in a program directly, while the access matrix models generally apply to larger objects such as files and processes. In an unpublished memorandum, Millen has pointed out that a set of rules specified according to the Bell and LaPadula model could preserve both the security and *-properties but could nevertheless contain storage channels, and that these channels would be exposed by an analysis of the information flow in the rules. The channels in Millen's example are introduced by the return code given when a request to apply a rule is denied. The information passed by this code is neglected under the axioms of the Bell and LaPadula model, but is disclosed by an information-flow analysis. Information-flow models thus appear to provide greater precision than access matrix models.

This is not to say that flow models eliminate the need for access control models; an access matrix can still be useful for specifying access policies (e.g., defining controls on user access to files). Like the Bell and LaPadula model, the flow model can detect both legitimate and storage channels but not timing channels. Also like the Bell and LaPadula model, there are programs that would be insecure in terms of the model, but would not in fact be insecure (i.e., the model provides constraints that are sufficient, but not necessary). For an example, see Ref. 60.

Apparently, Popek [61] was the first to note explicitly the applicability of partial orders in the context of access control. The ADEPT-50 system [11] had earlier implemented a lattice structure without noting its significance. Fenton [18] developed a structure in which data transfers were controlled according to a partial order relating the sensitivity of the data and the protection level of its destination. Walter, *et al.* [44] provided an early description of the military classification structure as a lattice, and Denning [19,62] introduced the concept of information flow as the ordering relation on the set of classifications. A certification mechanism for verifying the secure flow of information through a program is presented in Ref. 63. Andrews and Reitman have developed a logic for proving assertions about the flow properties of programs [64]; their work is presented following that of the Dennings. As described in a subsequent section, the flow model has in some respects been incorporated into the SRI version of the Bell and LaPadula model.

At about the same time Denning's work appeared, people at MITRE realized that the variables within the security kernel itself could act as information paths: information recorded in a kernel variable as a result of a kernel call by process A might be visible to a later kernel call by process B. If B has a lower security level than A, a violation may have occurred. This corresponds to a *flow* of information from a higher level to a lower level, even though the simple security and *-properties have both been enforced. Millen noted this problem [53].

The flow model, compared with the Bell and LaPadula model, is relatively uncomplicated. Instead of a series of conditions and properties to be maintained, there is the single requirement that information flow obey the lattice structure described below. Although the military security system had been earlier characterized as a lattice [44], Denning's presentation makes it clear that the lattice model is of more general significance.

An information-flow model has five components:

- (1) A set of objects, representing information receptacles (e.g., files, program variables, bits),

- (2) A set of processes, representing the active agents responsible for information flow,
- (3) A set of security classes, corresponding to disjoint classes of information,
- (4) An associative and commutative class-combining operator that specifies the class of the information generated by any binary operation on information from two classes, and
- (5) A flow relation that, for any pair of security classes, determines whether information is allowed to flow from one to the other.

Under a set of assumptions that is justified in Ref. 62 as applying to nearly any rationally specified flow model, the set of security classes, the flow relation, and the class combining operator form a lattice. Maintaining secure information flow in the modeled system corresponds to ensuring that actual information flows between objects do not violate the specified flow relation. This problem is addressed primarily in the context of programming languages.

Information flows from an object x to an object y whenever information stored in x is transferred directly to y or used to derive information transferred to y . Two kinds of information flow, *explicit* and *implicit*, are identified. A flow from x to y is explicit if the operations causing it are independent of the value of x (e.g., in a statement directly assigning the value of x to y). It is implicit if the statement specifies a flow from some other object z to y , but execution of the statement depends on the value of x (e.g., in the conditional assignment

$$\text{if } x \text{ then } y := z;$$

information about the value of x can flow into y whether or not the assignment is executed).

According to this model, a program is secure if it does not specify any information flows that violate the given flow relation. Denning primarily treats the case of static binding, in which objects are bound to security levels at compile-time (this assumption corresponds roughly to the tranquility property in the Bell and LaPadula model). In Ref. 63, rules for compile time certification of secure information flow are provided. The case of dynamic binding, in which the security level of some objects can change during program execution, is discussed briefly in Refs. 19 and 62. The work of Andrews and Reitman is based on dynamic binding; Reitman also presents a certification mechanism for parallel programs with static binding [65].

The formulation of information-flow models that Andrews and Reitman use is essentially the same as Denning's; they focus on *programs*, which have three relevant components: *variables*, which contain information; an *information state*, which is a mapping from the variables to the set of security classes; and *statements*, which modify variables and thus alter the information state. Statements correspond roughly to subjects since they are responsible for causing information flow, and variables correspond to objects. Since the security classes are assumed to be finite, partially ordered, and to have a least upper-bound operator, they again form a lattice. Variables are dynamically bound to security classes; transfer of information into a variable causes the variable to assume a classification in accordance with the transfer.

In addition to the explicit and implicit flows identified in the Dennings' work, (referred to in Ref. 64 as direct and indirect flows), Andrews and Reitman distinguish two types of implicit (indirect) flows: local and global. A local flow is an implicit flow within a statement, such as the flow from a boolean condition to the statements within an alternation or iteration statement. Global flows are implicit flows between statements. Sources of global flows include process synchronization statements and iterations (in the case that termination of the iteration is not guaranteed, execution of the following statements conveys the information that the loop terminated). To characterize these flows, two auxiliary variables, named "local" and "global," are introduced to record the current classification of local and global flows.

Based on this model, the authors develop axioms for the information-flow semantics of assignment, alternation, iteration, composition, and procedure invocation. These axioms correspond to the axioms developed by Hoare [66] for the semantics of a programming language, but they deal only with the information flows (both explicit and implicit) that can be generated by the various types of statements in a language. The axioms are based on the lattice model for information flow, but do not otherwise incorporate a specific policy. Following the development of these axioms for sequential programs, axioms for concurrent execution and for synchronization via the semaphore operations "wait" and "signal" are developed. These allow proofs of information-flow properties to be constructed for a class of parallel programs.

Andrews and Reitman distinguish an *access policy*, which specifies the rights that subjects have to objects, from an *information-flow policy*, which specifies the classes of information that can be contained in objects and the relations between object classes. To a point, these policies are interchangeable, or at least dependent: restrictions on a subject's access to an object will presumably restrict the flow of information (and hence the information that can be contained in a particular object). Conversely, restrictions on flow will have an effect on what access rights a given subject can exercise for a given object. Nevertheless, this distinction clarifies the perspectives from which an access matrix model and an information-flow model view a computer system.

Flow proofs demonstrate that a given set of flow assertions (e.g., that the security class of x is dominated by the security class of y) holds at a particular place in a program. A *flow policy*, if formulated as a set of flow assertions, can then be proven to hold (or not to hold) at particular points in the execution of a program. Andrews and Reitman distinguish two types of policies: *final value policies*, which only require that the assertions hold on termination of the program, and *high water mark policies*, which must be true for each information state in a program.

Reitman [65] presents a compile-time certification mechanism for parallel programs with static binding of variables to security classes. This mechanism is essentially an extension of the one developed by the Dennings [63] to include the structures for parallel programming for which flow axioms are developed [64]. Because it requires static binding, this mechanism is less general than the flow proofs [64], but this restriction makes possible compile-time certification that a program obeys a particular policy. The policies that may be used, of course, must also abide by the static binding restriction.

6.7 Extensions and Applications of the Bell and LaPadula Model

Since its original exposition, the Bell and LaPadula model has been extended, applied, and reformulated by several different authors. Modifications and applications of three kinds are described in this section: (1) addition of the concept of integrity to the model, (2) the application and extension to model database management systems, and (3) the reformulation of the model for use with automated verification tools.

Integrity

The Bell and LaPadula model is concerned with preventing the improper disclosure of information. Consequently, subjects are prevented from reading information for which they are not cleared and from writing (transmitting) information to subjects at lower clearance levels. Biba [20] noticed a class of threats, based on the improper *modification* of information, that this model neglects. These threats arise because there is often information that must be visible to users at all security levels but should only be modified in controlled ways by authorized agents. The controls on modification in the Bell and LaPadula model do not cover this case because they are based only on the sensitivity to the *disclosure* of that information.

As a remedy, Biba introduces the concept of integrity levels and integrity policy. The integrity level of information is based on the damage to national security its unauthorized modification could cause. Integrity compartments are defined analogously to security compartments, with different compartments reflecting different functional areas.

The integrity levels and compartments are ordered, as are sensitivity levels and compartments, to form an integrity lattice. Biba uses the same names for integrity levels as are used for security levels, with top secret integrity corresponding to information most sensitive to unauthorized modification (or in his words, sabotage). This choice is unfortunate, since information with "top secret" integrity may not be secret at all. Integrity levels for subjects correspond to clearances.

Biba also provides some example integrity policies that correspond to security policies. A "low water mark" integrity policy sets the integrity level of a subject to the lowest level of any object observed by that subject, and a subject can only modify objects dominated by the subject's current integrity level. Alternatively, the integrity level of any modified object can be reduced to the minimum of its current integrity level and the current integrity level of the subject performing the modification. A policy of "strict integrity" is the dual of the Bell and LaPadula security policy (interchanging "read" and "write" and substituting "integrity" for "security" in the original rules): a subject can only read objects with integrity at least as great as its own, and can only write objects with integrity less than or equal to its own. Bonyun [67] asserts that a policy of "strict integrity" will be too constraining to be useful and proposes an alternative that is slightly weaker than Biba's; Bonyun refers to it as a "semi-dual" of security and integrates it with an approach to handling the aggregation problem.

There has been little experience to date with integrity policies. In manual systems, the integrity problem is substantially reduced, since it is difficult to accidentally or maliciously modify information without detection. Both the KSOS and SCOMP kernels are to provide integrity controls according to the strict integrity model, but the integrity levels to be supported have only been specified as system administrator (highest), operator, and user (lowest). It is unclear exactly how integrity levels will be assigned to various system objects. Although the integrity problem has apparently only been examined in a military context to date, it seems clear that it can arise in civilian applications as well: consider the effect of an unauthorized modification of a mailing address in an electronic mail system.

Database Management Systems

An application that has been of particular interest since the beginning of work on secure computer systems is the implementation of a secure database management system (DBMS). As part of the work sponsored by the Air Force to develop a secure version of MULTICS, Hinke and Schaefer [22] provided an interpretation of the Bell and LaPadula model for a relational database implemented on top of a MULTICS file system. The considerations as to how classifications should be applied to the elements of a relational database and how the database can be mapped onto the objects protected by a secure MULTICS are lengthy, but the underlying Bell and LaPadula model is used essentially unchanged. As this work preceded Biba's, integrity is not a part of the model employed. The authors do consider the use of access control lists (referred to as "need-to-know" lists) to regulate discretionary access to files, but they note that strict observance of the *-property would require that the access list for a new file would be the mutual intersection of all of the previous access lists of objects read by the process performing the write. They recommend against such a policy, on the grounds that it is likely to result in the user writing data that only he can read (i.e., the intersection of all of the access control lists referenced will tend to be only the ID associated with the process doing the reading).

In applying classifications to the structure of a relational DBMS implemented on a MULTICS-based security kernel, Hinke and Schaefer recommend that classifications be attached to the fields of a relation, as opposed to classifying specific entries of fields of relations or classifying entire relations at a single level. For example, if a relation were defined between the fields "supplier" and "part," all entries of the supplier field would be classified at a single level and all entries of the part field would be

classified at a single (perhaps different) level. Each field would be stored in a MULTICS segment with a classification corresponding to that field. Classifications of segments and accesses to segments would be controlled by the MULTICS security kernel, so that the security of the database management system would depend only on the security kernel. The authors also develop a number of rules concerning the ordering of classifications of various field of relations, depending on whether a field is a primary key for the relation. These rules are generally derived from the properties of the Bell and LaPadula model.

Later work by Grohn [23] takes a more formal approach to modeling a secure database management system. Starting with the Bell and LaPadula model, Grohn extends it to include integrity levels and compartments. Each object in the model has both a security level and an integrity level; together these compose its protection level. Integrity properties are defined as the formal duals of the security properties. The tranquility property applies to both security and integrity, and there is a discretionary integrity property in addition to discretionary security.

Grohn also alters the directory structure of the Bell and LaPadula model. In his model the directory structure partitions the objects by protection level: each directory contains the identifiers of all objects of a given protection level. The directory itself is assigned that same protection level. A set of lattice directory functions is also defined which, given the level of a directory, generates a list of all existing directories that dominate that level and a list of all directories dominated by that level. These functions allow a subject to enumerate all of the objects accessible to it (directories are exempt from discretionary access controls). In the Bell and LaPadula model, the directory structure is hierarchical, with the requirement that any node must have a classification that is dominated by its successors (i.e., the root of each tree must be its least classified node). There is no guarantee that a subject can enumerate all of the objects classified at or below its clearance.

The approach Grohn takes to implementing a relational DBMS on his model differs from that of Hinke and Schaefer in the unit to which a classification is applied. Instead of classifying each field (domain) of a relation, he favors classifying only the relation as a whole. He argues that, for convenience of observation and modification, all fields would have to be at the same level of classification anyway, and that this requirement is equivalent to placing a classification on the relation as a whole instead of on each field.

Reformulation for Use with Automated Program Verifiers

As part of efforts to specify a Provably Secure Operating System (PSOS) and to verify the Kernelized Secure Operating System (KSOS), Feiertag and others from SRI International reformulated the model and altered it slightly to simplify its use in proving theorems about systems specified with formal languages. The reformulation allows the proof of security to be factored into smaller pieces by assigning each function reference (a function reference is a function call with a particular set of arguments) and state variable a specific security level, so that the security level of each data item referenced in the specification of a function can be compared to the level of the function reference. Proofs in KSOS are intended to cover only the security kernel, while in PSOS the entire system specification is to be verified. Although the revised models described in Refs. 21, 58, and 68 are presented as reformulations of the Bell and LaPadula model, they embody the concepts of information-flow models. Because it is the most recent version of the model and the one that has been carried farthest in implementation of a production operating system, the KSOS version of the model is described here.

The KSOS model informally defines a system as multilevel secure if, for any two processes HS (operating at a high security level) and LS (operating at a lower* security level), HS can do nothing to affect in any way the operation of LS. In this case, LS can know nothing of HS (it may not even know that HS exists) since it could only gain such knowledge if HS had somehow influenced its behavior.

*Actually, lower or incomparable: since security levels are only partially ordered, it is possible that for levels L1 and L2, neither $L1 \geq L2$ nor $L2 \geq L1$. An example of two such levels would be "SECRET, NUCLEAR," and "SECRET, NATO."

Since information cannot be transmitted from a process at a higher security level to one at a lower level, information can only flow upward in the security lattice or remain at the same level. The similarity of this general model to the information-flow models is apparent. Integrity is defined informally in a parallel way: a process LI (operating at a low integrity level) can do nothing to affect the operation of a process HI (operating at an integrity level greater than or incomparable with that of LI).

This informal model is developed formally in two steps. First, a general model is defined in which the system is characterized by its function references and a relation called "information transmission." One function reference transmits information to another if there is any possibility that the information returned by the second function reference is affected by the first function reference. Security and integrity levels are associated with each function reference; in practice, these levels would be the current levels of the process issuing the function reference.

This model is brought closer to actual system specifications by including state variables as well as function references. Each state variable has an assigned security and integrity level. Function references may depend on some state variables and may change other state variables. (These functional dependencies replace the notions of reading and writing state variables in the Bell and LaPadula model.) The constraints on functional dependencies are the following:

- a. If function reference f depends on state variable v , then the security level of v is less than or equal to the security level of f and the integrity level of v is greater than or equal to the integrity level of f ;
- b. If function reference f may cause the value of state variable v_2 to change in a way dependent on state variable v_1 , then the security level of v_1 is less than or equal to that of v_2 and the integrity level of v_1 is greater than or equal to that of v_2 ;
- c. If function reference f may affect the value of state variable v , then the security level of v is greater than or equal to that of f and the integrity level of f is greater than or equal to that of v .

Properties a and c correspond approximately to the simple security property and the *-property, respectively, of the Bell and LaPadula model. Property b addresses functions that may both read and write state variables; in the Bell and LaPadula model such functions could only be obtained by composition of operations that individually either read or write state variables.

Finally, property b is split and modified to handle the complexities introduced by systems that have "trusted" functions and that allow side effects at higher levels of functions invoked from lower levels.* By enforcing only the desired half of the split version, a system can allow information to flow downward in restricted cases. The underlined parts of b'' denote differences from b' :

- b' . If function reference f may cause the value of state variable v_2 to change in a way dependent on state variable v_1 , then
 - (i) either the security level of v_1 is less than or equal to that of v_2 or the security level of f is greater than or equal to that of v_1 and
 - (ii) either the integrity level of v_1 is greater than or equal to that of v_2 or the integrity level of f is less than or equal to that of v_1 .

*Discussions with R. Feiertag, one of the authors of the model, disclosed that the published version of property b' [68] contains two erroneous equal signs, and property b'' is lacking.

b''. If function reference f may cause the value of state variable v_2 to change in a way dependent on state variable v_1 , then

(i) either the security level of v_1 is less than or equal to that of v_2 or the security level of v_2 is greater than or equal to that of f and

(ii) either the integrity level of v_1 is greater than or equal to that of v_2 or the integrity level of f is greater than or equal to that of v_2 .

Typically, a trusted function may be allowed to violate the *-property (for security) or the simple integrity property, but will be required to enforce the other properties of the model. In the final model, properties $P2a$ and $P2b'$ compose the simple security property; properties $P2b''$ and $P2c$ compose the *-property. If a system function is to be allowed to violate the *-property, only $P2a$ and $P2b'$ must be enforced. Functions allowed to violate simple integrity must still obey properties $P2b''$ and $P2c$.

This adaptation of the Bell and LaPadula model gains much of its importance from its integration with the automated tools for program-specification (using the SPECIAL language [69]) and theorem-proving (using the Boyer-Moore theorem prover [70]) also developed at SRI. These tools, including the Multilevel Security Formula Generator, which incorporates this model, are being used to verify the security properties of system specifications in a number of current projects (e.g., work reported by McCauley on KSOS and by Bonneau on SCOMP [55,56]).

6.8 Programs as Channels for Information Transmission

In different ways, each of the final three models views a program as a medium for information transmission. The key question for them becomes exactly what information is conveyed by the execution of a program and what deductions about protected information are possible. The appeal of these approaches lies in their comprehensiveness. Their drawback is that in their present state, none of them is ready to be applied to actual development of a system.

The work by Jones and Lipton on filters views a protection mechanism as a filter on the information passed from a program's inputs to its outputs. Cohen first takes an information-theoretic view: a program transmits information to the extent that variety in its initial state induces variety in its final state. Later, he develops a similar structure based on the deductions that can be made about the initial state of a set of variables given the final state of a variable. Millen and Furtek also attempt to formalize the notion of deduction in their work on constraints. They view the execution of a program as a sequence of states determined by the input to the program and the transitions allowed by the program's structure. An observer, knowing the allowable states of the system and able to view portions of the actual sequence of states that occurs, may be able to deduce things about the remaining (hidden) portions of the state sequence.

Filters

Jones and Lipton have formulated a model to capture definitions of both a security policy and a protection mechanism intended to enforce that policy [24,71]. A policy, given in nonprocedural form, defines who may use what information in the system. The protection mechanism tells how the information is to be protected and is stated procedurally. The soundness of a mechanism with respect to a policy is determined by how well it enforces the policy.

A *program* is characterized as a function from the Cartesian product of the domains of all program inputs to the domain of the output. The *observability postulate* asserts that all available information about the program inputs (including, for example, execution time) must be encoded in the output value. A *protection mechanism* is defined as a function relative to a program: it maps the input domain of the program to an output domain expanded to include a set of *violation notices*. Given an input, the

protection mechanism must either produce the same output as does the program or it must produce a violation notice (e.g., it may refuse to provide a requested piece of information). A *security policy* is also defined with respect to a program: a security policy is a function that maps from the input domain of the program to some subset of that domain. The policy thus acts as a *filter* on the program inputs. An observer of the program's output should only be able to get information about the subset of the inputs that passes through the security policy filter. In the cited references, this subset is always formed by simply eliminating some of the input variables. A finer control of policy might be accomplished by restricting a variable to a particular range of values instead of eliminating it altogether. A protection mechanism is *sound* if the outputs it produces are the same as if it had received the input as filtered by the security policy instead of the actual input.

A partial ordering of protection mechanisms for a given program and security policy is developed. A sound mechanism M_1 is more *complete* than another sound mechanism M_2 if, for all inputs on which M_2 returns the same value as the original program, M_1 also returns that value, and, for at least one input for which M_2 returns a violation notice, M_1 does not return one. Based on this definition, it is established that for a given program and policy, there is a "maximal" protection mechanism.

To illustrate the use of this framework, the authors develop a "surveillance protection mechanism" to enforce a restricted class of security policies on programs constructed in a simple flowchart language. For each variable (input, output, temporary, and location counter) in the original program, a corresponding surveillance variable is added. The value of the surveillance variable is a set of indices that define which program variables have influenced the value of the variable under surveillance. Each component of the original flowchart program is replaced by a modified one that both performs the original function and updates the appropriate surveillance variables. At the termination of the program, the values of the surveillance variables can be checked against the requirements of the security policy to determine whether a particular result can be reported to a user or not.

The surveillance protection mechanism is proven to be sound if running times are not observable, and a modified version of it is proven sound even if running times are visible. Finally, it is shown that there is no effective procedure for finding a maximal protection mechanism for an arbitrary program and security policy.

The surveillance protection mechanism seems to have much in common with the flow-analysis approach Denning applied to programming language constructs [19]. The model as a whole provides clean formal definitions of security policy, protection mechanism, soundness, and completeness, but has found little application in practice. To model a system constrained by military security policy, a program-independent formulation of that policy within the model framework would be required.

Strong Dependency

In an effort to provide a formal basis for reasoning about information transmission in programs, Cohen has developed an approach he calls *strong dependency* [25,72]. This approach is based on the notion, fundamental to information theory, that information is the transmission of *variety* from a sender to a receiver. For example, if the sender can be in one of three states, and it sends a different message to the receiver corresponding to each state, all of the variety in the sender is transmitted to the receiver. If only two different messages are possible, some of the variety in the sender is not available to the receiver.

Consider a sequential program P with a set of variables A and a particular variable b . If two executions of P , starting in initial states that differ only in their values for some variable(s) in A , can lead to terminations with two different values for b , then b is said to be *strongly dependent* on A over the execution of P . Note that this definition only requires that *some* two different values for variables in A lead to different values for b , not all.

Two variables can be strongly dependent, even if there are states in which no variety is conveyed from one to the other. For example, the program statement

$$\text{if } m = 0 \text{ then } b := a + 1;$$

never transfers information from a to b if m is always nonzero when the statement is executed. In this case, the assertion $[m \text{ nonzero}]$ eliminates the variety that would have been conveyed by the program.

If a set of constraints covers all possible cases of the variables in a set A and if the truth value of each constraint is unaffected by the values of variables outside that set, the set is said to be a *cover* for A and to be *A-strict*. A variable b is defined to be *selectively independent* of A over the execution of program P with respect to a constraint set if that constraint set is a cover and is *A-strict*, and if for each individual constraint b is not strongly dependent on A . Cohen also develops the idea of *relative autonomy* of a constraint with respect to a set A of variables. Roughly, a constraint is autonomous relative to A if it does not relate variables within A to variables outside of A .

Based on these definitions, Cohen [25] formalizes the same sort of information-flow properties for lattice structures as did Denning [19]. He also provides a statement of the confinement problem in this framework. In more recently published work [72], Cohen discusses difficulties with strong dependency when non-autonomous constraints are used. Taking a deductive, instead of information-theoretic, approach, he develops a formally equivalent model related to the work of Jones and Lipton [24,71]. The deductive view asserts that information has been transferred from set A to variable b by the execution of a program, if the final value of b can be used to deduce information about the initial values of variables in A .

Working with this reformulation and using a formalism derived from projective logic, he develops *definitive dependency* and *contingent dependency*. Definitive dependency arises from the idea that one constraint may provide more information about the state of A than another; the stronger constraint is the more definitive. In some cases, a constraint may provide definitive information about A only if some additional information, not concerning A , is given. Such a constraint is *A-contingent*, and contingent dependency is defined accordingly. Cohen demonstrates that contingent dependency is equivalent to strong dependency if relatively autonomous constraints (or no constraints) are given. In addition, contingent dependency can model the transmission of information with nonautonomous constraints.

Although Cohen has developed proof rules based on strong dependency for four basic programming language constructs (assignment, sequence, alternation, and iteration), his work has yet to be applied in current efforts to model and to build secure systems.

Constraints

Given the definition of a finite state machine, the set of possible sequences of states through which the machine can cycle can be determined. In addition, it is possible to define sequences of states that cannot occur. For example, in a typical pushdown stack, the sequence "push the value 1 onto the stack" followed immediately by "pop the stack, returning value 0" would be excluded. Thus an observer, given the machine definition and an observation of the event (pop,0) at transition n could deduce that (push,1) could not have occurred at transition $n-1$. Furtek and Millen have developed a theory of *constraints* that models the deductions a user can make about a system in this way [26,27,73-76].

A constraint specifies a sequence of states that cannot occur; the development of constraints is analogous to the development of implicants in switching theory. A variable can assume any of a range of values. If v is a possible value for a variable a , then a_v is a *condition*. The condition is *satisfied* by a system state q if $a = v$ in q . A *term* R is a *conjunction of conditions in which each variable in the system appears at most once*. A *term is satisfied by a system state if all of its conditions are satisfied*. To introduce restrictions on a sequence of system states, a Cartesian product (called a *symbolic product*) of conditions is formed; the symbolic product is satisfied by a state sequence only if each state in the sequence satisfies

the corresponding term in the product. If there is no sequence of states (called a *simulation*) allowed by the machine's definition that satisfies a given symbolic product, that product is called a *constraint*. A constraint that is a symbolic product of n terms is called an *n-place constraint*. A constraint is *prime* if and only if deleting any of its conditions results in a symbolic product that is not a constraint. If the values of all but one of the variables occurring in a prime constraint are known to (or can be controlled by) an observer, then he can deduce something about the remaining one. (Specifically, he can exclude at least one possible value for that variable at one point in the simulation.) A *cover* is a set of two-place constraints such that each disallowed state transition satisfies some constraint in the cover.

Security can be introduced into this model by providing a mapping for each variable to one of the levels in the usual security lattice. The variables of the system may also be partitioned into those that are internal to the system (not directly observable) and those that are external input or output variables. A system may then be defined to be secure against unauthorized disclosures if no user at a level s can deduce anything about the value of an individual input at a higher or incomparable level by observing external variables at level s or below and/or control of inputs at any level. This definition is equivalent to requiring that, for any prime constraint in which only input and output variables occur, the least upper bound of the levels of the inputs is less than or equal to the least upper bound of the levels of the outputs.

In practice, the prime constraints for even a simple system can be arbitrarily long, and there can be arbitrarily many prime constraints; however, the set of all prime constraints for a given system forms a regular language. Furtek [75] has written a program that accepts a set of two-place constraints and generates a finite-state acceptor for the set of all prime constraints. Millen [27] develops a sufficient condition for security in systems that can be characterized by *simple* constraints of the form:

$$p \times a_v$$

where p is an arbitrary term and a_v represents any single condition. This condition is related to the *-property of Bell and LaPadula, and is called the *monotonicity condition*. Given an assignment of external variables to security levels, an extension of that assignment to all variables is *monotone* with respect to a simple cover (a cover consisting of simple constraints) if, for all variables a and constraints $p \times b_v$ in the cover, if a occurs in p then the security level assigned to a is less than or equal to the level assigned to b . Systems for which there is a monotone extension of the external level assignment are shown to be secure in the sense defined above [74].

The appeal of this approach lies in its ability to define a necessary and sufficient condition for a system to be secure (this definition is the first one given; the monotonicity condition is sufficient but not necessary). As in Cohen's approach, the authors carefully define what a deduction is and then construct a model in which deductions can be controlled. Unfortunately, the specification of a system in terms of its constraints can be a difficult problem even with automated aids for generating prime constraints from two-place constraints, and so the practicality of the approach remains to be demonstrated.

7.0 DISCUSSION

Table 1 compares the models discussed above with respect to motivation, approach, view of security, and use. A useful comparison of models should examine both the definition of security and the feasibility of implementing a computer system that performs the application required of it and can be verified to simulate the model. Unfortunately, such a comparison is difficult because few implementations based on these models have reached a stage where their performance can be reliably estimated or where verification of their security properties can be attempted. Nevertheless, some of these models are better candidates as bases for future secure systems than others.

Each model defines its own world and its own concept of security in that world, and a computer system that truly simulates any of the models will be secure in the sense defined by that model. To say

Table 1 — Comparison of Properties of Models

Properties		Models [†]									
		A.M.	UCLA	T-G	HWM	B+L1	B+L2	Flow	Filt	S.D.	Cons
Motivation	Developed primarily to represent existing system(s)	X	X*		X*						
	Developed to guide construction of future systems			X		X	X	X	X	X	X
View of Security	Models access to objects without regard to contents	X	X	X	X	X					
	Models flow of information among objects						X	X			
	Models inferences that can be made about protected data								X	X	X
Approach	Model focuses on system structures (files, processes)	X	X	X	X	X	X				X
	Model focuses on language structures (variables, statements)							X	X	X	
	Model focuses on operations on capabilities		X	X							
	Model separates protection mechanism and security policy	X	X	X					X		
	Systems based on or represented by this model have been implemented	X	X		X		X				

[†] A.M. = Access Matrix; UCLA = Data Secure Unix; T-G = Take-grant; HWM = High Water Mark; B+L1 = Bell and LaPadula (original); B+L2 = Bell and LaPadula (revised); Flow = Information flow; Filt = Filters; S.D. = Strong dependency; Cons = Constraints

* While this model describes a single existing system, it could be used to guide the construction of future systems.

that certain channels are not "detected" by a model is really to say that certain structures and information flows found in implementations are difficult to map into the structures defined by that model. A problem common to all of the models is that they define security as absolute: an operation either is secure or it is not. This approach does not help the designer or implementer who must make trade-offs between security and performance.

In assessing the protection afforded by safes, for example, ratings are given based on the time it would take to break into the safe with tools reasonably available to the attacker. Cryptographic codes are rated based on their work factors—the time it would take to "break" the code given the tools of the cryptographer. Similar measures suitable for assessing the time it would take to defeat a particular safeguard or the rate of information flow over a particular timing channel in computer systems have yet to be formalized.

With respect to their definitions of security, the models can be divided roughly into three groups: those that are concerned only with controlling direct access to particular objects (access matrix model, UCLA DSU model, take-grant model); those that are concerned with information flows among objects assigned to security classes (information-flow model, revised Bell and LaPadula model); and those that are concerned with an observer's ability to deduce any information at all about particular variables (filters, strong dependency, constraints). (The high-water mark model falls between the first and second groups since it is concerned with the security levels of the objects a process touches over time, but it only controls direct accesses to objects.) The appropriateness of a particular model naturally depends on the application for which it is to be used. For the purposes of multilevel secure military systems, those in the first category require the addition of military security policy and the assessment of indirect information flows (e.g., timing and storage channels) in the implementation. Those in the second group are probably the closest in structure to the requirements for military applications, but often applications require more flexibility than these models permit. The models in the third category are the least-tested and would probably be the most difficult to use. Although their mathematical formulations are appealing, the restriction that users be unable to deduce anything at all about restricted information would be likely to lead to impractical restrictions on system behavior.

Formal verification of properties of system designs is still an active research topic. Security properties of the UCLA DSU model were proven to hold for substantial portions of that system, but only the Bell and LaPadula model has been applied in more than one formally specified system. This anomaly is explained by the fact that DoD has specified that the latter model be used in several of its secure system developments. The properties specified by the high-water mark, access matrix, and take-grant models could probably be stated in a form suitable for automated verification techniques should the demand arise. The properties required by the constraint, strong dependency, and filter models could be expressed similarly, but actually developing a system specification in the terms required by those models appears an insurmountable task at this time.

Most of the secure system developments using the (revised) Bell and LaPadula model have been based on the concept of a security kernel, and there have been problems in extending its use beyond the operating system to application systems. The question of whether the "three layer" approach—application programs running on an operating system emulator and the emulator running on a security kernel—can produce a system with acceptable performance is still open. As of this writing, the only kernel-based systems that appear to have adequate performance are based on an application program running directly on top of a kernel specially tailored for that application.

Initial performance measurements for KSOS-11 [55] indicate that it provides about one-tenth the computing power of similar hardware operating under unmodified UNIX. A UNIX interface is no longer planned for the SCOMP [56], but the hardware architecture of the Level-6 and the Security Protection Module developed for it are expected to yield better performance than observed in KSOS-11. Performance of KVM/370 [57] is estimated to be about half that of VM/370 on comparable hardware. None of these results has been published as of this writing, and all systems may improve with tuning.

Detailed questions of implementation and performance are beyond the scope of this survey, but it is clear that security is not to be had without a price.

What then lies ahead? In the immediate future, efforts to develop models for trusted processes operating within the framework of the Bell and LaPadula model will continue [77,78]. If the current developments of security-kernel based systems are successful and kernels become widely used in military systems, it is likely that civilian applications for security kernels will be identified as well. Though there will be exceptions, the lattice model will probably fit many of the requirements for security and privacy in the private sector.

An alternative to adding special models for trusted processes on top of the Bell and LaPadula model for the operating system is to develop integrated models tailored to particular applications [79]. A security model designed for a particular application could be used as a basis for the development of an application-specific security kernel. A key problem in this approach is to assure that the model incorporates the desired notion of security while permitting the operations required in the application.

Further off, if capability-based systems are successfully developed, models more appropriate to their structures may be used. The take and grant model is a possible candidate in this area, though it would require tailoring for specific applications. The Provably Secure Operating System (PSOS) [59], if built, could provide an appropriate vehicle for experimentation. The goal of PSOS is to apply verification techniques to the entire operating system specification rather than just to a security kernel. There are pressures in the private sector, as well, to produce systems that enforce privacy. A large timesharing vendor has recently undertaken the development of a capability-based system for the IBM 370 series architecture, largely in order to provide better guarantees of privacy between its customers [80].

8.0 CONCLUSION

Formal models for computer security are needed in order to organize the complexity inherent in both "computer" and "security." Without a precise definition of what security means and how a computer can behave, it is meaningless to ask whether a particular computer system is secure.

If complete isolation between certain users and certain sets of data is required, the modeling problem appears tractable. Most of the models surveyed above could adequately represent a system that provided such segregation. To be sure, difficulties remain: for example, in modeling the finiteness of system resources and programs that convey information through their usage of such resources over time. A more serious difficulty is that in most applications, total segregation is not acceptable.

Controlling the sharing of information in a computer is in fact a critical problem in operating system design. It should not be surprising that it is as slippery a problem when treated from the standpoint of computer security as it is in any other context.

Recognizing these difficulties, the designer of an application that has security requirements is well advised to state in advance the specific security properties (or, more generally, constraints on information transfer) desired of the system. If he is fortunate, these properties and the structure of the system may correspond directly to one of the models surveyed above. More likely, they will differ in some respects from all of the models. He must then choose whether to apply an existing model and to make explicit the cases that violate the model or else to create a new model based on the particular requirements of the application. In the military environment, the former approach is taken by systems being constructed based on the Bell and LaPadula model that utilize trusted processes to circumvent the rules of the model as particular applications require, but only relatively straightforward applications have been attempted. In nonmilitary systems, the sometimes conflicting demands of the laws governing access to medical and financial records challenge the designer of future models for computer security.

9.0 ACKNOWLEDGMENTS

First, thanks are due to the authors whose work is surveyed above. Many of them consented to review a draft of this paper and provided comments that were helpful in revising and reorganizing it. Particularly helpful were the comments I received from J. Millen, G. Andrews, R. Feiertag, D. Bonyun, R. Schell, L. Snyder, F. Furtek, P. Denning, and D. Denning. K. Shotting of the Department of Defense and E. Britton of the Defense Communications Agency also provided helpful reviews of the initial draft, as did my colleagues C. Heitmeyer and D. Parnas at the Naval Research Laboratory.

10.0 REFERENCES

1. D.E. Denning and P.J. Denning, "Data Security," *Computing Surveys* 11 (3), Sept. 1979, pp. 227-249.
2. OPNAVINST 5239.1, Department of the Navy, Chief of Naval Operations, Op-942E, Apr. 2, 1979.
3. D.E. Denning, P.J. Denning, and M.D. Schwartz, "The Tracker: A Threat to Statistical Database Security," *ACM Trans. Database Syst.*, 4 (1), Mar. 1979, pp. 76-96.
4. R.A. DeMillo, D. Dobkin, and R.J. Lipton, "Even Databases That Lie Can Be Compromised," *IEEE Trans. on Software Eng. SE-4* (1), Jan. 1977, pp. 74-75.
5. D. Dobkin, A.K. Jones, and R.J. Lipton, "Secure Data Bases: Protection Against User Influence [sic]," *ACM Trans. on Database Syst.*, 4 (1), Mar. 1979, pp. 97-106.
6. M.D. Schwartz, D.E. Denning, and P.J. Denning, "Linear Queries in Statistical Databases," *ACM Trans Database Syst.*, 4 (2), June 1979, pp. 156-167.
7. B.W. Lampson, "A Note on the Confinement Problem," *Commun. ACM*, 16 (10), Oct. 1973, pp. 613-615.
8. J.P. Anderson, *Computer Security Technology Planning Study*, ESD-TR-73-51, Vol. 1, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Oct. 1972 (NTIS AD-758 206).
9. S.H. Wilson, J.W. Kallander, N.M. Thomas, III, L.C. Klitzkie, and J.R. Bunch, Jr., "Military Message Experiment Quick Look Report," NRL Memorandum Report 3992, Naval Research Laboratory, Washington, D.C., April 1979, p. 10.
10. G. Birkhoff and T.C. Bartee, *Modern Applied Algebra*, McGraw-Hill, New York, 1970, p. 260.
11. C. Weissman, "Security Controls in the ADEPT-50 Time Sharing System," *Proc. 1969 AFIPS Fall Jt. Computer Conf.*, Vol. 35, AFIPS Press, Montvale, N.J., pp. 119-133.
12. M. Bishop and L. Snyder, "The Transfer of Information and Authority in a Protection System," *Proc. Seventh Symp. on Operating Systems Principles, ACM SIGOPS Operating Systems Rev.* 13 (4), Dec. 1979, pp. 45-54.
13. G.J. Popek and D.A. Farber, "A Model for Verification of Data Security in Operating Systems," *Commun. ACM*, 21 (9), Sept. 1978, pp. 737-749.

14. D.E. Bell and L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, ESD-TR-73-278, Vol. 1, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Nov. 1973 (MTR-2547, Vol. 1, MITRE Corp., Bedford, Mass.).
15. R.R. Schell, P.J. Downey, and G.J. Popek, "Preliminary Notes on the Design of Secure Military Computer Systems," MCI-73-1, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Jan. 1973.
16. W.R. Price, *Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems*, Ph.D. thesis, Carnegie-Mellon University, 1973.
17. D.L. Parnas and W.R. Price, "Using Memory Access Control as the Only Protection Mechanism," in *Proc. Int. Workshop on Protection in Operating Systems*, IRIA/LABORIA, Rocquencourt, France, Aug. 1974, pp. 177-181.
18. J.S. Fenton, "Memoryless Subsystems," *Computer Journal*, **17** (2), May 1974, pp. 143-147.
19. D.E. Denning, *Secure Information Flow in Computer Systems*, Ph.D thesis, Purdue Univ., May, 1975.
20. K.J. Biba, *Integrity Considerations for Secure Computer Systems*, ESD-TR-76-372, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Apr. 1977 (MITRE MTR-3153, NTIS AD A039324).
21. R.J. Feiertag, K.N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design," in *Proc. Sixth ACM Symp. on Operating Syst. Principles*, ACM SIGOPS *Operating Syst. Rev.*, **11** (5), Nov. 1977, pp. 57-65.
22. T.H. Hinke and M. Schaefer, *Secure Data Management System*, RADC-TR-75-266, Rome Air Dev. Center, AFSC, Griffiss AFB, N.Y., Nov. 1975 (NTIS AD A019201).
23. M.J. Grohn, *A Model of a Protected Data Management System*, ESD-TR-76-289, ESD/AFSC, Hanscom AFB, Mass., June 1976 (I.P. Sharp, Ottawa, Canada, NTIS ADA 035256).
24. A.K. Jones and R.J. Lipton, "The Enforcement of Security Policies for Computation," *Proc. Fifth Symp. on Operating Syst. Principles*, ACM SIGOPS *Operating Systems Rev.*, **9** (5), Nov. 1975, pp. 197-206.
25. E. Cohen, "Information Transmission in Computational Systems," *Proc. Sixth Symp. on Operating Syst. Principles*, ACM SIGOPS *Operating Syst. Rev.*, **11** (5), Nov. 1977, pp. 133-140.
26. F.C. Furtek, *A Validation Technique for Computer Security Based on the Theory of Constraints*, ESD-TR-78-182, ESD/AFSC, Hanscom AFB, Mass., Dec. 1978 (MITRE MTR-3661, NTIS ADA065111).
27. J.K. Millen, "Constraints and Multilevel Security," in *Foundations of Secure Computation*, R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, ed., Academic Press, New York, 1978, pp. 205-222.
28. B.W. Lampson, "Protection," in *Proc. Fifth Princeton Symp. on Information Sciences and Syst.*, Mar. 1971, pp. 437-443, reprinted in *ACM SIGOPS Operating Syst. Rev.*, **8** (1), Jan. 1974, pp. 18-24.
29. P.J. Denning, "Third Generation Computer Systems," *ACM Computing Surveys*, **3** (4), Dec. 1971, pp. 175-216.

30. G.S. Graham and P.J. Denning, "Protection—Principles and Practice," in *Proc. 1972 AFIPS Spring Jt. Computer Conf.*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 417-429.
31. M.A. Harrison, W.L. Ruzzo, and J.D. Ullman, "Protection in Operating Systems," *Commun. ACM*, **19** (8), Aug. 1976, pp. 461-471.
32. E.I. Organick, *The Multics System: An Examination of its Structure*, MIT Press, Cambridge, Mass., 1972.
33. G.J. Popek and C. Kline, "A Verifiable Protection System," in *Proc. Int. Conf. on Reliable Software, ACM SIGPLAN Notices*, **10** (6), June 1975, pp. 294-304.
34. G.J. Popek, C.S. Kline, and E.J. Walton, "UCLA Secure Unix," UCLA Technical Report (Draft), Feb. 1978.
35. G.J. Popek, M. Kampe, C.S. Kline, A. Stoughton, M. Urban, and E.J. Walton, "UCLA Secure Unix," in *Proc. AFIPS Nat. Computer Conf.*, **48**, AFIPS Press, Montvale, N.J., 1979, pp. 355-364.
36. B.J. Walker, R.A. Kemmerer, and G.J. Popek, "Specification and Verification of the UCLA Unix Security Kernel," *Commun. ACM*, **23** (2), Feb. 1980, pp. 118-131.
37. M.D. Yonke, "The XIVUS Environment: XIVUS Working Paper No. 1," USC/Information Sciences Institute, Marina del Rey, Cal., Apr. 1976.
38. R.A. Kemmerer, *Verification of the UCLA Security Kernel: Abstract Model, Mapping, Theorem Generation and Proof*, Ph.D. thesis, UCLA, Los Angeles, Cal., 1979.
39. A.K. Jones, R.J. Lipton, and L. Snyder, "A Linear Time Algorithm for Deciding Subject-Object Security," *Proc. 17th Annual Foundations of Computer Science Conf.*, Houston, 1976, pp. 33-41.
40. R.J. Lipton and L. Snyder, "A Linear Time Algorithm for Deciding Subject Security," *J. ACM* **24** (3), July 1977, pp. 455-464.
41. L. Snyder, "On the Synthesis and Analysis of Protection Systems," *Proc. Sixth Symp. on Operating Systems Principles, ACM SIGOPS Operating Systems Rev.*, **11** (5), Nov. 1977, pp. 141-150.
42. A.K. Jones, "Protection Mechanism Models: Their Usefulness," in *Foundations of Secure Computation*, R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton ed., Academic Press, New York, 1978, pp. 237-254.
43. L. Snyder, "Formal Models of Capability-based Protection Systems," Technical Report 151, Dept. of Computer Science, Yale University, New Haven, Conn., Apr. 1979.
44. K.G. Walter, W.F. Ogden, W.C. Rounds, F.T. Bradshaw, S.R. Ames, and D.G. Shumway, *Primitive Models for Computer Security*, ESD-TR-74-117, AF/ESD L.G. Hanscom Field, Bedford, Mass., Jan., 1974 (NTIS AD-778 467).
45. K.G. Walter, S.I. Schaen, W.F. Ogden, W.C. Rounds, D.G. Shumway, D.D. Schaeffer, K.J. Biba, F.T. Bradshaw, S.R. Ames, and J.M. Gilligan, "Structured Specification of a Security Kernel," in *Proc. Int. Conf. on Reliable Software, ACM SIGPLAN Notices*, **10** (6), June 1975, pp. 285-293.

46. K.G. Walter, W.F. Ogden, J.M. Gilligan, D.D. Schaeffer, S.I. Schaen, and D.G. Shumway, *Initial Structured Specifications for an Uncompromisable Computer Security System*, ESD-TR-75-82, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., July 1975 (NTIS AD-A022 490).
47. D.E. Bell and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model*, ESD-TR-73-278, Vol. 2, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Nov. 1973 (MTR-2547, Vol. 1, MITRE Corp., Bedford, Mass.).
48. D.E. Bell, *Secure Computer Systems: A Refinement of the Mathematical Model*, ESD-TR-73-278, Vol. 3, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Apr. 1974 (MTR 2547, Vol. 3, MITRE Corp., Bedford, Mass.).
49. D.E. Bell and L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model*, M74-244, Oct. 1974, MITRE Corp., Bedford, Mass.
50. D.E. Bell and L.J. LaPadula, *Secure Computer System: Unified Exposition and Multics Interpretation*, MTR-2997, MITRE Corp., Bedford, Mass., July 1975.
51. M.D. Schroeder, D.D. Clark, and J.H. Saltzer, "The Multics Kernel Design Project," in *Proc. Sixth ACM Symp. on Operating Syst. Principles*, ACM SIGOPS *Operating Syst. Rev.*, **11** (5), Nov. 1977, pp. 43-56.
52. W.L. Schiller, *The Design and Specification of a Security Kernel for the PDP-11/45*, ESD-TR-75-69, The MITRE Corp., Bedford, Mass., Mar. 1975.
53. J.K. Millen, "Security Kernel Validation in Practice," *Commun. ACM*, **19** (5), May 1976, pp. 243-250.
54. S.R. Ames and D.R. Oestreicher, "Design of a Message Processing System for a Multilevel Secure Environment," in *Proc. AFIPS Nat. Computer Conf.*, **47**, AFIPS Press, Montvale, N.J., 1978, pp. 765-771.
55. E.J. McCauley and P.J. Drongowski, "KSOS: The Design of a Secure Operating System," in *Proc. AFIPS Nat. Computer Conf.*, **48**, AFIPS Press, Montvale, N.J., 1979, pp. 345-353.
56. C.H. Bonneau, *Secure Communications Processor Kernel Software, Detailed Specification, Part I, Rev. D*, Honeywell Inc., Avionics Division, St. Petersburg, Florida, 1980.
57. B.D. Gold, R.R. Linde, R.J. Peeler, M. Schaefer, J.F. Scheid, and P.D. Ward, "A Security Retrofit of VM/370," in *Proc. AFIPS Nat. Computer Conf.*, **48**, AFIPS Press, Montvale, N.J., 1979, pp. 335-342.
58. 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*, SRI International, Menlo Park, California, Feb. 1977.
59. R.J. Feiertag and P.G. Neumann, "The Foundations of a Provably Secure Operating System (PSOS)," in *Proc. AFIPS Nat. Computer Conf.*, **48**, AFIPS Press, Montvale, N.J., 1979, pp. 329-334.
60. J.K. Millen, "An Example of a Formal Flow Violation," in *Proc. IEEE Computer Soc. Second International Computer Software and Applications Conf.*, Nov. 1978, pp. 204-208.

61. G.J. Popek, *Access Control Models*, ESD-TR-73-106 ESD/AFSC, L.G. Hanscom Field, Bedford, Mass., Feb. 1973 (NTIS AD-761 807).
62. D.E. Denning, "A Lattice Model of Secure Information Flow," *Commun. ACM*, **19** (5) May 1976, pp. 236-243.
63. D.E. Denning and P.J. Denning, "Certification of Programs for Secure Information Flow," *Commun. ACM*, **20** (7) July 1977, pp. 504-512.
64. G.R. Andrews and R.P. Reitman, "An Axiomatic Approach to Information Flow in Programs," *ACM Trans. on Prog. Languages and Syst.*, **2** (1), Jan. 1980, pp. 56-76.
65. R.P. Reitman, "A Mechanism for Information Control in Parallel Systems," in *Proc. Seventh Symposium on Operating Systems Principles*, *ACM SIGOPS Operating Systems Review*, **13** (4), Dec. 1979, pp. 55-63.
66. C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *Commun. ACM*, **12** (10), Oct. 1969, pp. 576-583.
67. D. Bonyun, "A New Model of Computer Security with Integrity and Aggregation Considerations," I.P. Sharp Assoc., Ottawa, Ontario, Mar. 1978.
68. *KSOS Verification Plan*, WDL-TR7809, Ford Aerospace and Communications Corp., Western Dev. Lab. Div., Palo Alto, Calif., and SRI International, Menlo Park, Calif., 1978.
69. O. Roubine and L. Robinson, *SPECIAL Reference Manual*, 3rd ed., SRI International, Menlo Park, California, 1977.
70. R.S. Boyer and J.S. Moore, *A Theorem-prover for Recursive Functions: A User's Manual*, Technical Report CSL-91, SRI International, Menlo Park, Calif., June, 1979.
71. A.K. Jones and R.J. Lipton, "The Enforcement of Security Policies for Computation," *Journal of Computer and Syst. Sciences*, **17** (1), Aug. 1978, pp. 35-55.
72. E. Cohen, "Information Transmission in Sequential Programs," in *Foundations of Secure Computation*, R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, eds., Academic Press, New York, 1978, pp. 297-336.
73. F.C. Furtek, "Constraints and Compromise," in *Foundations of Secure Computation*, R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, eds., Academic Press, New York, 1978, pp. 189-204.
74. J.K. Millen, *Causal System Security*, ESD-TR-78-171, ESD/AFSC, Hanscom AFB, Mass., Oct. 1978 (MITRE MTR-3614).
75. F.C. Furtek, "Doing Without Values," *Proc. Ninth Int. Symp. on Multiple-Valued Logic*, Apr. 1979, IEEE Cat. no. CH1408-4C, pp. 114-120.
76. F.C. Furtek, "Specification and Verification of Real-time, Distributed Systems Using the Theory of Constraints," Technical Report P-1027, The Charles Stark Draper Laboratory, Inc., Cambridge, Mass., Apr. 1980.
77. P.T. Withington, *The Trusted Function in Secure Decentralized Processing*, MITRE MTR-3892, MITRE Corp., Bedford, Mass., Sept. 1979.

78. S.R. Ames and J.G. Keeton-Williams, "Demonstrating Security for Trusted Applications on a Security Kernel," MITRE Corporation, Bedford, MA, Apr. 1980.
79. C.E. Landwehr, "Assertions for Verification of Multi-level Secure Military Message Systems," contribution to Workshop on Formal Verification, SRI International, Menlo Park, CA, April, 1980. Reprinted in *ACM Software Eng. Notes*, 5 (3), July 1980, pp. 46-47.
80. *GNOSIS External Specifications*, Tymshare, Inc., Cupertino, Cal., 1980.