ESD-TR-75-306

MTR-2997 Rev. 1

SECURE COMPUTER SYSTEM: UNIFIED EXPOSITION AND MULTICS INTERPRETATION

MARCH 1976

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Bedford, Massachusetts

Approved for public release; distribution unlimited.

Project No. 522B
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-76-C-0001

REVIEW AND APPROVAL

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

ROGER R. SCHELL, Major, USAF
Techniques Engineering Division

WILLIAM R. PRICE, 1Lt, USAF
Techniques Engineering Division

FOR THE COMMANDER

STANLEY P. DERESKA, Colonel, USAF
Chief, Techniques Engineering Division
Information Systems Technology
Applications Office
Deputy for Command and Management Systems

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-75-306	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* SECURE COMPUTER SYSTEM: UNIFIED EXPOSITION AND MULTICS INTERPRETATION		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER MTR-2997 Rev. 1
7. AUTHOR(s) D. E. Bell L. J. La Padula		8. CONTRACT OR GRANT NUMBER(s) F19628-75-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation Box 208 Bedford, MA 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 522B
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Command and Management Systems Electronic Systems Division, AFSC Hanscom Air Force Base, Bedford, MA 01731		12. REPORT DATE MARCH 1976 13. NUMBER OF PAGES 129
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

This report supersedes ESD-TR-75-306 dated January 1976.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

ASTERISK-PROPERTY
MATHEMATICAL MODEL
SECURE COMPUTER SYSTEM
SECURITY
TRUSTED SUBJECT

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A unified narrative exposition of the ESD/MITRE computer security model is presented. A suggestive interpretation of the model in the context of Multics and a discussion of several other important topics (such as communications paths, sabotage and integrity) conclude the report. A full, formal presentation of the model is included in the Appendix.

ACKNOWLEDGEMENT

Project 522B was performed by The MITRE Corporation under sponsorship of the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Bedford, Massachusetts.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

INTRODUCTION

For the past several years ESD has been involved in various projects relating to secure computer systems design and operation. One of the continuing efforts, started in 1972 at MITRE, has been secure computer system modeling. The effort initially produced a mathematical framework and a model [1, 2] and subsequently developed refinements and extensions to the model [3] which reflected a computer system architecture similar to that of Multics [4]. Recently a large effort has been proceeding to produce a design for a secure Multics based on the mathematical model given in [1, 2, 3].

Any attempt to use the model, whose documentation existed in three separate reports until this document was produced, would have been hampered by the lack of a single, consistent reference. Another problem for designers is the difficulty of relating the abstract entities of the model to the real entities of the Multics system. These two problems are solved by this document.

All significant material to date on the mathematical model has been collected in one place in the Appendix of this report. A number of minor changes have been incorporated, most of them notational or stylistic, in order to provide a uniform, consistent, and easy-to-read reference. A substantive difference between the model of the Appendix and that of the references [2, 3] is the set of rules: the specific rules presented in Appendix have been adapted to the evolving Multics security kernel design.

Because the model is by nature abstract and, therefore, not understandable in one easy reading, Section II gives a prose description of the model.

In order to relate the mathematical model to the Multics design, Section III exhibits correspondences from Multics and security kernel entities to model entities.

Section IV discusses further considerations--topics which lie outside the scope of the current model but which are important issues for security kernel design.

As background for the remainder of this document, we briefly establish a general framework of related efforts in the rest of this section.

Work on secure computer systems, in one aspect or another, has been reported fairly continuously since the mid 1960s. Three periods are discernible: early history, transitional history, and current events.

The work by Weissmann [5] on the ADEPT-50 system stands out in the early history period. Not only was a fairly formal structuring of solution to a security problem provided, but ADEPT-50 was actually built and operated. In this early period the work of Lampson [6] is most representative of attempts to attack security problems rigorously through a formal medium of expression. In Lampson's work, the problem of access control is formulated very abstractly for the first time, using the concepts of "subjects," "object," and "access matrix." The early period, which ended in 1972, understandably did not provide a complete and demonstrable mathematical formulation of a solution.

The transitional period (1972 - 1974) is characterized by markedly increased interest in computer security issues as evidenced by the Anderson panel [7]. One of the principal results of this panel was the characterization of a solution to the problem of secure computing (using the concept of a "reference monitor") together with the reasoned dictum that comprehensive and rigorous modeling is intrinsic to a solution to the problem. This period also saw the development of the first demonstrated mathematical models [1, 2, 13] as well as ancillary mathematical results which characterized the nature of the correctness proof demonstration [2, 8]. A second modeling effort, also sponsored by the Electronic Systems Division of the United States Air Force and performed at Case-Western Reserve University, was also undertaken in this period [9]. In this model, the flow of information between repositories was investigated, initially in a static environment (that is, one in which neither creation nor deletion of agents or repositories is allowed) and subsequently in a dynamic environment. Many other papers appeared during this period. An implementation of a system based on a mathematical model was carried out at MITRE by W. L. Schiller [10]. An extension and refinement of the first model was developed [3] to tailor the model to the exigencies of a proposed Multics implementation of the model; included in this extension was a concept promulgated at Case-Western Reserve concerning compatibility between the Multics directory structure and the classifications of the individual files. A great number of other computer security issues were investigated and characterized [11, 12, 13, 14, 15] during this time.

Current work succeeding the work reported above is a project sponsored by ESD and ARPA. In this project, the Air Force, the MITRE Corporation, and Honeywell are working cooperatively

to develop a design for a security kernel for the Honeywell Multics (HIS level 68) computer system. Other significant efforts include work at UCLA [16], and the Stanford Research Institute [17].

This report summarizes, both narratively and formally, the particular version of the mathematical model that is relevant to the development of a Multics security kernel. The report not only presents the model in convenient and readable form, but also explicitly relates the model to the emerging Multics kernel design to help bridge the gap between the mathematical notions of the model and their counterparts in the Multics security kernel.

SECTION II

DESCRIPTION OF THE MODEL

The model can be viewed as having three major facets--a descriptive capability (the elements), general mechanisms (the limiting theorems), and specific solutions (the rules). In this section, we shall discuss these three facets narratively, make explicit the inclusions and exclusions of meaning (that is, interpretations) that can be correctly associated with the model itself rather than with its interpretation in any given context. A summary of the model is included in the Appendix; however reference to the Appendix should not be necessary for complete understanding of this section.

DESCRIPTIVE CAPABILITY

The model has the ability to represent abstractly the elements of computer systems and of security that are relevant to a treatment of classified information stored in a computer system. †The essential problem is to control access of active entities to a set of passive (that is, protected) entities, based on some security policy. Active entities are called subjects (denoted S_i individually and S collectively); passive entities are called objects (denoted O_j and O). No restriction is made regarding entities that may be both subjects and objects: a given interpretation of the model could have no subject/objects, some subject/objects, or all subjects could be objects. It is merely required that, when an entity's active (respectively, passive) role is being considered, that entity be constrained by the model's treatment of subjects (respectively, objects).

†Note that the model is in no way restricted to a computer system (although that is the topic here). It has also been applied to physical and procedural security controls.

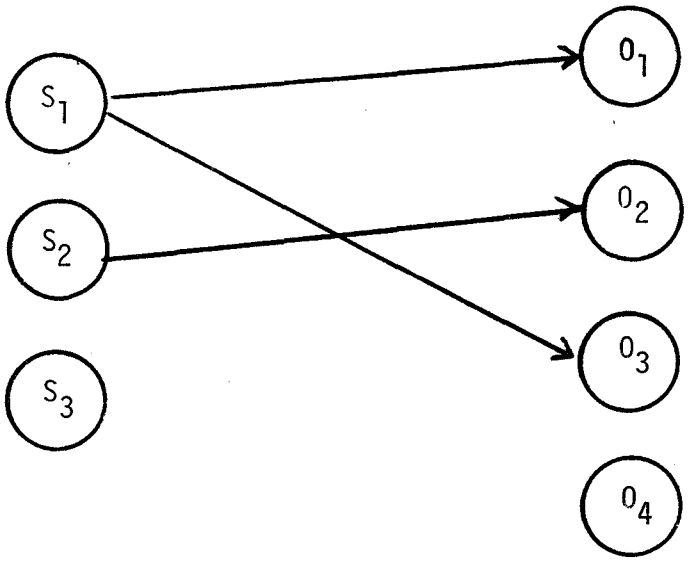


Figure 1. Subjects Accessing Objects

As in computer systems, access in the model can assume different modes. The modes of access in the model are called access attributes (denoted x and A). The access attributes are abstracted from actual access modes in computer systems.

The two effects that an access can have on an object are the extraction of information ("observing" the object) and the insertion of information ("altering" the object). There are thus four general types of access imaginable:

- no observation and no alteration;
- observation, but no alteration;
- alteration, but no observation; and
- both observation and alteration.

An access attribute for each of these possibilities is included in the model:

- e access (neither observation nor alteration);
- r access (observation with no alteration);
- a access (alteration with no observation); and
- w access (both observation and alteration).

The symbols e, r, a, and w are derived from the generalized access modes execute, read, append, and write, and in fact, the underlined words are used interchangeably with the shorter letter symbols. The meaning of any access attribute, however, is not at all constrained by an actual access mode with the same name. †Rather each actual access mode must be analyzed and paired with the access attribute which matches its own access characteristics. The only intrinsic semantics that pertain to every interpretation of the model access attributes are those listed in the preceding paragraph.

It is now possible to begin a description of a system state in the model. The state will be expressed as a set of four values, each referred to as a component.

The first component of a system state is the current access set, denoted b. A current access by a subject to an object is represented by a triple:

(subject, object, access-attribute).

This triple means that "subject" has current "access-attribute" access to "object" in the state. The current access set b is a set of such triples representing all current accesses.

The next element of a system state within the model concerns a structure imposed on the objects. What we stipulate is that a

†Note that this abstract notion of "execute" access is not what is typically implemented (enforced) by computer hardware since the results of the execution reflect the contents and thus constitute "observation" of the executed element.

parent-child relation be maintained which allows only directed, rooted trees and isolated points as shown:

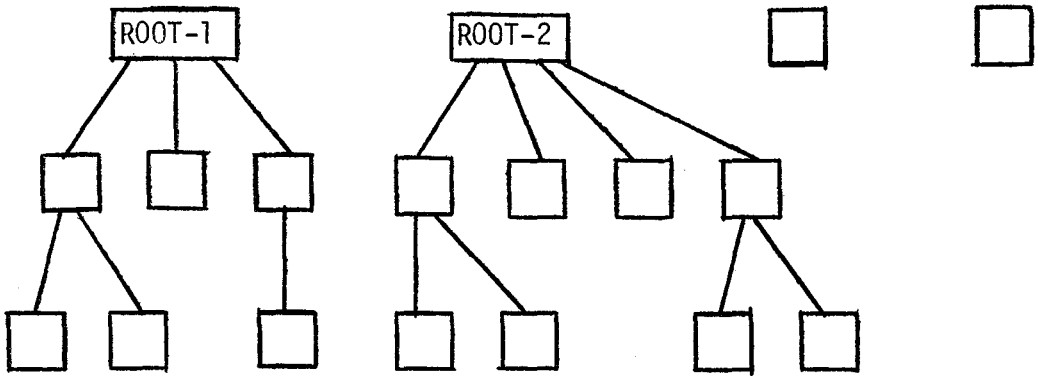


Figure 2. The Desired Object Structure

This particular structure is desired in order to take advantage of the implicit control conventions of and the wealth of experience with logical data objects structured in this way. The construct used is called a hierarchy (denoted H and *H*); a hierarchy specifies the progeny of each object so that structures of the type mentioned are the only possibilities.

The next state component which we consider involves access permission. Access permission is included in the model in an access matrix[†] M.

[†]Notice that M is a matrix only in the model's conceptual sphere: any interpretation of M which records all the necessary information is acceptable.

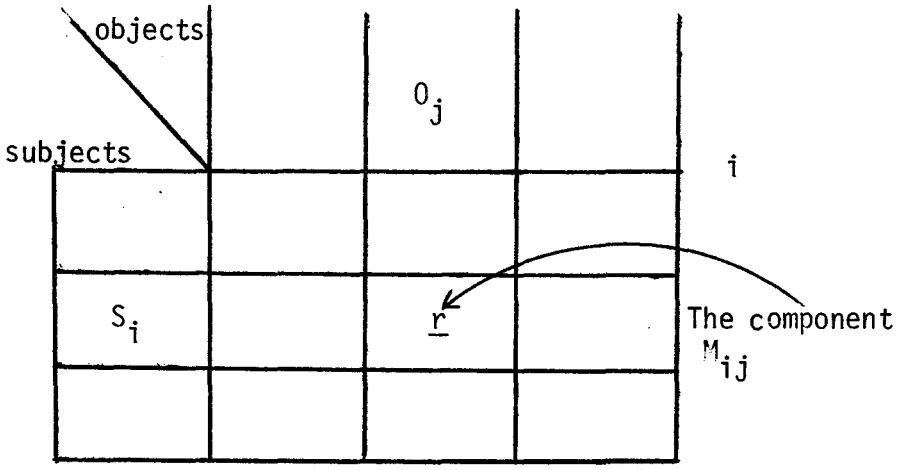


Figure 3. An Access Matrix

The component M_{ij} records the modes in which subject S_i is permitted to access object O_j. Thus the entries of M are subsets of the set A of access attributes.

The last component of a system state is a level function, the embodiment of security classifications in the model. In a military or governmental environment, people and documents can receive two types of formal security designations: one is classification or clearance (unclassified, confidential, secret, and top secret are usual) and the other is formal category (such as Nuclear, NATO, and Crypto). A total security designation is a pair:

(classification, set of categories).

Such a pair we call a "security level." A necessary condition for an individual's possession of a document is that his security level must dominate the security level of the document. One level dominates another:

(class 1, category-set 1) dominates (class 2, category-set 2)

if and only if

class 1 is greater than or equal to class 2 and
category-set 1 includes category-set 2 as a subset.

This rather complicated requirement is abbreviated in this discussion by using abstract security levels (denoted L_u and L) and a dominance ordering $\succ$ (read "dominates") which is required to be a partial ordering.[†]

The classification of subjects and objects assigns to each subject and to each object a security level. The (maximum) security level of a subject S_i is denoted "$f_S(S_i)$" in the formal development in the Appendix, but for the purposes of this section will be denoted "level(S_i)." Similarly, the security level of an object O_j is denoted formally and informally as $f_O(O_j)$ and level(O_j). One further assignment to subjects identifies the current security level of the subject. The current level allows a subject to operate at less than its maximum security level, a feature that is very important under some of the security constraints to be developed later.[††] The current security level of a subject S_i is denoted $f_C(S_i)$ and current-level(S_i); it is required that level(S_i) dominate current-level(S_i).

[†]That the relation $\succ$ must be a partial ordering requires only that 1) L_u dominates L_u for every level L_u; 2) L_u dominates L_v and L_v dominates L_w, then L_u dominates L_w; and 3) if L_u and L_w dominate each other, then they are the same.

[††]In particular, the current security level makes feasible the requirement that high-level information not be put into low-level objects.

A triple of security level assignment functions (f_S, f_0, f_C) or (level(·), level(·), current-level(·)) is called a level function and is denoted f(or, collectively, F).

A state of the model is a 4-tuple of the form:

(current access set, access permission matrix, level function, hierarchy).

The model notation for a state is (b, M, f, H).

We refer to inputs to the system as requests (R_k and R) and outputs as decisions (D_m and D). The system is all sequences of (request, decision, state) triples with some initial state (z_0) which satisfy a relation W on successive states.

The system defined in this way can be used in two ways--analysis and synthesis. The use of the model for analysis involves:

1. the specification of R and D for the system being analyzed, and
2. the determination of W.

The operation of the system of concern can then be addressed by examining the relation W which characterizes the system as a model. The use made of the model in the security kernel design work is synthesis: the job involves first the specification of system characteristics that we desire to be maintained, and then the definition of a relation W that is sufficient to the task. The definition of an appropriate relation W is the topic of SPECIFIC SOLUTIONS; we conclude this discussion with an exposition

of the system characteristics that we desire to be maintained. These characteristics we speak of collectively as "security."

The first aspect of security which we consider is the simple security property (ss-property hereafter). The ss-property is satisfied if every "observe" access triple (subject, object, attribute) in the current access set b has the property that level (subject) dominates level (object). More concisely, the ss-property stipulates that if (subject, object, observe-attribute) is a current access, then level (subject) dominates level (object).

The ss-property is the strict interpretation of the current security regulations for documents, with one modification. In a document system, "access" refers to physical possession which implies the ability to extract information. Where there is the possibility of access without observation, as in this model, access does not necessarily imply the ability to extract information. Hence, the security regulations for documents were applied in the model only to attributes that entail observation (viz. w and r).

The ss-property was considered to be the whole of security in our early efforts at modeling [1]. A brief look at the expected interpretation of the model will show that this property is indeed only a "simple" statement of the problem.

The expected interpretation of the model anticipates protection of information containers rather than of the information itself. Hence a malicious program (an interpretation of a subject) might pass classified information along by putting it into an information container labeled at a lower level than the information itself.

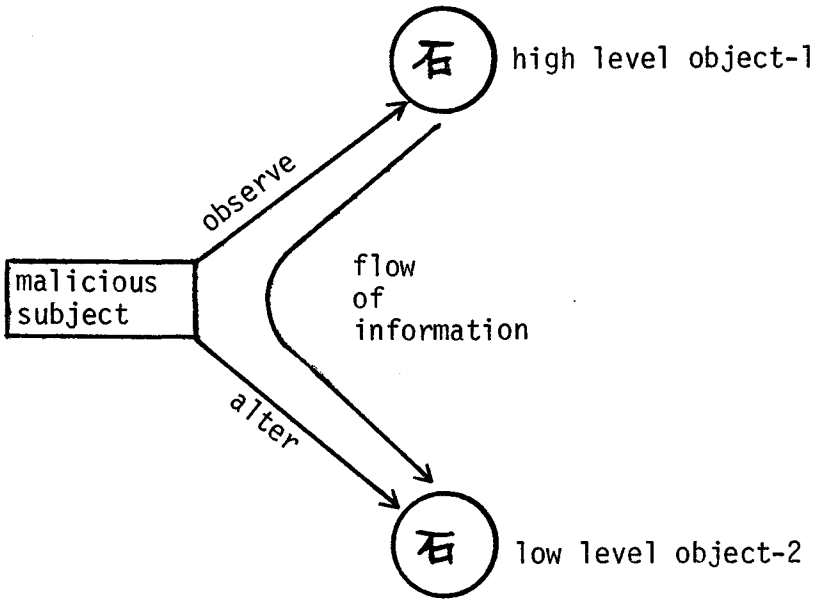


Figure 4: Information Flow Showing the Need for *-Property

Thus, another security property, called the *-property† (for historical reasons), is added to the ss-property in the specification of "security." The *-property is satisfied if:

> in any state, if a subject has simultaneous "observe" access to object-1 and "alter" access to object-2, then level (object-1) is dominated by level (object-2).

This definition clearly disallows the situation pictured (Figure 4). Under this restriction, however, the levels of all objects accessed by a given subject are neatly ordered:

> level (a-accessed-object) dominates level (w-accessed-object);
> level (w-accessed-object-1) equals level (w-accessed-object-2); and
> level (w-accessed-object) dominates level (r-accessed-object).

†read "star-property."

Thus the definition of *-property is now refined in terms of current-level (subject):

> in any state, a current access (subject, object, attribute) implies:
>
> > level (object) dominates current-level (subject) if attribute is a;
> > level (object) equals current-level (subject) if attribute is w; and
> > level (object) is dominated by current-level (object) if attribute is r.

There are two important comments to be made about the *-property. First, it does not apply to trusted subjects: a trusted subject is one guaranteed not to consummate a security-breaching information transfer even if it is possible.† Second, it is important to remember that both ss-property and *-property are to be enforced. Neither property by itself ensures the "security" we desire.

There is one further aspect of security that we address: the problem is called discretionary security and it is also based on current military/governmental policy (known as "need-to-know"). The enforcement of classification/clearance matching is mandated by executive order, directive and regulation: an individual may not exercise his own judgment to violate this standard. Similarly, the enforcement of categories (also called formal need-to-know compartments) is mandatory. These two restrictions make up nondiscretionary security policy and are

---

†The topic of trusted subjects is treated at more length in Section IV.

embodied in the model as the ss-property and *-property. Discretionary security policy allows an individual to extend to another individual access to a document based on his own discretion, constrained by non-discretionary security policy: that is, discretionary security policy allows an individual to extend access to a document to anyone that is allowed by non-discretionary security to view the document.

This exact property is included in the model in the discretionary security property (ds-property). A state satisfies the ds-property provided every current access is permitted by the current access permission matrix M. More specifically, the ds-property, requires that:

> if (subject-i, object-j, attribute-x) is a current access (is in b), then attribute-x is recorded in the (subject-i, object-j) - component of M (x is in M_{ij}).

The term "discretionary" security is appropriate in the context of the specific solutions of this model since the capability to alter M (the permission structure) is included in the model.

Note that restrictions of the concept of security will not require reproof of the properties already established because additional restrictions can only reduce the set of reachable states. The notion of "security" was purposefully made extensible in this way to allow for later refinements of the concept of security.†

GENERAL MECHANISMS

This discussion of the general mechanisms of the model is tripartite. First, the "inductive nature" of security within the

---

†Some discussion of other security-related topics which might be included in later definitions of security is given in Section IV.

model is established. Then a general construct--the rule--for the modular specification of system capabilities is defined. Finally, the relation of rule properties to system properties is established.

The first general result in the model is the basic security theorem (Corollary A1 in the Appendix). This theorem states that security (as defined) can be guaranteed systemically when each alteration to the current state does not itself cause a breach of security. Thus security can be guaranteed systemically if, whenever (subject, object, attribute) is added to the current access set b, then:

1. level(subject) dominates level(object) if attribute involves observation (to assure the ss-property);
2. current-level(subject) and level(object) have an appropriate dominance relation (to assure the *-property); and
3. attribute is contained in the (subject, object) component of the access permission matrix M (to assure the ds-property).

We say that the basic security theorem establishes the "inductive nature" of security in that it shows that the preservation of security from one state to the next guarantees total system security.

The importance of this result should not be underestimated. Other problems of seemingly comparable difficulty are not of an inductive nature. The problems of data- and resource-sharing, for example, are not inductive. In fact, the most trivial example of deadlock (Figure 5) can arise in any nontrivial sharing system that

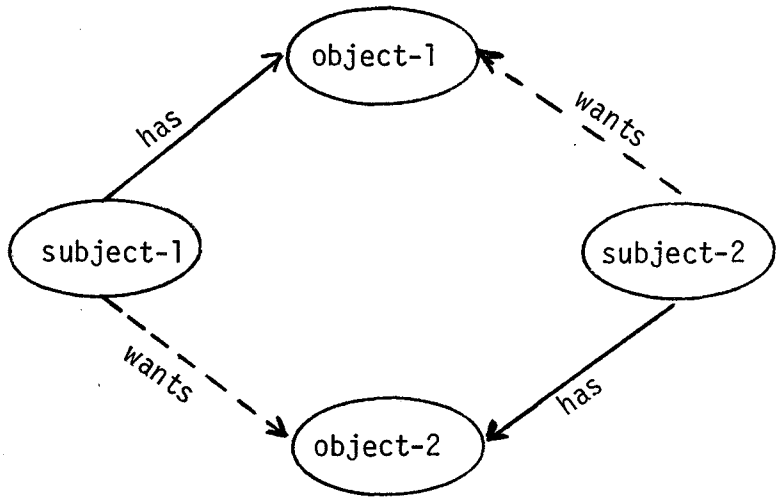


Figure 5. Deadlock

decides immediately to grant or deny a request for access. Resolution of this problem requires knowledge of future possibilities, queues of requests, and process priorities [18]. The result, therefore, that security (as defined in the model) is inductive establishes the relative simplicity of maintaining security: the minimum check that the proposed new state is "secure" is both necessary and sufficient for full maintenance of security.

The second step of constructing general mechanisms within the model is a direct consequence of the basic security theorem. Since the systemic problems of security can be dealt with one state transition at a time, a general framework for isolating single transitions was devised. This framework relies on the "rule," a function for specifying a decision (an output) and a next-state for every state and every request (an input):

$$(\text{request}, \text{current-state}) \xrightarrow{\text{rule}} (\text{decision}, \text{next-state}).$$

The idea is to analyze each class of requests separately in a rule designed to handle that particular class. To provide clarity, no two rules (in a given system) are allowed to specify non-trivial changes for a given (request, current-state) pair; total system "response" to the pair (request, current-state) is then defined as the response of the rule written to handle the request. This framework allows different approaches to a given class of requests to be worked out independently in different rules. A final set of rules to specify a desired system could be chosen to reflect idiosyncratic needs; the only restriction is that rules with overlapping responsibility cannot be used together. This approach gives the model a modular flexibility which can be of great use in tailoring the model to a particular application, as illustrated by Section III.

The last development which is classed a general development centers on the relation of rule properties to system properties. It has been shown that the entire system specified by a set of rules satisfies all three security properties--the ss-property, the *-property, and the ds-property--provided each rule itself introduces no exception to these properties. Moreover, the requisite demonstration that a rule preserves security can in most cases be reduced to the direct consideration of the small number of state alterations involved in the given state transition (Corollary A3 in the Appendix).

In summary, the general mechanisms of the model:

- bound the scope of investigation to single transitions of state;
- provide the ability to investigate desired features of the system independently of one another using the rule framework; and

- reduce the systemic problem to very restricted rule-based problems of the preservation of security properties over one transition.

SPECIFIC SOLUTIONS

The rules presented in this document represent one specific solution to the requirement for a "secure" computer system. This particular solution is in no sense unique, but has been specifically tailored for use with a Multics-based information system design. For this use, the solution has to satisfy two requirements: the provision of generally useful functions and appropriate accommodations to the effects of the Multics design on an implementation of this model.

A number of general functions can be suggested for any computer-based information system. With reference to the model described earlier, the functions can be grouped in four classes:

- functions to alter current access (the set b);
- functions to alter the level functions (the values level(subject), level(object), and current-level(subject));
- functions to alter the current access permission structure (the matrix M); and
- functions to alter the object structure (the hierarchy H).

This list covers changes to each of the elements of a system state in the model. Our particular solution includes the capability to cause the following changes to the system state:

- altering current access:
    - to get access (add a triple (subject, object, attribute) to the current access set b), and
    - to release access (to remove an access triple from the current access set b);
- altering level functions:
    - to change object level (to change the value of level(object) for some object), and
    - to change current level (to change the value of current-level(subject));
- altering access permission:
    - to give access permission (to add an attribute to some component of the access permission matrix M), and
    - to rescind access permission (to delete an attribute from some component of the access permission matrix M); and
- altering the hierarchy:
    - to create an object (to attach an object to the current tree structure as a leaf), and
    - to delete a group of objects (to detach from the hierarchy an object and all other objects "beneath" it in the hierarchy).

Section III presents a more detailed discussion of the particular rules presented in this document.

These rules reflect several characteristics of the Multics operating system. The main Multics characteristic that affects the model is the hierarchical object structure which has been mentioned previously. The principal reason for the inclusion of the

hierarchy in the model is the desire to disturb the Multics operating system as little as possible while adding the capability to process simultaneously information of varying security levels. The basic Multics mechanisms for access control rely heavily on the object structure: to retain that basic structure it is necessary to investigate our restrictions on access control in the Multics setting of an object hierarchy--that is, in the setting of Multics control structures.

The second Multics characteristic involves the physical counterpart of the access permission matrix M. This structure (called the Access Control List (ACL) in Multics), its location, and its manipulation have direct effects on the capability to get access, to give access, and to rescind access in Multics. The Access Control List in Multics is a list of "(process, ring bracket)" pairs [†] (for our purposes here, the Multics analogue of subjects) allowed to access a segment (that is, an object) and the modes of access allowed. There is one Access Control List for every segment/object. Thus the information contained in the Access Control List for object-j includes the information contained in the j-th column of the access permission matrix M in the model. The most important fact about the Multics ACLs is that they are contained in a segment's parent directory (parent object in the model) and are manipulated by manipulation of the object's parent. Hence, "control" over an object (to extend access, to rescind access, or to destroy the object althogether) is equivalent in Multics to write permission to the object's parent. Moreover, since "creation" of a segment in Multics is the insertion of a new entry (called a "branch") in a directory segment, the "control" over creation is equivalent to write or append access (that is, read/write or pure-write access) to the directory segment that will be the parent of the created segment (directory Z in Figure 7).

[†]The entry into the ACL by process is actually indirect: a process maps to a "user-id" (essentially a set of processes associated with a particular user) which in turn maps to an ACL entry. To simplify the exposition here, this indirect entry is represented directly.

Matrix M

	. . .	O_j	. . .
S_1		r e w a	
S_2		ϕ	
S_3		r e	

is represented by

ACL for O_j

process	attributes	ring brackets
S_1	r e w a	
S_3	r e	
. . .		

Figure 6. The Correspondence of M Columns to ACLs

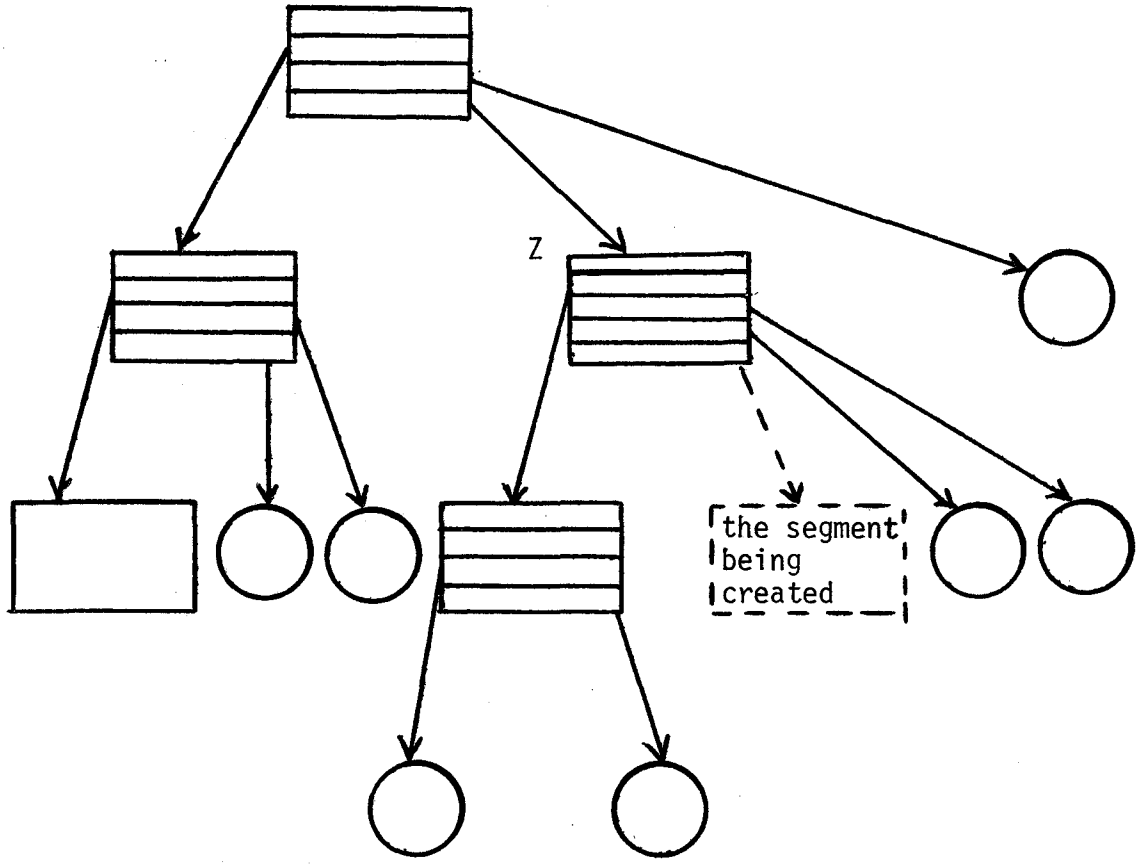


Figure 7. The "Creation" of a Segment in Multics

These Multics characteristics are taken into account in the model's rule where, for example, a request to give access to an object is allowed only if (among other things) the requesting subject has current w access to the parent of the object (implying that the usual Multics operation of extending access can be carried out).

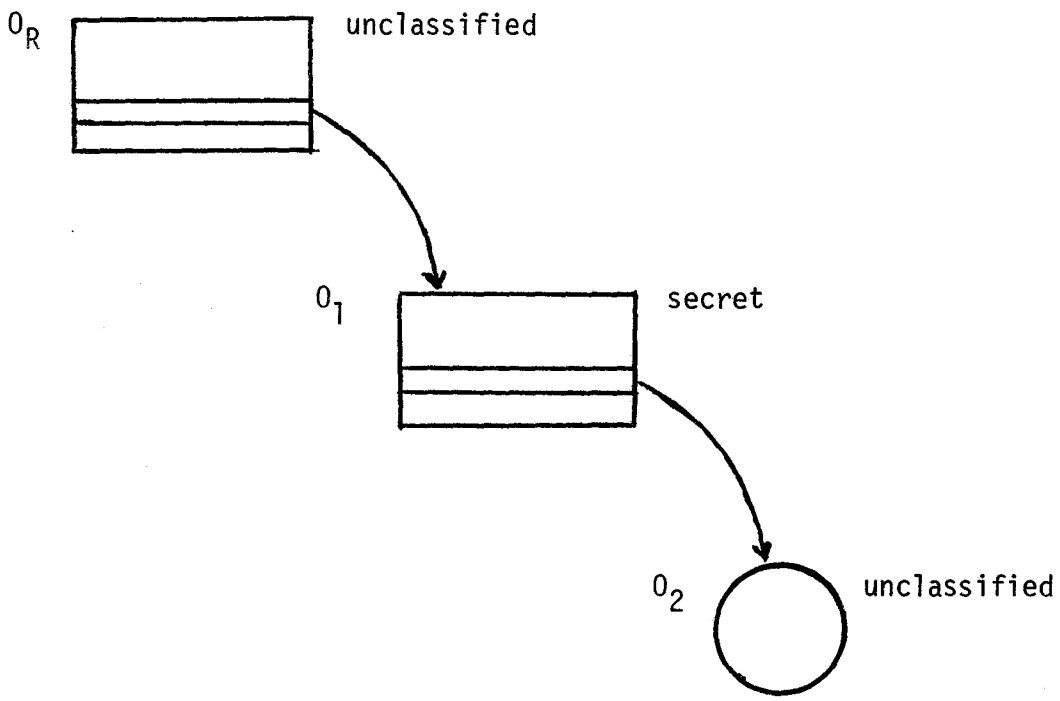


Figure 8. The Need for Compatibility

The way access to an object is carried out in Multics is the final characteristic reflected in the model. A user request to access a segment causes the user's surrogate (his process) to access every object in the hierarchy in the path from the root directory (the object O_R in the model) to the segment of interest. This fact implies that in the situation shown in Figure 8, an unclassified subject would have to observe the secret object O_1 in order to access the unclassified object O_2: an unclassified subject cannot observe the secret object O_1 because of the ss-property. Moreover, the *-property combined with the requirement to "write" in O_1 in order to "create" object O_2 make any situation similar to that in Figure 8 useless. Hence, it is required in the rules of the model that the security level of an object dominate the security level of its parent.[†] The rules to allow creation of objects and to cause changes in an object's security level reflect this requirement, which is termed "compatibility."[††]

The rules of this document provide a particular specification for a secure computer system that supplies a full complement of information processing capabilities while matching the special requirements of the Multics operating system environment.

[†]Remember that if the two levels are the same, this requirement is met.

[††]The concept termed "compatibility" here was initially proposed and investigated at Case Western Reserve University [9].

SECTION III

MORPHISM FROM MULTICS TO MODEL

INTRODUCTION

The discussion of the correspondence of the Multics security kernel design to the mathematical model† will be phrased in terms of a "morphism;" this stance is taken because of the verification strategy that has been proposed for the Multics kernel design [19].

A morphism is a mapping from one system to another which preserves one or more operations of the system. This concept can be stated mathematically in concise form. Exposition of the concept is better achieved by example. Suppose [I, +, ·] is the following algebraic system:

I is the set of integers from 0 to 9.
+ is the ordinary arithmetic sum operator except addition is to be done modulo 10; that is, ordinary sum equal to 10 becomes 0, 11 becomes 1, 12 becomes 2, and so forth.
· is the ordinary arithmetic product operator except multiplication is to be done modulo 10.

Suppose [A, $\oplus$, $\odot$] is the following algebraic system:

A is the set of letters a, b, c, d, e.
$\oplus$ is a binary operator defined as follows:

---

†The term "model" refers specifically to the model presented in the Appendix.

a $\oplus$ any letter in A = that letter	c $\oplus$ c = e
b $\oplus$ a = b	c $\oplus$ d = a
b $\oplus$ b = c	c $\oplus$ e = b
b $\oplus$ c = d	d $\oplus$ d = b
b $\oplus$ d = e	d $\oplus$ e = c
b $\oplus$ e = a	e $\oplus$ e = d

which can be shown in table form:

$\oplus$	a	b	c	d	e
a	a	b	c	d	e
b	b	c	d	e	a
c	c	d	e	a	b
d	d	e	a	b	c
e	e	a	b	c	d

.

$\odot$ is a binary operator defined by:

$\odot$	a	b	c	d	e
a	a	a	a	a	a
b	a	b	c	d	e
c	a	c	e	b	d
d	a	d	b	e	c
e	a	e	d	c	b

.

Now define the mapping M from the system [I, +, $\cdot$] to the system [A, $\oplus$, $\odot$] as follows:

$0 \longrightarrow a$
$1 \longrightarrow b$
$2 \longrightarrow c$
$3 \longrightarrow d$
$4 \longrightarrow e$
$5 \longrightarrow a$
$6 \longrightarrow b$
$7 \longrightarrow c$
$8 \longrightarrow d$
$9 \longrightarrow e$

M is then a morphism from $[I, +, \cdot]$ to $[A, \oplus, \odot]$ since it "preserves" the operations of + and $\cdot$. This means that the value of the expressions $i + j$ and $i \cdot j$ in the system $[I, +, \cdot]$ have corresponding values in $[A, \oplus, \odot]$ under the mapping M which is the same as the value obtained by $\oplus$ ing and $\odot$ing the elements in $[A, \oplus, \odot]$ which correspond under M to i and j in $[I, +, \cdot]$. Symbolically we can express this as follows:

$$M(i + j) = M(i) \oplus M(j) \text{ and } M(i \cdot j) = M(i) \odot M(j).$$

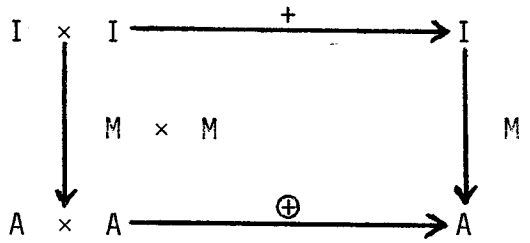
By inspecting the previous definitions we can verify, for example, that:

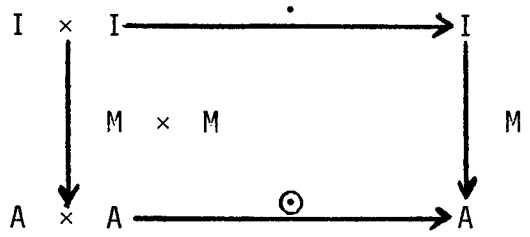
$M(1 + 3) = M(4) = e$ and
$M(1) \oplus M(3) = b \oplus b = e$ so
$M(1 + 3) = M(1) \oplus M(3)$,

Similarly,

$M(7 \cdot 3) = M(7) \odot M(3)$ since
$M(7 \cdot 3) = M(1) = b$ and
$M(7) \odot M(3) = c \odot d = b.$

The "preservation" property of M can be shown diagrammatically:

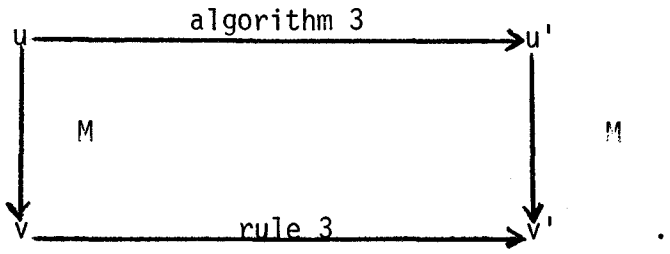




These diagrams are said to be "commutative." In each, one can get from $I \times I$ to A by two paths; each path leads to the same place, that is, given two elements in I (an ordered pair in $I \times I$) the same element in A is arrived at by both paths.

The math model of a secure system is like the system $[A, \oplus, \odot]$. Corresponding to the set A is a set of elements of the model. The analogy is most enlightening if we consider elements in A to correspond to states in the model. Corresponding to the operators $\oplus$ and $\odot$ is a set of eleven rules. The Multics system we shall discuss is like the system $[I, +, \cdot]$. Corresponding to the set I is a set of elements of the system; again, consider the latter to be

states of the system. Corresponding to the operators + and · is a set of algorithms. Now, just as we established a morphism from [I, +, ·] to [A, ⊕, ⊙], we wish to establish a morphism from Multics to the model. In other words, given a set of algorithms for "secure" operation, which correspond to rules of the model, we wish to establish a mapping from the elements of Multics to the elements of the model in such a way that the algorithms (operations) are preserved. For each algorithm we wish to be able to specify a commutative diagram; for example:



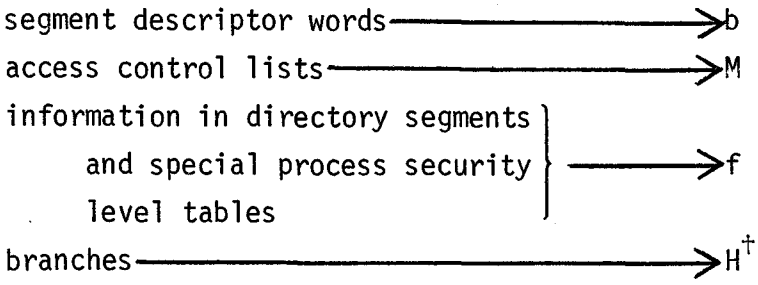
In this document the mapping M is partially specified. The algorithms then are to be so specified as to be able to show that M preserves operations; this specification is outside the scope of this report.

In the remainder of this section we identify the elements of Multics and then show a preliminary correspondence of the identified elements to the elements of the model. It remains for future effort to show that the correspondence is a morphism.

ELEMENTS OF A SECURE MULTICS

State Elements

Corresponding to a state (b, M, f, H) in the model is a set of information structures in Multics. The following correspondences have been identified:



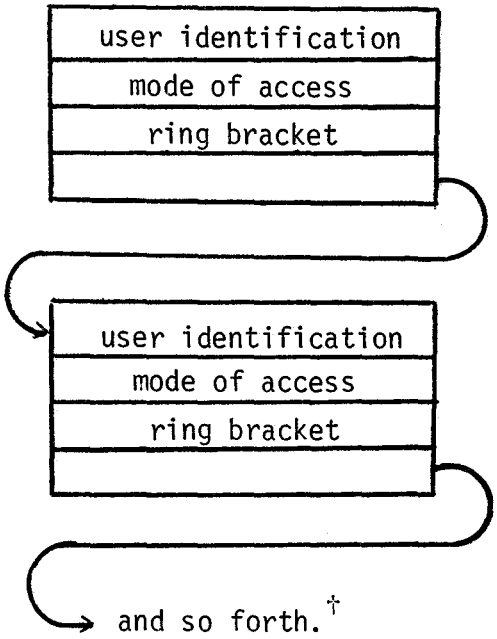
An element $(S_i, O_j, \underline{x})$ in b indicates that subject S_i has current access to object O_j in access mode $\underline{x}$. In Multics the same information is contained in a descriptor segment base register (DSBR), a temporary pointer register (TPR), and a segment descriptor word (SDW). An address field in the DSBR is a pointer to the head of the descriptor segment for the process (subject) that is currently running on the processor to which the DSBR belongs. The TPR gives an offset, in the descriptor segment, to the SDW associated with the segment (object) to which the process has access. In the SDW is a field which indicates access permission (namely, read, execute, or write). When a process is ready or waiting (not running) the information in the DSBR and TPR is saved in the active segment table.

In case the object referred to in a triple of the form $(S_i, O_j, \underline{x})$ is something other than a segment, say a socket††, correspondences like those shown above must pertain.

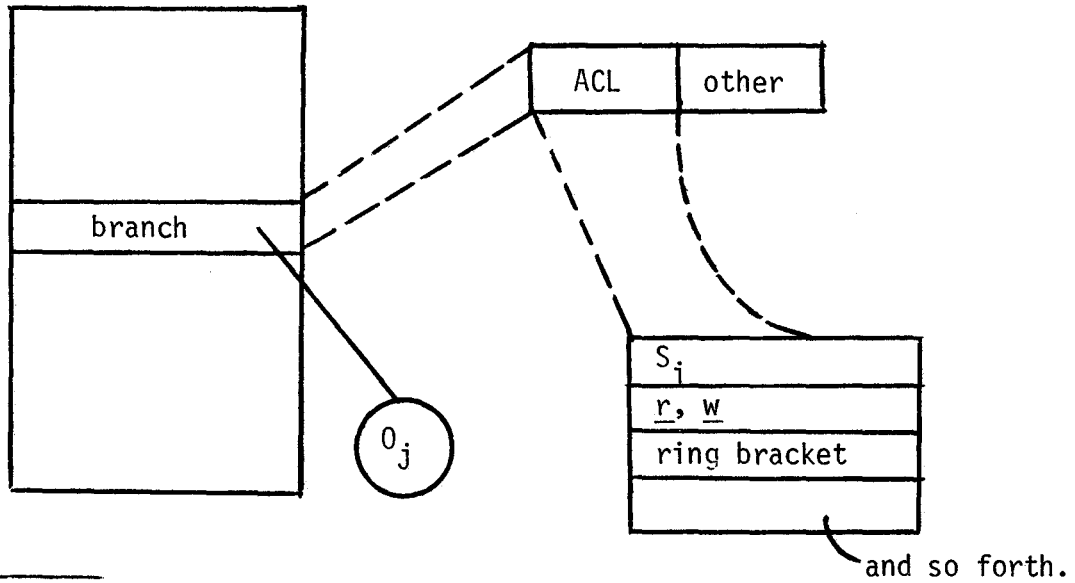
An entry $\alpha_{ij} = \{\underline{r}, \underline{w}\}$ in M indicates that subject S_i has read and write permission with respect to object O_j. Suppose O_j is a data segment. In Multics this information is kept in an access control list. An access control list has the following form:

†The Multics described in this report is derived from Organick's The Multics System [4]. Multics, as an evolving system, currently may not fit this description, but at this writing, the variations were of little importance to the discussion.

††The term "socket" denotes a connection from a process to a physical device for input or output operations.



The access control list (ACL) together with other information (e.g., physical location) makes up a branch. A collection of branches is a directory segment. Corresponding to α_{ij} then we have:



†Currently, ring brackets are associated with segments rather than ACL's; this presentation follows Organick.

The security level function f of the model has the three components:

f_S: maximum security level of subjects;
f_C: current operating security level of subjects;
f_O: security level of objects.

For example, $f_O(O_j)$ = confidential means that O_j is classified confidential. This information would be kept in a directory segment in Multics, perhaps as an extension of a branch. Specific information structures for representing f_S and f_C have not yet been chosen at this writing; we postulate appropriate tables at a high level of abstraction for establishing correspondence to the model.

The hierarchy H of the model is structured to reflect the tree structure among segments realized by branches in Multics; correspondence is quite straightforward. If O_i and O_j are objects in the model and $H(O_i)$ includes O_j, then O_i is the parent of O_j; the Multics structural equivalent of this situation is shown in Figure 9.

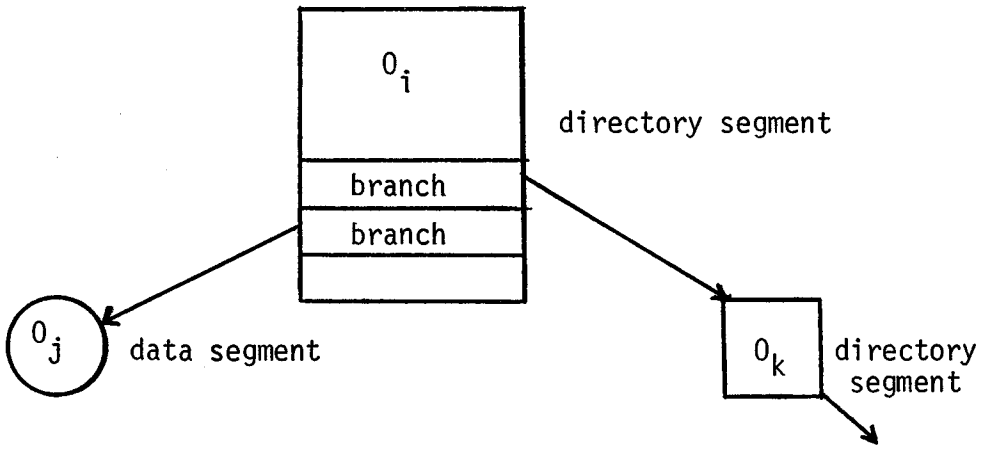


Figure 9. Multics Hierarchy Equivalent

With respect to the model, the Multics link is considered a shorthand for a symbolic pathname: therefore, it introduces no additional structure.

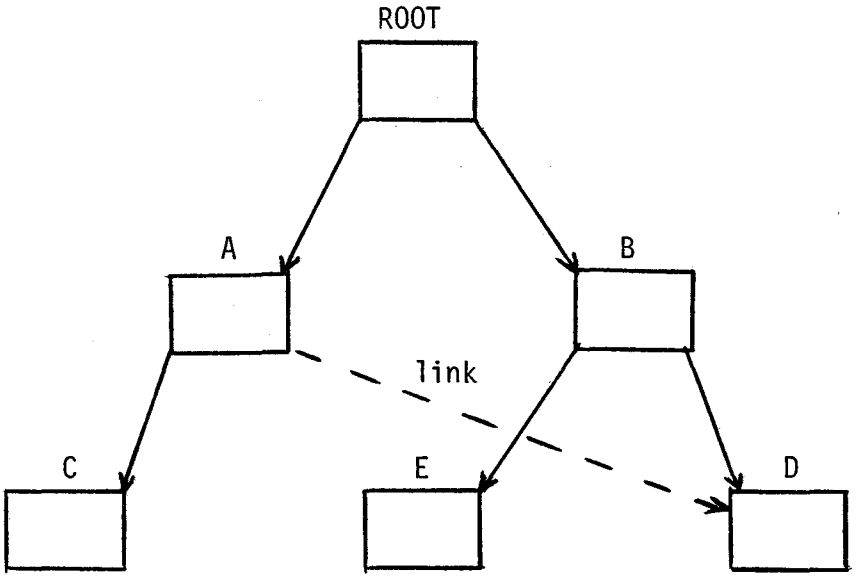


Figure 10. The Interpretation of Links

From directory A in Figure 10, the symbolic name "D" is shorthand for ">B>D."

Subjects and Objects

A process-ring pair (process, ring) in Multics corresponds to a subject in the model. Corresponding to objects in the model are, at least, directory segments, data segments, certain I/O devices, certain address spaces, and sockets.

Attribute Elements

The set A ={r, e, w, a} is used in the model for access mode designation with the following meanings:

r--read; observe only
e--execute; neither observation nor alteration
w--write; observe and alter
a--append; alter only.

For data segments in Multics the usage attributes correspond as follows:

Multics	Model
read	r
execute	r, e
read and write	w
write	a.

For directory segments the correspondences are:

Multics	Model
status	r
status and modify	w
append	a
search	e.

For other objects in Multics the access attributes have not yet been specified sufficiently to permit exact correspondences to be established at the time of this writing.

Corresponding to the set $C = \{C_1, C_2, \ldots, C_q\}$ of classifications in the model is a set of classifications in Multics:

top secret $\longrightarrow C_1$
secret $\longrightarrow C_2$
confidential $\longrightarrow C_3$
unclassified $\longrightarrow C_4$.

Corresponding to the categories $K = \{K_1, K_2, \ldots, K_r\}$ of the model is a set of formal categories in Multics. The four classifications above have been adopted for general use [5]; the formal categories used in any particular installation will vary. For example, an installation might establish the correspondence:

NATO $\longrightarrow K_1$
CRYPTO $\longrightarrow K_2$
NOFORN $\longrightarrow K_3$.

For the present implementation, a maximum of 7 categories has been adopted as the standard.

SECURITY PROPERTIES IN A SECURE MULTICS

With the Multics/model element correspondences as a foundation, the examination of a secure Multics can proceed with an examination of the properties of Multics which will be deemed "security" properties. Among these properties are the Multics analogues of the security properties in the model; the identification of other security properties in Multics is also included here.

The first model property reflected in a secure Multics is the ss-property, or simple-security property. This property embodies the military/governmental policy on disclosure, tailored to a computer environment. In the model, the ss-property requires that every current access involving observation (an element (subject, object, observe-attribute) in the current access set b) must imply that the level of the subject dominates the level of the object observed

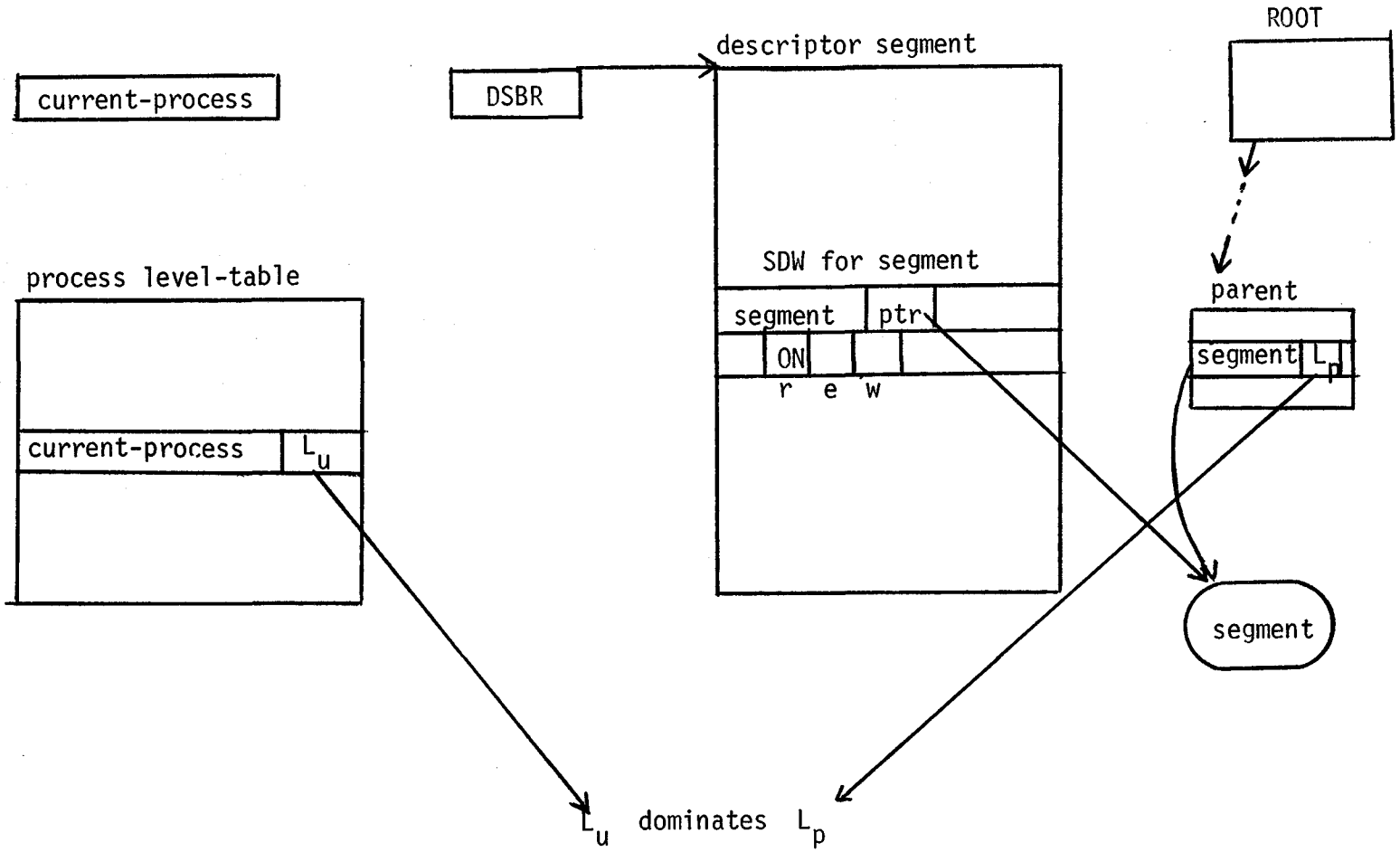


Figure 11. The ss-Property in Multics

(level(subject) $\geq$ level(object)). In Multics, an SDW in an active segment's descriptor segment with the r indicator on indicates a current observe for that process. (Recall that in Multics "read" is the only observe access to data segments; "status" plays the identical role for directory segments.) Thus, for an active process, compliance with the ss-property means that the r (or s) indicator is on only in those SDWs where the level of the process dominates the level of the segment described by the SDW (see Figure 11). For an inactive process, compliance with the ss-property means that on activation the currently stored process information would conform to the requirements for an active process.

In the model, the *-property places restrictions on current access triples (subject, object, attribute) based on the value of current-level(subject). Specifically,

- if attribute is read, current-level(subject) dominates level(object);
- if attribute is append, current-level(subject) is dominated by level(object);
- if attribute is write, current-level(subject) equals level(object); and
- if attribute is execute, current-level(subject) and level(object) have no required relation.

In Multics, the *-property can be phrased for active processes, the requirement for inactive processes being, as for the ss-property, that on activation the restrictions on active processes be satisfied. For any SDW of an active process's descriptor segment, the current-level of the process:

- must dominate the level of a segment having the r indicator on and the w indicator off (respectively, the s indicator

on and the m indicator off) as shown for segment-1 in Figure 12.a;

- must be dominated by the level of a segment having the r indicator off and the w indicator on (respectively, the s indicator off and the a indicator on) as shown for segment-2 in Figure 12.b;
- must equal the level of a segment having both the r and w (respectively, s and m) indicators on (segment-3 in Figure 12.c);
- must dominate the level of a segment having the e indicator on and the w indicator off (segment-4 in Figure 12.d).

In the model, the ds-property requires that every current access (a triple (subject, object, attribute) in the current access set b) be permitted by the current access permission matrix M (attribute is an element of the (i, j)-component of M). The exactly analogous condition in Multics is required for the satisfaction of the ds-property. For every SDW and every access indicator that is on in the SDW, the branch in the segment's parent to the segment described by the SDW has the same access indicator on. In Figure 13, α_1 = ON implies β_1 = ON; α_2 = ON implies β_2 = ON; and α_3 = ON implies β_3 = ON. Note that (α_1, α_2, α_3) = (ON, OFF, OFF) and β_1, β_2, β_3) = (ON, ON, ON) satisfy the ds-property. Note that the maximum access permitted need not be present in the SDW. As before, an inactive process is required to be described dormantly so that on activation the above condition holds true.

There are several other important security properties being considered in the development of a secure Multics. Two important correlative properties are sabotage and communication paths. "Sabotage" in this context means the malicious alteration or destruction of data, especially data related to the operation of

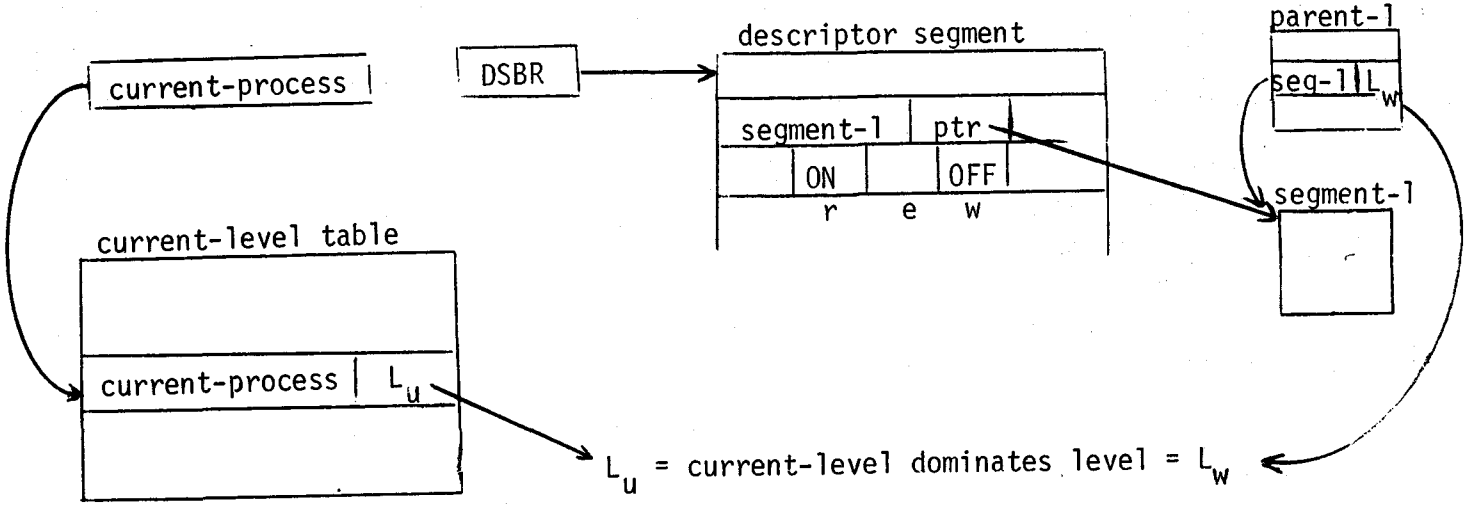


Figure 12a. The *-Property for Multics read

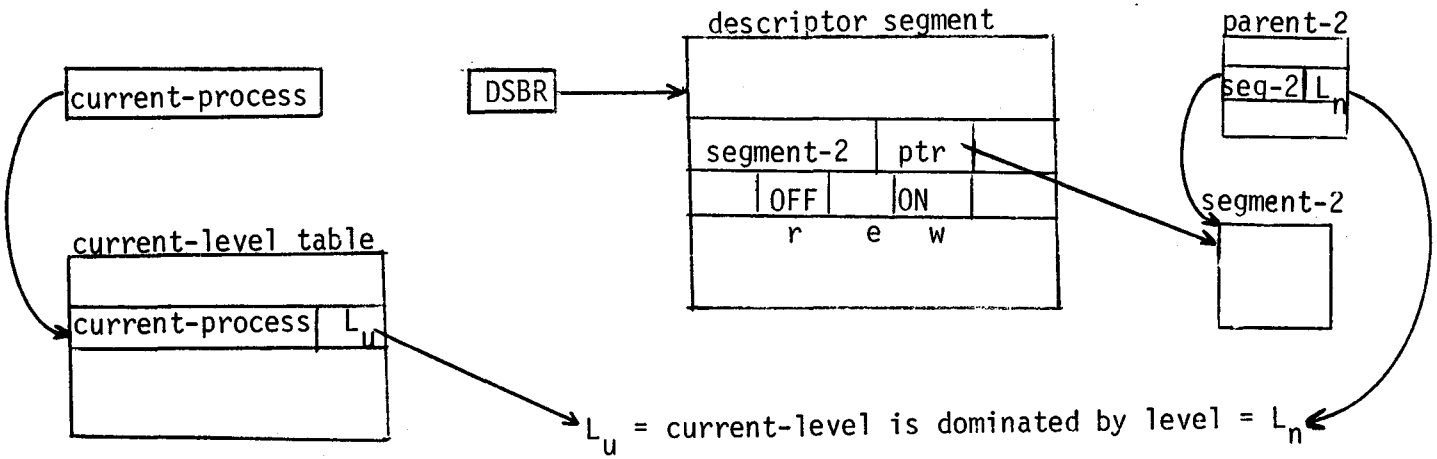


Figure 12 b. The *-Property for Multics write (only)

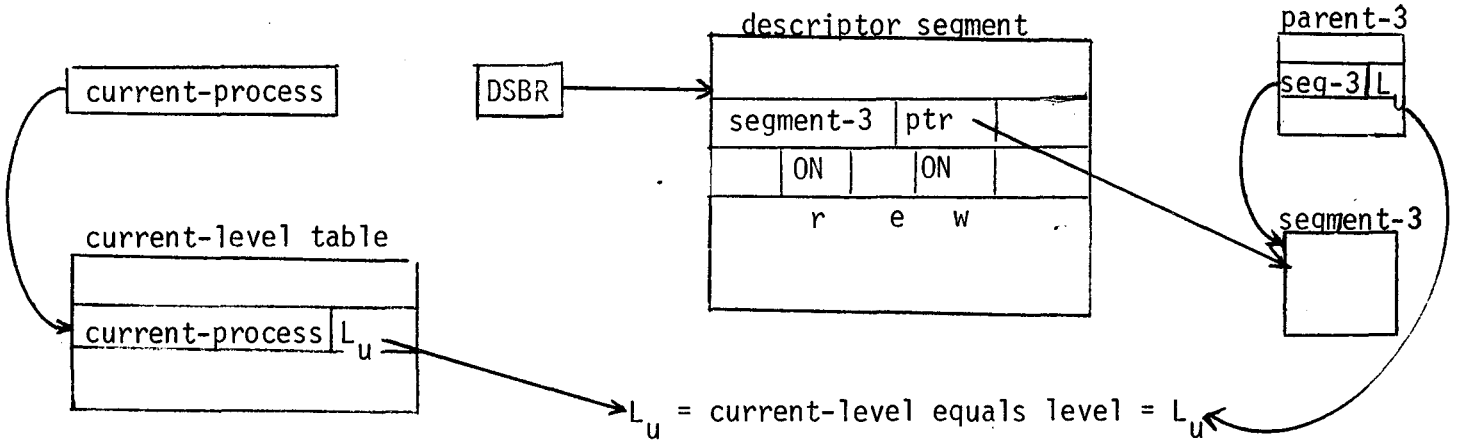


Figure 12c. The *-Property for Multics read-write

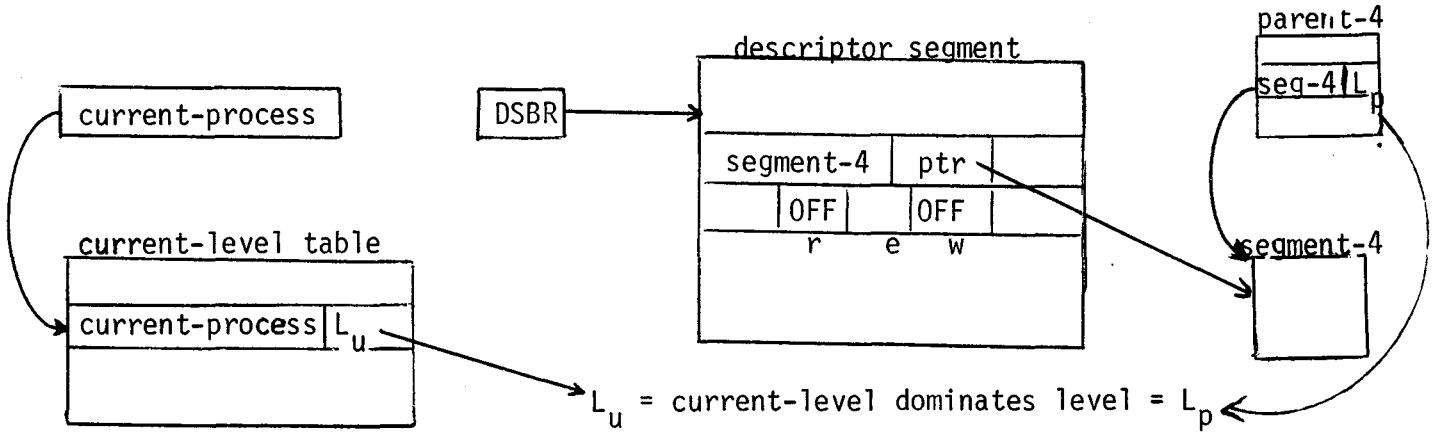


Figure 12d. The *-Property for Multics execute

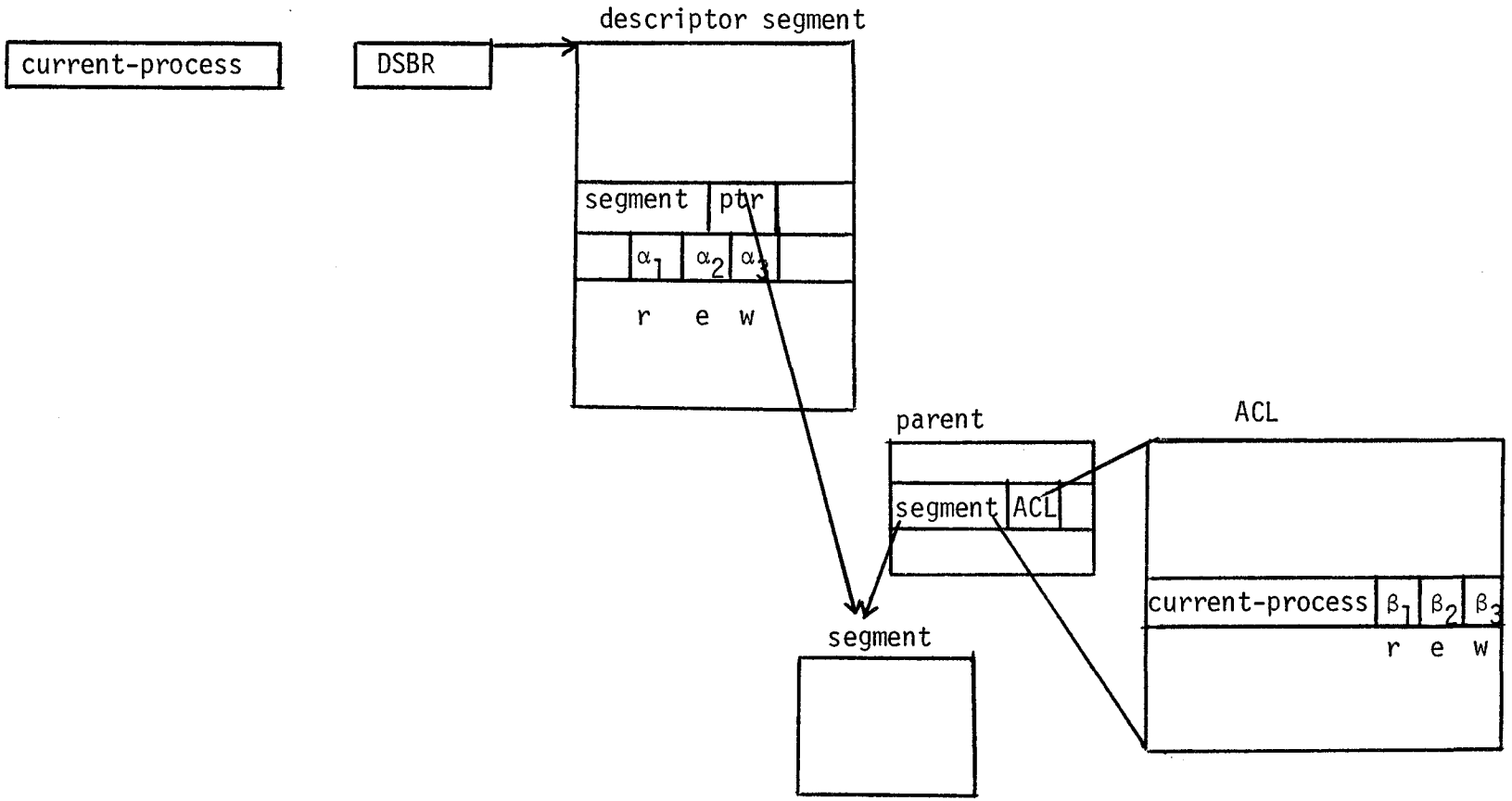


Figure 13. The ds-property in Multics

critical programs. The matter of communication paths centers on the possibility of information transmission using observable system characteristics and a prearranged code to semaphore critical information to an undercleared subject/process. Neither of these topics is directly addressed by the mathematical model, although both can be satisfactorily resolved using the model as a paradigm; discussion of these security properties is included in the section FURTHER CONSIDERATIONS.

RULES OF OPERATION FOR A SECURE MULTICS

Kernel primitives for a secure Multics will be derived from a higher level user specification and will serve to match the user specification to the particulars of the Multics architecture. Current planning is based on the desire to change the Multics architecture as little as possible; this will account to a large extent for radical differences in form between actual kernel primitives and the rules of the model.

In the interests of exposition and better understanding, a set of imaginary kernel primitives is presented here. They are essentially a transliteration of the model rules using Multics terminology and elements. In this exposition the get-access rules of the model are translated into separate kernel functions, one for each of read, write-only write, execute attributes of the model. In Multics the current operation is such that only one access function serves: when a segment fault occurs (for example, as a result of a load or store), an SDW is created, if possible and allowable, with all allowable bits on (the r, e, and w indicators) which are on in the user's ACL.

Another difference between the set of model rules and the projected kernel primitives is that there will be neither a change-subject-

current-security-level nor a change-object-security-level kernel primitive. Nevertheless, descriptions of these rules as well as the other nine rules of the model will be given here.

For purposes of exposition each informally specified kernel function is given a name of the form kernel function i (kfi) with kf1 corresponding the rule 1, kf2 corresponding to rule 2, and so forth. Objects will be considered to be data segments; similar operations would pertain for other objects.

kernel-function 1: get-read

Request has the elements:

(a) get-access
(b) process-id
(c) segment-id
(d) read

Process process-id requests that access to data segment segment-id in usage mode read be enabled.

The following conditions are checked:

(i) the ACL (in the directory segment which is the parent of segment-id unless segment-id = Root) lists process-id with read usage (for segment-id).

(ii) the security level of process-id, as given in the security level table, dominates the security level of segment-id, as given in the branch extension in the directory segment which is the parent of segment-id.

(iii) process-id is a trusted subject or the current security level of process-id, as given in the current security level table, dominates the security level of segment-id.

If conditions (i) - (iii) are met, then a segment descriptor word (SDW) is added to the descriptor segment of process-id.† The

†If the SDW already exists, then the following actions are still appropriate--essentially the appropriate access mode bit is turned on in the existing SDW. This remark pertains in following rules also.

SDW has the read bit on, is pointed to by a temporary pointer register (TPR), and points to segment-id. The process-id receives an affirmative response.

Otherwise process-id receives a negative response from the kernel.

kernel function 2: get-write-only

Request has the elements:

(a) get-access
(b) process-id
(c) segment-id
(d) write.

Process process-id requests that access to data segment segment-id in usage mode write be enabled.

The following conditions are checked:

(i) the ACL in the directory segment which is the parent of segment-id lists process-id with write usage.
(ii) process-id is a trusted subject or the security level of segment-id dominates the current security level of process-id.

If conditions (i) - (ii) are met, then a SDW is added to the descriptor segment of process-id. The SDW has the write bit on, is pointed to by the TPR, and points to segment-id. The process process-id receives an affirmative response.

Otherwise process-id receives a negative response from the kernel.

kernel function 3: get-execute

From the viewpoint of usefulness (not security), this function is appropriate only if the segment identified in the request for access is a procedure segment.

Request has the elements:

(a) get-access
(b) process-id
(c) segment-id (procedure-id)
(d) execute

Process-id requests that execute access to procedure-id be enabled.

An appeal to rule kfl is made with "execute" replacing "read" in condition (i) and in the action description.

kernel-function 4: get-read-write

One of a number of possible forms for kf4 is shown here.

Request has the elements:

(a) get-access
(b) process-id
(c) segment-id
(d) read and write

Process-id requests that read and write access to segment-id be enabled.

Action of kf4:

(a) appeal to kf1
(b) if response from kf1 is affirmative then appeal to kf2; otherwise response is negative
(c) if response from kf2 is affirmative, then response is affirmative; otherwise, response is negative.

kernel-function 5: release-read/execute/write

Request has the elements:

(a) release-access
(b) process-id
(c) segment-id
(d) usage attribute

Process-id requests that read, execute, or write access to segment-id be disabled.

The read, execute, or write bit in the SDW pointed to by TPR is turned off. If no other access bits are on, then the SDW is removed from the descriptor segment of process-id.

kernel-function 6: give-read/execute/write

Request has the elements:

(a) give-access
(b) requesting-process-id
(c) receiving-process-id
(d) segment-id
(e) usage-attribute (read, execute, or write)

Requesting-process-id gives to receiving-process-id usage-attribute access to segment-id.

The following conditions are checked:

(i) neither the parent of segment-id nor the segment segment-id itself is the root of the directory hierarchy and the SDW for the parent of segment-id has the write indicator on.

(ii) the segment segment-id is the root object of the directory hierarchy or is directly inferior to the root and requesting-process-id is allowed to give access permission to the segment in the current state.

If either condition (i) or condition (ii) is met and segment-id is not the root object, then an entry is added to the ACL in the directory segment which is the parent of segment-id; this ACL lists receiving-process-id with usage-attribute usage (to segment-id). If condition (ii) is met and segment-id is the root, then permission

for receiving-process-id to access segment-id in usage-attribute mode is recorded. Requesting-process-id receives an affirmative response.

Otherwise requesting-process-id receives a negative response.

kernel-function 6: give-read/execute/write

Request has the elements:

(a) give-access
(b) requesting-process-id
(c) receiving-process-id
(d) segment-id
(e) usage-attribute (read, execute, or write)

Requesting-process-id gives to receiving-process-id usage-attribute access to segment-id.

The following conditions are checked:

(i) neither the parent of segment-id nor the segment segment-id itself is the root of the directory hierarchy and the SDW for the parent of segment-id has the write indicator on.

(ii) the segment segment-id is the root object of the directory hierarchy or is directly inferior to the root and requesting-process-id is allowed to give access permission to the segment in the current state.

If either condition (i) or condition (ii) is met and segment-id is not the root object, then an entry is added to the ACL in the directory segment which is the parent of segment-id; this ACL lists receiving-process-id with usage-attribute usage (to segment-id). If condition (ii) is met and segment-id is the root, then permission

for receiving-process-id to access segment-id in usage-attribute mode is recorded. Requesting-process-id receives an affirmative response.

Otherwise requesting-process-id receives a negative response.

kernel-function 7: rescind-read/execute/write

Request has the elements:

(a) rescind-access
(b) requesting-process-id
(c) receiving process-id
(d) segment-id
(e) usage-attribute

Requesting-process-id takes from receiving-process-id usage-attribute access to segment-id.

The conditions checked are the same as the conditions of kf6 except, of course, "rescind" replaces "give" in condition (ii).

If either condition (i) or condition (ii) is met, then the usage-attribute is removed from the receiving-process-id's ACL entry in the directory segment which is the parent of segment-id; if no other usage attributes are left in this entry, then the entry is deleted. Requesting-process-id receives an affirmative response.

Otherwise a negative response is given.

kernel-function 8: create-object

Request has the elements:

(a) generate-leaf-segment
(b) process-id
(c) segment-id
(d) security-level (sec-level)

Process process-id requests that a segment be added to the directory hierarchy directly below directory segment segment-id; the added segment is requested to have level sec-level.

The following conditions are checked:

(i) the SDW in the descriptor segment corresponding to the directory segment-id has the w bit turned on.

(ii) sec-level dominates the security level of segment-id, which is recorded in the branch to segment-id, found in its parent directory.

If conditions (i) - (ii) are met, then a branch is created in segment-id to the created segment, using a supplied name, say new-segment; the level of new-segment is set to sec-level. The process process-id receives an affirmative response.

Otherwise, process-id receives a negative response from the kernel.

kernel function 9: delete-object-group

Request has the elements:

(a) process-id
(b) segment-id

Process-id requests that segment-id be deleted (detached from the directory hierarchy). This results in deletion of all segments in the directory hierarchy which are inferior to segment-id.

The following condition is checked:

(i) same conditions as condition (i) of kf6.

If the condition is met, then the following recursive algorithm is invoked:

(i) set current-segment-id to segment-id.
(ii) if there are no branches in current-segment-id then do the following:

    (a) delete all SDWs which refer to current-segment-id.
    (b) delete current-segment-id from the hierarchy.
    (c) delete the branch of current-segment-id in its parent directory segment.
    (d) set current-segment-id to the segment-id of the parent of the segment just deleted.
    (e) if current-segment-id refers to the parent of segment-id (the original segment-id), then finished; else do action (ii).

otherwise, set current-segment-id to the segment-id given in any branch and do action (ii).

kernel-function 10: change-subject-current-security-level

Request has the elements:

(a) process-id
(b) sec-level

Process process-id requests that its current security level be changed to sec-level.

The following conditions are checked:

(i) process-id is listed in a table of trusted processes or for every SDW for a segment in the descriptor segment for process-id,

- if the r indicator is on, sec-level dominates the level of the segment, and
- if the w indicator is on, sec-level is dominated by the level of the segment.

(ii) the security level of process-id, given in the security level table, dominates sec-level.

If conditions (i) - (ii) are met, then the current security level of process-id in the current-security-level table, is changed to sec-level. The process process-id receives an affirmative response.

Otherwise, process-id receives a negative response from the kernel.

kernel-function 11: change-object-security-level

Request has the elements:

(a) revise-security-level
(b) process-id
(c) segment-id
(d) sec-level.

Process process-id requests that the security level of segment-id be revised to the value sec-level.

The following conditions are checked:

(i) process-id is a trusted process and the current security level of process-id, recorded in the current security level table, dominates the security level of segment-id, found in the branch to segment-id in segment-id's parent directory,

(ii) for every SDW for a process and segment-id that has the r indicator on, the current level of process in the current-security-level table dominates sec-level,

(iii) for every SDW for a process and segment-id that has the w indicator on, sec-level dominates the current level of process,

(iv) the security-level field of every branch in segment-id dominates sec-level and sec-level dominates the level of the parent of segment-id,

(v) process-id is allowed to change segment-id's security level.

If conditions (i) - (v) are met, then the security-level field of the branch to segment-id found in the parent directory of segment-id is changed to sec-level. The process process-id receives an affirmative response.

Otherwise, process-id receives a negative response from the kernel.

SECTION IV

FURTHER CONSIDERATIONS

INTRODUCTION

In this section we discuss topics that are related to the mathematical model only indirectly. The first of these is the concept of "trusted subjects": an attempt is made here to explicate the functional characteristics of trusted subjects and the formal justification required to make a subject "trusted." The other topics discussed are problems that might admit modeling in an extension of the current model but that have not been investigated in this way. These topics are "communication paths" (the indirect disclosure of sensitive information), "sabotage" (the deliberate alteration or destruction of sensitive information), and "integrity" (a property addressing approved modification of information).

The topics covered in this section become important in the certification and implementation phases of the development of a secure computer system. Moreover, resolutions of the problems have not been devised as yet. Hence, the discussion in this section will attempt to identify the issues, making use of specific examples in a Multics environment in the exposition. The discussion will of necessity not provide definitive answers: the intent is to formulate the questions.

TRUSTED SUBJECTS

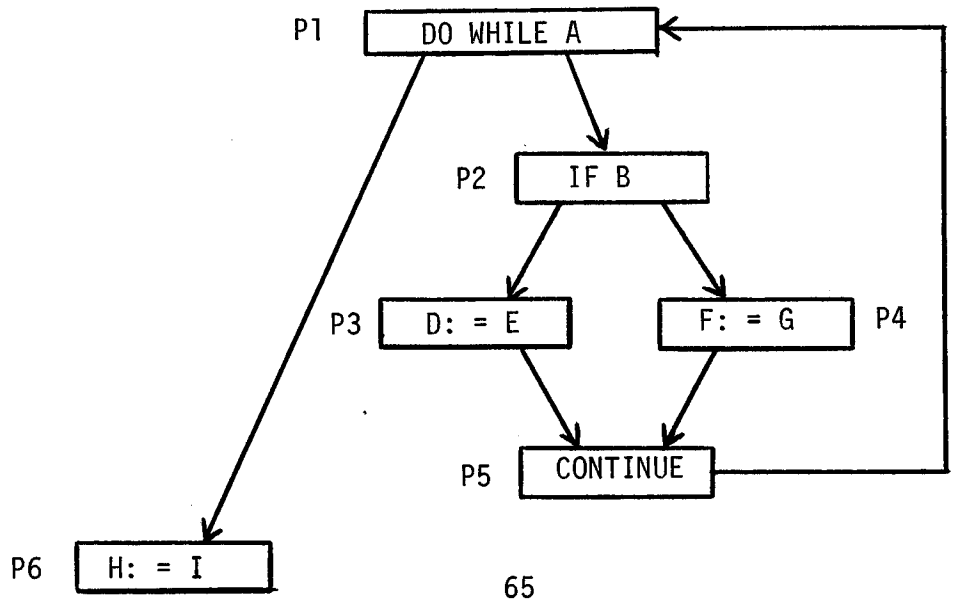
Within the model, trusted subjects are those subjects not constrained by the *-property. Outside the model, a subject, to be designated "trusted," must be shown not to consummate the undesirable transfer of high level information that *-property constraints prevent untrusted subjects from making. The demonstration that a process can be a "trusted" process is the concern of this discussion.

It is important to emphasize here that a "trusted subject" is only required not to copy high-level information into a low-level segment (object). It is also important to guarantee that the operation of a trusted subject (procedure) cannot be used as a medium of clandestine communication. That is, trusted subjects are not involved in communications paths, a topic we will discuss in a later section. The focus here is on "trustedness" — not copying information into inappropriate objects.

A sufficient (but not necessary) condition for declaring a process trusted is that the process is conceptually equivalent to a set of subprocedures each of which performs an operation constrained by the *-property and then chooses a successor. For example, the simple procedure:

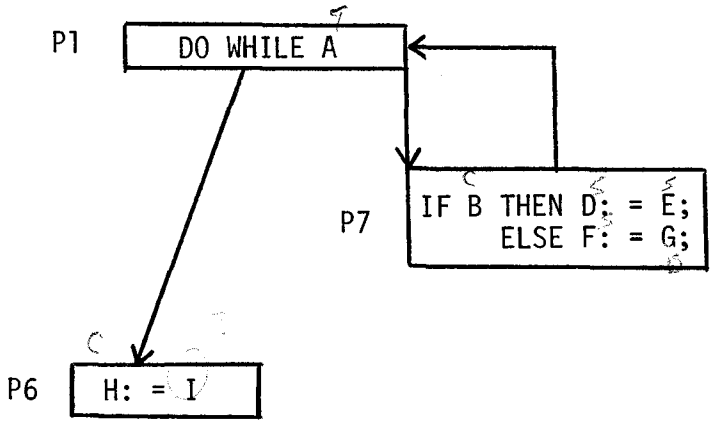
```
P:  DO WHILE A;
       IF B THEN D: = E;
            ELSE F: = G;
       END;
    H: = I;
    END;
```

is conceptually equivalent to the subprocedures P1, . . ., P6 defined and organized as shown:



If none of the subprocedures violates the *-property (using the minimal conceptual current access for each Pi), then P itself would not violate the *-property, even if, say, A were top secret and H were confidential.

Two remarks are in order. First, the division into subprocedures here is possibly overdone. If, for instance, D, E, and F are secret, B is confidential, and G is unclassified, then subprocedures P2, P3, P4 and P5 could be combined into a single subprocedure P7. P could then be represented as follows:



Since P7 does not violate the *-property, P could be shown not to violate *-property using this subdivision also. The merits of subdivision to instruction level vs. subdivision only as needed can be worked out to suit individual tastes; the result will be the same in either case.

The second point to be made about this type of demonstration is that the condition that the process be equivalent to a number of subprocedures obeying the *-property constraints is not necessary for the establishment of trusted processes. In particular, if and when a semantically correct "write-down" from a high-level file to a low-level file can be guaranteed, the process responsible could be

demonstrated to be trusted. The latter situation leads directly to the formulary concept, which is treated at some length elsewhere [20].

EXTRA-MODEL SECURITY PROPERTIES

Communication Paths

The first extra-model security property to be discussed is communications paths. By this term is meant the indirect disclosure of sensitive information, as opposed to the direct disclosure of information which is addressed by the security properties of the model. Indirect disclosure can be effected by transmitting data piecemeal using observable system characteristics as the code medium.

A large number of observable system characteristics can be used to transmit information, frequently a bit at a time. Possibly the most difficult medium to rule out as a communication path is real time: intervals of real time, delimited by prearranged observable events and varied by using the system, can be used to transmit information in bit strings (see Figure 14).

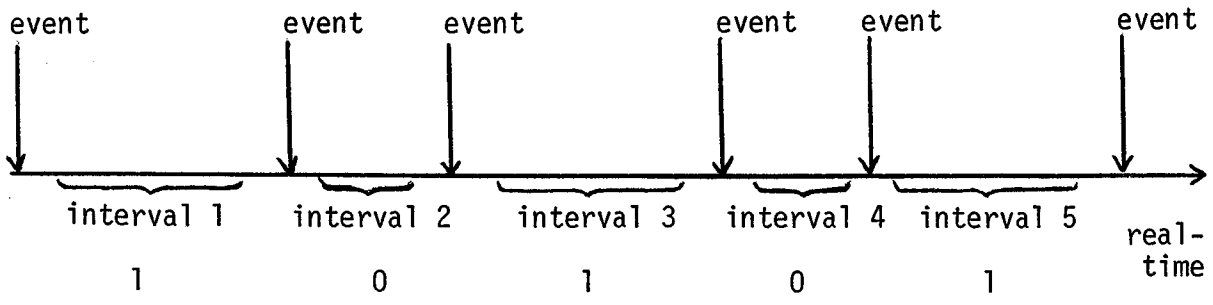


Figure 14. Communication Using Real-Time Intervals

Examples of system uses to vary real-time intervals are computing-to-IO ratios and paging rate. There is the possibility that synchronous paths cannot be entirely eliminated from any system that shares data. Examples of this type of communication can be found in B. W. Lampson's discussion of system-performance information channels [21] and Lipner's discussion of improvements (viz., lowering bandwidths of paths) [23].

Indirect communication using nonsynchronous paths remains a very complicated problem. Since a nonsynchronous path must make use of files, system variables, and the like to transmit a message, close and careful consideration of every possible action in a system will discover every nonsynchronous communication path. Within the model, however, there is no guidance for this enumerative exercise. In addition, the exercise itself can involve very subtle interactions of a number of objects.† Two examples will be presented to demonstrate the subtleties involved. Both examples involve the capability to create and destroy objects.

Suppose in the first instance that secret-process can create and destroy confidential segments whose existence can be detected by confidential-process (see Figure 15).

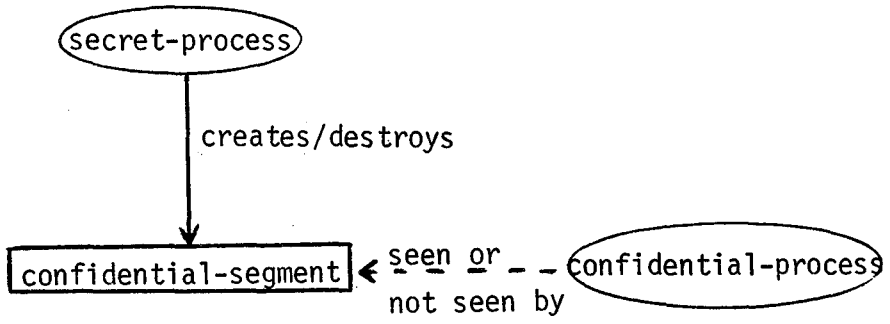


Figure 15. An Example of a One-Bit Message

†A description of a solution to this problem may be found in [22].

A string of such confidential segments could easily be used to transmit a bit string to a confidential process, by destroying those segments which correspond to zeroes in the bit string (Figure 16). This situation is clearly undesirable.

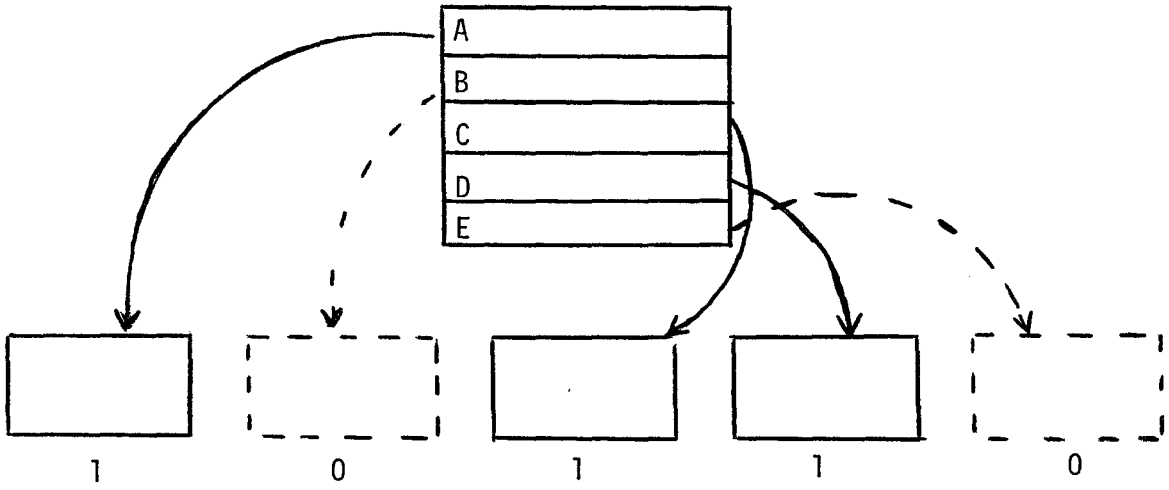


Figure 16. The Transmission of the Bit-String 10110

For the second example, suppose that confidential-process is denied a request to destroy a confidential directory if there is a secret segment inferior to it (see Figure 17).

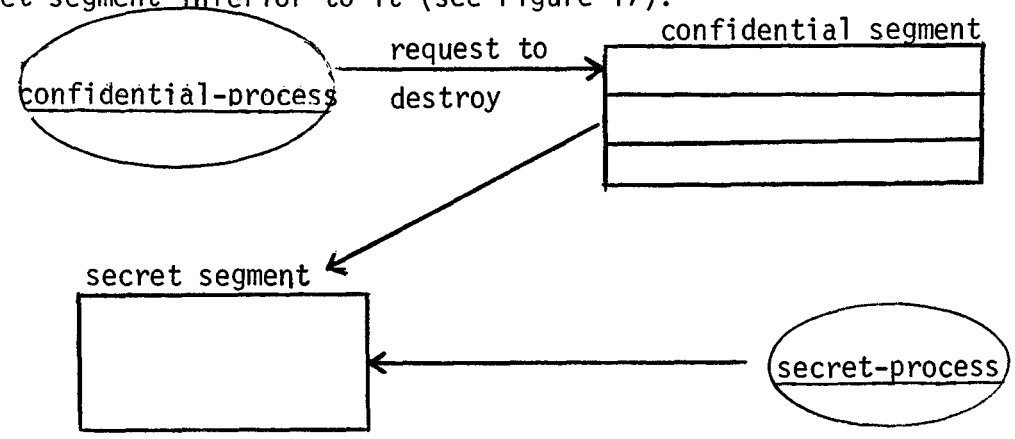


Figure 17. Another One-Bit Message

In this case, secret-process can alter the system's response to a request to destroy the confidential segment by creating or destroying a subordinate secret segment. This situation too is undesirable.

Neither of these situations is possible in the secure Multics design. The first example is disallowed by compatibility: to destroy a segment one must read/write the segment's parent which, by compatibility, has a level lower than or equal to that of the segment itself. The second example is disallowed because the destruction of objects specified by rule 9, delete-object-group, does not prohibit a confidential process from destroying a secret object inferior to the root object of the destroyed subtree. However, the care with which creation and destruction algorithms must be designed illustrates the complexities of enumerating the full list of objects which can be used in nonsynchronous communications paths.

Sabotage and Integrity

Sabotage, in this context, means undesired alteration or destruction of information by the purposeful action of an agent; integrity is a property determined by approved modification of information. To clarify the meanings of the two terms "sabotage" and "integrity" the intended meanings of the adjectives "undesired" and "approved" must be explicated. An alteration or destruction of information is undesirable if the intended and well-intentioned users of the system deem it so; a modification is approved if these same users consider the resulting semantic content of the modified information to be correct. Hence, in the context of information stored in a computer-based information system, sabotage and integrity are closely related.

An act of sabotage can have two principal effects: improper functioning of the system and incorrect semantic content. An integrity policy attempts to prevent acts of sabotage within the information system or to localize the effects to an acceptable degree.

Work on a model or integrity policy implementation is proceeding at MITRE [23]. A major problem is to specify an acceptable and appropriate policy to govern the modification of data segments. We consider here a simple model of integrity, leaving policy largely unspecified, in order to further the exposition of the problem.

Suppose that a set S of "integrity levels" is given: consider as an example the set:

nonsensitive < sensitive < critical < very critical

The semantics of these terms is suggestive; the integrity policy is, nevertheless, not specified by them since they are not formally defined. Suppose further that integrity level functions, analogous to security level functions, are defined:

I_S: {subjects} $\longrightarrow$ {integrity levels} and
I_O: {objects} $\longrightarrow$ {integrity levels}.

I_S(subject) denotes the maximum integrity level of an object that subject is allowed to modify; I_O(object) denotes the minimum level of any subject that is allowed to modify object.

Redefine a state v of the system by the inclusion of $I = (I_S, I_O)$:

$v = (b, M, f, I, H)$.

We can define a simple-integrity-property (si-property), analogous to the ss-property, as follows:

> a state satisfies the si-property provided for every current alter-access (subject, object, alter-attribute), the integrity level of subject (I_S(subject)) is greater than or equal to the integrity level of object (I_O(object)).

More formally, $v = (b, M, f, I, H)$ satisfies the si-property provided:

$$[(S_i, O_j, \underline{x}) \text{ in } b \text{ and } \underline{x} \text{ in } \{\underline{w}, \underline{a}\}] \text{ implies } I_S(S_i) \geq I_O(O_j).$$

There is an alternative formulation of the si-property, as there is for the ss-property:

> the state $v = (b, M, f, I, H)$ satisfies the si-property provided every $(S_i, O_j, \underline{x})$ in b satisfies the simple-integrity condition relative to I (SIC rel I); $(S_i, O_j, \underline{x})$ in b satisfies SIC rel I provided ($\underline{x} = \underline{w}$ or $\underline{x} = \underline{a}$) implies that $I_S(S_i) \geq I_O(O_j)$.

Given the above extension of the model, needed modifications to the rules of operation are obvious; moreover, intuition indicates that assuring the si-property systemically is inductive and can be accomplished by demonstrating si-property preservation over one state change (as is the case for secure state preservation). No analogue to the *-property exists, since the problem of information transfer within the realm of disclosure has no analogue in the

realm of sabotage. Finally, an inverse compatibility property for the hierarchy seems attractive; this would dictate that the integrity level of objects be monotone non-increasing on paths away from the root. This latter property relates to "localizing" damaging effects of sabotage action. Actual sabotage of sensitive-directory in Figure 18 indirectly sabotages inferior segments, which are necessarily nonsensitive or sensitive under inverse compatibility; the effect of sabotaging sensitive-directory by a sensitive process running amok would not extend to its parent, critical-directory, nor to unrelated segments such as critical-segment, sensitive-segment, and nonsensitive-segment.

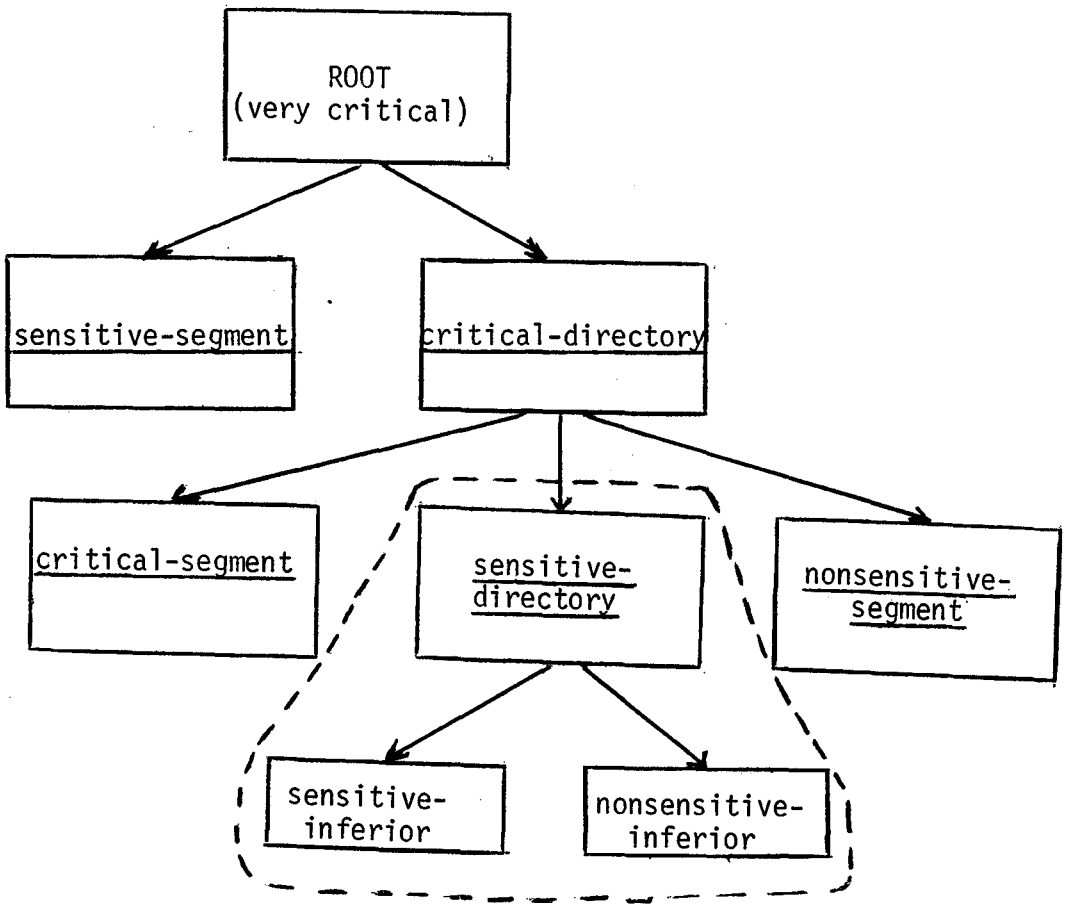


Figure 18. The Subtree Affected by Sabotage of Sensitive-Directory

APPENDIX

Introduction

The formal mathematical model is presented in this Appendix. No interpretation or explanation is offered, except as subsequently noted. The intended interpretations and correspondences to Multics architectural elements are given in the body of this report. In the section of this Appendix on rules, a narrative statement of each rule is given in order to reduce the reader's inconvenience in dealing with highly abstract symbology and in order to provide a natural language statement of intention by which errors or policy misdirections in the formal statements may be more easily discovered.

Elements

The elements of the mathematical model are presented in Table 1. Some items are not self-explanatory and they are explained here.

partial ordering relation $\propto$:

A relation R is a partial ordering relation if R is reflexive, antisymmetric, and transitive.

Suppose that U is a set and R is a binary relation defined on U, with elements of U denoted by small letters a, b, c, . . . etc.

reflexive: R is reflexive if xRx for each x in U.

antisymmetric: R is antisymmetric if [xRy and yRx] implies

$x = y$ (x is identically y) for each x and y in U. (In other words, we have xRy and yRx (symmetry) only in case $x = y$.)

<u>transitive</u>: R is transitive if [xRy and yRz] implies xRz for each x and y and z in U.

$L = \{L_1, L_2, \ldots, Lp\}$ where $L_i = (C_j, K)$ and C_j is in C and K is a subset of K. Define the relation ∞ on L as follows:

$$(L_i, L_j) \in \infty \equiv L_i \infty L_j \equiv (C_i, K) \infty (C_j, K') \text{ iff}$$

(i) $C_i \geq C_j$, and

(ii) $K \supseteq K'$.

Since both "$\geq$" and "$\supseteq$" are partial orderings, a straightforward argument shows that "∞" is also a partial ordering.

Suppose $C = \{S, C, U\}$, $S > C > U$, and $K = \{K_1, K_2, K_3\}$ and $L_1 = (S, \{K_1, K_2\})$, $L_2 = (S, \{K_1\})$, $L_3 = (C, \{K_1, K_2\})$, $L_4 = (C, \{K_1\})$, $L_5 = (S, \{K_2, K_3\}$, $L_6 = (C, \{K_2\})$, and $L_7 = (U, \{K_1\})$. The partial ordering of these elements of L is illustrated as a digraph in Figure A1.

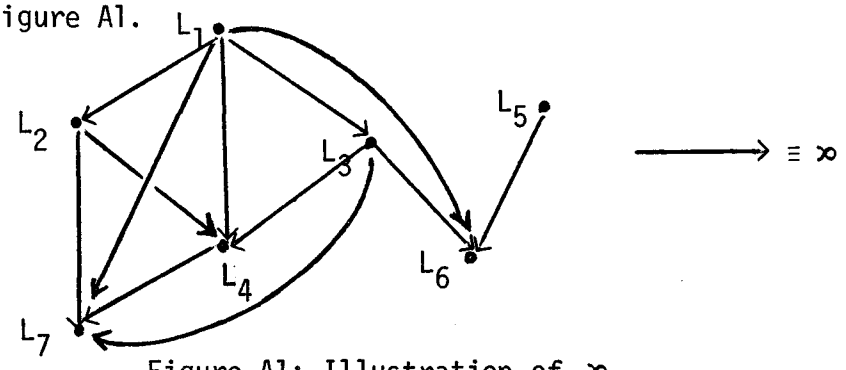


Figure A1: Illustration of ∞.

Table I

Elements of The Model

SET	ELEMENTS	SEMANTICS
S	$\{S_1, S_2, \ldots, S_n\}$	subjects: processes; programs in execution
O	$\{O_1, O_2, \ldots, O_m\}$	objects: data; files; programs; subjects; I/O devices
C	$\{C_1, C_2, \ldots, C_q\}$, $C_1 > C_2 > \ldots > C_q$	classifications: clearance level of a subject; classification of an object
K	$\{K_1, K_2, \ldots, K_r\}$	categories: special access privileges
L	$\{L_1, L_2, \ldots, L_p\}$ with partial ordering relation $\propto$; $L_i = (C_j, K)$, where C_j is in C and K is a subset of K	security levels:

Table I (Cont.)

SET	ELEMENTS	SEMANTICS
A	{r, e, w, a}	access attributes: r: read-only; e: execute (no read, no write); w: write (read and write); a: append (write-only)
RA	{g, r}	request elements: g: get, give r: release, rescind
S'	a subset of S	subjects subject to *-property:
S_T	$S - S'$	trusted subjects: subjects not subject to *-property but 'trusted' not to violate security with respect to it.

Table I (Cont.)

SET	ELEMENTS	SEMANTICS
R	$\bigcup_{1 \leq i \leq 5} R^{(i)}$, where $R^{(1)} = RA \times S \times O \times A$ $R^{(2)} = S \times RA \times S \times O \times A$ $R^{(3)} = RA \times S \times O \times L$ $R^{(4)} = S \times O$ $R^{(5)} = S \times L$	requests: $R^{(1)}$: requests for get- and release-access $R^{(2)}$: requests for give- and rescind-access $R^{(3)}$: requests for generation and reclassification of objects $R^{(4)}$: requests for destruction of objects $R^{(5)}$: requests for changing security level
D	{yes, no, error, ?}; an arbitrary element of D is written D_m	decisions:

Table I (Cont.)

SET	ELEMENTS	SEMANTICS
T	$\{1, 2, \ldots, t, \ldots\}$	indices: elements of a time set; identification of discrete moments; an element t is an index to request, decision, and state sequences
F	an element $f = (f_s, f_o, f_c)$ is in $F \subseteq L^S \times L^O \times L^S$ if and only if for each S_i in S $f_s(S_i) \propto f_c(S_i)$	security level vectors: f_s: subject security level function f_o: object security level function f_c: current security level 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:

Table I (Cont.)

SET	ELEMENTS	SEMANTICS
M	$\{M_1, M_2, \ldots, M_{nm2}4\}$; an element of M, say M_k, is an nxm matrix with entries from PA; the (i,j) - entry of M_k shows S_i's attributes relative to O_j; the entry is denoted by M_{ij}	access matrices: current access-permission structure; embodiment of discretionary security
H	an element H is in $H \subseteq (PO)^O$ if and only if: (1) $O_i \neq O_j$ implies $H(O_i) \cap H(O_j) = \phi$ (2) there does not exist a set $\{O_1, O_2, \ldots, O_w\}$ of objects such that O_{r+1} is in $H(O_r)$ for each r, $1 \leq r \leq w$, and $O_{w+1} \equiv O_1$	hierarchies: a hierarchy is a forest possibly with stumps, i.e., a hierarchy can be represented by a collection of rooted, directed trees and isolated points.
$\underline{tree}_H$	$[\cup H(O)] \cup H^{-1}(PO-\{\phi\})$	the "forest part of the hierarchy: if the hierarchy has a single tree, then $\underline{tree}_H$ can be represented by a single rooted, directed tree.

Table 1 (Concl.)

SET	ELEMENTS	SEMANTICS
grass	{system-wide variables} ∪ {non-forest I/O devices} ∪ {any other non-forest objects}	miscellanies:
A(H)	tree_H ∪ grass	the active objects:
B	$P(S \times O \times A)$; an arbitrary element of B is written b	current access set: record of current access of subjects to objects in various modes
V	B × $\mathcal{A}$ × F × H; 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:

Suppose $[U, R]$ is a partially ordered system. An element m in U is called a minimal element in U if mRx implies xRm for each x in U; if m is unique it is called a minimum. For $[L, \infty]$, as in the previous example, there are three minimal elements, (U, K_1), (U, K_2), and (U, K_3) and there is no minimum. If $K' = K \cup \{\phi\}$, then (U, ϕ) is a minimum in $[C \times K', \infty]$.

the notation A^B:

Suppose A and B are sets. The notation A^B denotes the set of all functions from B to A. Suppose $A = \{a, b\}$ and $B = \{1, 2\}$; then A^B consists of

$f_1 = \{(1, a), (2, b)\}$,
$f_2 = \{(1, b), (2, a)\}$,
$f_3 = \{(1, a), (2, a)\}$, and
$f_4 = \{(1, b), (2, b)\}$.

cartesian product:

Suppose A and B are sets. The cartesian product of A and B, denoted $A \times B$, is defined by

$$A \times B = \{(a, b): a \in A \text{ and } b \in B\},$$

i.e., $A \times B$ is the set of all ordered pairs of the form (a, b) where a is in A and b is in B. Suppose $A = \{a, b\}$ and $B = \{1, 2\}$. Then $A \times B = \{(a, 1), (a, 2), (b, 1), (b, 2)\}$. Notice that $B \times A = \{(1, a), (2, a), (1, b), (2, b)\} \neq A \times B$. Notice also that $f_1 \subset B \times A$, f_1 defined above.

the notation PX:

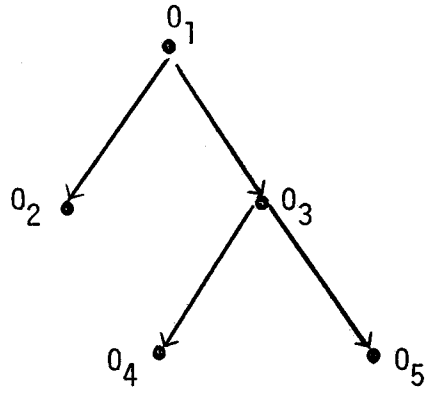
Suppose X is a set, say $X = \{a, b, c\}$. PX means the set of all subsets of X. In this case, $PX = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$ where ϕ denotes the empty set.

hierarchies:

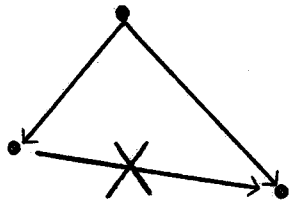
Suppose $H \subseteq (PO)^O$ where $O = \{O_1, O_2, O_3, O_4, O_5\}$. Restrict membership in H by the conditions (1) and (2) (see Table 1, entry for H). Define $H \in H$ as follows:

$$H = \{(O_1, \{O_2, O_3\}), (O_2, \phi), (O_3 \{O_4, O_5\}), (O_4, \phi), (O_5, \phi)\}.$$

H can be described also by a diagraph:



Condition (1) rules out a structure such as



and condition (2) rules out a structure such as

If an element of H imposes a forest structure on the objects with exactly one tree, as in the example, we identify the root of the tree by the notation O_R. If H is a tree structure then O_R is that object in O for which

$$H(O_R) \neq \phi \quad \text{and}$$
$$O_R \notin H(O) \quad \text{for any} \quad O \in O.$$

If O_j is an object in O then $O_{s(j)}$ denotes that object with respect to H such that $O_j \in H(O_{s(j)})$; in other words $O_{s(j)}$ is "superior" to O_j by H.

System

Suppose that $W \subset R \times D \times V \times V$. The system $\Sigma\ (R, D, W, z_o) \subset X \times Y \times Z$ is defined by

$(x, y, z) \in \Sigma(R, D, W, z_o)$ iff
$(x_t, y_t, z_t, z_{t-1}) \in W$ for each t in T,
where z_o is an initial state of the system, usually
of the form (ϕ, M, f, H).

Properties

We define properties in terms of the members of a state sequence. We then say that the system has a specified property if each state of

every state sequence of the system has the property. The following notation is defined.

$$b(S: \underline{x}, \underline{y}, \ldots, \underline{z}) = \{O: (S, O, \underline{x}) \in b \text{ or } (S, O, \underline{y}) \in b \text{ or } \ldots \text{ or } (S, O, \underline{z}) \in b\}$$

simple-security

A state $v = (b, M, f, H)$ satisfies the simple-security property (ss-property) iff

$$S \in S \Rightarrow [(O \in b\ (S: \underline{r}, \underline{w})) \Rightarrow (f_S(S) \propto f_O(O))].$$

It is convenient also to define:

$(S, O, \underline{x}) \in b$ satisfies the simple security condition relative to f (ssc rel f) iff

(i) $\underline{x} = \underline{e}$ or $\underline{a}$, or
(ii) $\underline{x} = \underline{r}$ or $\underline{w}$ and $f_S(S) \propto f_O(O)$.

Then it is easily shown that a state $v = (b, M, f, H)$ satisfies ss-property iff each $(S, O, \underline{x}) \in b$ satisfies SSC rel f.

*-property

Suppose S' is a subset of S. A state $v = (b, M, f, H)$ satisfies the *-property relative to S' iff

$$S \in S' \Rightarrow \begin{cases} (O \in b(S: \underline{a})) \Rightarrow (f_o(O) \propto f_c(S)) \\ (O \in b(S: \underline{w})) \Rightarrow (f_o(O) = f_c(S)) \\ (O \in b(S: \underline{r})) \Rightarrow (f_c(S) \propto f_o(O)). \end{cases}$$

An immediate consequence is: if v satisfies *-property rel S' and $S \in S'$ then

$$[O_j \in b(S: \underline{a}) \text{ and } O_k \in b(S: \underline{r})] \Rightarrow f_o(O_j) \propto f_o(O_k).$$

discretionary-security

A state $v = (b, M, f, H)$ satisfies the discretionary-security property (ds-property) iff

$$(S_i, O_j, \underline{x}) \in b \Rightarrow \underline{x} \in M_{ij}.$$

secure system

A state v is a secure state iff v satisfies the ss-property and *-property rel S' and ds-property. A state sequence z is a secure state sequence iff z is a secure state for each $t \in T$. Call $(x, y, z) \in \Sigma(R, D, W, z_0)$ an appearance of the system. $(x, y, z) \in \Sigma(R, D, W, z_0)$ is a secure appearance iff z is a secure sequence. Finally, $\Sigma(R, D, W, z_0)$ is a secure system iff every appearance of $\Sigma(R, D, W, z_0)$ is a secure appearance. Similar definitions pertain for the notions.

(i) the system $\Sigma(R, D, W, z_0)$ satisfies the ss-property,

(ii) the system satisfies *-property rel S', and

(iii) the system satisfies the ds-property.

Definition of Rule

A rule is a function $\rho: R \times V \rightarrow D \times V$. A rule therefore associates with each request-state pair (input) a decision-state pair (output).

A rule ρ is secure-state-preserving iff v^* is a secure state whenever $\rho(R_k, v) = (D_m, v^*)$ and v is a secure state. Similar definitions pertain for the notions

(i) ρ is ss-property-preserving,

(ii) ρ is *-property-preserving, and

(iii) ρ is ds-property-preserving.

Suppose $\omega = \{\rho_1, \rho_2, \ldots, \rho_s\}$ is a set of rules. The relation $W(\omega)$ is defined by

$(R_k, D_m, v^*, v) \in W(\omega)$ iff $D_m \neq \underline{?}$ and
$(D_m, v^*) = \rho_i (R_k, v)$ for a unique i, $1 \leq i \leq s$.

Theorems

$(R_i, D_j, v^*, v) \in R \times D \times V \times V$ is an action of $\Sigma(R, D, W, z_0)$ iff there is an appearance (x, y, z) of $\Sigma(R, D, W, z_0)$ and some $t \in T$ such that $(R_i, D_j, v^*, v) = (x_t, y_t, z_t, z_{t-1})$.

theorem A1:

$\Sigma(R, D, W, z_0)$ satisfies the ss-property for any initial state z_0 which satisfies ss-property iff W satisfies the following

conditions for each action $(R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H))$:

(i) each $(S, O, \underline{x}) \in b^*\text{-}b$ satisfies the simple security condition relative to f^* (SSC rel f^*);

(ii) each $(S, O, \underline{x}) \in b$ which does not satisfy SSC rel f^* is not in b^*.

argument:

($\Leftarrow$)

Suppose $z_o = (b, M, f, H)$ is an initial state which satisfies ss-property. Pick $(x, y, z) \in \Sigma(R, D, W, z_o)$ and write $z_t = (b^{(t)}, M^{(t)}, f^{(t)}, H^{(t)})$ for each $t \in T$.

z_1 satisfies ss-property

(x_1, y_1, z_1, z_o) is in W. In order to show that z_1 satisfies ss-property we need to show that each $(S, O, \underline{x})$ in $b^{(1)}$ satisfies SSC rel $f^{(1)}$.

Notice that $b^{(1)} = (b^{(1)} - b^{(0)}) \cup (b^{(0)} \cap b^{(1)})$ and $(b^{(1)} - b^{(0)}) \cap (b^{(1)} \cap b^{(0)}) = \varphi$. Suppose $(S, O, \underline{x})$ is in $b^{(1)}$. Then either $(S, O, \underline{x})$ is in $(b^{(1)} - b^{(0)})$ or is in $(b^{(1)} \cap b^{(0)})$. Suppose $(S, O, \underline{x})$ is in $(b^{(1)} - b^{(0)})$. Then $(S, O, \underline{x})$ satisfies SSC rel $f^{(1)}$ according to (i). Suppose $(S, O, \underline{x})$ is in $(b^{(0)} \cap b^{(1)})$. Then $(S, O, \underline{x})$ satisfies SSC rel $f^{(1)}$ according to (ii). Therefore z_1 satisfies ss-property.

<u>if z_{t-1} satisfies ss-property, then z_t satisfies ss-property</u>.

The argument given for "z_1 satisfies ss-property" applies with "t-1" substituted for "0" and "t" substituted for "1".

By induction, z satisfies ss-property so that the appearance (x, y, z) satisfies ss-property. (x, y, z) being arbitrary, $\Sigma(R, D, W, z_0)$ satisfies the ss-property.

(=>)

Suppose $\Sigma(R, D, W, z_0)$ satisfies the ss-property for any initial state z_0 which satisfies ss-property.

Argue by contradiction. Contradiction yields the proposition

"there is some action (x_t, y_t, z_t, z_{t-1}) such that either

(iii) some $(S, O, \underline{x})$ in $b^{(t)} - b^{(t-1)}$ does not satisfy SSC rel $f^{(t)}$ or

(iv) some $(S, O, \underline{x})$ in $b^{(t-1)}$ which does not satisfy SSC rel $f^{(t)}$ is in $b^{(t)}$, i.e., is in $b^{(t-1)} \cap b^{(t)}$."

Suppose (iii). Then there is some $(S, O, \underline{x})$ in $b^{(t)}$ which does not satisfy SSC rel $f^{(t)}$. Suppose (iv). Then there is some $(S, O, \underline{x})$ in $b^{(t)}$ which does not satisfy SSC rel $f^{(t)}$. Therefore z_t does not satisfy ss-property, (x, y, z) does not satisfy ss-property, and so $\Sigma(R, D, W, z_0)$ does not satisfy ss-property, which contradicts initial assumption of the argument.

The argument is complete.

theorem A2: $\Sigma(R, D, W, z_0)$ satisfies the *-property relative to $S' \subset S$ for any initial state z_0 which satisfies *-property relative to S' iff W satisfies the following conditions for each action $(R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H))$:

(i) for each $S \in S'$,

(a) $0 \in (b^* - b)(S:\underline{a}) \Rightarrow f_0^*(0) \propto f_c^*(S)$, and
(b) $0 \in (b^* - b)(S:\underline{w}) \Rightarrow f_0^*(0) = f_c^*(S)$, and
(c) $0 \in (b^* - b)(S:\underline{r}) \Rightarrow f_c^*(S) \propto f_0^*(0)$;

(ii) for each $S \in S'$,

(a') $[0 \in b(S:\underline{a})$ and $f_0^*(0) \not\propto f_c^*(S)] \Rightarrow$
$0 \notin b^*(S,\underline{a})$, and

(b') $[0 \in b(S:\underline{w})$ and $f_0^*(0) \neq f_c^*(S)] \Rightarrow$
$0 \notin b^*(S,\underline{w})$, and

(c') $[0 \in b(S:\underline{r})$ and $f_c^*(S) \not\propto f_0^*(0)] \Rightarrow$
$0 \notin b^*(S:\underline{r})$.

argument:

($\Leftarrow$)

Suppose $z_0 = (b, M, f, H)$ is an initial state which satisfies *-property rel S'. Pick (x, y, z) in $\Sigma(R, D, W, z_0)$ and write $z_t = (b^{(t)}, M^{(t)}, f^{(t)}, H^{(t)})$ for each $t \in T$.

z_1 satisfies *-property rel S'

(x_1, y_1, z_1, z_0) is in W. In order to show that z_1 satisfies *-property rel S' we need to show that:

$$\text{(iii) } S \in S' \Rightarrow \begin{cases} 0 \in b^{(1)}(S:\underline{a}) \Rightarrow f_o^{(1)}(0) \propto f_c^{(1)}(S) \\ 0 \in b^{(1)}(S:\underline{w}) \Rightarrow f_o^{(1)}(0) = f_c^{(1)}(S) \\ 0 \in b^{(1)}(S:\underline{r}) \Rightarrow f_c^{(1)}(S) \propto f_o^{(1)}(0). \end{cases}$$

Suppose $(S, 0, \underline{x}) \in b^{(1)}$, $S \in S'$, $\underline{x} \in \{\underline{a}, \underline{w}, \underline{r}\}$. Then either $(S, 0, \underline{x})$ is in $(b^{(1)} - b^{(0)})$ or $(S, 0, \underline{x})$ is in $(b^{(1)} \cap b^{(0)})$. Suppose $(S, 0, \underline{x})$ is in $(b^{(1)} - b^{(0)})$. Then (iii) is satisfied according to (i). Suppose $(S, 0, \underline{x})$ is in $b^{(1)} \cap b^{(0)}$. Then (iii) is satisfied according to (ii). Therefore z_1 satisfies *-property rel S'.

if z_{t-1} satisfies *-property rel S', then z_t satisfies *-property rel S'

The argument given for "z_1 satisfies *-property rel S'" applies with "t-1" substituted for "0" and "t" substituted for "1".

By induction, z satisfies *-property rel S' so that the appearance (x, y, z) satisfies *-property rel S'. (x, y, z) being arbitrary, $\Sigma(R, D, W, z_0)$ satisfies *-property relative to S'.

($\Rightarrow$)

Suppose $\Sigma(R, D, W, z_0)$ satisfies *-property relative to S' for any initial state z_0 which satisfies *-property rel S'.

Argue by contradiction. Contradiction yields the proposition

"there is some action (x_t, y_t, z_t, z_{t-1}) such that either

(iv) (i) is false or
(v) (ii) is false."

Suppose (iv). Then there is some $S \in S'$ such that (a) is false or (b) is false or (c) is false. Then z_t does not satisfy *-property rel S'. Suppose (v). Then there is some $S \in S'$ such that (a') is false or (b') is false or (c') is false. Then z_t does not satisfy *-property rel S'. This leads to "(x, y, z) does not satisfy *-property rel S' and so $\Sigma(R, D, W, z_0)$ does not satisfy *-property rel S'", which contradicts initial assumption of the argument.

The argument is complete.

<u>theorem A3</u>: $\Sigma(R, D, W, z_0)$ satisfies the ds-property iff z_0 satisfies the ds-property and W satisfies the following condition for each action $(R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H))$:

(i) $(S_a, O_{a'}, \underline{x}) \in b^* - b \Rightarrow \underline{x} \in M^*_{a,a'}$; and

(ii) $(S_a, O_{a'}, \underline{x}) \in b$ and $\underline{x} \notin M^*_{a,a'} \Rightarrow (S_a, O_{a'}, \underline{x}) \notin b^*$.

(<=)

If $(S_a, O_{a'}, \underline{x}) \in b^{(1)} - b^{(0)}$, $\underline{x} \in M_{a,}{}^{(1)}{}_{a'}$, by (i). Suppose $(S_a, O_{a'}, \underline{x}) \in b^{(1)} \cap b^{(0)}$. If $x \notin M_{a,}{}^{(1)}{}_{a'}$ then $(S_a, O_{a'}, \underline{x}) \notin b^{(1)}$, contrary to our supposition. Thus $x \in M_{a,}{}^{(1)}{}_{a'}$.

$(S_a, O_{a'}, \underline{x}) \in b^{(1)} = (b^{(1)} - b^{(0)}) \cup (b^{(1)} \cap b^{(0)})$, $x \in M^{(1)}_{a,a'}$ and z_1 satisfies the ds-property.

$(\Rightarrow)$

Suppose $\Sigma(R, D, W, z_0)$ satisfies the ds-property.

Argue by contradiction. Contradiction yields the proposition

"there is an initial state z_0 satisfying the ds-property and there is some action (x_t, y_t, z_t, z_{t-1}) such that there is some $(S_a, O_{a'}, \underline{x}) \in b^{(t)}$ such that $\underline{x} \notin M^{(t)}_{a, a'}$."

Therefore z_t does not satisfy ds-property, (x, y, z) does not satisfy ds-property, and so $\Sigma(R, D, W, z_0)$ does not satisfy ds-property, which contradicts the initial assumption of the argument.

The argument is complete.

corollary A1: $\Sigma(R, D, W, z_0)$ is a secure system iff z_0 is a secure state and W satisfies the conditions of theorems A1, A2, and A3 for each action.

theorem A4: Suppose ω is a set of ss-property-preserving rules and z_0 is an initial state which satisfies ss-property. Then $\Sigma(R, D, W(\omega), z_0)$ satisfies ss-property.

argument

Suppose $\Sigma(R, D, W(\omega), z_0)$ does not satisfy ss-property.

Then there is (x, y, z) in $\Sigma(R, D, W(\omega), z_0)$ which does not satisfy ss-property. Suppose t is the least element of T such that z_t does not satisfy ss-property. Since z_0 satisfies ss-property, $t > 0$. By choice of t, z_{t-1} satisfies ss-property and $z_{t-1} \neq z_t$. By definition of $\Sigma(R, D, W(\omega), z_0)$, $(x_t, y_t, z_t, z_{t-1}) \in W(\omega)$. By the definition of $W(\omega)$, there is some rule $\rho \in \omega$ such that $\rho(x_t, z_{t-1}) = (y_t, z_t)$. Since z_{t-1} satisfies ss-property and $\rho(x_t, z_{t-1}) = (y_t, z_t)$ and ρ is ss-property-preserving, z_t satisfies ss-property. The contradiction shows that $\Sigma(R, D, W(\omega), z_0)$ satisfies ss-property.

The argument is complete.

theorem A5: Suppose ω is a set of *-property preserving rules and z_0 is an initial state which satisfies *-property. Then $\Sigma(R, D, W(\omega), z_0)$ satisfies *-property.

argument: The argument is that of theorem A4 with the substitution of *-property for ss-property.

theorem A6: Suppose ω is a set of ds-property preserving rules and z_0 is an initial state which satisfies ds-property. Then $\Sigma(R, D, W(\omega), z_0)$ satisfies ds-property.

corollary A2: Suppose ω is a set of secure-state-preserving rules and z_0 is an initial state which is a secure state. Then $\Sigma(R, D, W(\omega), z_0)$ is a secure system.

theorem A7: Suppose $v = (b, M, f, H)$ is a state which satisfies ss-property, $(S, O, \underline{x}) \notin b$, $b^* = b \cup \{(S, O, \underline{x})\}$, and $v^* = (b^*, M, f, H)$. Then v^* satisfies ss-property iff

(i) ($\underline{x} = \underline{e}$ or $\underline{x} = \underline{a}$) or
(ii) ($\underline{x} = \underline{r}$ or $\underline{x} = \underline{w}$) and $f_s(S) \not\propto f_o(0)$.

argument

($\Rightarrow$)

Suppose $v^* = (b^*, M, f, H)$ satisfies ss-property. Then $0 \in b^*(S:\underline{r}, \underline{w}) \Rightarrow f_s(S) \not\propto f_o(0)$ by definition. Therefore (i) or (ii) holds since $\underline{x} \in \{\underline{e}, \underline{w}, \underline{r}, \underline{a}\}$.

($\Leftarrow$)

Suppose (i). Then v^* satisfies ss-property since v does.

Suppose (ii). Then for any $S \in \mathcal{S}$ we have $0 \in b^*(S:\underline{r}, \underline{w}) \Rightarrow f_s(S) \not\propto f_o(0)$ since v satisfies ss-property. Therefore v^* satisfies ss-property.

theorem A8: Suppose $v = (b, M, f, H)$ is a state which satisfies *-property rel $\mathcal{S}' \subset \mathcal{S}$, $S \in \mathcal{S}'$, $(S, 0, \underline{x}) \notin b$, $b^* = b \cup \{(S, 0, \underline{x})\}$, and $v^* = (b^*, M, f, H)$.

v^* satisfies *-property† iff

(i) if $\underline{x} = \underline{a}$, then $f_o(0) \not\propto f_c(S)$;
(ii) if $\underline{x} = \underline{w}$, then $f_c(S) = f_o(S)$; and
(iii) if $\underline{x} = \underline{r}$, then $f_c(S) \not\propto f_o(0)$.

---

† "rel $\mathcal{S}'$" is understood.

argument:

(⇒) Suppose v* satisfies *-property. The definition of *-property applied to S, O and (S, O, $\underline{x}$) yields conditions (i), (ii), and (iii) directly.

(⇐) Suppose conditions (i) - (iii) hold. Let $(S_i, O_j, \underline{y}) \in b^*$ with $S_i \in S'$. If $(S_i, O_j, \underline{y}) \in b$, the *-property conditions hold for f by the assumption that v satisfies *-property. If $(S_i, O_j, \underline{y}) \notin b$, $(S_i, O_j, \underline{y}) = (S, O, \underline{x})$ and the *-property conditions hold by the initial assumption of conditions (i) - (iii). Hence v* satisfies *-property as desired.

theorem A9: Suppose v = (b, M, f, H) is a state which satisfies ds-property, $(S_i, O_j, \underline{x}) \notin b$, $b^* = b \cup \{(S_i, O_j, \underline{x})\}$, and $v^* = (b^*, M, f, H)$. Then v* satisfies ds-property iff $\underline{x} \in M_{ij}$.

argument:

(⇒) Suppose v* satisfies ds-property. Then $\underline{x} \in M_{ij}$ by definition.

(⇒) Suppose $\underline{x} \in M_{ij}$. Then, since $(S_i, O_j, \underline{x}) \in b^*$, the proposition $((S_i, O_j, \underline{x}) \in b^* \Rightarrow \underline{x} \in M_{ij})$ is true; therefore, v* satisfies ds-property.

corollary A3: Suppose v = (b, M, f, H) is a secure state, $(S_i, O_j, \underline{x}) \notin b$, $b^* = b \cup \{(S_i, O_j, \underline{x})\}$, and $v^* = (b^*, M, f, H)$. Then v* is a secure state iff

(i) $S_i \in S_T$ and the conditions of theorems A7 and A9 are met, or

(ii) $S_i \in S'$ and the conditions of theorems A7, A8, and A9 are met.

theorem A10: Let ρ be a rule and $\rho(R_{k,\ v}) = (D_m,\ v^*)$, where $v = (b, M, f, H)$ and $v^* = (b^*, M^*, f^*, H^*)$.

(i) If $b^* \subseteq b$ and $f^* = f$, then ρ is ss-property-preserving.

(ii) If $b^* \subseteq b$ and $f^* = f$, then ρ is *-property-preserving.

(iii) If $b^* \subseteq b$ and $M_{ij}^* \supseteq M_{ij}$ for all i and j, then ρ is ds-property-preserving.

(iv) If $b^* \subseteq b$, $f^* = f$, and $M_{ij}^* \supseteq M_{ij}$ for all i and j, then ρ is secure-state-preserving.

argument:

(i) If v satisfies the ss-property, then $(S, O, \underline{x}) \in b^*$ with $\underline{x} = \underline{w}$ or $\underline{r}$ implies $(S, O, \underline{x}) \in b$ so that $f_s(S) \propto f_o(O)$ by assumption. Hence $f_s^*(S) \propto f_o^*(O)$ since $f^* = f$. Thus v^* satisfies ss-property and ρ is ss-property-preserving.

(ii) and (iii) are proved in ways exactly analogous to the proof of (i). Implications (i), (ii), and (iii) prove implication (iv).

Rules

notation

The symbol "$\setminus$" will be used in expressions of the form "A$\setminus$B" to mean "proposition A except as modified by proposition B". Some examples follow. Suppose f is a function from the set {A, B, C} to the set {0, 1, 3} defined by:

$f(A) = 1$ or $(A, 1) \in f$,
$f(B) = 0$ or $(B, 0) \in f$,
$f(C) = 3$ or $(C, 3) \in f$.

Then $f \setminus (C, 1)$ or $f \setminus f(C) = 1$ means

$f(A) = 1$,
$f(B) = 0$,
$f(C) = 1$.

Suppose M is a matrix. Then $M \setminus M_{ij} \leftarrow a$ means the matrix obtained from M by replacing the $(i, j)^{th}$ element by a. $M \setminus M_{ij} \cup \{\underline{x}\}$ means the matrix obtained from M by adding the element $\underline{x}$ to the $(i, j)^{th}$ set entry. Similarly, the notation $f \setminus f_o \leftarrow f_o \cup (O_{NEW(H)}, L_u)$ [see Rule 8] means the function obtained from f by replacing f_o by f_o plus the ordered pair $(O_{NEW(H)}, L_u)$ $[f_o (O_{NEW(H)}) = L_u]$. The notation NEW(H) denotes a selection function with respect to the hierarchy H which specifies an arbitrary inactive object index.

definitions of rules

The definitions of Rules 1 to 11 are given in the following

pages. These rules preserve compatibility and assume the presence of trusted subjects.

Rule 1 (R1): get-read

Domain of R1: all $R_k = (g, S_i, O_j, \underline{r})$ in $R^{(1)}$. (Denote domain of R_i by dom (R_i).)

Semantics: Subject S_i requests access to object O_j in read-only mode ($\underline{r}$).

*-property function: $*1\ (R_k, v) = \text{TRUE} \Leftrightarrow f_c(S_i) \propto f_o(O_j)$.

The rule:

$$R1(R_k,v) = \begin{cases} (\underline{?}, v) & \text{if } R_k \notin \text{dom } (R1); \\ (\underline{yes}, (b \cup (S_i, O_j, \underline{r})^{\dagger}, M, f, H)) & \text{if } [R_k \in \text{dom } (R1)] \ \&\ [\underline{r} \in M_{ij}] \ \&\ [f_s(S_i) \propto f_o(O_j)] \ \&\ [S_i \in S_T \text{ or } *1(R_k,v)]; \\ (\underline{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R1:

if $R_k \notin$ dom (R1) then $R1(R_k, v) = (\underline{?}, v)$; else if $\underline{r} \in M_{ij}$ and $\langle [S_i \in S' \text{ and } *1(R_k, v)] \text{ or } [S_i \in S_T \text{ and } f_s(S_i) \propto f_o(O_j)] \rangle$
then $R1(R_k, v) = (\underline{yes}, (b \cup (S_i, O_j, \underline{r}), M, f, H))$;
else $R1(R_k, v) = (\underline{no}, v)$;

end;

†more precisely $b \cup \{(S_i, O_j, \underline{r})\}$; braces are left out for legibility and compactness.

Rule 2 (R2): get-append

Domain of R2: all $R_k = (g, S_i, O_j, \underline{a}) \in R^{(1)}$.

Semantics: Subject S_i requests access to object O_j in append ($\underline{a}$) mode.

*-property function: $*2(R_k, v) = \text{TRUE} \Leftrightarrow f_o(O_j) \succ f_c(S_i)$.

The rule:

$$R2(R_k, v) = \begin{cases} (\underline{?}, v) & \text{if } R_k \notin \text{dom (R2)}; \\ (\underline{yes}, (b \cup (S_i, O_j, \underline{a}), M, f, H)) & \text{if } [R_k \in \text{dom (R2)}] \;\&\; [\underline{a} \in M_{ij}] \;\&\; [S_i \in S_T \text{ or } *2(R_k, v)]; \\ (\underline{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R2:

if $R_k \notin$ dom (R2) then $R2(R_k, v) = (\underline{?}, v)$; else if $\underline{a} \in M_{ij}$ and $\langle [S_i \in S' \text{ and } *2(R_k, v)] \text{ or } [S_i \in S_T] \rangle$
then $R2(R_k, v) = (\underline{yes}, (b \cup (S_i, O_j, \underline{a}), M, f, H))$;
else $R2(R_k, v) = (\underline{no}, v)$;

end;

Rule 3 (R3): get-execute

Domain of R3: all $R_k = (g, S_i, O_j, \underline{e}) \in R^{(1)}$.

Semantics: Subject S_i requests access to object O_j in execute ($\underline{e}$) mode.

*-property function: $*3\ (R_k, v) = \text{TRUE}$.

The rule:

$$R3(R_k,v) = \begin{cases} (\underline{?}\ , v) & \text{if } R_k \notin \text{dom } (R3); \\ (\underline{yes}, (b \cup (S_i,O_j,\underline{e}),M,f,H)) & \text{if } [R_k \in \text{dom } (R3)] \ \&\ [\underline{e} \in M_{ij}]; \\ (\underline{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R3:

if $R_k \notin$ dom (R3) then $R3(R_k, v) = (\underline{?}, v)$; else if $\underline{e} \in M_{ij}$ then $R3\ (R_k, v) = (\underline{yes}, (b \cup (S_i,O_j,\underline{e}),M,f,H))$;
else $R3(R_k, v) = (\underline{no}, v)$;

end;

Rule 4 (R4): get-write

Domain of R4: all $R_k = (g, S_i, O_j, \underline{w}) \in R^{(1)}$.

Semantics: Subject S_i requests access to object O_j in write ($\underline{w}$) mode.

*-property function: $*4(R_k, v) = \text{TRUE} \Leftrightarrow f_c(S_i) = f_o(O_j)$.

The rule:

$$R4(R_k, v) = \begin{cases} (\underline{?}, v) & \text{if } R_k \notin \text{dom } (R4); \\ (\underline{yes}, (b \cup (S_i, O_j, \underline{w}), M, f, H)) & \text{if } [R_k \in \text{dom } (R4)] \;\&\; [\underline{w} \in M_{ij}] \;\&\; [f_s(S_i) \rtimes f_o(O_j)] \;\&\; [S_i \in S_T \text{ or } *4(R_k, v)]; \\ \text{otherwise.} \end{cases}$$

Algorithm for R4:

if $R_k \notin$ dom (R4) then $R4(R_k, v) = (\underline{?}, v)$; else if $\underline{w} \in M_{ij}$ and $\langle [S_i \in S_T$ and $f_s(S_i) \rtimes f_o(O_j O]$ or $[S_i \in S'$ and $*4(R_k, v)]\rangle$
then $R4(R_k, v) = (\underline{yes}, (b \cup (S_i, O_j, \underline{w}), M, f, H))$;
else $R4(R_k, v) = (\underline{no}, v)$;

end;

Rule 5 (R5): release-read/execute/write/append

Domain of R5: all $R_k = (r, S_i, O_j, \underline{x}) \in R^{(1)}$, $\underline{x} \in A$.

Semantics: Subject S_i signals the release of access to object O_j in mode $\underline{x}$, where $\underline{x}$ is $\underline{r}$ (read-only), $\underline{e}$ (execute), $\underline{w}$ (write), or $\underline{a}$ (append).

*-property function: $*5(R_k, v) = TRUE$.

The rule:

$$R5(R_k, v) = \begin{cases} (\underline{yes}, (b - (S_i, O_j, \underline{x}), M, f, H)) & \text{if } R_k \in \text{dom (R5)}; \\ (\underline{?}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R5:

if $R_k \notin$ dom (R5) then $R5(R_k, v) = (\underline{?}, v)$;
else $R5(R_k, v) = (\underline{yes}, (b - (S_i, O_j, \underline{x}), M, f, H))$;

end;

Rule 6: give-read/execute/write/append

Domain of R6: all $R_k = (S_\lambda, g, S_i, O_j, \underline{x}) \in R^{(2)}$, $\underline{x} \in A$.

Semantics: Subject S_λ gives subject S_i access permission to O_j in mode $\underline{x}$, where $\underline{x}$ is $\underline{r}$, $\underline{e}$, $\underline{w}$, or $\underline{a}$.

*-property function: *6 (R_k, v) = TRUE.

The rule:

$$R6(R_k, v) = \begin{cases} (\underline{?}, v) & \text{if } R_k \notin \text{dom (R6)}; \\ (\underline{yes}, (b, M \setminus M_{ij} \cup \{\underline{x}\}, f, H)) & \text{if } [R_i \in \text{dom (R6)}] \ \& \\ & [<[O_j \neq O_R] \ \& \ [O_{s(j)} \neq O_R] \ \& \ [O_{s(j)} \in b(S_\lambda: \underline{w})]> \ \text{or} \\ & <[O_{s(j)} = O_R] \ \& \ [\text{GIVE } (S_\lambda, O_j, v)]> \ \text{or} \\ & <[O_j = O_R] \ \& \ [\text{GIVE } (S_\lambda, O_R, v)]>]; \\ (\underline{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R6:

if $R_k \notin$ dom (R6) then $R6(R_k, v) = (\underline{?}, v)$;
else if $[<[O_j \neq O_R]$ and $[O_{s(j)} \neq O_R]$ and $[O_{s(j)} \in b(S: \underline{w})]>$ or $<[O_{s(j)} = O_R] \ \& \ [\text{GIVE}(S_\lambda, O_j, v)]>$
or $<[O_j = O_R]$ and [GIVE $(S_\lambda, O_R, v)]>]$
then $R6(R_k, v) = (\underline{yes}, (b, M \setminus M_{ij} \cup \{\underline{x}\}, f, H))$;
else $R6(R_k, v) = (\underline{no}, v)$;

end;

GIVE (S_λ, O_k, v) = TRUE iff S_λ is allowed to give access permission to O_k in state v, for $O_k = O_R$ or $O_{s(k)} = O_R$.

Rule 7 (R7): rescind-read/execute/write/append

Domain of R7: all $R_k = (S_\lambda, r, S_i, O_j, \underline{x}) \in R^{(2)}$, $\underline{x} \in A$.

Semantics: Subject S_λ rescinds subject S_i's access permission to O_j in mode $\underline{x}$, where $\underline{x}$ is $\underline{r}$, $\underline{e}$, $\underline{w}$, or $\underline{a}$.

*-property function: $*7(R_k, v) = \text{TRUE}$.

The rule:

$$R7(R_k, v) = \begin{cases} (\underline{?}, v) & \text{if } R_k \notin \text{dom } (R7); \\ (\underline{yes}, (b - (S_i, O_j, \underline{x}), M \setminus M_{ij} - \{\underline{x}\}, f, H)) & \text{if } [R_k \in \text{dom } (R7)] \ \& \\ & [<[O_j \neq O_R] \ \& \ [O_{s(j)} \in b(S_\lambda: \underline{w})]> \ \text{or} \\ & \ <[O_j = O_R] \ \& \ [\text{RESCIND } (S_\lambda, O_R, v)]>]; \\ (\underline{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R7:

if $R_k \notin$ dom (R7) then $R7(R_k, v) = (\underline{?}, v)$; else if $[<[O_j \neq O_R]$ and $[O_{s(j)} \in b(S_\lambda: \underline{w})]>$ or
$<[O_j = O_R]$ and [RESCIND $(S_\lambda, O_R, v)]>]$
then $R7(R_k, v) = (\underline{yes}, (b - (S_i, O_j, \underline{x}), M \setminus M_{ij} - \{\underline{x}\}, f, H))$;
else $R7(R_k, v) = (\underline{no}, v)$;

end;

RESCIND (S_λ, O_R, v) = TRUE iff S_λ is allowed to rescind access permission to O_R in state v.

Rule 8 (R8): create-object (preserving compatibility)

Domain of R8: all $R_k = (g, S_i, O_j, L_u) \in R^{(3)}$.

Semantics: Subject S_i "generates" an object. S_i requests the "creation" (i.e., attachment) of an object, denoted $O_{NEW(H)}$, having security level L_u, directly below O_j in the hierarchy H (i.e., $O_{NEW(H)} \in H(O_j)$).

*-property function: $*8\ (R_k, v)$ = TRUE.

The rule:

$$R8(R_k, v) = \begin{cases} (?, v) & \text{if } R_k \notin \text{dom (R8)}; \\ (\text{yes}, (b, M, f \setminus f_o \leftarrow f_o \cup (O_{NEW(H)}, L_u), H \cup (O_j, O_{NEW(H)}))) & \text{if } [R_k \in \text{dom (R8)}] \,\&\, [O_j \in b(S_i: w, a)] \,\&\, [L_u \propto f_o(O_j)]; \\ (\text{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R8:

if $R_k \notin$ dom (R8) then (?, v); else if $\langle [O_j \in b(S_i: w, a)]$ and $[L_u \propto f_o(O_j)] \rangle$ then
$R8(R_k, v) = (\text{yes}, (b, M, f \setminus f_o \leftarrow f_o \cup (O_{NEW(H)}, L_u), H \cup (O_j, O_{NEW(H)})))$;
else $R8(R_k, v) = (\text{no}, v)$;

end;

Rule 9 (R9): delete-object-group

Domain of R9: all $R_k = (S_i, O_j) \in R^{(4)}$

Semantics: Subject S_i requests that object O_j be deleted (i.e., detached from the hierarchy). This results in deletion of O_j and all objects inferior to O_j in the hierarchy.

*-property function: $*9\ (R_k, v) = \text{TRUE}$.

The rule:

$$R9(R_k, v) = \begin{cases} (?, v) & \text{if } R_k \notin \text{dom } (R9); \\ (\text{yes}, (b - \text{ACCESS}(O_j), M \setminus M_{uw} \leftarrow \phi: 1 \le u \le n, O_w \in \text{INFERIOR}(O_j), f, H - \text{SUBTREE}(O_j)) & \\ \qquad \text{if } [R_k \in \text{dom } (R9)] \,\&\, [O_j \neq O_k] \,\&\, [O_{s(j)} \in b(S_i: w)]; & \\ (\text{no}, v) & \text{otherwise.} \end{cases}$$

Algorithm for R9:

if $R_k \notin$ dom (R9) then $R9(R_k, v) = (?, v)$;
else if $[O_j \neq O_R]$ and $[O_{s(j)} \in b(S_i: w)]$ then
$R9(R_k, v) = (\text{yes}, b - \text{ACCESS}(O_j), M \setminus M_{uw} \leftarrow \phi: 1 \le u \le n, O_w \in \text{INFERIOR}(O_j), f, H\text{-SUBTREE}(O_j))$;
else $R9(R_k, v) = (\text{no}, v)$;

end;

$\text{INFERIOR}(O_j) = \{O_k: [O_k = O_j]$ or [there is a set of objects $\{O_1, O_2, \ldots, O_h\}$ such that $O_k \in H(O_1)$, $O_1 \in H(O_2), \ldots, O_k \in H(O_j)\}$.
$\text{SUBTREE}(O_j) = \{(O_{s(k)}, O_k): O_k \in \text{INFERIOR}(O_j)\}$.
$\text{ACCESS}(O_j) = (S \times \text{INFERIOR}(O_j) \times A) \cap b$.

Rule 10 (R10): change-subject-current-security-level

Domain of R10: all $R_k = (S_i, L_u) \in R^{(5)}$.

Semantics: Subject S_i requests a change in its current security (value of $f_c(S_i)$) to L_u.

*-property function: $*10(R_k, v) = \text{TRUE} \Leftrightarrow [O_j \in b\ (S_i: a) \Rightarrow f_o(O_j) \asymp L_u]\ \&$
$[O_j \in b\ (S_i: w) \Rightarrow L_u = f_o(O_j)]\ \&$
$[O_j \in b\ (S_i: r) \Rightarrow L_u \asymp f_o(O_j)]\ \&$

The rule:

$$R10(R_k, v) = \begin{cases} (?, v) & \text{if } R_k \notin \text{dom } (R10); \\ (yes, (b, M, f \diagdown f_c(S_i) \leftarrow L_u, H)) & \text{if } [R_k \in \text{dom } (R10)\ \&\ [f_s(S_i) \asymp L_u] \\ & \quad \&\ [S_i \in S_T \text{ or } *10(R_k, v)]; \\ (no, v) & \text{otherwise.} \end{cases}$$

Algorithm for R10:

if $R_k \notin$ dom (R10) then R10 $(R_k, v) = (?, v)$; else if $[S_i \in S_T$ or $*10(R_k, v)]$ and $[f_s(S_i) \asymp L_u]$ then
$R10(R_k, v) = (yes, (b, M, f \diagdown f_c(S_i) \leftarrow L_u, H))$;
else $R10(R_k, v) = (no, v)$;

end;

Rule 11 (R11): change-object-security-level

Domain of R11: all $R_k = (r, S_i, O_j, L_u) \in R^{(3)}$.

Semantics: Subject S_i requests that the security level of object O_j be changed (reclassified) to L_u.

*-property function: $*11(R_k, v) = TRUE \Leftrightarrow$ for each $S_\lambda \in S'$,

$$[(S_\lambda, O_j, \underline{a}) \in b \Rightarrow L_u \propto f_c(S_\lambda)] \;\&$$
$$[(S_\lambda, O_j, \underline{w}) \in b \Rightarrow f_c(S_\lambda) = L_u] \;\&$$
$$[(S_\lambda, O_j, \underline{r}) \in b \Rightarrow f_c(S_\lambda) \propto L_u].$$

The rule:

$$R11(R_k, v) = \begin{cases} (\underline{?}, v) & \text{if } R_k \notin \text{dom } (R11); \\ (\underline{yes}, (b, M, f \setminus f_o(O_j) \leftarrow L_u, H)) & \text{if } [R_k \in \text{dom } (R11)] \;\&\; [\langle S_i \in S_T \;\&\; f_c(S_i) \propto f_o(O_j)\rangle \text{ or } \langle f_c(S_i) \propto L_u \propto f_o(O_j)\rangle] \\ & \;\&\; [\text{for each } S \in S\; [(O_j \in b(S: \underline{r}, \underline{w})) \Rightarrow (f_s(S) \propto L_u)]] \\ & \;\&\; [*11(R_k, v)] \;\&\; [\text{COMPAT } (v, O_j, L_u)] \;\&\; [\text{CHANGE } (v, O_j, L_u)]; \\ (\underline{no}, v) & \text{otherwise.} \end{cases}$$

where $\text{COMPAT } (v, O_j, L_u) = TRUE \Leftrightarrow [L_u \propto f_o(O_{s(j)})$ and $f_o(O_w) \propto L_u$ for each $O_w \in H(O_j)]$, and

$\text{CHANGE}(v, O_j, L_u) = TRUE \Leftrightarrow [S_i$ is allowed to change O_j's security level in state $v]^\dagger$

†CHANGE is included in order to allow for additional policy enforcement for a particular system.

Algorithm for R11:

<u>if</u> $R_k \notin$ dom (R11) <u>then</u> $R11(R_k, v) = (\underline{?}, v)$;

<u>else if</u> $[S_i \in S_T$ <u>or</u> $f_c(S_i) \asymp L_u \asymp f_o(O_j)]$ <u>and</u> $[f_c(S_i) \asymp f_o(O_j)]$ <u>and</u>
[for each $S \in S\ [O_j \in b(S: \underline{r}, \underline{w})) \Rightarrow (f_s(S) \asymp L_u)]]$ <u>and</u> $[*11(R_k, v)]$ <u>and</u>
$[COMPAT\ (v, O_j, L_u)]$ <u>and</u> $[CHANGE\ (v, O_j, L_u)]$ <u>then</u>
$R11(R_k, v) = (\underline{yes}, (b, M, f \diagdown f_o(O_j) \leftarrow L_u, H))$;
<u>else</u> $R11(R_k, v) = (\underline{no}, v)$;

<u>end</u>;

descriptions of rules

rule 1: get-read

Request is of the form (g, S_i, O_j, r).

Subject S_i requests access to object O_j in read-only mode (r).

If request is not of the proper form, then response is ? with no state change.

Otherwise, the following conditions are checked:

(i) S_i has current access permission to O_j in read-only mode.

(ii) the security level of S_i dominates the security level of O_j.

(iii) S_i is a trusted subject or the current security level of S_i dominates the security level of O_j.

If conditions (i) - (iii) are met, then the response is yes and the state changes by adding an entry in the current access list indicating that S_i has read-only access to O_j.

Otherwise the response is no with no state change.

rule 2: get-append

Request is of the form $(g, S_i, O_j, \underline{a})$.

Subject S_i requests access to object O_j in append mode ($\underline{a}$).

If request is not of the proper form, then response is ? with no state change.

Otherwise the following conditions are checked:

(i) S_i has current access permission to O_j in append mode.

(ii) S_i is a trusted subject or the security level of O_j dominates the current security level of S_i.

If conditions (i) - (ii) are met, then the response is yes and the state changes by adding an entry to the current access list indicating that S_i has append access to O_j.

Otherwise the response is no with no state change.

rule 3: get-execute

Request is of the form $(g, S_i, O_j, \underline{e})$.

Subject S_i requests access to object O_j in execute mode ($\underline{e}$).

If request is not of the proper form, then the response is ? with no state change.

Otherwise the following condition is checked:

(i) S_i has current access permission to O_j in execute mode.

If condition (i) is met, then the response is yes and the state changes by adding an entry to the current access list indicating that S_i has execute access to O_j.

Otherwise the response is no with no state change.

rule 4: get-write

Request is of the form (g, S_i, O_j, w).

Subject S_i requests access to object O_j in write mode (w).

If request is not of the proper form, then the response is ? with no state change.

Otherwise the following conditions are checked:

(i) S_i has current access permission to O_j in write mode.

(ii) the security level of S_i dominates the security level of O_j.

(iii) S_i is a trusted subject or the current security level of S_i equals the security level of O_j.

If conditions (i) - (iii) are met, then the response is yes and the state changes by adding an entry to the current access list indicating that S_i has write access to O_j.

Otherwise the response is no with no state change.

rule 5: release-read/execute/write/append

Request is of the form (r, S_i, O_j, x).

Subject S_i signals the release of access to object O_j in access mode x.

If request is not of the proper form, then the response is ? with no state change.

Otherwise the response is yes and the state changes by removing an entry from the current access list indicating that S_i no longer has access to O_j in mode x.

rule 6: give-read/execute/write/append

Request is of the form $(S_\lambda, g, S_i, O_j, x)$,

Subject S_λ gives to subject S_i access permission to O_j in mode x.

If request is not of the proper form, then response is ? with no state change.

Otherwise the following condition is checked:

(i) object O_j is not the root object of the hierarchy and subject S_λ has current access in write mode to O_j's immediately superior object ($O_{s(j)}$) in the hierarchy

or

O_j is the root object and S_λ is allowed to give access permission to the root object in the current state.

If condition (i) is met, then the response is yes and the state is changed by adding access permission for S_i to O_j in mode x to the access permission matrix.

Otherwise the response is no with no state change.

rule 7: rescind-read/execute/write/append

Request is of the form $(S_\lambda, r, S_i, O_j, x)$.

Subject S_λ rescinds subject S_i's access permission to O_j in mode x.

If request is not of the proper form, then response is ? with no state change.

Otherwise the following condition is checked:

(i) object O_j is not the root object of the hierarchy and subject S_λ has current access in write mode to O_j's immediately superior object ($O_{s(j)}$) in the hierarchy,

or

O_j is not the root object and S_λ is allowed to rescind access permission to the root object in the current state.

If condition (i) is met, then response is yes and the state changes as follows:

(i) removal of an entry from the current access list indicating that S_i no longer has access to O_j in mode x.

(ii) removal of access permission for S_i to O_j in mode x from the access permission matrix.

Otherwise the response is no with no state change.

rule 8: create-object

Request is of the form (g, S_i, O_j, L_u).

Subject S_i generates an object. S_i requests creation (i.e., attachment) of an object, denoted $O_{NEW(H)}$, having security level L_u, directly below object O_j in the hierarchy H ($O_{NEW(H)} \varepsilon H(O_j)$).

If request is not of the proper form, then response is ? with no state change.

Otherwise the following conditions are checked:

(i) S_i has current access to O_j in write or append mode.

(ii) the security level L_u dominates the security level of O_j.

If conditions (i) - (ii) are met, then response is yes and the state changes as follows:

(i) the security level function is updated by adding the ordered pair ($O_{NEW(H)}$, L_u) (i.e., the security level of $O_{NEW(H)}$ is recorded as L_u).

(ii) the object $O_{NEW(H)}$ is added to the hierarchy such that $O_{NEW(H)}$ is directly below O_j($O_{NEW(H)} \varepsilon H(O_j)$).

Otherwise response is no with no state change.

rule 9: delete-object-group

Request is of the form (S_i, O_j).

Subject S_i requests that object O_j be deleted (detached from the hierarchy). This results in deletion of all objects in the hierarchy which are inferion to O_j.

If request is not of the proper form, then response is ? with no state change.

Otherwise the following condition is checked:

(i) S_i has current write access to the object immediately superior to O_j ($O_{s(j)}$) and O_j is not the root object.

If condition (i) is met, then response is yes and the state changes as follows:

(i) all entries in the current access list giving subjects access to O_j or any object inferior to O_j in any mode are removed from the current access list.

(ii) all entries in the access permission matrix giving subjects access permission to O_j or any object inferior to O_j in any mode are removed from the access permission matrix.

(iii) O_j and all objects inferior to O_j are removed from the hierarchy.

Otherwise response is no with no state change.

rule 10: change-subject-current-security-level

Request is of the form (S_i, L_u).

Subject S_i requests that its current security level be changed to L_u.

If request is not of the proper form, then response is ? with no state change.

Otherwise the following conditions are checked:

(i) S_i is a trusted subject or if S_i's security level were changed to L_u, then the resulting state would satisfy *-property.

(ii) the security level of S_i dominates L_u.

If conditions (i) - (ii) are met, then response is yes and the state changes by changing the current security level of S_i to L_u.

Otherwise response is no with no state change.

rule 11: change-object-security-level

Request is of the form (r, S_i, O_j, L_u).

Subject S_i requests that the security level of object O_j be changed to L_u.

If request is not of the proper form, then response is ? with no state change.

Otherwise the following conditions are checked:

(i) S_i is a trusted subject and the current security level of S_i dominates the security level of O_j

or

the current security level of S_i dominates L_u and L_u dominates the security level of O_j.

(ii) if any subject S has current access to O_j in read or write mode, then the current security level of S dominates L_u.

(iii) if O_j's security level were changed to L_u, then the resulting state would satisfy *-property.

(iv) if O_j's security level were changed to L_u, then compatibility would be preserved in the hierarchy.

(v) S_i is allowed to change O_j's security level.

If conditions (i) - (v) are met, then response is yes and the state changes by changing the security level of O_j to L_u.

Otherwise response is no with no state change.

proofs

rule 1

Suppose v satisfies ss-property, *-property rel S', and

ds-property and $R_k \in R$. $R1(R_k, v) = (D_m, v^*)$ with:

(i) $v^* = v$ or

(ii) $v^* = (b \cup (S_i, O_j, \underline{r}), M, f, H)$

If (i), then v^* satisfies ss-property, *-property, and ds-property since v does.

Suppose (ii). If $(S_i, O_j, \underline{r}) \in b$, then $v^* = v$. Suppose $(S_i, O_j, \underline{r}) \notin b$. Then, since $f_s(S_i) \propto f_o(O_j)$ according to R1, v^* satisfies ss-property by theorem A7 and, since $f_c(S_i) \propto f_o(O_j)$ if $S_i \in S'$ according to R1, v^* satisfies *-property rel S' by theorem A8 and, since $\underline{r} \in M_{ij}$ according to R1, v^* satisfies ds-property by theorem A9.

Therefore R1 is secure-state-preserving by corollary A3.

rule 2

Suppose v satisfies ss-property, *-property rel S', and ds-property and $R_k \in R$. $R2(R_k, v) = (D_m, v^*)$ with

(i) $v^* = v$ or

(ii) $v^* = (b \cup (S_i, O_j, \underline{a}), M, f, H)$

Suppose (ii). If $(S_i, O_j, \underline{a}) \in b$, then $v^* = v$. Suppose $(S_i, O_j, \underline{a}) \notin b$. Then v^* satisfies ss-property by theorem A7 and, since $f_o(O_j) \propto f_c(S_i)$ if $S_i \in S'$ according to R2, v^* satisfies *-property rel S' by theorem A8 and, since $\underline{a} \in M_{ij}$

according to R2, v* satisfies ds-property by theorem A9.

Therefore R2 is secure-state-preserving by corollary A3.

rule 3

Suppose v is a secure state and $R_k \in R$.

Suppose $v^* = (b \cup (S_i, O_j, \underline{e}), M, f, H)$ and $(S_i, O_j, \underline{a}) \notin b$. Then v* satisfies ss-property by theorem A7 and v* satisfies *-property rel S' by theorem A8 and, since $\underline{e} \in M_{ij}$ according to R3, v* satisfies ds-property by theorem A9.

Therefore R3 is secure-state-preserving by corollary A3.

rule 4

Suppose v is a secure state and $R_k \in R$.

Suppose $v^* = (b \cup (S_i, O_j, \underline{w}), M, f, H)$ and $(S_i, O_j, \underline{w}) \notin b$. Then, since $f_s(S_i) \propto f_o(O_j)$ according to R4, v* satisfies ss-property by theorem A7 and, since $f_c(S_i) = f_o(O_j)$ if $S_i \in S'$, v* satisfies *-property rel S' by theorem A8 and, since $\underline{w} \in M_{ij}$ according to R4, v* satisfies ds-property by theorem A9.

Therefore R4 is secure-state-preserving by corollary A3.

rule 5

Suppose v is a secure state.

According to R5 $b^* \subseteq b$, $M^* = M$, and $f^* = f$. Therefore v^* is a secure state and R5 is secure-state-preserving by theorem A10 (iv).

rule 6

Suppose v is a secure state.

According to R6 $b^* = b$ and $M^* = M \cup \{\underline{x}\}$. Therefore v^* is a secure state and R6 is secure-state-preserving by theorem A10 (iv).

rule 7

Suppose v is a secure state.

According to R7 $v^* = v$ or $v^* = (b - (S_i, O_j, \underline{x}), M \setminus M_{ij} - \{\underline{x}\}, f, H)$. If the latter then it is still the case that $(S_a, O_b, \underline{x}) \in b \Rightarrow \underline{x} \in M_{ab}$. R7 is ss-property-preserving and *-property-preserving by theorem A10 (i) and (iv). Therefore v^* is a secure state and R7 is secure-state-preserving.

rule 8

Suppose v is a secure state.

According to R8 $b^* = b$ and $M^* = M$. Since $(S_\lambda, O_{NEW(H)}, \underline{x}) \notin b$ for any S_λ in S and $\underline{x}$ in A, v^* is a secure state and R8 is secure-state-preserving.

rule 9

Suppose v is a secure state.

According to R9 if $(S_a, O_{a'}, \underline{x}) \in b^*$, then $\underline{x} \in M_{aa'}$ so v* is a secure state. Therefore R9 is secure-state-preserving.

rule 10

Suppose v is a secure state.

According to R10 if $f^* \neq f$ then $f^* = f \diagdown f_c(S_i) \leftarrow L_u$ and *10 (R_k, v) is true so v* is a secure state. Therefore R10 is secure-state-preserving.

rule 11

Suppose v is a secure state.

According to R11 if $f^* \neq f$ then $f^* = f \diagdown f_o(O_j) \leftarrow L_u$ and *11 (R_k, v) is true so v* is a secure state. Therefore R11 is secure-state-preserving.

REFERENCES

1. D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations," ESD-TR-73-278, Vol. I, AD 770 768, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, November 1973.

2. Leonard J. La Padula and D. Elliott Bell, "Secure Computer Systems: A Mathematical Model," ESD-TR-73-278, Vol. II, AD 771 543, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, November 1973.

3. D. Elliott Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," ESD-TR-73-278, Vol. III, AD 780 528, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, April 1974.

4. Elliott I. Organick, The Multics Systems, The MIT Press, Cambridge, Massachusetts, 1972.

5. Clark Weissman, "Security Controls in the ADEPT-50 Time-Sharing System," AFIPS Conf. Proc. 35, FJCC 1969, 119-133.

6. B.W. Lampson, "Dynamic protection structures," AFIPS Conf. Proc. 35, FJCC 1969, 27-38.

7. James P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Vol. I, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, October 1972.

8. D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations and Model," M74-244, The MITRE Corporation, Bedford, Massachusetts, October 1974.

9. K.G. Walter et al., "Primitive Models for Computer Security," ESD-TR-74-117, Electronic Systems Division (MCIT), Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, January 1974.

10. W. Lee Schiller, "Design of a Security Kernel for the PDP-11/45," ESD-TR-73-294, AD 772 808, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, December 1973.

11. Leroy A. Smith, "Architectures for Secure Computing Systems," ESD-TR-75-51, AD A009 221, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, April 1975.

12. Steven B. Lipner, "A Minicomputer Security Control System," MTP-151, The MITRE Corporation, Bedford, Massachusetts, February 1974.

13. Roger R. Schell, Peter J. Downey, and Gerald J. Popek, "Preliminary Notes on the Design of Secure Military Computer Systems," MCI-73-1, Electronic Systems Division, Hanscom AFB, Bedford, Massachusetts, January 1973.

14. R. Bisby, II and Gerald J. Popek, "Encapsulation: An Approach to Operating System Security," USC/Information Sciences Institute, Marina del Ray, California, October 1973.

15. D.K. Hsiao, E.J. Kerr, and E.J. McCauley, III, "A Model for Data Secure Systems (Part I)," Computer & Information Science Research Center, OSU-CICRC-TR-73-8, Ohio State University, February 1974.

16. Gerald J. Popek and Charles S. Kline, "Verifiable Protection Systems," Proceedings, 1975 International Conference on Reliable Software, Los Angeles, April 20-23, 1975.

17. Peter G. Neumann et al., "On the Design of a Provably Secure Operating System," presented at the International Workshop on Protection in Operating Systems, IRIA, August 1974.

18. Leonard J. La Padula and D. Elliott Bell, "Harmonious Cooperation of Processes Operating on a Common Set of Data, ESD-TR-72-147, Vol. III, AD 757 904, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, December 1972.

19. D. Elliott Bell and Edmund L. Burke, "A Software Validation Technique for Certification: The Methodology," ESD-TR-75-54, AD A009 849, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, April 1975.

20. Daniel F. Stork, "Downgrading in a Secure Multilevel Computer System: The Formulary Concept," ESD-TR-75-62, AD A011 696, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, May 1975.

21. B.W. Lampson, "A Note on the Confinement Problem," Communications ACM 16 (1973), 613-615.

22. Jonathan K. Millen, "Security Kernel Validation in Practice," ESD-TR-75-54, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, Massachusetts, June 1975.

23. Steven B. Lipner, "A Comment on the Confinement Problem," MTP-167, The MITRE Corporation, Bedford, Massachusetts, November, 1975.