BA

BEDFORD OPERATIONS

MTR-3153

Integrity Considerations for Secure Computer Systems

K. J. Biba

30 JUNE 1975

MITRE

MITRE Technical Report

MTR-3153

Integrity Considerations for Secure Computer Systems

K. J. Biba

30 JUNE 1975

CONTRACT SPONSOR	ESD
CONTRACT NO.	F19628-75-C-0001
PROJECT NO.	522B
DEPT.	D-73

18865

THE MITRE CORPORATION

BEDFORD, MASSACHUSETTS

Department Approval: Charles R. Sheehan

MITRE Project Approval: Edward J. Bloustein

ABSTRACT

An integrity policy defines formal access constraints which, if effectively enforced, protect data from improper modification. We identify the integrity problems posed by a secure military computer utility. Integrity policies addressing these problems are developed and their effectiveness evaluated. A prototype secure computer utility, Multics, is then used as a testbed for the application of the developed access controls.

ACKNOWLEDGEMENTS

The author would like to thank S. Ames, E. Burke, S. Lipner, W. Price, and R. Schell for helpful comments, criticisms, and suggestions.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

OVERVIEW

Insuring legitimate access to privileged information has become a major area of concern for information processing technology. The rapidly growing use of complex resource sharing information systems[1] has emphasized the need to carefully identify and guarantee who has which access to what data. Experience has indicated that the protection issue is two-pronged: concerned both with the proper dissemination of information and with that information's validity. Our concern, in this paper, is an examination of how information validity may be maintained.

Our context is the Secure General Purpose Computer Project of the Air Force's Electronic Systems Division [1]. Its purpose is the design, construction, and formal validation of a secure computer utility for the military environment. The term secure computer utility refers to an interactive, multiprogrammed, multiprocessor computer system supporting resource sharing in a manner determined by an information protection policy. Its effective enforcement of the protection policy must be formally validated before it can be certified, by the appropriate authority, for use with classified information.

The protection policy of particular interest in a military environment [2] [3] takes the form of the DoD security regulations. To date, security has been interpreted to address the authorized observation (dissemination) of classified information. We intend to extend the certified function of the utility to address the proper modification of information within the computer system.

---

[1] These systems are realized in many forms including: network packet-switches, time-shared mainframes, microcomputer arrays, and dedicated data base systems.

BACKGROUND

Before beginning a discussion of information validity, a review of fundamental notions upon which this paper (and the project) is built is appropriate. These are:

1) the reference monitor;

2) a formalized security policy; and

3) the concept of a kernel.

Those readers familiar with these concepts may skip ahead to Section II where we begin our analysis of an integrity policy.

The Reference Monitor

An investigation [4] into the protection problems of military computer systems proposed the abstract notion of a reference monitor as a generic solution.

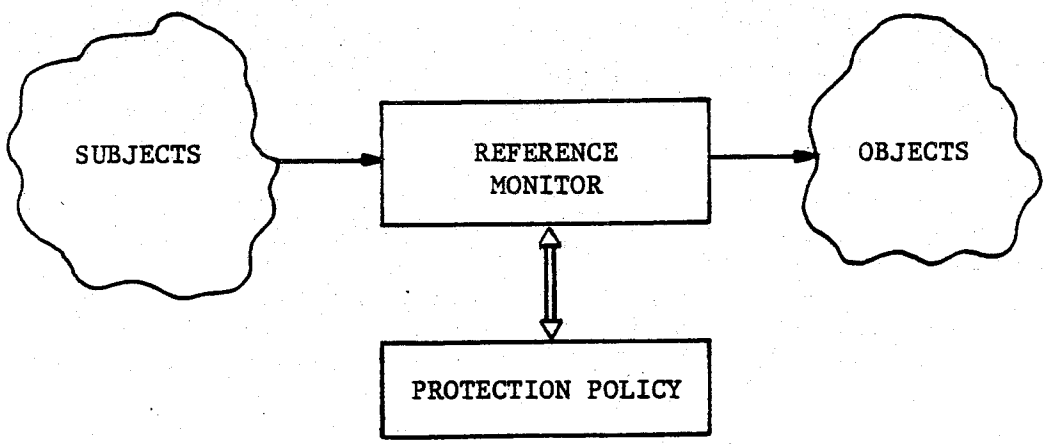

Figure 1. A Reference Monitor

A reference monitor is an entity that monitors and decides the allowability of all accesses by information processors (subjects) to information repositories (objects). The protection policy, enforced by the reference monitor, is the data base in which the allowability decision is encoded.

A reference monitor must satisfy three logical properties:

1) it is complete: all accesses by subjects to objects are monitored and enforced;

2) it is protected: its function may not be maliciously or accidently modified by unauthorized forces; and

3) it has provably proper behavior: it must faithfully enforce the specified protection policy.

We will see that the second property is concerned with the maintenance of the validity of the reference monitor.

The protection policy has historically [2] [3] [5] been represented as a set of axioms constraining access by subjects to objects. A generic policy model will illustrate the concept. We define a set S of subjects, a set O of objects, a relation $a \subset S \times O$ denoting that a subject s may access an object o, and a function domain: S -> POWERSET(O). The axiom A1.1 defines the function of the reference monitor.

(A1.1) $\forall s \in S, o \in O \quad s\ a\ o \Rightarrow o \in domain(s)$.

The function domain is an encoding of a protection policy identifying the accessible name space of its subject argument. A1.1 specifies that a subject s may access an object o only if o is in the domain of s. A protection policy takes the form of a decision algorithm defining the name spaces (or partitions of name spaces) assigned to subjects. The dotted lines of Figure 2 represent the access domains of the two subjects S1 and S2. The arrows have the following interpretation: arrows pointing to an object represent a modification of an object by a subject; arrows pointing to a subject from an object represent an observation of the object by the subject.

Access Modes

The above example presented a rather abstract form of access, independent of the semantics associated with an access. In general, access domains are partitioned on the basis of the mode (semantics) of the access. In this paper we will be concerned with three abstract modes of access: observation, modification, and invocation. Access permission, for each of these modes, will be explicitly identified.

Observation, as the name implies, relates to the viewing of information by a subject. Central to observation is the testing of information. We state that observation is the testing of information which results in a choice of distinct states of the observing subject (and possibly distinct outputs).

Modification may be defined in terms of observation. A subject modifies information if its value is changed so that an observation, by a subject (possibly distinct from the modifier), results in a different state than previous observations (a discernable change).

Invocation is a logical request for service from one subject to another. Since the control state of the invoked subject is a function of the fact that the subject was invoked, invocation is a special case of modification. Invocation is an abstraction of the primary control construct for transferring control between distinct subjects in (possibly) differing access domains. In general we require that the invoking subject not be informed of the success or failure of the operation. Such information may be passed by subsequent invocations, the sequencing determined by some control protocol. Interprocess communication is one instance of an invocation mechanism where the wakeup signal passed from the invoking process to the invoked process constitutes the invocation. A subroutine type of intersubject control structure may be accomplished by two invocations: the invoker first signals the invoked subject and then waits for the invoked subject to reactivate it (return) via a subsequent invocation. A necessary consequence of the subroutine type of control structure is that the invoking and invoked subjects must each have invocation privilege to the other.

It should be noted that "execute" access is quite different than invocation. While invocation represents a control access between distinct subjects in (possibly) differing domains, execution is the access, by a subject, to an object for the purpose of obtaining instructions. Since during execution a subject obtains its instructions by observing the object in which the instructions reside, we will consider execute access to be equivalent to observe access for our purposes.

From these elementary constructs, we may compose complex subsystems accessing many objects (subjects) with combinations of primitive access modes.

Security Policy

The protection policies investigated, to date, have addressed the problem of information security. Security denotes the property of protection against compromise: unauthorized dissemination of information. The security policy defines access domains of subjects based on considerations derived from DoD security attributes of subjects and objects. Several axiomatic systems [2] [3] represent this policy. We present a model defining this policy below.

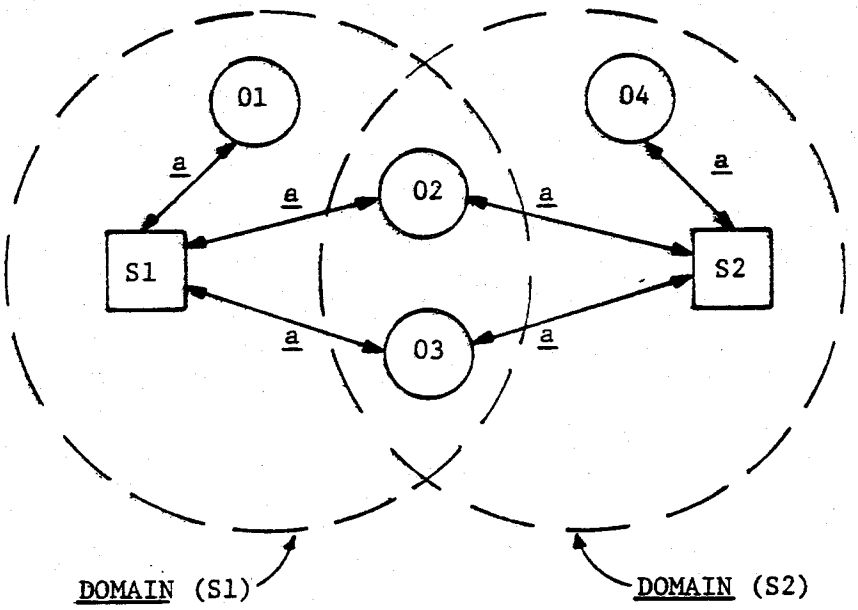

Figure 2. Access Domains

First, we define the elements of the model.

S: a set of subjects;

O: a set of objects where the intersection of S and O is null;

SL: a partially ordered set of security levels that forms a lattice;

sl: S U O -> SL, a function mapping subjects and objects into security levels;

$\leq$: a subset of SL x SL defining the partial ordering "less than or equal";

o: a subset of S x O defining the access capability for observation; and

m: a subset of S x O defining the access capability for modification.

The access domain of each subject is partitioned with respect to the mode of access. The following axioms[2] define the access domain.

(A1.2) $\forall s \in S, o \in O \quad s \; o \; o \Rightarrow sl(o) \leq sl(s)$.

(A1.3) $\forall s \in S, o \in O \quad s \; m \; o \Rightarrow sl(s) \leq sl(o)$.

The access domain for each subject s is thus the set:

$$\left| o \in O \mid sl(o) \leq sl(s) \quad or \quad sl(s) \leq sl(o) \right|.$$

A subject may observe the information contained in an object if its security level is greater than or equal to that of the object. A subject may modify the information contained in an object if its security level is less than or equal to that of the object. These constraints insure [2] [3] that information may be transferred only "upward" in security level, even by subjects untrusted to behave properly.

We may also extend the above axioms to intersubject invocation by the relation i: subset of S x S defining the access capability for intersubject invocation.

---

[2] Formal statements are labelled according to the following convention: a section specific number prefaced by "A" for axioms, "T" for theorems, and "D" for definitions.

(A1.4) ∀ s[1],s[2] ∈ S s[1] $\underline{i}$ s[2] => $\underline{sl}$(s[1]) $\underline{\leq}$ $\underline{sl}$(s[2]).

The Kernel Concept

The realization of a reference monitor within a computer system is termed a kernel. Conceptually, the kernel is a central, localized hardware/software system that enforces the protection policy of the reference monitor it implements [1]. The kernel defines an abstract machine composed of logical objects and operations[3] access to which is determined by the protection policy. A crucial part of the kernel development process is the formal verification of the property: the protection policy is enforced for all designated accesses.

The first kernel in the ESD program was constructed for the PDP-11/45 [6]. Its successful implementation encouraged the design and implementation of a large-scale prototype [7], based on the Honeywell Information Systems' Multics [8]. The kernel design for this system [9] incorporates the protection considerations described in this paper.

OUTLINE

We will address the following issues:

1) identification of relevant integrity problems;
2) definition of protection policies that address these problems; and
3) identification of the computer system elements (subsystems) to which these protection policies should be applied.

Our analysis of integrity policies begin in Section II with a discussion of integrity problems within a military computer utility. Section III formally proposes several integrity policies designed to cope with the problems posed in Section II. Section IV considers the application of these policies within a prototype computer utility.

[3] Implemented either in software, firmware, or hardware.

A caution to the reader. This investigation occurs in the context of a Multics' kernel development. Therefore many of the examples (and particularly the jargon) are taken from this milieu. While the issues (and conclusions) generalize, some familiarity with Multics [8] is assumed.

SECTION II

THE INTEGRITY PROBLEM

INTEGRITY DEFINED

What do we mean by integrity in computer systems? In this subsection we will investigate our intended meaning of integrity and establish the context for the integrity formulations of the succeeding sections.

Webster's dictionary provides an initial integrity definition:

> integrity - 1a: an unimpaired or unmarred condition: entire correspondence with an original condition: SOUNDNESS
> 1b: an uncompromising adherence to a code of moral, artistic or other values.

This definition, and our informal notions of integrity point to a similar conception: integrity does not imply guarantees concerning the absolute behavior of systems. For instance, a person thought to have the property of integrity is only considered to behave consistently with respect to some standard: no statement (or decision) about the quality of the standard is implied.

This concept can be applied to computer systems. We consider a subsystem to possess the property of integrity if it can be trusted to adhere to a well-defined code of behavior. No *a priori* statement as to the properties of this behavior are relevant.

The concern of computer system integrity is thus the guarantee that a subsystem will perform as it was intended to perform by its creator. We assume that a subsystem has been initially certified (by some system external agency) to perform properly. We then wish to insure that the subsystem cannot be corrupted to perform in a manner contrary to its certification. The integrity problem is the formulation of access control policies and mechanisms that provide a subsystem with the isolation necessary for protection from subversion. Based on an initial assumption of proper behavior (according to some system external standard), we are primarily concerned with protection from intentionally malicious attack: unprivileged, intentionally malicious modification.

INTEGRITY THREATS

How can integrity be compromised? That is, how can a system be improperly persuaded (forced) to change its behavior? The following paragraphs briefly classify integrity threats. The abstract form of an integrity threat is a subsystem modification not considered in the subsystem's initial verification of proper behavior. We term such improper modifications sabotage.

Our viewpoint is that of the subsystem: some subset of a system's subjects and objects isolated on the basis of function or privilege. We will consider two dimensions of integrity threats to a subsystem: source and type. Integrity threat sources identify where a threat might originate while threat type identifies the manner in which the threat might be made.

Threat Sources

We consider two threat sources:

1) subsystem external; and

2) subsystem internal.

Their names unambiguously describe their origin. An external threat is posed by one subsystem attempting to change (improperly) the behavior of another by the supplying of false data, improperly invoking functions, or direct modification of its own behavior. Improperly performed (or not considered in the specification of its behavior) such modification can sabotage subsystem function.

Threat Types

A second classification of threats can be made on the basis of type. We consider two:

1) direct (overt); and

2) indirect (covert).

Direct threats involve "direct" means: a write into a protected data base object. We must consider, in this case, the protection properties only of the accessing subject and accessed object. Indirect threats subsume a much larger class of scenarios. Generally, indirect threats refer to improper modifications resulting from the use of data or procedures developed (modified) by a malicious subsystem. This data (procedure), by not fulfilling expected

requirements, may then sabotage its user's functions. This structure requires knowledge, at each access, of both the ultimate source and transfer path of the accessed information.

Examples

We can illustrate this threat taxonomy by an example drawn from the "people system." Let us consider threats to the physical integrity of a person. External threats take the form of another person perpetrating physical harm. Internal threats take the form of self-inflicted physical harm. For instance:

- external direct: a direct assault by another person with, say, a knife;
- external indirect: an assault by another person via surreptitious means, say, by poison covertly placed in food;
- internal direct: suicide via direct means, say, a gun; and
- internal indirect: unsuspecting suicide via, say, poor care for one's body.

A protection policy can be formulated to block each of these threats. For example, a policy which identifies persons who might commit assault can be used to segregate them so that they have no opportunity (no access) to commit assault and to insure that they are not hired as cooks or servers. In these cases, selective isolation is sufficient protection. However, what about internal threats? These can only be blocked by some certification of an individual's ability and desire to take proper care of himself.

The example can be easily extended to computer systems. For instance:

- external direct: one subsystem maliciously modifies a necessary data base, say executable code, of another subsystem;
- external indirect: one subsystem foists a maliciously behaving subroutine onto another subsystem (Trojan Horse attack);
- internal direct: self-modifying code; and
- internal indirect: inadvertent self-modifying code, say, via improperly initialized pointers.

External and internal threats are illustrated in Figure 3. Subject S1 maliciously modifies object O1, causing subject S2's behavior to

change. Subject S3 modifies its own data base (object 02) in an unanticipated manner, causing changes in its subsequent behavior.

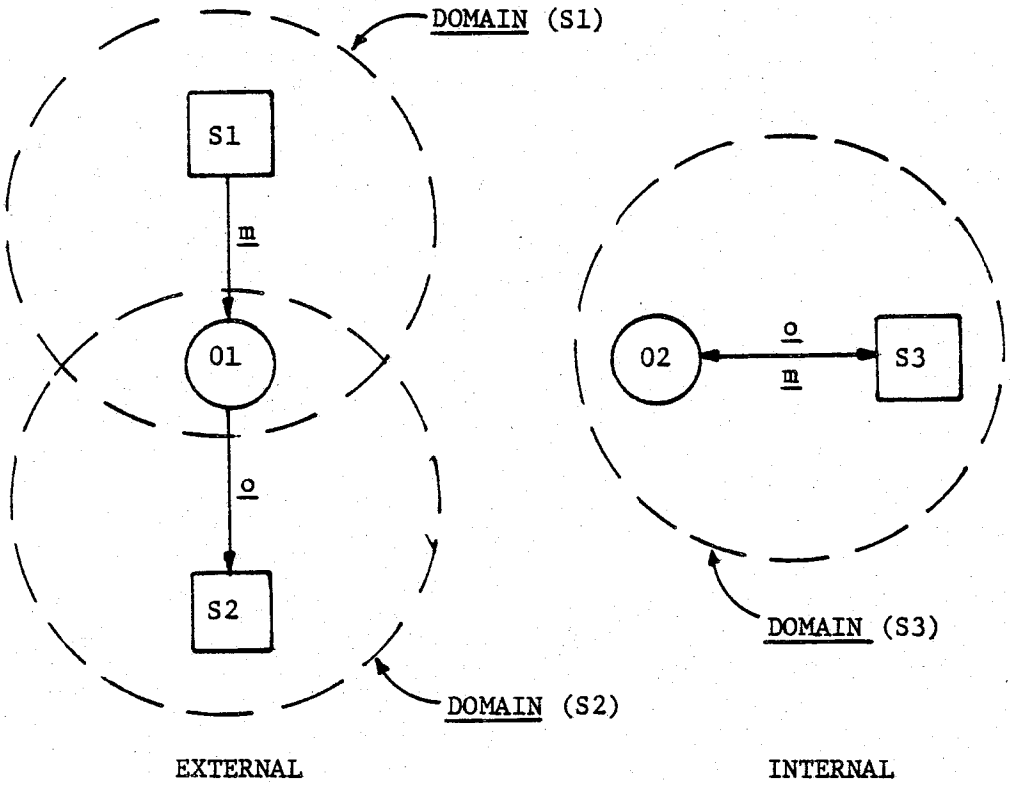

Figure 3. External and Internal Integrity Threats

INTEGRITY POLICY ENFORCEMENT

In each of the above cases, our primary concern is an identification of those modifications which preserve the validity (intended properties) of computer system elements. Integrity policy concisely organizes this information so that the propriety of a given access can be easily evaluated by an enforcement mechanism. Two issues govern the types of integrity policy a given system may support:

1) available enforcement mechanisms; and

2) the integrity problems the system must address.

This subsection will address the former issue, the following subsection the latter.

The space of enforcement mechanisms is diagrammed in Figure 4. We identify two dimensions: enforcement frequency and enforcement granularity. Frequency refers to the time at which access control enforcement occurs. If enforcement is performed only once (viz., at the time a program is created) the frequency is termed static. If enforcement is performed at each access by a program to an object, the frequency is termed dynamic. One-time program verification of access propriety is a static enforcement mechanism. Hardware descriptor mechanisms support dynamic enforcement.

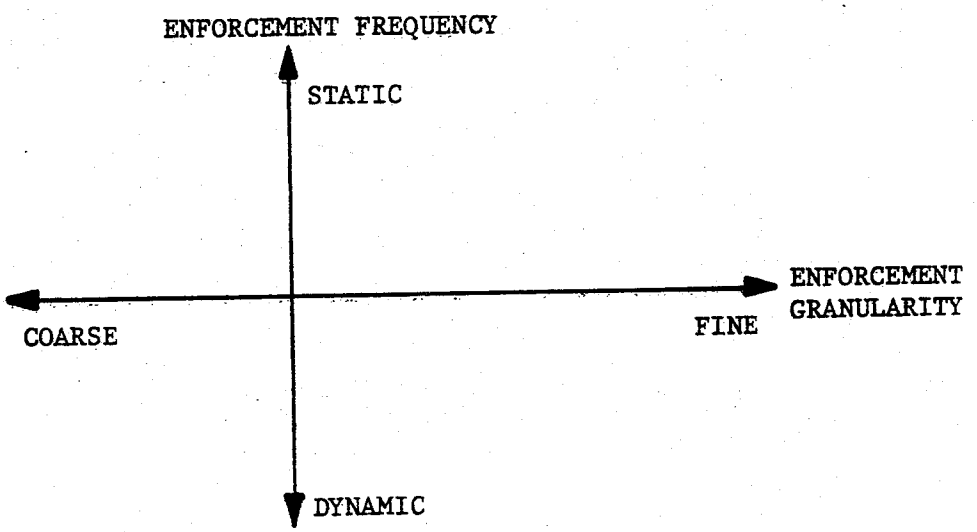

Figure 4. Enforcement Mechanisms

Granularity refers to the size and resolution of the protected system elements. For effective enforcement of an integrity policy, the granularity of enforcement must match the granularity of the policy. For example, if a protection policy controls access to parts of a file, an enforcement mechanism that only controls access to the entire file cannot effectively implement the policy.

Another example mechanism supports static enforcement at a coarse granularity. The propriety of access by a program to coarsely defined objects (entire files) may be statically evaluated by verification of the program text at the time the program was created. We can insure, via this technique, that the program only accesses certain named files in a proper manner. However, if the renaming of files is a facility supported by the system, this verification can be invalidated by the renaming of files accessed by the program. We note that the solution to this renaming problem centers about the matching of the frequency and granularity of access enforcement with the frequency and granularity of the binding of program names to objects.

These examples illustrate that policies that alter the protection attributes and existence of subjects and objects must have dynamic enforcement at the granularity of the attribute alteration to support effective access control. Since a computer utility (our intended application) supports a dynamic environment with these properties, our subsequent discussions focus on integrity policies suited for a dynamic environment.

The integrity of information is maintained by guaranteeing that only proper modifications are made. As indicated above, this can be done in a number of ways ranging from access control at execution time to program verification. We find that internal threats, in general, cannot be addressed by dynamic access control mechanism. Indeed, we note that for any given computer system, the hardware access control mechanism defines a certain level of subsystem access control granularity. Any access restriction below this level of granularity (or based on other than hardware defined access modes) must be guaranteed by static enforcement.

The static nature of internal threat policy enforcement places it beyond the scope of this report. The prevention of internal threats is more the province of program verification. Therefore, our concern focuses on integrity policies for external threats.

INTEGRITY PROBLEMS

Specific interpretations of the notion of "proper" modification and the consequent protection policies are dependent on problem specific protection requirements. This subsection does not address all conceivable protection problems: an obviously impossible task. It does identify certain problems that have, historically, proved troublesome.

We begin by restating the fundamental principle of integrity policy: identification and enforcement of proper modifications. Our job, then, must start with the isolation of relevant, proper modifications. For a secure DoD computer utility, three classes of proper modifications are of immediate interest:

1) propriety of modification with respect to the national security importance of information;

2) propriety of modification with respect to the identity of an accessing user ("need-to-modify"); and

3) propriety of modification with respect to application specific reasons.

We consider each in turn.

National Security

The integrity of national security information is clearly of paramount importance for the intended user community. The malicious modification of crucial information can, in some circumstances, be of greater importance than its compromise (unauthorized observation). Consider the case of global data bases accessible to a large community of innocuous applications yet of critical importance to a few applications. For example, a data base defining interstate transportation routes is useful to a large number of non-critical tasks. Yet the sound construction of this data base is crucial to the proper operation of logistics programs in time of national emergency. Clearly, this data base must be protected from modification to a degree commensurate with its importance to national security (its accessing applications), while still being observable to a variety of applications at differing security levels.

A similar situation pertains to access to procedures. Consider the ill-conceived use of a library subroutine by a critically important program. An optimal routing subroutine of a logistics management package is a good example. If the subroutine is not protected from malicious modification at least to the degree of the subject using it, sabotage of its function may cause improper behavior of its caller. Yet the library routine should be executable (observable) by the entire user community, at many levels of importance to national security.

Thus, a contradiction exists between the protection necessary for observation and modification. A single security level cannot be assigned to these objects so as to satisfy both protection requirements: (observable by everyone) and (modifiable by no one). This

apparent difficulty must be addressed by an integrity policy distinct from the security (compromise) policy.

Available DoD policy [10] requires access controls that guarantee data integrity: that is, the accessibility, maintenance, movement, and disposition of data shall be governed on the basis of security classification and need-to-know. However, the interpretation of such classification and need-to-know. However, the interpretation of such controls has centered on the security issue: information compromise. Our requirement is the specification of protection policies addressing the sabotage of information important to the national security.

We should note that similar considerations apply to applications other than the military. The need to compartmentalize data modification exists in a variety of application areas.

User Identity

Protection against improper modification based on user identity takes a somewhat different form. Consider a data base owned by user A. User B wishes to read this data base. User A does not trust user B to non-maliciously access an only copy and would like to extend read-only access to this specific data base only to user B and his subjects (from the set of all users). If user B misuses his privilege to the data base, user A must have the capability to revoke user B's access. The protection mechanism must support dynamic revocation based on user identify. It should be noted that while the previous problem isolates classes of users (by national security privilege) it is orthogonal with respect to this problem since a given user may operate at many levels of national security privilege.

Protected Subsystems

The preceding problems addressed access restrictions based on properties of the person who either invokes or creates a subsystem. A distinct class of protection problems arise from the use of system services. These services take the form of subjects, system-supplied, that may be invoked by user-supplied subjects to perform privileged functions. These subjects (and the data and procedure they access) must be explicitly protected from their invoker. Likewise, the invoking subject often requires protection from the invoked subject.[4]

[4]This situation has been referred to as the "mutually suspicious problem" [11].

Two important instances of protected system services are:

1) a security kernel; and

2) a distributed operating system.

However, in these instances the security kernel and operating system require access to user defined data bases (viz., parameters). Thus, in the simple case, the operating system has unlimited access to user space, but the user has only carefully restricted access to operating system space.[5] Our consideration of this problem will not be general. Rather, we concentrate on explicating a policy sufficient to isolate a kernel.

All three of the above examples of specific problems are instances of the archetypal model presented in Section I. Each policy attempts to isolate domains of access privilege, each (possibly) disjoint from those defined by the other policies. The access frequency and general properties of each suggest different implementation mechanisms. Indeed, the fundamental property of rate of change of privilege, either by the accessing subject or by accessed object motivates the tailoring of policy and mechanism to the properties of the access.

PROTECTION ENVIRONMENTS

Subsystems within a computer system perform differing functions and thus often require differing integrity policies. The protection requirements are also a function of the perspective with which subsystems are viewed. We make the following definitions of perspective:

1) each system user has (at least) one process executing in his/her behalf;

2) each process is composed of a number of subjects and objects, partitioned into domains of access privilege within each process; and

3) some subset of the most privileged subjects and objects within each process define the protected subsystem that comprises the security kernel.

---

[5] Primarily through "system service calls" (viz., "gates" in Multics) that permit controlled invocation of operating system functions by the user. An important special case is the invocation of the kernel from uncertified user subsystems.

Based on these definitions, we can identify two environments with distinct integrity requirements:

1) the set of subjects and objects not contained in the security kernel, these will be collectively referred to as the user virtual environment; and
2) the set of subjects and objects that compose the security kernel, these will be collectively referred to as the kernel environment.

Section IV applies the integrity policies developed in Section III to each of these protection environments.