Secure Computer Systems: A Mathematical Model

November, 1996

An electronic reconstruction
by
Len LaPadula
of
the original

MITRE Technical Report 2547, Volume II
titled "Secure Computer Systems: Mathematical Foundations"
by Leonard J. LaPadula and D. Elliott Bell
dated 31 May 1973

ABSTRACT

This paper presents a set of rules of operation which guarantee that a computer system can remain secure while exhibiting certain desired external characteristics. The rules are easily transformable into algorithms suitable for implementation on a digital computer.

PREFACE

In our introductory paper on secure computer systems (MTR–2547, Vol. I) we introduced a framework for mathematical modeling based upon general systems theory. We then developed a basic model of a secure computer system in very abstract terms. That model is the basis from which we proceed in this paper.

We present a set of rules of operation which guarantee that the system remains secure while exhibiting certain desired external characteristics. The rules of operation will be seen to be easily transformable into algorithms suitable for implementation on a digital computer.

This work was supported by Project 522B, Secure Computer, a part of Air Force Project 5220, Project 85.

Table of contents

Introduction....5
Background....5
APPROACH....5
SUMMARY AND REFERENCES....6
External Characteristics of the Model....7
Introduction....7
ACCESS ATTRIBUTES....7
REQUESTS AND DECISIONS....9
Security Condition....12
Interaction Of Multiple Accesses....12
SUMMARY AND REFERENCES....14
Formulation of the Model....16
Introduction....16
Security Condition....19
*−PROPERTY....19
RULES....20
Proof Approach....21
Rules of Operation....22
Introduction....22
The Rules....22
Rule 1: get−read: $\rho 1(Rk,v) \equiv$....23
Rule 1 is security−preserving....23
Rule 1 is *−property−preserving....23
Rule 2: get−append: $\rho 2(Rk,v) \equiv$....24
Rule 3: get−execute: $\rho 3(Rk,v) \equiv$....24
Rule 4: get−write: $\rho 4(Rk,v) \equiv$....24
Rule 5: release−read/write/all/execute: $\rho 5(Rk,v) \equiv$....25
Rule 6: give−read/write/all/execute: $\rho 6(Rk,v) \equiv$....25
Rule 7: rescind−read/write/all/execute: $\rho 7(Rk,v) \equiv$....25
Rule 8: change−f: $\rho 8(Rk,v) \propto$....26
Rule 9: create−object: $\rho 9(Rk,v) \equiv$....26
Rule 10: delete−object: $\rho 10(Rk,v) \equiv$....26
The System $\sum(R,D,W,z0)$....27
Summary....27
Design Considerations....28
Introduction....28
A Covering, Disjoint Set of Rules....28
System Data Base....29
Protection....30
Rescinding Access....30
Summary and References....30
Bibliography....32

SECTION I

Introduction

Background

This paper is a direct extension of the work reported in [1], SECURE COMPUTER SYSTEMS: MATHEMATICAL FOUNDATIONS. Therein we developed a mathematical framework and showed the basic methodology by which the resulting simple model could be extended. Starting with an abstract notion of general system, we showed how a relation can specify behavior of a system and how it is possible to prove that a system has (or does not have) a given property. However, the model of [1] is far from being usable for the design of a system. A key factor here is that a function (a specialized form of relation) is more amenable to translation into algorithm for a digital computer than is a general relation. Additionally, the set of access attributes, the set of requests, and the set of system decisions were all left abstract in [1]. Finally, we showed that the system under discussion had no security compromise; however, the definition of security compromise was such that we had only shown that the system was internally correct we had not related the internal behavior of the system to its external behavioral characteristics.

APPROACH

Our general approach is easily deduced from the preceding remarks. We shall make specific the sets of requests, decisions, and access attributes, guided by the characteristics we would like to see in the resulting system. Our model, while less general, will then be closely matched to the design problem. This approach is an exemplification of the general principles of modeling which we discussed in the introductory chapter of [1].

More particularly, we shall now consider external characteristics of the model (Section II) as they pertain to secure system operation. Section II provides justification for the formal properties of the model developed in Section III. It is in Section III that we develop a model which will be seen to be more specific than the model presented in [1] and particularly suited to the development of rules of operation which are functions.

Section III, then, provides a system for the formalization of rules; in addition, we shall develop therein a proof approach by which we shall show that the rules satisfy certain desired properties. Section IV presents the rules of operation, each accompanied by a formal proof of correctness Ñ that is, we show that each rule preserves security and satisfies an externally imposed property. Section V is devoted to design considerations; in particular, we shall discuss the notion of a covering, disjoint set of rules Ñ a notion irrelevant to the mathematical model but of some significance for the design engineer, for a covering, disjoint set of rules has no redundancy in the domains of operation of the rules. Other subjects relevant to design are also discussed in Section V.

SUMMARY AND REFERENCES

We have given a very brief introduction to the material of this paper. A lengthy introduction and motivation for our approach can be found in [1].

1. Bell, D. Elliott; La Padula, L. J., "Secure Computer Systems: Mathematical Foundations", MTR–2547 Vol. I, The MITRE Corporation, Bedford, Massachusetts, 1 March 1973.

SECTION II

External Characteristics of the Model

Introduction

The problem addressed in this report is the specialization of the model described in [1]. The goal is to make the model more useful for the investigation of rules for the operation of a secure computer system. To realize that goal, we must consider the characteristics of the system we are designing as guides for the specialization process. In particular, we must devise a reasonable set of access attributes and accompanying sets of requests and decisions. The next task is to consider the reasonableness of the security condition of [1]; in this case, we shall revise our concept of security to make it reflect the real situation more accurately. We shall end this chapter with a discussion of the possible interaction caused by multiple accesses. This discussion will result in the statement of a property which we shall incorporate into the model axiomatically as the *−property.

ACCESS ATTRIBUTES

We consider four basic types of access in a complex computer system. In this section, we shall describe these four access attributes Ñread−only access, append access, execute access, and read/write access, together with a fifth attribute, control access. The discussion of each access attribute is intended to provide our interpretation of that type of access and to justify our conclusions in the sections entitled Security Conditions and Interaction of Multiple Accesses.

The read access we shall include in our set A of access attributes can be considered a "pure read" access in the sense that a user with read access to an object cannot affect the contents of that object. This access mode will enable us to model two very different situations, each of which poses a significant modeling problem.

A file containing information which can be referenced, but which should not be altered, will be accessed in read mode. An obvious example of such information would be the classification and clearance listings of the subjects and objects in the system: in order to enforce security rules, this information must be accessible and unalterable.

Read–only access provides the appropriate combination of availability and protection.

Our model needs to take into account input and output devices, as well as information storage units. In particular, an input device such as a card reader, which has no inherent content, can be included in our model as a read–only object. Analogously, output devices will be included in the model using the append access mode.

The append access we shall include is a "pure write" access. By this we mean that append access allows alteration of the object (in particular, the addition of information to a file) while preventing information extraction from the object. We intend, therefore, that append access to a file should not divulge any of the contents of the file. In addition, we include within the purview of append access the use of an output device such as a printer which transfers information beyond the control of the system. In doing so, we assume that printers which are in nonclassified areas will be nonclassified and that matching the classifications of output information and the output device will suffice to prevent unauthorized personnel from viewing restricted material. The append access that we have described denotes a pure write operation anywhere in a file. We chose the word "append", however, because it connotes addition to the end of a file, i.e., appending data to an existing file, and because we anticipate that subsequent implementation will use an actual append operation.

Another type of object we want to include in our model is the executable object such as a program or routine. We therefore add an execute access mode to our model. Execute access only allows a subject to trigger the executable object: the general user cannot read or write the program if he has only execute access. If the program produces information, the information is fed back to the executor under system control. The usual procedure would be for the program to write its results in a file and for the executor to read the file. If the information produced were classified higher than the executor's clearance level, then the system would deny the executor read access to the file where the generated information had been stored.

The last type of access we shall consider is an interactive read/write access. We shall call this type of access write–access for simplicity. Write–access is the type of access that would be used in editing or updating a file.

The final access attribute we shall include we call "control access". This attribute is intended to formalize the notion of control over an object and the access to it. Locking and unlocking a file to another subject will be governed by the control access attribute.

We formalize five access attributes later in this paper based upon the preceding considerations:

- read access; a subject having read access attribute with respect to some object has a read–only capability with respect to that object;
- append access; a subject having append access attribute with respect to some object

has a write–only capability with respect to that object;

- execute access; a subject having execute access attribute with respect to some object has the capability to cause execution of that object (in the sense of running a program) in its behalf;
- write access; a subject having write access attribute with respect to some object has the capability to both read and write that object;
- control access; a subject having control access with respect to some object has the capability to extend to another subject one or more of the other four access attributes it may have with respect to that object.

REQUESTS AND DECISIONS

Assuming an environment wherein processes are surrogates for users, we shall speak of processes making requests for access to objects, requests to make changes to the access matrix of the system, requests to create objects, or requests to delete objects. The reader, then, should understand that such requests represent the intentions of the users of the system. We shall speak of the responses of the system to requests by processes as decisions.

The access matrix we have just mentioned is formally defined in Section III. For our purposes here it will suffice for the reader to have the notion that the access matrix of the system is simply a record–keeping device which remembers, for each possible subject–object pairing, a list of access attributes associated with that subject–object pairing. For example, the access matrix might show that the access attributes write (w)and control (c) are associated with subject S7 and object O9, where S7 is a process and O9 is a file. The access matrix can be envisioned as suggested in Figure 2–1.

		objects	
subjects		O9	
S7		w,c	

Figure 2–1

As we have briefly indicated at the beginning of this section, we shall provide for four types of requests:

(i) a request by a subject to be granted access to an object in a particular mode;

(ii)a request by a subject that another subject be given some access attribute with respect to some object;

(iii)a request by a subject to create an object in the system; and

(iv)a request by a subject to delete an object from the system.

With respect to (i), only requests involving read, write, append, or execute are valid. We shall use the control attribute in such a way that for a subject to request control access to an object makes no sense. Thus, a subject may request that it be given read, write, execute, or append access to an object. When such a request is made, a number of checks need to be made by the system. One of these, and perhaps the most obvious, is to check that the subject making the request is allowed to access the object in the mode which it is requesting. This is done by checking in the access matrix. For example, subject S_7 requesting write access to object O_9 passes the test, according to Figure 2–1, but S_7 requesting execute access to O_9 fails the test. Now, other checks must be made before a final decision can be rendered by the system; these have to do with preserving security and *–property Ñ this is discussed in the next two sections.

With respect to (ii), we require that a number of conditions be met in order for this type of request to be valid. Imagine a request by S_6 to give the read (r) access attribute to subject S_8 with respect to object O_9. First of all, the access matrix must show an entry of c at S_6 ,O_9 Ñ that is S_6 must have the .control access attribute with respect to O_9. Next, S_6 must also have the access attribute with respect to O_9 which it is attempting to extend to S_8 Ñ in this case, S_6 itself must have the read access attribute for O_9. In other words, a subject cannot give (or take away) access attributes unless it itself has control access and the access attribute it is attempting to give (or take away). Finally, we shall not allow a subject to extend the control access attribute to another subject. Other checks need to be made also Ñ these are similar to those for type (i) requests and are covered later.

With respect to (iii), we have divided the notion of creation of an object into two parts. Our reason for this approach lies in our method of modeling the process of creation: we see the creation of an object as the activation of an unused object index. This point of view was chosen to simplify the discussion of creation and deletion of objects by avoiding the need to dynamically alter Both the domain of the classification function f and the dimension of the access matrix M. This approach is justified since the activation of an object index is logically equivalent to the addition of an entirely new object to the set O. Using this point of view, however, the created object O_j may have a classification and a set of categories which do not match the requirements of the requesting subjects. (In the model, every object, active or not, has a classification and a set of categories assigned to it.) Thus we have included

(i) requests to alter the classification and category assignment of unused objects, and

(ii)requests to create (i.e., activate) unused objects.

Inasmuch as these types of requests neither alter the classifications or categories of active objects nor alter the clearances or categories of subjects nor change the current allocation of objects to subjects, the granting of these requests hinges not on security considerations but on appropriateness of the request. That is, a request to alter the classification of an active object or a request to create an object which is currently active would be rejected as inappropriate.

With respect to (iv), we again find that security considerations play no significant role. A request to delete an object Oj, when made by a subject with <u>control</u> access, causes the deactivation of O_j and the immediate withdrawal of all access privilege to O_j, both current and future. As in the preceding discussion, only the appropriateness of the request is considered by the system since neither compromise nor any other problem of concern can result.

In summary, the access matrix records the access attributes associated with each subject–object pair. The access matrix does not change except as noted Ñ that is, the <u>read,</u> <u>write,</u> <u>execute,</u> and <u>append</u> access attributes for active objects may be given and rescinded only by certain subjects. During normal operation of the system the <u>control</u> access attribute is entered into or deleted from the access matrix only when an object is created or deleted. We assume that other changes involving this access attribute are made by a control officer while the system is in a special mode of operation and not generally available to users.

The two basic decisions which the system must make are <u>yes</u> and <u>no</u> Ñ yes, the request is granted; no, the request is not granted. In addition, however, we specify an <u>error</u> decision and a "question" (?) decision. We shall use the <u>error</u> decision in a special way: it will indicate that the decision–making mechanism of the system is confused; its principal use would be in debugging during initial stages of checkout. The <u>error</u> decision will result when two or more rules (developed in Chapter 4) are applicable to the same request (as defined in Chapter 3, Section 3–5, Rules). However, the <u>error</u> mechanism could be retained in the system beyond its initial checkout to cover the eventuality that a change in the rules might at some time be desirable.

The decision <u>?</u> will be used to indicate that the request is not recognized. In the formal model this simply means that no rule is applicable to the request being made. As we shall see in Section V, the <u>?</u> decision can be restricted to be an internal response of the system, with the decision <u>illegal</u> being given to the subject (indicating an illegal request).

Security Condition

We desire the system to satisfy the condition that security compromise not be allowed to occur within it. With the establishment of specific access attributes and requests behind us, we can now be quite specific about the security condition we wish to satisfy.

Security compromise will be used in the usual sense of military and governmental definitions of security compromise Ñ the unauthorized disclosure of information. With this strict interpretation of security compromise we gain two advantages. First, we need not change nor strain our usual understanding of compromise; second, the security condition can be precisely and succinctly formalized in our model and can be kept distinctly separate from other properties or conditions we may wish to include in the model.

Given the access attributes we have defined and granted the strict interpretation of security compromise which we have indicated, the following notions develop quite naturally. The security condition to be developed formally in the model should be so defined as to prevent a subject of a given clearance level with a given set of categories (project clearances, special clearances, password, and so forth) from having read access to any object which is or can be the source of information with a classification which exceeds the clearance of the subject or an associated category set which includes an element (such as a project–need–to–know) which the subject does not possess. And that is all. Notice that we shall define the security condition (in Section 3) so that it pertains to immediate, actual ability to read (receive) information in an unauthorized way Ñ it does not involve potential, probable, or possible unauthorized access. This latter point illustrates what we have said previously. Potential unauthorized access is dealt with in the next section and leads to a new property to be incorporated into the model. The security condition, however, remains intact, simple and to the point.

Interaction Of Multiple Accesses

In certain cases, a combination of two or more accesses, none of which in itself constitutes a security compromise, can lead to a potential for later compromise. More importantly, these situations involve a loss of control on the part of the security system: it becomes possible for high classification material to be added to a low classification file without appropriate changes being made to the security classification lists. To avoid such potential for compromise and to retain control of the security situation, we have developed a set of criteria which describe precisely which situations we wish to avoid. The remainder of this section presents a specific example of potential compromise to illustrate the problem more clearly and an informal description of the *–property which characterizes those undesirable situations with which we are concerned.

Suppose that a subject S_7 currently has read access to a file O_{11}. Suppose further that the clearance level of S_7 is c_i and that the classification of O_{11} is also c_i. Assume also that the categories associated with O_{11} are included within those possessed by S_7 so that the security condition is satisfied. According to the security condition we have discussed, if S_7 should request append access to some object O_{13} we are able to grant the access regardless of the classifications, clearances, and categories associated with S_7 and O13. But suppose that the classification of O_{13} is c_j and $c_j < c_i$, meaning O_{13} is of lower classification than O_{11}; for example, O_{13} might be classified confidential and O_{11} secret. We have now set up a typical example of potential compromise.

Suppose that S_7, either inadvertently or intentionally, reads secret information from O_{11} and appends it to O_{13}. Then it is possible for some other subject whose clearance level is only confidential to gain access to O_{13} and read secret information(security compromise).

Several courses are open to us. For example, we might attempt to allow S_7 to do what we have indicated, in a controlled way of course, thereby necessarily changing the classification of O_{13} (by upgrading it from confidential to secret). An approach similar to this alternative has, in fact, been reported by Weissman [1]. This approach does keep the system's information about clearances and classifications in agreement with the actual clearances and classifications. However, we choose not to change the classifications of objects during the normal operation of the system. If this choice were reversed, the change could, of course, be incorporated into the model.

Therefore, we must find some other way to guarantee that the system's information reflects reality. Thus, we shall deny S_7's request to access O_{13} in append mode while it has access to O_{11} in read mode; this decision is based not on the satisfying of the security condition but on the fact that the potential for compromise would exist if the request were granted.

A number of other examples could be constructed; each could illustrate some aspect of the general problem we have been discussing. The reader should have no difficulty in seeing that we are led to the following requirement Ñ that the system, at each moment, satisfy a property (to be called *–property) which will guarantee that the potential for compromise does not exist.

We give now an informal description of *–property (Section III formalizes this as part of the mathematical model). Consider some subject S. We require that if S has <u>write</u> or <u>append</u> access to some objects and <u>read</u> or <u>write</u> access to some objects, then the classifications of the objects to which S has <u>write</u> or <u>append</u> access must exceed or equal the classifications of the objects to which S has <u>read</u> or <u>write</u> access. Clearly, this requirement prevents the situation which we detailed in the example at the beginning of this section. Another obvious situation is avoided. If S can read information from a card reader and print information on a printer, the *−property guarantees that information of a classification higher than that which the printer should handle will not be sent to it by S from the card reader, which might be allowed to handle the information.

SUMMARY AND REFERENCES

In this section we have discussed certain aspects of an hypothetical multi−user system which is to be secure in some sense. We have attempted to give motivation and heuristic arguments for the precise sense in which we shall consider a system to be secure later in this paper.

We have not always indicated why a particular approach has been taken. For example, in developing our arguments for the *−property, we indicated that "we choose not to change the classifications of objects during the normal operation of the system." In this case the decision is the result of analyzing the requirements for the system, the general consensus being that changing classifications of objects during normal operation is somewhat unnatural, akin to upgrading a confidential document while one is reading it by writing secret marginalia in it.

The approaches we have taken can be succinctly recounted and categorized as being either required or discretionary. The set of access attributes contains both required and discretionary attributes. The attributes read and write (that is, read−only and read−write) we consider to be necessary in a reasonable computer system; the attributes execute and append are optional in that a computer system could operate without them. Given the access attributes, the requests we have included are necessary to provide a minimally usable system; the same is true of all the decisions except ?. ? was included in order to simplify the discussion of sets of rules. Three other principles combined with the access attributes, the requests, and the decisions we chose, directly resolve every other decision that was made. Those three principles are the following: the security principle; the interactivity principle; and the tranquility principle. The security principle, enunciated in the definition of security, is necessary in a military−governmental situation. The interactivity principle, embodied in the model as the *−property, is also a necessary ingredient in a usable system. The tranquility principle, that the classification of active objects will not be changed during normal operation, is not required. As we mentioned above, its adoption was accepted as a result of a first analysis of the requirements for secure computer systems. These decisions specify our model. In particular, situations such as the one presented in the preceding paragraph can be handled in only one way: queueing a request is ruled out by the unavailability of that decision in our model and upgrading a document to preserve the *−property is ruled out by the tranquility principle. Thus, the model as presented can be significantly altered only by changing elements of the model or by rejecting the tranquility principle.

1. Weissman, C.: "Security Controls in the ADEPT−50 Time−Sharing System," AFIPS Conference Proceedings 35, FJCC 1969, 119−133.

SECTION III

Formulation of the Model

Introduction

We are now in a position to develop a model more specifically suited to the investigation at hand than the model presented in [1].

Having introduced specific <u>access attributes</u>, <u>requests</u>, and <u>properties</u> of the system which we deem desirable, two things are immediately evident:

(i) we can specialize the state space of the system;

(ii) we can specialize the general relation W (cf. [1]) to be the union of partial functions, each function being a rule of operation for access control.

We show the elements of the model in Table I, wherein we identify sets, elements of the sets, and an interpretation of the elements of the sets. This table can be seen to be that of [1] extended as we have previously indicated.

Table I

Elements of the Model

Set	Elements	Semantics
S	$\{S_1,S_2, \cdots ,S_n\}$	<u>subjects</u>; processes, programs in execution
O	$\{O_1,O_2, \cdots ,O_m\}$	<u>objects</u>; data, files, programs, subjects, I/O devices
C	$\{C_1,C_2, \cdots ,C_q\}$ $\{C_1 > C_2 > \cdots > C_q$	<u>classifications</u>; clearance level of a subject, classification of an object

K	$\{K_1, K_2, \cdots, K_r\}$	categories: special access privileges
A	{r,w,e,a,c}	access attributes; read, write, append, execute, and control
RA	{g,r,c,d}	request elements; g: get, give r: release, rescind c: change, create d: delete
R	$S^+ \times S^+ \times O \times X$ where $S^+ = S \cup \{\phi\}$ and $X = A \cup \{\phi\} \cup F$; an arbitrary element of R is written R_k	requests: inputs, commands, requests for access to objects by subjects
D	{yes, no, error, ?} an arbitrary element of D is written D_m	decisions
T	$\{1,2,\cdots,t,\cdots\}$	indices; elements of the time set; identification of discrete moments; an element t is an index to request, decision, and state sequences
F	$C^S \times C^O \times (PK)^S \times (PK)^O$ an arbitrary element of F is written $f = (f_1, f_2, f_3, f_4)$	classification/need–to–know vectors; f_1: subject–classification function f_2: object–classification function f_3: subject–category function f_4: object–category function
X	R^T an arbitrary element of X is written x	request sequences
Y	D^T an arbitrary element of Y is written y	decision sequences

M	$\{M_1, M_2, \cdots, M_c\}$, $c = nm2^5$; an element of M, say M_k, is an $n \times m$ matrix with entries from PA; the (i,j)–entry of M_k shows S_i's access attributes relative to O_j	access matrices
V	$P(S \times O \times A) \times M \times F$ an arbitrary element of V is written v	states
Z	V^T an arbitrary element of Z is written z; $z_t \in z$ is the t–th state in the state sequence z	state sequences

We have defined the states of the system in such a way as to include all the information considered pertinent to preservation of security and other desirable properties. A state $v \in V$ is an ordered triple (b,M,f) where

$b \in P(S \times O \times A)$, indicating which subjects have access to what objects in what mode in the state v;

$M \in M$, indicating the entries of the access matrix in the state v; and

$f \in F$, indicating the clearance level of all subjects, the classification level of all objects, and the categories associated with each subject and object in the state v.

Let $W \subset R \times D \times V \times V$. The system $\sum(R,D,W,z_o) \subset X \times Y \times Z$ is defined by $(x,y,z) \in \Sigma(R,D,W,z_o)$ iff (if and only if) $(x_t,y_t,z_t,z_{t-1}) \in W$ for each $t \in T$, where z_o is a specified initial state, usually of the form (ϕ,M,f).

W has been defined as a relation. As we shall see, however, W will be the union of partial functions which constitute the rules of operation of the system with respect to preservation of security and the externally imposed characteristic (to be called *–property).

Security Condition

$(s,o,\underline{x}) \in S \times O \times A$ satisfies the <u>security condition relative to f</u> (SC rel f) iff

(i) $\underline{x} = \underline{e}$ or $\underline{x} = \underline{a}$ or $\underline{x} = \underline{c}$, or
(ii)($\underline{x} = \underline{r}$ or $\underline{x} = \underline{w}$) and ($f_1(s) \geq f_2(o)$ and $f_3(s) \supseteq f_4(o)$).

A state $v = (b,M,f) \in V$ is a <u>secure state</u> iff each $(s,o,\underline{x}) \in b$ satisfies SC rel f. A state v is a <u>compromise state</u> (<u>compromise</u>) iff it is not a secure state.

A state sequence $z \in Z$ <u>has a compromise</u> iff z_t is a compromise for some $t \in T$. z is a <u>secure state sequence</u> iff z_t is a secure state for each $t \in T$. We call

$$(x,y,z) \in \Sigma(R,D,W,z_0)$$

an <u>appearance</u> of the system. $(x,y,z) \in \Sigma(R,D,W,z_0)$ is a <u>secure appearance</u> iff z is a secure state sequence. The appearance (x,y,z) <u>has a compromise</u> iff z has a compromise.

$\Sigma(R,D,W,z_0)$ is a <u>secure system</u> iff every appearance of $\Sigma(R,D,W,z_0)$ is secure. $\Sigma(R,D,W,z_0)$ <u>has a compromise</u> iff some appearance of $\Sigma(R,D,W,z_0)$ has a compromise.

****−PROPERTY***

We introduce the following notation to make later development more succinct.

Let $b(s:\underline{x},\underline{y}, \ldots ,\underline{z})$ denote the set

$$\{o: o \in O \text{ and } [(s,o,\underline{x}) \in b \text{ or } (s,o,\underline{y}) \in b \text{ or } \cdots \text{ or } (s,o,\underline{z}) \in b]\}.$$

A state $v = (b,M,f) \in V$ <u>satisfies *−property</u> iff for each $s \in S$ the proposition

$$\begin{aligned}&[[b(s:\underline{w},\underline{a}) \neq \emptyset \text{ and } b(s:\underline{r},\underline{w}) \neq \emptyset] \text{ implies } [f_2(o_1) \geq f_2(o_2) \text{ and } f_4(o_1) \supseteq f_4(o_2),\\ &\text{for all } o_1 \in b(s:\underline{w},\underline{a}),\ o_2 \in b(s:\underline{r},\underline{w})]] \qquad (3\text{−}1)\end{aligned}$$

is true. A state v <u>violates *−property</u> iff v does not satisfy *−property.

A state sequence $z \in Z$ <u>satisfies *−property</u> iff z_t satisfies *−property for each t

$\in$ T. $(x,y,z) \in \Sigma(R,D,W,z_0)$ satisfies *–property iff z satisfies *–property. $\Sigma(R,D,W,z_0)$ satisfies *–property iff every appearance of $\Sigma(R,D,W,z_0)$ satisfies *–property.

RULES

A rule is a function ρ: $R \times V \to D \times V$. The interpretation of a rule is that, given a request and a state, a rule decides a response and a state change. A rule is analogous to the concatenation of the output and decision (next–state) functions of a finite–state machine (or sequential machine).

A rule ρ is security–preserving iff the proposition

$$[[\rho(R_k,v) = (D_m,v^*) \text{ and } v \text{ is secure}] \text{ implies } [v^* \text{ is secure}]]$$

holds for all elements $(R_k,v) \in R \times V$.

A rule ρ is *–property–preserving iff the proposition

$$[[\rho(R_k,v) = (D_m,v^*) \text{ and } v \text{ satisfies *–property}] \text{ implies } [v^* \text{ is secure}]]$$

holds for all elements $(R_k,v) \in R \times V$.

Let $\omega = \{\rho_1, \rho\rho_2, \cdots, \rho\rho_t\}$ be a set of rules relative to R, D, and V. The relation $W(\omega)$ is defined by:

(i) $(R_k,\underline{?},v,v) \in W(\omega)$ iff $\rho_i(R_k,v) = (\underline{?},v)$ for each i, $1 \le i \le t$;

(ii)$(R_k,\underline{error},v,v) \in W(\omega)$ iff there exist i_1, i_2, $1 \le i_1 < i_2 \le t$, such that $\rho_{i1}(R_k,v) = (\underline{?},v^*)$ and $\rho_{i2}(R_k,v) = (\underline{?},v^{**})$, for some $v^*,v^{**} \in V$;

(iii)$(R_k,D_m,v^*,v) \in W(\omega)$ iff there exists a unique i, $1 \le i \le t$, such that $(\underline{?},v^{**}) \ne \rho_i(R_k,v) = (D_m,v^*)$, for some v^* and any $v^{**} \in V$.

One may interpret a rule to be a formal realization of one's intuitive notion of how the system handles requests, making decisions based on the combination of current situation and specific request. If rule ρ_i does not handle a request R_k , then $\rho\rho_i(R_k,v) = (\underline{?},v)$, meaning that ρ_i is not applicable to (R_k,v); considering ρ_i as a function, this means that (R_k,v) is not in the domain of ρ_i. If we have some (R_k,v) for which every rule is inapplicable, then $W(\omega)$ is inapplicable and the system response is $(\underline{?},v)$ (cf. (i) under the definition of $W(\omega)$). By our definition of $W(\omega)$ we have imposed the condition that each type of request be handled by no more than one rule. If more than one rule is applicable, then the system response is (error, v) (cf. (ii) under the definition of $W(\omega)$). If exactly one rule is applicable to (R_k,v) resulting in (D_m,v^*), then the system response is (D_m,v^*) (cf. (iii) under the definition of $W(\omega)$).

Proof Approach

Esta sección la omití deliberadamente pues trata únicamente sobre la forma de probar que las reglas cumplen ciertas propiedades.

SECTION IV

Rules of Operation

Esta sección difiere de la original pues no se muestran las pruebas de que cada operación verifica las propiedades (excepto para la primera).

Introduction

In this section we develop a set ω of rules which are security–preserving and *–property–preserving and show that the resulting system $\sum(R,D,W(\omega),z_0)$, given z_0 secure and satisfying *–property, is secure and satisfies *–property.

Theorem 3–4 is the principal tool by which we shall prove that a rule is security–preserving; Theorem 3–5 is the principal tool by which we shall prove that a rule is *–property–preserving; and Theorems 3–2 and 3–3 enable us to prove that the resulting system is secure and satisfies *–property if its initial state is secure and satisfies *–property.

The Rules

Let $R_k = (\sigma_1, \gamma, \sigma_2, O_j, \underline{x})$, where σ_1 and σ_2 in S^+, γ in RA, $\underline{x}$ in X, v = (b,M,f). When $\sigma_1 \in S$, we shall denote it by "S_λ"; we shall use S_i for σ_2 whenever $\sigma_2 \in S$. Also, let Φ denote an arbitrary subset of A.

For notational convenience, make the definitions:

- $augb(R_k,v) = (b \cup (\sigma_2,O_j,\underline{x}),M,f)$
- $dimb(R_k,v) = (b - (\sigma_2,O_j,\underline{x}),M,f)$
- $M \, (+) \, [\Phi]_{ij}$ will be the matrix M^+ where

  $M^+[st] = M[st]$ if $(s,t) \neq (i,j)$

$M^{+}[st] = M[st] \cup \Phi$ if $(s,t) = (i,j)$.

- $M\,(-)[\Phi]_{ij}$ will be the matrix M^{-} where

  $M^{-}[st] = M[st]$ if $(s,t) \neq (i,j)$

  $M^{-}[st] = M[st] - \Phi$ if $(s,t) = (i,j)$.

- $A(M) = \{j: 1 \leq j \leq m$ and $M_{ij} \neq \Phi$ for some $i\}$.

Rule 1: <u>get−read</u>: $\rho_1(R_k,v) \equiv$

<u>if</u> $\sigma_1 \neq \phi$ or $\gamma \neq$ g or $\underline{x} \neq \underline{r}$ or $\sigma_2 = \phi$
    <u>then</u> $\rho_1(R_k,v) = (\underline{?},v)$;
<u>if</u> $\underline{r} \notin M_{ij}$ or $[f_1(S_i) < f_2(O_j)$ or $f_3(S_i) \not\supseteq f_4(O_j)]$
    <u>then</u> $\rho_1(R_k,v) = (\underline{no},v)$;
<u>if</u> $U\rho_1 = \{O{:}O \in b(S_i{:}\underline{w},\underline{a})$ and $[f_2(O_j) > f_2(O)$ or $f_4(O_j) \not\subseteq f_4(O)]\} = \emptyset$
    <u>then</u> $\rho_1(R_k,v) = (\underline{yes},augb(R_k,v))$
    <u>else</u> $\rho_1(R_k,v) = (\underline{no},v)$;

<u>end</u>.

Rule 1 is security−preserving

Let v be secure and $R_k \in R$. If $\rho_1(R_k,v) = (D_m,v^*)$, then $v^* = v$ or $v^* = augb(R_k,v)$, according to ρ_1. If $v^* = v$, then v^* is secure since v is secure. Suppose $v^* = augb(R_k,v)$. Now $b^* - b = \{(S_i,O_j,\underline{r})\}$ or $\emptyset$. If empty, then done (since $(S_i,O_j,\underline{r}) \in b$). Suppose not empty: then, since $f_1(S_i) \geq f_2(O_j)$ or $f_3(S_i) \supseteq f_4(O_j)$ according to ρ_1, v^* is secure by Theorem 3−4.

Rule 1 is *−property−preserving

Let v satisfy *−property and $R_k \in R$. If $\rho_1(R_k,v) = (D_m,v^*)$. Then $v^* = v$ or $v^* = augb(R_k,v)$. If $v^* = v$, then v^* satisfies *−property. Suppose $v^* = augb(R_k,v)$. Now

$augb(R_k,v) = (b^*,M,f)$, where $b^* = b \cup \{(S_i,O_j,\underline{r})\}$. According to ρ_1 we have $U\rho_1 = \emptyset$. Thus, if $O \in b(S_i:\underline{w},\underline{a})$, then $[f_2(O_j) > f_2(O)$ or $f_4(O_j) \not\subseteq f_4(O)]$ is false. That is, we have $f_2(O_j) \leq f_2(O)$ and $f_4(O_j) \subseteq f_4(O)$ so that v* satisfies *−property by Theorem 3−5.

Rule 2: get−append: $\rho_2(R_k,v) \equiv$

> $\underline{\text{if}}$ $\sigma_1 \neq \phi$ or $\gamma \neq$ g or $\underline{x} \neq \underline{a}$ or $\sigma_2 = \phi$
> $\underline{\text{then}}$ $\rho_2(R_k,v) = (\underline{?},v)$;
> $\underline{\text{if}}$ $\underline{a} \notin M_{ij}$
> $\underline{\text{then}}$ $\rho_1(R_k,v) = (\underline{no},v)$;
> $\underline{\text{if}}$ $U\rho_2 = \{O:O \in b(S_i:\underline{w},\underline{r})$ and $[f_2(O_j) < f_2(O)$ or $f_4(O_j) \not\supseteq f_4(O)]\} = \emptyset$
> $\underline{\text{then}}$ $\rho_2(R_k,v) = (\underline{yes},augb(R_k,v))$
> $\underline{\text{else}}$ $\rho_2(R_k,v) = (\underline{no},v)$;
>
> $\underline{\text{end}}$.

Rule 3: get−execute: $\rho_3(R_k,v) \equiv$

> $\underline{\text{if}}$ $\sigma_1 \neq \phi$ or $\gamma \neq$ g or $\underline{x} \neq \underline{e}$ or $\sigma_2 = \phi$
> $\underline{\text{then}}$ $\rho_3(R_k,v) = (\underline{?},v)$;
> $\underline{\text{if}}$ $e \notin M_{ij}$
> $\underline{\text{then}}$ $\rho_3(R_k,v) = (\underline{no},v)$;
> $\underline{\text{else}}$ $\rho_3(R_k,v) = (\underline{yes},augb(R_k,v))$;
>
> $\underline{\text{end}}$.

Rule 4: get−write: $\rho_4(R_k,v) \equiv$

> $\underline{\text{if}}$ $\sigma_1 \neq \phi$ or $\gamma \neq$ g or $\underline{x} \neq \underline{w}$ or $\sigma_2 = \phi$
> $\underline{\text{then}}$ $\rho_4(R_k,v) = (\underline{?},v)$;
> $\underline{\text{if}}$ $\underline{w} \notin M_{ij}$ or $[f_1(S_i) < f_2(O_j)$ or $f_3(S_i) \not\supseteq f_4(O_j)]$

$\underline{\text{then}}$ $\rho_4(R_k,v) = (\underline{\text{no}},v)$;

$\underline{\text{if}}$ $U\rho_4 = \{O{:}O \in b(S_i{:}\underline{a})$ and $[f_2(O_j) > f_2(O)$ or $f_4(O_j) \not\subseteq f_4(O)]\} \cup$

$\{O{:}O \in b(S_i{:}\underline{r})$ and $[f_2(O_j) < f_2(O)$ or $f_4(O_j) \not\supseteq f_4(O)]\} \cup$

$\{O{:}O \in b(S_i{:}\underline{w})$ and $[f_2(O_j) \neq f_2(O)$ or $f_4(O_j) \neq f_4(O)]\} = \emptyset$

$\underline{\text{then}}$ $\rho_4(R_k,v) = (\underline{\text{yes}},augb(R_k,v))$

$\underline{\text{else}}$ $\rho_4(R_k,v) = (\underline{\text{no}},v)$;

$\underline{\text{end}}$.

Rule 5: $\underline{\text{release–read/write/all/execute}}$: $\rho_5(R_k,v) \equiv$

$\underline{\text{if}}$ $(\sigma_1 \neq \phi)$ or $(\gamma \neq r)$ or $(\underline{x} \neq \underline{r},\underline{w},\underline{a}$, or $\underline{e})$ or $(\sigma_2 = \phi)$

$\underline{\text{then}}$ $\rho_5(R_k,v) = (\underline{?},v)$;

$\underline{\text{else}}$ $\rho_5(R_k,v) = (\underline{\text{yes}}, dimb(R_k,v))$;

$\underline{\text{end}}$.

Rule 6: $\underline{\text{give–read/write/all/execute}}$: $\rho_6(R_k,v) \equiv$

$\underline{\text{if}}$ $(\sigma_1 \neq S_\lambda)$ or $(\gamma \neq g)$ or $(\underline{x} \neq \underline{r},\underline{w},\underline{a}$, or $\underline{e})$ or $(\sigma_2 = \phi)$

$\underline{\text{then}}$ $\rho_6(R_k,v) = (\underline{?},v)$;

$\underline{\text{if}}$ $\underline{x} \notin M_{\lambda j}$ or $\underline{c} \notin M_{\lambda j}$

$\underline{\text{then}}$ $\rho_6(R_k,v) = (\underline{\text{no}},v)$;

$\underline{\text{else}}$ $\rho_6(R_k,v) = (\underline{\text{yes}},(b,M\ (+)\ [\underline{x}]_{ij},f))$;

$\underline{\text{end}}$.

Rule 7: $\underline{\text{rescind–read/write/all/execute}}$: $\rho_7(R_k,v) \equiv$

$\underline{\text{if}}$ $(\sigma_1 \neq S_\lambda \in S)$ or $(\gamma \neq r)$ or $(\underline{x} \neq \underline{r},\underline{w},\underline{a}$, or $\underline{e})$ or $(\sigma_2 = \phi)$

$\underline{\text{then}}$ $\rho_7(R_k,v) = (\underline{?},v)$;

$\underline{\text{if}}$ $\underline{x} \notin M_{\lambda j}$ or $\underline{c} \notin M_{\lambda j}$

$\underline{\text{then}}$ $\rho_7(R_k,v) = (\underline{\text{no}},v)$;

else $\rho_7(R_k,v) = (\underline{yes},(b-\{(S_i,O_j,\underline{x})\},M\ (-)\ [\underline{x}]_{ij},f))$;

end.

Rule 8: change–f: $\rho_8(R_k,v) \propto$

if $\sigma_1 \neq \phi$ or $\gamma \neq$ c or $\underline{x} \neq$ F or $\sigma_2 \neq \phi$

then $\rho_8(R_k,v) = (\underline{?},v)$;

if $f_1{}^* \neq f_1$ or $f_3{}^* \neq f_3$ or $[f_2{}^*(O_j) \neq (f_2(O_j)$ or $f_4{}^*(O_j) \neq f_4(O_j)$ for some $j \in A(M)]$

then $\rho_8(R_k,v) = (\underline{no},v)$;

else $\rho_8(R_k,v) = (\underline{yes},b,M,f^*)$;

end.

Rule 9: create–object: $\rho_9(R_k,v) \equiv$

if $\sigma_1 \neq \phi$ or $\gamma \neq$ c or $(\underline{x} \neq \underline{e}$ or $\phi)$ or $\sigma_2 = \phi$

then $\rho_9(R_k,v) = (\underline{?},v)$;

if $j \in A(M)$

then $\rho_9(R_k,v) = (\underline{no},v)$;

if $\underline{x} = \phi$

then $\rho_9(R_k,v) = (\underline{yes},(b,M\ (+)\ [\underline{r},\underline{w},\underline{a},\underline{c}\}]_{ij},f))$;

else $\rho_9(R_k,v) = (\underline{yes},(b,M\ (+)\ [\underline{r},\underline{w},\underline{a},\underline{c},\underline{e}\}]_{ij},f))$;

end.

Rule 10: delete–object: $\rho_{10}(R_k,v) \equiv$

if $\sigma_1 \neq \phi$ or $\gamma \neq$ d or $(\underline{x} \neq \phi)$ or $\sigma_2 = \phi$

then $\rho_{10}(R_k,v) = (\underline{?},v)$;

if $c \notin M_{ij}$

else $\rho_{10}(R_k,v) = (\underline{yes},(b,M\ (-)\ [\underline{r},\underline{w},\underline{a},\underline{c},\underline{e}\}]_{ij,1\leq i\leq n},f))$;

end.

The System $\sum(R,D,W,z_0)$

Theorem 4–1 Let $\omega = \{\rho_1, \cdots, \rho_{10}\}$, the ρ_i as defined in the section entitled The Rules, and z_0 be a secure state which satisfies *–property. Then $\sum(R,D,W,z_0)$ is a secure system which satisfies *–property.

Proof Follows directly from the proofs for the ρ_i in the section entitled The Rules, and from Theorems 3–2 and 3–3.

Summary

In this section we have presented a set of rules, suitable for implementation on a digital computer, which, within the framework of definitions, properties, and capabilities discussed in Sections II and III, provide the algorithms by which a secure system satisfying the externally derived characteristic called *–property can operate successfully. The proofs presented herein provide certification of the assertion just made.

SECTION V

Design Considerations

Introduction

In this section we review briefly a subset of the design considerations which will necessarily precede a translation of the model presented herein into a set of design specifications for a system. We shall assume a Multics–like architecture, both software and hardware, for this section. Such an assumption provides a representative system so that our discussion may have reasonable empirical content.

A Covering, Disjoint Set of Rules

Definition 5–1. Let ω be the set of rules for $\Sigma(R,D,W(\omega),z_0)$. ω is a covering set of rules if $[(R_i,D_m,v^*,v) \in W(\omega)]$ implies $[Dm \neq \underline{?}]$.

Definition 5–2. Let ω be the set of rules for $\Sigma(R,D,W(\omega),z_0)$. ω is a disjoint set of rules if for each $\rho_i \in \omega$ the proposition

$$[[\rho_i(R_k,v) \neq (\underline{?},v^*)] \text{ implies } [\rho_j(R_k,v) = (\underline{?},v^{**})] \text{ for each } \rho_j \in \omega - \{\rho_i\}]$$

is true.

The set $\omega = \{\rho_1, \cdots, \rho_{10}\}$ developed in Section IV is easily seen to be a disjoint but not a covering set of rules. On the other hand, the system response $\underline{?}$, which can occur since ω is not a covering set, is not very informative. The augmented set $\omega^+ = \omega \cup \{\rho_{11}\}$ could be used to rectify this as follows:

- add the element illegal to the set D (decisions)
- define $\rho_{11}(R_k,v) \neq (\underline{illegal},v)$ for all $R_k \in R$ and $v \in V$ such that $\rho_i(R_k,v) = (\underline{?},v)$ for $1 \leq i \leq 10$.

Thus if Rules 1–10 are inapplicable, then Rule 11 is applicable and system response is illegal, meaning that the request was not a valid one. The resulting set of rules, ω^{+} is now a covering set. An example of an illegal request is (S_{λ},g,S_i,O_j,c).

System Data Base

In speaking of the system data base we mean that portion of the total data base which contains the information required by the rules of the system. Therefore, the elements of the system data base are:

- an access matrix (M of the state (b,M,f))
- a list of 3–tuples (b of the state (b,M,f)) which shows which subjects have access to which objects in what mode
- a list of the classifications, clearances, and categories associated with subjects and objects (f of the state (b,M,f)).

The contents of b and M and the mapping f may change during normal system operation. All three can be implemented in any way one may choose so long as it is guaranteed that they be protected from unauthorized modification; only the rules of the system are allowed to change b, M, and f during normal operation.

Since b expands and contracts dynamically during operation, the information should be organized for a reasonable trade–off between access time and update time.

With respect to M, at least two alternatives suggest themselves. M could be treated in the same general manner as b or it could be a fixed–size matrix which is always kept in the main store of the computer system.

The preceding considerations raise several points worth considering here. The model clearly suggests that the access matrix be established (or changed) in a special mode of operation of the system by a control officer. This suggests further that a special program, which is itself certified and protected, be run after the control officer establishes the access matrix to ascertain that the access matrix satisfies some appropriate set of properties. We have made no explicit provision for changing the number of subjects in the system. There are several ways of making provision for this option. We envision the access matrix to be of fixed size ($n \times m$) during normal operation. However, the size of M is established to handle peak load and subjects could come and go during normal operation, with a mapping of subject name to access matrix row designation changing.

The ability to create and delete objects, which is provided by Rules 8 – 10, is most easily accomplished by retaining the notion of a fixed size matrix. The matrix would have empty columns in it from time to time and these would represent available address space locations for establishing the existence of a new object. Similarly, when an object is deleted, a column in the matrix becomes blank and available. Mapping of object names to column designations (indices) would be accomplished by the system at the allocation level Ñ outside and (in a hierarchical sense) above the rules which manage the access matrix.

Protection

The rules of access developed in Section IV are not invoked for every access to an object by a subject. For example, once Rule 1 grants a subject read−access to some object, actual access (reading words of a file) is not monitored by the rules. However, each access must be monitored in order to guarantee that the system remains secure and preserves *−property. Assuming a machine architecture suitable for a Multics−like system, this kind of protection is easily provided by the use of hardware−interpreted registers. For each $(S,O,\underline{r})$ we have an associated register by which S must access O. Clearly, S must not be allowed to change the contents of the register directly; changes must be made by a certified portion of the system (i.e., the ÒkernelÓ of the system). The register then monitors every access to O by S, insuring at each moment of activity that S does only what it is allowed to do.

Rescinding Access

Given the hardware registers of the type just discussed in the previous section, implementation of the rescind−access rule (Rule 7) is quite straightforward. The action

$$b \text{ is replaced by } b - \{(S,O,\underline{x})\}$$

of Rule 7 means simply that the appropriate enabling portion of the hardware register associated with $(S,O,\underline{x})$ in b is disabled, thereby denying S any further access to O in the mode $\underline{x}$.

Summary and References

In this section we have briefly discussed some design and implementation considerations; this discussion, while by no means comprehensive, should serve as a first step from the mathematical model to design considerations and specifications.

We should point out that the notion of a fixed–size access matrix with blank columns has been discussed in [1], which will also provide the reader with a general understanding of the Multics architecture.

1. Bensoussan, A.: Clingen, C. T.; Daley, R. C.: ÒThe Multics Virtual Memory: Concepts and Design,Ó Communications of the ACM, Vol. 15, No. 5, May 1972, pp. 308–318.

Bibliography

1. Bensoussan, A.: Clingen, C. T.; Daley, R. C.: ÒThe Multics Virtual Memory: Concepts and Design,Ó Communications of the ACM, Vol. 15, No. 5, May 1972, pp. 308−318.
2. Bell, D. Elliott; La Padula, L. J., "Secure Computer Systems: Mathematical Foundations", MTR−2547 Vol. I, The MITRE Corporation, Bedford, Massachusetts, 1 March 1973.
3. Weissman, C.: "Security Controls in the ADEPT−50 Time−Sharing System," AFIPS Conference Proceedings 35, FJCC 1969, 119−133.