FOREWORD

The National Computer Security Center is issuing *A Guide to Understanding Security Modeling in Trusted Systems* as part of the "Rainbow Series" of documents produced by our Technical Guidelines Program. In the Rainbow Series, we discuss, in detail, the features of the *Department of Defense Trusted Computer System Evaluation Criteria (DoD 5200.28-STD)* and provide guidance for meeting each requirement. The National Computer Security Center, through its Trusted Product Evaluation Program, evaluates the security features and assurances of commercially-produced computer systems. Together, these programs ensure that organizations are capable of protecting their important data with trusted computer systems. Security modeling, in its various forms, is an important component of the assurance of the trust of a computer system.

A Guide to Understanding Security Modeling in Trusted Systems is intended for use by personnel responsible for developing models of the security policy of a trusted computer system. At lower levels of trust, this model is generally the system"s philosophy of protection. At higher trust levels, this also includes informal and formal models of the protection mechanisms within a system. This guideline provides information on many aspects of security modeling, including the process of developing a security policy model, security modeling techniques, and specific ways to meet the requirements of the *Department of Defense Trusted Computer System Evaluation Criteria.*

As the Director, National Computer Security Center, I invite your suggestions for revising this document. We plan to review and revise this document as the need arises.

Patrick R. Gallagher, Jr. October 1992
Director
National Computer Security Center

ACKNOWLEDGMENTS

Special recognition and acknowledgment for their contributions to this document are extended to the following: Dr. James Williams, as primary author of this document; Barbara Mayer, Capt. James Goldston, USAF, and Lynne Ambuel, for project management.

Special thanks are also given to the many evaluators, vendors, and researchers whose careful reviews and suggestions contributed to this document, with special recognition for the advice of David Bell, Daniel Schnackenberg, Timothy Levin, Marshall Abrams, Deborah Bodeau, Leonard La Padula, Jonathan Millen, William L. Harkness, Thomas A. Ambrosi, Paul Pittelli, Dr. Michael Sinutko, Jr, and COL Rayford Vaughn, USA.

TABLE OF CONTENTS

1. INTRODUCTION

This document provides guidance on the construction, *evaluation*, and use of security policy models for *automated information systems (AIS)* used to protect *sensitive information* whose unauthorized disclosure, alteration, loss, or destruction must be prevented. In this context, sensitive information includes classified information as well as unclassified sensitive information.

1.1 BACKGROUND

The National Computer Security Center (NCSC) was established in 1981 and acquired its present name in 1985. Its main goal is to encourage the widespread availability of trusted AISs. In support of this goal, the *DoD Trusted Computer System Evaluation Criteria (TCSEC)* was written in 1983. It has been adopted, with minor changes, as a DoD standard for the protection of sensitive information in DoD computing systems. [NCSC85] The *TCSEC* is routinely used for the evaluation of commercial computing products prior to their *accreditation* for use in particular environments. This evaluation process is discussed in *Trusted Product Evaluations: A Guide for Vendors.* [NCSC90c]

The *TCSEC* divides AISs into four main divisions, labeled D, C, B, and A, in order of increasing security protection and *assurance.* Divisions C through A are further divided into ordered subdivisions referred to as classes. For all classes (C1, C2, B1, B2, B3 and A1), the *TCSEC* requires system documentation that includes a *philosophy of protection*. For classes B1, B2, B3, and A1, it also requires an *informal* or *formal security policy model*.

Although the *TCSEC* is oriented primarily toward operating systems, its underlying concepts have been applied much more generally. In recognition of this, the NCSC has published a *Trusted Network Interpretation* [NCSC87] and a *Trusted Database Management System Interpretation.* [NCSC91] In addition, the NCSC also provides a series of guidelines addressing specific TCSEC requirements, of which this document is an example.

1.2 PURPOSE

This guideline is intended to give vendors and evaluators of trusted systems a solid understanding of the modeling requirements of the *TCSEC* and the *Trusted Network Interpretation of the TCSEC (TNI).* It presents modeling and philosophy of protection requirements for each of the classes in the *TCSEC,* describes possible approaches to modeling common *security requirements* in various kinds of systems, and explains what kinds of *mode is* are useful in an evaluation. It is intended for vendors, evaluators, and other potential builders and *users* of security models.

This guideline discusses the philosophy of protection requirement, explains how it relates to security modeling, and provides guidance on documentation relating to the system's philosophy of protection and security policy model. It explains the distinction between informal and formal security policy models as well as the relationships among application-independent models, *application-dependent security models,* and model interpretations. It also explains which techniques may be used to meet the modeling requirements at levels B1 through Al as well as the advantages and disadvantages of the various modeling techniques. Finally, it discusses the specific *TCSEC* modeling requirements.

This guideline also addresses human aspects of modeling. It describes how modeling captures basic security requirements and how the security modeling effort leads to better systems and provides a basis for increased assurance of security.

Finally, this guideline answers common questions about NCSC recommendations regarding the construction of security models. Security policies and models are supplied by vendors in response to customer needs and *TCSEC* requirements; they are not supplied by the NCSC or the *TCSEC*. The *TCSEC* does, however, set minimum requirements in the areas of *mandatory and discretionary* access control. The *TCSEC* does not require particular implementation techniques; this freedom applies, by extension, to modeling techniques. Acceptability of a technique depends on the results achieved. More specifically, a security model must provide a clear and accurate description of the security policy requirements, as well as key ideas associated with their enforcement. Moreover, one must be able to validate the model using assurance techniques appropriate to the proposed evaluation class. Any vendor-supplied modeling technique which appears to be compatible with the security modeling requirements will be seriously considered during the course of a product evaluation.

Topics which are closely related to security modeling include formulation of security policy objectives, design specification and *verification, covert channel analysis,* and implementation correspondence analysis. These topics are addressed only to the extent necessary to establish their influence on the security modeling process. All but the first of these topics are addressed in other guidelines in this series. Security objectives for trusted systems traditionally include nondisclosure, *integrity,* and *denial of service. [*cf NCSC88*, AIS Security]* However, the modeling of denial of service is not addressed in this guideline, due to the lack of adequate literature on this topic. This guideline is written with the understanding that security modeling will continue to evolve in response to new policy objectives and modeling techniques associated with future trusted system designs.

Reading and understanding this guideline is not sufficient to allow an inexperienced individual to develop a model. Rather, he or she should read this document in conjunction with one or more of the published models in the list of references. This guideline assumes a general familiarity with the *TCSEC* and with computer security. A general text such as *Building a Secure Computer System* [GASS87] may provide useful background reading. Additional mathematical background needed for formal modeling can be found in general texts on mathematical structures such as *Introduction to Mathematical Structures* [GALO89].

The approaches to security modeling presented in this document are not the only possible approaches to satisfying *TCSEC* modeling requirements, but are merely suggested approaches. The presentation of examples illustrating these approaches does not imply NCSC endorsement of the products on which these examples are based. Recommendations made in this document are not supplementary requirements to the *TCSEC.* The *TCSEC* itself (as supplemented by announced interpretations [NCSC87, NCSC88a, NCSC88b, NCSC91]) is the only metric used by the NCSC to evaluate trusted computing products.

1.3 CONTROL OBJECTIVES

The requirements of the *TCSEC* were derived from three main control *objectives:* assurance, security policy, and accountability. The security modeling requirements of the *TCSEC* support the assurance and security policy control objectives in systems rated B1 and above.

1.3.1 THE ASSURANCE OBJECTIVE

The *TCSEC* assurance control objective states, in part,

> "Systems that are used to process or handle classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policy and must not distort the intent of that policy." [NCSC85, § 5.3.3]

Assurance in this sense refers primarily to technical assurance rather than social assurance. It involves reducing the likelihood that security mechanisms will be subverted as opposed to just improving people's confidence in the utility of the security mechanisms.

1.3.2 THE SECURITY POLICY OBJECTIVE

The TCSEC security policy control objective states, in part,

> "A statement of intent with regard to control over access to and dissemination of information, to be known as the security policy, must be precisely defined and implemented for each system that is used to process sensitive information. The security policy must accurately reflect the laws, regulations, and general policies from which it is derived." [NCSC85, § 5.3.1]

The security policy objective given in the *TCSEC* covers "mandatory security policy," "discretionary security policy," and "marking" objectives. The mandatory security policy objective requires the formulation of a mandatory access control (MAC) policy that regulates access by comparing an individual's clearance or authorization for information to the classification or sensitivity designation of the information to be accessed. The discretionary access control objective requires the formulation of a discretionary access control (DAC) policy that regulates access on the basis of each individual's *need-to-know,* Finally, the marking objective gives labeling requirements for information stored in and exported from systems designed to enforce a mandatory security policy.

The access controls associated with security policies serve to enforce nondisclosure and integrity. *Nondisclosure controls* prevent inappropriate dissemination of information. Integrity controls prevent inappropriate modification of information.

The security policy control objective requires that the security policy accurately reflect the laws, regulations, and general policies from which it is derived. These may include the revised DoD Directive 5200.28, *Security Requirements for Automated Information Systems (AISs).* [DOD88a] Section D of Directive 5200.28 gives policy relating to both integrity and nondisclosure. It mentions safeguards "against sabotage, tampering, denial of service, espionage, fraud, misappropriation, misuse, or release to unauthorized users." Enclosure 2 defines relevant terms and, in particular, extends the concept of "user" to include *processes* and devices interacting with an automated information system on behalf of users. Enclosure 3 gives general security requirements, including data-integrity requirements. The revised 5200.28 does not use the MAC/DAC terminology. Instead, it requires accurate marking of sensitive information and the existence of an (undifferentiated) access control policy based partly on the identity of individual users. It treats need-to-know as a form of *least privilege.*

Security policy relating to nondisclosure stems from Executive Order 12356 [REAG82] and, in the case of DoD systems, from DoD 5200.1-R. [DOD86] Depending on the application, the security policy may be constrained by a variety of other regulations as well. The collection, maintenance, use, and dissemination of personal information is protected by DoD Directive 5400.11 [DOD82, § E.2] and by Public Law 100-503 [CONG88]. Classified and other sensitive information needed for the conduct of federal programs is protected by the Computer Security Act of 1987, Public Law 100-235 [CONG87], which requires safeguards against loss and unauthorized modification.

1.4 HISTORICAL OVERVIEW

The starting point in modeling a MAC policy is the observation that *security levels* are *partially ordered* (see Appendix A). This fact was first reflected in the design of the ADEPT-50, an IBM 360-based, time-sharing system with both mandatory and discretionary controls. [WEIS69, LAND81] The central role of partial orderings in MAC policy models was then recognized in subsequent work. [POPE73; BELL73; BELL74a] These partial orderings on security levels have become known as *dominance* relations.

Independently of this, access *control* matrices were used by Lampson and Graham [LAMP71; GRAH72] to represent protection data. These works modeled discretionary access control using matrices whose rows and columns represented subjects and objects. *Subjects* were active entities such as processes and users. *Objects* were entities such as data structures that played a passive role, but often included subjects as well. Each entry in an access matrix told what a given subject was allowed to do with a given object by giving a set of allowed operations such as read, write, execute, or change ownership. The AIS used the access matrix to mediate all accesses by subjects to objects.

An Air Force *Computer Security Technology Planning Study* [ANDE72], referred to as "the Anderson Report," discussed use of a reference monitor to enforce nondisclosure policies and advanced the idea of *formal security verification*— the use of rigorous mathematical proofs to give (partial) assurance that a program design satisfies stated security requirements. This idea was investigated by Schell, Downey and Popek. [SCHE73]

At that time, Bell and La Padula adapted access control matrices for use in modeling both mandatory and discretionary access controls and codified the reference monitor concept using *state machines.* Their machine states included access matrices and other security-relevant information and are now often referred to as *protection states.* Having established the necessary framework, they observed that a portion of DoD policy for nondisclosure of classified information could be formalized as *invariant* properties of protection states [LAPA73], that is, as properties which hold in all *reachable* states of the system. Their invariants constrained the allowed accesses between subjects and objects. The most significant of these was their **-property* It included a form of the *simple security property* and guaranteed that the security level of every object read by an untrusted subject was at or below the security level of every object written by that subject. [BELL74] These ideas and their use in enforcing nondisclosure are covered in detail in Sections 3.2.4 and 4.1.

To complete their state machine, Bell and La Padula introduced a set of state transformations, called *rules of operation,* that modeled basic changes in a protection state and then rigorously proved that the *r*ules of operation preserved the identified state invariants. [LAPA73, BELL74, BELL76] This work contains the first widely discussed use of mathematical proof for studying multilevel security.

In parallel with Bell and La Padula, Walter and others developed a similar approach to verifying multilevel security. [WALT74, WALT74a] A strong point of the work by Walter et al. is the use of modularity and data abstraction techniques that have since been accepted as indispensable for verifying large systems.

The intent of the state invariants identified by Bell and La Padula is that information is allowed to flow from one *entity* to another only if the second entity is at an equal or higher security level than the first. Denning and others attempted to formalize this idea directly using "information flow" models. [DENN76, DENN77, COHE77, REIT79, ANDR80]. These models differed in style from access control models in that they referred explicitly to information flow, but, in contrast to more recent investigations, did not actually define information flow. Denning's work did, however, point out an interesting distinction. In the conditional assignment

$$\textbf{if } a = 0 \textbf{ then } b := c,$$

information flows explicitly from c to b (when a = 0) and implicitly from a to b (when b ≠ c). Denning also pointed out that, in the case of the ADEPT-50, access control can easily allow implicit information flow. [DENN77] This problem is discussed in detail in Section 3.2.

Discrepancies between information flow and access control open up the possibility of *covert channels*— paths of information flow that malicious processes can use to bypass the access control mechanism and thereby violate underlying security objectives. [cf LAMP73] Information flow models and concern over covert channels have led to the development of techniques and tools for covert channel analysis that are capable of detecting a variety of covert channels. [MILL76, MILL81, FEIE77, FEIE80]

The developments just described provided the technical background for the security modeling requirements given in the *TCSEC.* A variety of important developments not explicitly reflected in the *TCSEC* have taken place in the last decade and will be presented later in this guideline. The class of security policies that can be formally modeled has been greatly extended, beginning with Biba's (pre-TCSEC) work on integrity. [BlBA77] Work has also gone into tailoring nondisclosure security to particular applications. Finally, various *external-interface models* have been developed that do not constrain internal system structure, an example of which is the noninterference model of Goguen and Meseguer. [GOGU82] These models provide a rationale for rigorously assessing new approaches to access control.

1.5 CONTENT AND ORGANIZATION

Section 2 presents an overview of the security modeling process, with emphasis on correctness and utility. Section 3 presents technical detail on how to model concepts of interest, including nondisclosure and integrity policies, mandatory and discretionary access controls, and exemption from access control within the Trusted Computing Base (TCB).

Section 4 shows how to apply the techniques from Section 3 to various kinds of systems; including operating systems, networks, database systems, and multilevel workstations. Finally, Section 5 summarizes all of the *TCSEC* security modeling requirements for B1, B2, B3, and A1 computing systems.

Appendix A presents facts about lattices and partially ordered sets that are needed for Sections 3 and 4. Appendix B contains brief descriptions of available support tools. Appendices C and D contain suggested outlines for a philosophy of protection and a security policy model. Finally, Appendix E is a glossary giving definitions of technical terms. This glossary includes all terms that are introduced in italics throughout the guideline.

When *TCSEC* requirements are discussed in this guideline, they are identified either by the key words "must" or "shall" or by explicit quotations from the *TCSEC*. By way of contrast, when desirable but optional actions and approaches are discussed, they are presented without exhortation and are instead accompanied by an explanation of specific advantages or benefits. In a few cases, possible requirements are designated by "should," because the implications of the *TCSEC* are not fully understood or agreed upon.

2. OVERVIEW OF A SECURITY MODELING PROCESS

A security model precisely describes important aspects of security and their relationship to system behavior. The primary purpose of a security model is to provide the necessary level of understanding for a successful implementation of key security requirements. The *security policy* plays a primary role in determining the content of the security model. Therefore, the successful development of a good security model requires a clear, well-rounded security policy. In the case of a formal model, the development of the model also must rely on appropriate mathematical techniques of description and analysis for its form.

Sections 2.1 and 2.2 explain what security models describe, why they are useful, and how they are used in the design of *secure systems*. Section 2.3 introduces general definitions relating to security models and explains how security models are created. Finally, Section 2.4 discusses the presentation of a security model in a modeling document.

2.1 SECURITY MODELS AND THEIR PURPOSE

Early security models focused primarily on nondisclosure of information. More recently, the importance of data as a basis for decisions and actions has stimulated interest in integrity models. [DOD88a, WHIT84] For example, nondisclosure properties alone do not protect against viruses that can cause unauthorized, malicious modification of user programs and data.

A wide variety of concepts can impact nondisclosure and integrity in particular system designs. As a result, the content of security models is quite varied. Their primary purpose is to provide a clear understanding of a system's security requirements. Without such an understanding, even the most careful application of the best engineering practices is inadequate for the successful construction of secure systems.

Inadequacies in a system can result either from a failure to understand requirements or from flaws in their implementation. The former problem, defining what a system should do, is relatively difficult in the case of security. The definition must be precise in order to prevent undesired results, or subtle flaws, during the implementation of the system design.

During the entire design, coding, and review process, the modeled security requirements may be used as precise design guidance, thereby providing increased assurance that the modeled requirements are satisfied by the system. The precision of the model and resulting guidance can be significantly improved by casting the model in a formal language.

Once the system has been built, the security model serves the evaluation and accreditation processes. It contributes to the evaluators' judgement of how well the developers have understood the security policy being implemented and whether there are inconsistencies between security requirements and system design. Moreover, the security model provides a mechanism for tailoring the evaluation process to the vendor's needs because it presents the security concept that is supported by the system being evaluated. The inclusion of particular facts in the security model proclaims to evaluators and potential customers that those facts are validated at the level of assurance which the *TCSEC* associates with that system's evaluation class.

Upon successful evaluation and use, the security model provides a basis for explaining security-relevant aspects of system functionality. Later, during maintenance, it provides a basis for guidance in making security-relevant modifications. Finally, by suppressing inessential design detail, security models facilitate a broader understanding of security that can be applied to increasingly larger classes of systems. Among the many examples of such generalizations is the adaptation of traditional reference monitor concepts referenced in the *TNI* to provide a basis for understanding network security requirements. [NCSC87]

The intended purpose of a security model suggests several desirable properties. The requirements captured by a good model pertain primarily to security, so that they do not unduly constrain the functions of the system or its implementation. A good model accurately represents the security policy that is actually enforced by the system. Thus, it clarifies both the strengths and the potential limitations of the policy. (As an extreme example, if the system can simply declassify all objects and then proceed normally, as in McLean's System Z [MCLE87], a good model would show this.) Finally, a good model is simple and therefore easy to understand; it can be read and fully understood in its entirety by its intended audience. This last property cannot be achieved without significant care in choosing the contents, organization, and presentation of the security model. For example, the desire to provide a security model with the "look and feel" of UNIX® might need to be tempered by the need for simplicity and abstraction. [cf NCSC90b, Sec. 6.2]

2.2 SECURITY MODELING IN THE SYSTEM DEVELOPMENT PROCESS

Security requirements are best identified early in the system development process. Not identifying security requirements in a timely fashion is likely to have devastating effects on security assurance, security and application functionality, development schedule, and overall development costs. For example, in the case of a development using DOD-STD-2167A, [DOD88] this identification process would be part of the system requirements analysis. The identification of security requirements begins with the identification of high-level security objectives (as described in Section 1.3) and the methods by which they are to be met, including automated, procedural, and physical protection methods. This identification of security requirements and their derivation from identified higher-level security objectives is the initial material for a philosophy of protection (POP). As indicated in Appendix C, the philosophy of protection may also include a broad range of other topics such as the structure of the *trusted computing base (TCB)* and physical and procedural *security mechanisms*.

Those requirements in the philosophy of protection which deal with automated protection methods provide an initial definition of security for a *security policy model*. The model's purpose is to precisely state these requirements and to compare them with key aspects of the security enforcement mechanism. A security policy model in this sense contains two essential portions: a "definition of security" portion that captures key security requirements and a "rules of operation" portion that shows how the definition of security is enforced.[1]

The model's definition of security can be used to avoid major system development errors. It can be used to guide the design of basic software protection mechanisms and to influence the design, selection and use of underlying firmware and hardware protection mechanisms. The initial draft model, and supporting documentation, provides guidance as to system security during reviews of the system design. However, there often are discrepancies between the design and the model. Some of these are resolvable and can be identified and corrected during the normal design review process. In some cases, however, discrepancies are unavoidable and can only be resolved by making some assumptions that simplify the problem. These assumptions need to be justifiable, based on the model. These discrepancies can also be addressed through procedural restrictions on the use of the system. There are some portions of a security model that may require design information that is not initially available and must, therefore, be postponed. Possible examples include detailed rules of operation for a security kernel and security models for specific security-critical processes. In such cases, the system designer must ensure that discrepancies are noted and the completed system will satisfy the completed model.

To ensure that a design satisfies modeled security requirements, it is necessary to give a *model interpretation* which shows how the model relates to the system. For evaluation classes B1 and B2, this involves explaining how each concept in the model is embodied by the system design and informally demonstrating that the modeled requirements are met. Since the model's rules of operation must conform to the model's definition of security, the model interpretation need demonstrate only that the rules of operation are adhered to. For classes B3 and A1, the model interpretation is done in two steps. The design, as reflected in a *top-level specification (TLS),* is shown to be consistent with the model, and the implementation is shown to be consistent with the TLS. For Class B3, an informal *descriptive top-level specification (DTLS)* is used, an informal correspondence from the DTLS to the model is performed, and the implementation is shown to be consistent with the DTLS. At Class A1, the DTLS is supplemented with a *formal top-level specification (FTLS),* a formal verification proves that the FTLS is consistent with the model, and the implementation is shown to be consistent with the FTLS. A fuller summary of model-interpretation requirements is given in Section 5.

The role of security modeling in relation to other aspects of system development is summarized in Figure 2.1. Aspects that do not directly involve modeling, per se, are shaded in grey. Requirements concerning the model are summarized at the beginning of Section 5. The broadest document is the philosophy of protection (POP); it covers higher-level security objectives, derived security policies constraining the design and use of the system, and the protection mechanisms enforced by the TCB. The POP, the security policy, and the model all cover the system's definition

---

[1]The TCSEC Glossary identifies several kinds of formal security policy models but discusses modeling requirements only for state-machine models. For these, it identifies a "definition of security" portion, referred to as a "definition of a "secure state" and "rules of operation" portion, in an apparent summary of the Bell & La Padula approach. Both the definition itself and its subsequent use in product evaluations indicate that, while a state-machine approach is unnecessary, a model must contain both a definition of security and rules of operation which show how the definition of security is enforced. The definition of security should adequately support identified security policy objectives but does not necessarily have to consist of, or be limited to, a definition of secure state.

of security. Both the POP and the model cover key aspects of the TCB protection mechanisms. At B2 and above, the formal model supports a rigorous proof showing that the rules of operation enforce the definition of security, and the DTLS gives a functional description of the TCB protection mechanisms with emphasis on the TCB interface. At A1, the FTLS formalizes a large portion of the DTLS in order to verify that the TCB design satisfies the modeled security requirements.

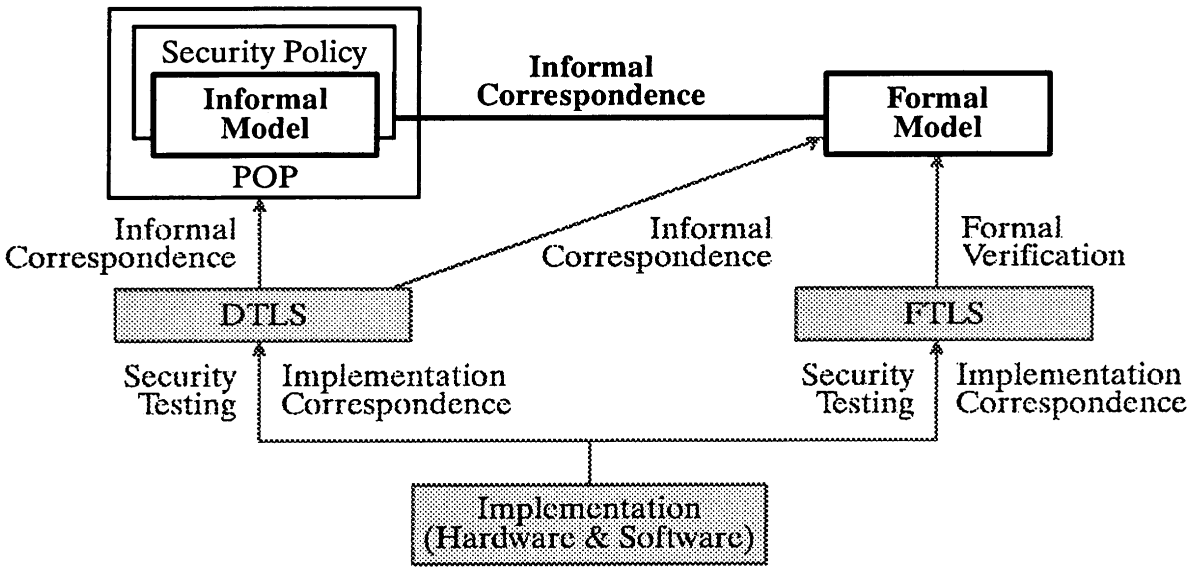

Figure 2.1. Security Documentation

The above paragraphs refer to a single security model but networks, database systems, and other complex systems may involve several security models or *submodels.* When a system is made up of several complex components or subsystems, interfaces between the components or subsystem layers must be modeled, if they play a key role in security protection. In this case, the best approach may be to develop separate models for each component or layer in order to show how the various subsystems contribute to overall system security. A danger with this approach, however, is that the combined effect of the various submodels may not be obvious. This is discussed further in Section 4.

2.3 IDENTIFYING THE CONTENTS OF A SECURITY MODEL

The most basic strategy in identifying the contents of a security model is perhaps that of divide and conquer. The modeling effort may be subdivided according to higher-level security objectives. Requirements on a system can be mapped to derived requirements on subsystems. Requirements for a particular objective can be classified according to level in a requirements hierarchy.

Security modeling provides a five-stage elaboration hierarchy for mapping a system's security policy requirements to the behavior of the system. As a result of this hierarchy, the phrases "security policy" and "security model" have taken on a variety of meanings. The five relevant stages are as follows:

1. Higher-level policy objectives
2. External-interface requirements

3. Internal security requirements
4. Rules of operation
5. Top-level specification

A higher-level objective specifies what is to be achieved by proper design and use of a computing system; it constrains the relationship between the system and its environment. The *TCSEC* control objectives belong to this first level of the requirements hierarchy. An external-interface requirement applies a higher-level objective to a computing system's external interface; it explains what can be expected from the system in support of the objective but does not unduly constrain internal system structure. *Internal security requirements* constrain relationships among system components and, in the case of a TCB-oriented design, among *controlled entities.* Rules of operation explain how internal requirements are enforced by specifying access checks and related behaviors that guarantee satisfaction of the internal requirements. A top-level specification is a completely functional description of the TCB interface. It also specifies behavior of system components or controlled entities.

In various contexts, the phrase security policy may refer to any or all of the first four stages of elaboration. In many contexts, the term security policy refers to organizational security policy. [cf NCSC85, Glossary] From a modeling perspective, organizational policies are the source of higher-level policy objectives. In much of the literature on security modeling, a system security policy is part of the requirements stages of development and provides the definition of security to be modeled. Additional thoughts on the importance of distinguishing among policy objectives, organizational policies, and system policies may be found in Sterne's article "On the Buzzword "Security Policy." [STER91] The terms "AIS security policy" and "automated security policy" are synonyms for "system security policy."

The security policy model is a model of the security policy enforced by the TCB. It is simultaneously a model of the policy and of the system for which the model is given. The portions of the system which are constrained by the model belong to the TCB by definition. The model's definition of security may contain external-interface requirements, but more traditionally consists of internal requirements. Its rules of operation may, in some cases, amount to a top-level specification. There is no firm boundary between rules of operation and top-level specifications: both explain how internal requirements are enforced by the TCB. However, more detail is required in a TLS, including accurate descriptions of the error messages, exceptions, and effects of the TCB interface. Security requirements occur implicitly in rules of operation and top-level specifications. In this form they are often referred to as security mechanisms. Internal requirements are sometimes classified as "policy" or "mechanism" according to whether or not they are derived from a specific higher-level policy objective.

Conceptually, the security modeling process takes place in two main phases. Requirements modeling takes place after a system security policy is fairly well understood. This is accomplished by constraining the system's design and use based on the higher-level objectives. Rules of operation may be deferred until after the definition of security is established and the basic architecture of the system has been identified.

The following paragraphs contain general suggestions for the construction of security models. Suggestions pertaining to specific kinds of security requirements and specific kinds of systems are given in Sections 3 and 4, respectively.

These suggestions are broken down into six steps that could be carried out with respect to an entire system and security policy or, more simply, for a given component and higher-level policy objective. The first step is to identify externally imposed security policy requirements on how the system (or component) must interact with its environment. The second is to identify the internal entities of the system and to derive the requirements portion of the model in such a way as to extend the external-interface requirements. The third is to give rules of operation in order to show how the modeled security requirements can be enforced. After completion of step three enough information generally is available to determine reliably whether the model is inherently new or can be formalized by known techniques. At classes B2 and above, this is the fourth step. The first three steps taken together result in the identification of several submodels. The fifth step is to demonstrate consistency among these various submodels, so that they fit together in a reasonable way to form a completed security policy model. Finally, the sixth step is to demonstrate relevance of the model to the actual system design. Typically, the majority of the security modeling effort is associated with these last two steps and associated revisions to the model. The total effort needed to produce a security policy model varies considerably, but is often between six staff-months and two staff-years.

2.3.1 STEP 1: IDENTIFY REQUIREMENTS ON THE EXTERNAL INTERFACE

The first step is to identify major security requirements and distinguish them from other kinds of issues. These identified requirements should adequately support known higher-level policy objectives for use of the system. An emphasis on external-interface requirements helps prevent an unrecognized mixing of security and design issues. Such mixing could interfere with the understanding of security and could impose unnecessary constraints on the system design.

External-interface requirements for a computer system can be described in several ways. An elegant, but possibly difficult, approach is to limit the discussion purely to data crossing the system interface; this is the “black-box” approach. The best known example of the black-box approach is noninterference (see Section 3.2.1). Alternatively, one can describe the system in terms of its interactions with other entities in its environment, such as other computing systems, users,[2] or processes. Finally, one can give a hypothetical description of internal structure that ensures the desired external-interface behavior.

In general, the system’s interaction with its environment is constrained by the sensitivity of the information handled and by the authorizations of the individuals and systems accessing the system. Identified user roles, associated privileges, and the extent to which certain roles are security-critical also limit the system’s interface to the environment. These constraints determine *security attributes* that are associated with the system's inputs and outputs. Security attributes may include classification, integrity, origin, ownership, source of authorization, and intended use, among others. The use of such attributes in the construction of security models is discussed in Section 3.

In the case of mandatory access control policies, information must be accurately labeled and handled only by authorized users. This requirement places restrictions on how information is input to the system and, implicitly, on how it will be processed. Authorized handling of information is modeled in terms of constraints on what information may flow from one user to another:

[2]In this guideline, a *user* is always an individual human user.

information input to the system as classified information should be output only to authorized recipients.

In addition to general regulations, there are often site-dependent and application-dependent constraints that may need to be modeled. In particular, there may be site-dependent constraints on allowed *security labels*.

Having identified the security requirements on the system interface, it is necessary to decide on the requirements to be covered in the model's definition of security. It must then be decided which of these requirements should be modeled directly or indirectly in terms of internal requirements on *system entities* and the TCB interface. The set of requirements to include is constrained by minimal *TCSEC* requirements, the need to adequately support relevant policy objectives, and the need for a simple, understandable model. The inclusion of more requirements may provide more useful information for accreditation once the system is evaluated, but it also increases the difficulty of the vendor's assurance task. The inclusion of more requirements also suggests a more careful structuring of the model in order to show how various aspects of security fit together.

Several factors may influence a decision of whether to directly include external-interface requirements in the security model. The direct inclusion of external-interface requirements can help explain how the model supports higher-level policy objectives. In the case of an application security model, the direct modeling of user-visible operations may be more relevant to end users, a point of view reflected in the SMMS security model. [LAND84] In the case of a network security model, understanding is facilitated by modeling of the network's interaction with hosts in its environment, as will be discussed in Section 4.2.

2.3.2 STEP 2: IDENTIFY INTERNAL REQUIREMENTS

To support the identified external requirements, the system must place constraints on the controlled entities of the system. These internal constraints traditionally form the model's definition of security. In a model whose definition of security contains external-interface requirements, internal constraints can provide a needed bridge between the definition of security and the rules of operation.

The controlled entities themselves should be identified at a level of granularity that is fine enough to allow needed security-relevant distinctions among entities. At class B2 and above, the controlled entities should include all (active and passive) system resources that are accessible outside of the TCB. For convenience, controlled entities may be grouped into subclasses in any way that facilitates understanding of the system. Such groupings will depend on the system to be modeled. For an operating system, the relevant controlled entities might include buffers, segments, processes, and devices. In most models, it has been convenient to group entities according to whether they play an active or a passive role in order to help show how the TCB implements the reference monitor concept[3] . For networks and other complex systems, identification of controlled entities may need to be preceded by identification of subsystems and their derived security requirements.

---

[3] In [ANDE72], active entities are called "subjects". This use of the word is slightly broader than that used in this guideline, where subjects are essentially controlled processes.

Constraints on controlled entities are best stated as general properties and relationships that apply to all (or a broad class of) entities and accesses. Greater generality eliminates unnecessary constraints on the system design, improves on's intuition about the model, and can greatly reduce the overall effort needed to validate correctness of the identified constraints.

The identification of necessary constraints on controlled entities and their interactions is a nontrivial task; familiarity with similar systems and security models can be of great benefit. To define the necessary constraints, it will be necessary to label each entity with appropriate security attributes and to identify possible kinds of interactions, or accesses, between controlled entities. As used here, an access is any interaction that results in the flow of information from one entity to another. [cf DOD88a]

In describing access constraints, it may be useful to know that some kinds of accesses are highly restricted, as when a process accesses a file by executing it. The notion of causality may also be important: a transfer of information from entity e_1 to e_2 might occur because e_1 <u>wrote</u> e_2, because e_2 <u>read</u> e_1,or because a third entity directly <u>copied</u> e_1 to e_2 without actually reading e_1. [cf MCLE90, Sec. 2]

In the particular case of state machine models, constraints often take the form of state *invariants,* as in the work of Bell and La Padula. Recent efforts suggest that *state-transition* constraints are an attractive alternative to state invariants for certain kinds of security requirements. [MCLE88; NCSC90b, Sec. 6.7] The simplest well-known, state-transition constraint is the *tranquility* principle of Bell and La Padula, which says that the security level of a controlled entity cannot change during a state transition. Tranquility can be artificially rephrased as a state invariant: in any reachable state, the level of an entity coincides with its level in the initial state. Consider, however, the DAC requirement that a subject which gains access to an object must possess the necessary DAC permissions when the access request is made. This is harder to rephrase as a state invariant because DAC permissions can change from one state to the next. Good tutorial examples of state invariants and state-transition constraints may be found in *Building a Secure Computer System* [GASS87, Sec. 9.5.1, 9.5.2]; more complex examples occur in the TRUSIX work (NCSC90b, Sec. 2].

The number of internal requirements in a typical model varies considerably depending on the desired level of assurance, the complexity of the policy and system being modeled, and the granularity of the individual requirements. For example, the Trusted UNIX Working Group (TRUSIX) model covers *TCSEC* access control requirements for a B3 Secure Portable Operating System Interface for Computer Environments (POSIX) system and has eleven state invariants and ten state-transition constraints. [NCSC90b]

The question of which security requirements to cover in the model's internal requirements is answered as much by issues of good engineering practice as by the *TCSEC.* At a minimum, the model must cover the control of processes outside the TCB. Such processes are potentially a significant security risk because they may be of unknown functionality and origin. According to current practice, those portions of the TCB that do not support user-requested computation do not have to be modeled. For example, the security administrator interface does not have to be modeled. However, if significant user-requested computation is performed by some portion of the TCB, it should be modeled. A typical example would be a trusted downgrading process available to authorized users (not necessarily the security administrator). A more extreme example would be a multilevel database management system implemented as a *TCB subject* whose database consisted of a single highly-classified file.

Finally, there is the possibility that some security requirements might be included in the model, but only in the rules of operation. The rules of operation may allow the application of good security practice closer to the implementation level. The rules of operation can then contain security conditions that are not explicitly required by the security policy. Rules of operation can not make up for an inadequate definition of security, however. The task of inferring a modeled security policy from the rules of operation may be excessively difficult, especially if the rules are complex. [cf NCSC90b, Sec. 6.8]

2.3.3 STEP 3: DESIGN RULES OF OPERATION FOR POLICY ENFORCEMENT

How can the modeled security requirements on system entities be enforced? The rules of operation answer this question in broad terms by describing abstract interactions among system entities with particular emphasis on access control and other policy-enforcement mechanisms. In the case of an operating system kernel, the rules of operation would typically describe state transformations and associated access checks needed to uphold the definition of security. In the case of a formal model, this step amounts to the construction of a miniature FTLS. It is a useful preliminary step, even if a full FTLS specification is planned. The rules of operation together with the modeled requirements on controlled entities form the security policy model.

In giving rules of operation, it is important that system behavior be adequately represented. However, actual interactions among controlled entities need not be directly specified when they can be described as a composition of several rules of operation. There is no need for an implementation to directly support an individual rule where several rules have been combined to describe an action. An appropriate level of detail for rules of operation is illustrated by the work of Bell and La Padula. [BELL76] Good tutorial examples may be found in *Building a Secure Computer System.* [GASS87, § 9.5.1 (Step 3)] More complex examples may be found in the TRUSIX work. [NCSC90b] The rules of operation will usually be more readable if they are not too detailed. At class B3 and above, it is especially important to avoid details that are not security relevant. This is because their inclusion in the model may force their inclusion in the TCB in order to achieve a successful model interpretation, thereby violating the TCB minimization requirements. [NCSC85, Sec.3.3.3.1.1]

In the case of a B1 informal model, the rules of operation may reasonably contain information that, for higher evaluation classes, would be found in a DTLS. Thus, a stronger emphasis on policy enforcement than on policy requirements may be useful. This is especially true if the TCB is large and complex, as, for example, when security is retrofitted onto a previously unevaluated system. [BODE88]

A formal model needs to formalize the idea that a particular state transformation is being executed on behalf of a particular subject. Two traditional approaches are to add an extra input to the state transformation that gives the subject id and to add a “current subject” field to the system state. With the former approach, accompanying documentation should explain clearly how the subject-identity parameter is passed, in order to avoid the erroneous impression that the subject is responsible for identifying itself to the TCB. The latter approach suggests that the system being modeled has a single central processing unit. This may be a problem for the model interpretation if the system actually contains multiple CPUs. See the TRUSIX work for further discussion. [NCSC90b, Sec. 6.13]

Finally, in designing rules of operation, it may be convenient to separate access decisions from other kinds of system functionality, including the actual establishment or removal of access. This separation facilitates exploration of new access control policies. Only the access decision portion is affected by a change in policy because access enforcement depends only on the access decision, not on how it was made. [LAPA90, ABRA90] Isolation of the access decision mechanism occurs in the LOCK system design [SAYD87] and in the security policy architecture of Secure Ware's Compartmented Mode Workstation [NCSC91b, Section 2.2.1.].

2.3.4 STEP 4: DETERMINE WHAT IS ALREADY KNOWN

Usually some, if not all, aspects of the identified security model will have been studied before. The identification of previously used terminology allows security issues to be presented in a manner that is more easily understood; and making the connection to previously studied issues may provide valuable insight into their successful solution.

For classes B2 and above, the modeling effort must be based on accepted principles of mathematical exposition and reasoning. This is because formal models and mathematical proofs are required. The chosen mathematical formalism must allow for an accurate description of the security model and should provide general mechanisms for its analysis. This is necessary so that specific interactions among the entities of the model can be verified as being appropriate.

In practice, needed mathematical techniques are usually adapted from previous security modeling efforts because security modeling, per se, is generally much easier than the development of new mathematical techniques. Extensive use of past modeling techniques is especially feasible in the case of fairly general and familiar policy requirements. System-dependent modeling is generally required for policy enforcement, but established techniques are often available for demonstrating consistency with the modeled requirements.

If new mathematical techniques are used, their credibility is best established by exposure to critical review, in order to uncover possible errors in the mathematics and its intended use. This review process is facilitated by the development of comparative results that give useful relationships between new models and old ones.

2.3.5 STEPS: DEMONSTRATE CONSISTENCY AND CORRECTNESS

Since the security provided by a system is largely determined by its security model, it is important that the model capture needed security requirements and that its rules of operation show how to enforce these requirements.

The first crucial step of identifying security requirements on the system interface cannot be directly validated by mathematical techniques. [cf MCLE85] Human review is needed to establish what security requirements need to be addressed and whether these requirements are reflected in later mathematical formalizations. For systems in class B2 and above, real-world interpretations are assigned to key constructs of the formal model in order to provide a common semantic framework for comparing the model and the security policy.

The appropriate technique for validating requirements on controlled entities (Step 2) depends partly on the novelty of the approach. Informal human review is required to assure that the modeled requirements support the original system security policy. Careful comparisons with previous models of the same or related policies may help to show how new modeling techniques relate to previously accepted techniques for handling particular policy concepts. (See, for example, [MCLE87; MCLE90, Sec. 3 & Sec. 4].) An alternate technique is to give an external-interface model and then mathematically prove that the constraints on system entities imply satisfaction of the external-interface model. The external-interface model is easily compared with a security policy or policy objective. The constraints on system entities are those identified in the requirements portion of the security policy model. This technique is illustrated in Section 3.2.

If the rules of operation are given correctly (Step 3), they will comply with the constraints given in the requirements portion of the model. A *formal proof* of compliance is required for classes B2 and above. In a state-transition model, state invariants are proved by induction: one proves that they hold for the initial state and are preserved by each state transition given in the rules of operation. (See, for example, [CHEH81].) State-transition constraints are straightforwardly proved by showing that each state transition satisfies the constraints.

2.3.6 STEP 6: DEMONSTRATE RELEVANCE

The rules of operation should be a correct abstraction of the implementation. A preliminary model interpretation that is done as part of the modeling process can provide an informal check on whether the rules of operation are sensible and relevant. This model interpretation shows how enforcement mechanisms modeled in the rules of operation are realized by the system. A model interpretation explains what each model entity represents in the system design and shows how each system activity can be understood in terms of the given rules of operation. Thus, for example, a "create file" operation in the system might be explained as involving a restricted use of the rules for "create object," "write object," and/or "set permissions" in the model, depending on the actual arguments to the "create file" command.

An appropriate, somewhat novel, example of a model interpretation is found in the TRUSIX work. [NCSC90b, Sec. 4] The model interpretation is described as an informal mapping from the TRUSIX DTLS to the TRUSIX model. This interpretation first explains how UNIX entities are represented in the model and the DTLS. For example, messages, semaphores, and shared memory are entities that the DTLS refers to as interprocess communication (IPC) objects but that are treated as "files" in the model. Thirty-six UNIX system interface functions are then described by giving pseudocode that shows how each function can be expressed in terms of the state transformations given by the rules of operation. In other words, the transformations given in the rules of operation form a toy security kernel that could be used to implement UNIX if efficiency were irrelevant.

The SCOMP and Multics model interpretations are examples based on *Secure Computer System: Unified Exposition and Multics Interpretation.* [BELL76]. In the SCOMP interpretation, a set of security-relevant kernel calls and trusted processes are identified as `SCOMP rules of operation." [HONE85] For each such SCOMP rule, a brief summary of security-relevant functionality is given and then interpreted as the combined effect of restricted use of one or more rules from the above work by Bell. [BELL76] The specific restrictions are enumerated in an appendix, along with a rationale for omitting some kernel calls and trusted functions from the interpretation. The correspondence between the rules in this work [BELL76] and actual SCOMP behavior is rather complex. The Multics interpretation, by way of contrast, is more sketchy, but the actual correspondence appears to be simpler. [MARG85]

2.4 PRESENTATION FOR EVALUATION AND USE

The most important factors in the successful presentation of a security model are the simplicity of the model, its appropriateness, and an understanding of its intended uses.

The presentation of the model must demonstrate that the model correctly describes the system security policy. A clear explanation of the policy from which the model is derived should be available for this demonstration. Sufficiency of the model may be demonstrated by presenting the relationship of the model to the policy and by carefully explaining and justifying the modeling decisions behind this relationship. In addition to modeled requirements, the system may support other security requirements that are not suitable for inclusion in the model, especially if it is a formal model. Unmodeled security requirements can be presented in an appendix so that TCB developers have all of the security requirements in a single document.

An overview of the model interpretation is needed so that readers can understand the relevance of the system's security model to the security policy and to the overall system-development process. All of these topics may be legitimately covered in a philosophy of protection. In fact, it is highly desirable that the philosophy of protection be the first document delivered by the vendor in a system evaluation, so that it can serve as a basis for further understanding of the security model and other system documentation.

In the case of a formal model, an informal explanation or presentation of the model prepared for a general audience is highly desirable. This is partly for reasons mentioned in Section 2.3.5 and partly because of the general need to acquaint system developers, implementors, and end users with the basic security principles that are to be supported by the system. The informal presentation can reasonably be included in the philosophy of protection.

A well-written explanation of the model's definition of security can be used by potential buyers of a secure system to determine compatibility between the computing system and their organization's security policies and needs. To evaluate the ability of a computing system to support these policies, potential users may wish to construct an informal model of their security policies and compare it to the definition of security supported by the computing system. This use of the security model and the fact that some aspects of security modeling can only be validated by social review suggest that portions of the security model and the philosophy of protection be made available to a relatively wide audience, for example, treating them as publicly released, nonproprietary documents.

For systems at the B2 level or above, the requirement of mathematical proof necessitates the presentation of a rigorous mathematical formalization as well. For a mathematical audience, standards of precision may preclude a style that will be appropriate for a general audience. In this case, an informal presentation of the model is also needed in order to reach the general audience.

At the A1 level of assurance, the formal model is directly involved in the formal verification of the FTLS. As a result, it may be appropriate to present the model in the formal specification language of an endorsed verification system (see Appendix B). Authors and readers must be fully aware of the nuances of the descriptive notation of the verification system used in order to avoid errors due to discrepancies between author intent and reader expectation. Furthermore, if a formal verification system is used to check proofs, it is necessary to achieve a correct translation of what is to be proved into the language of the verification system. Further discussion of these points may be found in [FARM86; NCSC90b, Sec. 6.3].

3. SECURITY MODELING TECHNIQUES

Having introduced the major topics involved in security modeling, we now consider detailed modeling issues with emphasis on mathematical structure as well as empirical justification. In this section, a review of basic concepts is followed by discussions on the modeling of various kinds of policies and objectives: nondisclosure, need-to-know, integrity, and special-purpose policies of particular TCB subjects.

In most cases, modeling techniques are introduced, with details handled via references to the available literature. Inclusion of these references does not imply NCSC endorsement of referenced products incorporating the techniques discussed.

3.1 BASIC CONCEPTS

As discussed in Section 2, the identification of controlled entities plays a crucial role in the development of a security policy model. At B2 and above, the controlled entities must include all system resources. If the policy is multifaceted, with separate subpolicies for mandatory and discretionary access controls, it may be acceptable to decompose the system into different sets of controlled entities for different subpolicies. A secure computing system may decompose into data structures, processes, information about users, I/O devices, and security attributes for controlled entities. In general, the number of different kinds of entities depends on what security-relevant distinctions are to be made. The following paragraphs discuss how to perform such a decomposition for typical kinds of entities, with the goal of modeling security requirements in such a way as to allow an accurate, useful model interpretation.

An *explicitly controlled entry* is one that has explicitly associated security attributes. In the TRUSIX model, for example, the explicitly controlled entities are referred to as “elements”. They consist of subjects and three kinds of objects; files, directories, and entries (i.e., links). [NCSC90b, Sec. 6.9-6.11] In addition to explicitly controlled entities, a system will have implicitly controlled entities. Such an entity might be contained in an explicitly controlled entity or might be a composite entity composed of explicitly controlled entities having potentially different security attributes.

The discussion of security attributes in this section emphasizes security levels used for mandatory access control because the *TCSEC* labeling requirements apply primarily to mandatory access control and auditing. However, much of what is said about MAC labels applies to other kinds of security attributes including, for example, discretionary access control lists.

It is necessary to model creation and destruction of most kinds of controlled entities. A simpler model results if the same approach is used in all cases. In a formal model, one can model the creation of new controlled entities by modeling changes in the set of all controlled entities. A fixed set of possible controlled entities, together with an indication of which entities are available for use in a particular state (e.g., which subjects are active, which objects or object-ids are allocated) can also be used. [cf NCSC90b]

3.1.1 DATA STRUCTURES AND STORAGE OBJECTS

In this guideline, a data structure[4] is a repository for data with an internal state, or value, that can be written (i.e., changed) and/or read (i.e., observed) by processes (and, possibly, devices) using well defined operations available to them. A data structure is distinct from its value at a particular time, which is, in turn, distinct from the information conveyed by that value. The information content depends not only on the value but also on how it is interpreted. Similarly, a data structure is distinct from a name for that data structure.

A data structure that is explicitly associated with exactly one security level by the security policy model is a *storage object*.[5] There are no a *priori* restrictions on the level of abstraction at which data structures and storage objects are modeled. In particular, objects may be abstract data structures that are accessed using high-level operations provided by an *encapsulation mechanism*. In this case, a user would have access to the high-level operations but could not access the encapsulated data directly via more concrete operations that may have been used to define the high-level operations. This lack of direct access would have to be enforced by the TCB.

Ordinarily, the storage objects in a security model are *disjoint*. This means that, in principle, changes to one object do not force changes to other objects. If a state-machine model is used, disjoint storage objects can be modeled as separate components of the underlying machine state. If two objects at different security levels were not disjoint, then access to data in their intersection would be allowed or denied according to which label is used. This ambiguity can be resolved by explicitly associating a level with their intersection. In this case, the associated level allows the intersection to also be regarded as a separate storage object, thereby restoring disjointness. Thus, disjointness in storage objects simplifies one's understanding of how security labels are used to control access. It has also been argued that disjointness simplifies covert channel analysis via shared resource matrices.

In some systems, collections of objects are combined to form *multilevel data structures* that are assigned their own security levels. Multilevel data structures called "containers" are used in the Secure Military Message System (SMMS) to address *aggregation problems*.[6] [LAND84] The level of a container must dominate the levels of any objects or containers inside it.

When an object or other controlled entity is created, it must be assigned security attributes and initialized in such a way as to satisfy the object reuse requirement. The modeling of object reuse policies (e.g., objects are erased when allocated) can be useful, but object reuse is not found in traditional *access control models* because the object reuse requirement deals with the content of objects and TCB data structures. The initialization of security attributes, however, plays an essential role in access control and can be modeled by distinguishing between "active" and "inactive" entities, as in [NCSC90b]. The security attributes of a newly created object may be taken from, or assigned by, the subject that created it. In many systems, creating an object is essentially a special case of writing to it: its value changes from undefined to null. If this change is visible to

[4]The need to distinguish between data structures and objects is not seen in the *TCEC* Glossary definitions but has arisen more recently from issues involving structured objects, multilevel data structures, and possible differences between the storage objects used for MAC and the *named objects* used for DAC.

[5]The need to distinguish between data structures and objects is not seen in the *TCEC* Glossary definitions but has arisen more recently from issues involving structured objects, multilevel data structures, and possible differences between the storage objects used for MAC and the *named objects* used for DAC

[6]An aggregation problem can occur when a user has access to individual pieces of data in a data set, but not to the entire data set. This problem is discussed further in Section 4.3.

non-TCB subjects and the new object is created by a non-TCB subject, then the security level of the new object must dominate that of the creating subject. DAC attributes, by way of contrast, are usually given by system- or user-supplied defaults.

3.1.2 PROCESSES AND SUBJECTS

A process may create, destroy, and interact with storage objects and other processes, and it may interact with I/O devices. It has an associated run-time environment and is distinct from the program or code which defines it. For instance, the program would ordinarily be modeled as information contained in a storage object. An explicitly controlled process is a subject. It normally has a variety of associated security attributes including a security level, hardware security attributes such as its process domain, the user and user group on whose behalf it is executing, indications of whether it belongs to the TCB, and indications of whether it is exempt from certain access control checks.

The issues regarding disjointness of storage objects mentioned in Section 3.1.1 apply to subjects as well. If two subjects share memory, it is advisable to model the shared memory as an object separate from either subject in order to achieve the disjointness needed for information-flow analysis. Local memory need not be separately modeled, but must be properly taken into account when modeling the subject. Registers (including the instruction pointer) are technically shared memory but can often be treated abstractly as unshared local memory. This is especially true if registers are saved and restored during process swaps and the information they contain is not shared between processes.

Security levels for processes are ordinarily similar to security levels for storage objects. The *TCSEC* requires that a subject have a nondisclosure level that is dominated by the clearance and authorization of its user. A TCB subject may reasonably be assigned separate levels for reading and writing in order to allow partial exemption from access control (see Section 3.4). Other TCB-related entities may also need separate levels for reading and writing in some systems. One such example is /dev/null in UNIX. [GLIG86] This object is system high in the sense that any process can write to it; but it is system low in the sense that any process can read from it because its value is always null.

In general, the security-relevant *attributes* of a subject are part of its run-time environment. Some attributes are relatively static and are inherited from the subject's executable code or are stored with its code. In UNIX, the property of being a set-user-id program is a statically determined attribute, as is the effective user-id. Other attributes are dynamically assigned and are inherited from a subject's parent or are actively assigned by the parent (assuming it has a parent process). In the latter case, the parent must be relied upon to assign the child's security attributes appropriately. As a result the parent usually belongs to the TCB in this latter case. A secure terminal server, for example, might create a subject and assign it security attributes based on the determined identity of a user logging in.

There are potential difficulties in associating user-ids with TCB subjects. A terminal server would appear to operate on behalf of the user currently logged in or on behalf of the Identification and Authentication mechanism if no one was logged in. A print spooler acts on behalf of the users in its print queue. Such TCB subjects are, in reality, associated with multiple user-ids, a fact which is relevant to the design of the audit mechanism. A TCB subject that does not act on behalf of a given user can be handled either as having a dynamically modifiable user-id or as acting on behalf of the system (equivalently, as acting on behalf of a system-defined pseudo-user).

When a subject creates a child subject by executing a program, the resulting child might belong to the TCB and be exempt from certain access control checks, even if the original subject was not in the TCB. This is possible if exemptions are associated with the program object itself.

With the advent of multitasking languages such as Ada, there is a question of when two threads of control belong to the same subject. Pragmatically, to be the same subject, they should have the same security attributes. To qualify as separate subjects, their separation should be enforceable by the TCB.[7] Finally, nothing explicitly prohibits a multithreaded process from consisting of several subjects operating at different security levels. Of course, intraprocess communication will be somewhat limited for such a process unless it contains TCB subjects that are exempt from some of the MAC constraints.

3.1.3 USERS AND USER ROLES

User-related topics often turn up in security models in spite of the fact that users are not controlled entities and thus do not need to be directly modeled. The users of a system may perform specific *user roles* by executing associated *role-support programs*. In general, a user may engage in a combination of several roles. As a matter of policy, a given user role may require system-mediated authorization and may provide specific system resources needed for performance of the role. Although the following paragraphs discuss only individual user roles, most of the basic concepts extend to roles for other kinds of *clients* in the environment of a computing system including user groups and, in the case of a network, hosts or subjects running on hosts.

The *TCSEC* requires support for certain *trusted user roles.* For systems at B2 and above, these roles include the *security administrator* role and the *system operator* role. The administrator role governs security attributes of users, moderates discretionary and mandatory access control, and interprets audit data. The operator role ensures provision of service and performs accounting activities. [NCSC90a] These roles may be subdivided so that they can be shared by multiple users. In addition, they do not preclude the addition of other trusted roles. In general, trusted user roles are characterized by the fact that they involve handling security-critical information in a way that violates access restrictions placed on non-TCB subjects. As a result, processes that support trusted roles can be misused and must have restricted access. By definition, such *trusted role processes* have security properties that are unacceptable without special constraints on the behavior of their users. These observations suggest (but do not mandate) the modeling of support for user roles, especially trusted user roles. Trusted role processes are usually treated as TCB subjects. Additional information on modeling them is given in Section 3.4.

Instructive examples of role definitions may be found in Secure Xenix [GLIG86] and the SMMS security model [LAND84]. Secure Xenix restructured the Guru role into four separate roles. The SMMS security model included a system security officer role, a downgrader role and a releaser role. In both of these systems, the roles support *separation of duty* in that each role has

---

[7]The *TCSEC* glossary mentions two uses of the word “subject,” the more general use from [ANDIE72] and a less general use from Multics. In Multics, a process and, hence, a subject is single-threaded. However, legitimate examples of multithreaded subjects have turned up in product evaluations, and their multithreadiness has been considered acceptable by the NCSC. Multithreaded processes are often referred to as process *families*.

privileges not available to the other roles. Separation of duty has been emphasized by Clark and Wilson, who have given a general scheme for constraining user roles by using triples of the form (user, program, object-list) in order to control how and whether each user may access a given collection of objects. [CLAR87] A similar role-enforcement mechanism is supported by *type enforcement.* [BOEB85] The type-enforcement mechanism itself assigns “types” to both subjects and objects; subject types are referred to as “domains.” The accesses of each subject are constrained by a “type table” based on subject domain and object type. Each user has a set of domains associated with subjects running on his behalf. These domains are used to define user roles. [cf THOM90]

3.1.4 I/O DEVICES

Although users are external to the computing system and need not be directly modeled, it is still useful to model their allowed interactions with the system in order to document security-relevant constraints that are enforced by the system. User interactions may be modeled in terms of constraints on I/O devices if there is a convention for discussing the current user of a device. Devices which are accessible to subjects outside the TCB should be modeled either implicitly or explicitly as part of the interface between the non-TCB subjects and the TCB. An additional reason for interest in I/O devices is that I/O typically accounts for a large portion of the TCB.[8] The following paragraphs present general information on the modeling of devices with emphasis on device security requirements and then discuss when they should be modeled as objects, as subjects, or as their own kind of entity.

Normally, security models discuss abstract encapsulated devices rather than actual physical devices. An *encapsulated device* is an abstract representation of a device, its associated device driver, and possibly other associated entities such as attached hardware and dedicated device buffers. Thus, for example, one might discuss a line printer connected to the system but not the actual RS-232 device used to achieve the connection or its associated device driver.

By definition, an input device injects information into the system in a way that the system cannot entirely control. The next state of the system may depend on the actual input as well as on the current state and the particular state transformation being executed. This fact can be accommodated in several ways in an FTLS or a model that addresses object content. One can treat the actual input as a parameter to the transformation. If there are only a few different input values, one can associate different transformations with different inputs. Finally, one can treat the transformation as being *nondeterministic,* meaning that several different states can result from executing the transformation in a given state.

Abstract encapsulated devices are often passive entities, in contrast to their underlying hardware. Security requirements for devices, however, differ significantly from those for either storage objects or controlled processes:

- External policy on use of the system requires that devices pass information only to authorized users.

[8] I/O handling often accounts for 30 percent or more of an entire operating system. [TANE87, Preface]

- Devices may transport either unlabeled data or labeled data and are classified as single *level* or *multilevel devices* accordingly.
- At B2 and above, the *TCSEC* requires that every device have a minimum and maximum device level that represents constraints imposed by the physical environment in which the device is located.

Authorized use of a device may be enforced by requiring that any piece of information output by the device have a security level that is dominated by the clearance and authorization of the recipient. A combination of procedural and automated methods are needed to correctly associate the security level of the information, the user who will receive that information, and the security level of that user. Typically, the device itself will also have a security level. The user who receives information from a device may well be different than the user who sent that information to the device. If a device with local memory (an intelligent terminal, for example) is allocated to different users or different security levels at different times, it will typically need to be reset between users in order to meet requirements on authorized transmission of information.

Both single-level and multilevel devices may handle multiple levels of information. A single-level device can only handle a single level at a time but may have some convention for changing levels. A multilevel device, by way of contrast, can handle different levels of data without altering its security designation, but still might be used in such a way as to carry data at only one fixed security level.

The *TCSEC* requirement for device ranges does not explain how they are to be used. The most common option is to require that the level of any object transmitted by the device be within the range. Another option is to require that the security clearance of any user be within the range. Separate decisions regarding device minimum and device maximum are also possible. In the Trusted Xenix system,[9] a user with a secret clearance can have an unclassified session on a terminal with a secret device minimum. The intended interpretation of the device minimum is that the device is in a restricted area that should contain only users whose clearance is at least the device minimum. If the device minimum is secret, then a login attempt by an uncleared user is treated as a violation of physical security. [GLIG86]

The question of whether devices should be modeled in the same way as other kinds of controlled entities depends on the complexity of the devices involved, observed similarities between devices and other modeled entities, and goals relating to the simplicity and accuracy of the model. The TRUSIX group decided to model devices in the same way as file objects, a decision that depended heavily on both the nature of UNIX and the specific goals of the TRUSIX effort. [NCSC90b, § 6.9] The SMMS model treats devices as containers in order to capture the fact that the device maximum must dominate the level of all data handled by the device. [LAND84] Yet another alternative is to model devices by modeling their device drivers as TCB subjects. In general, more sophisticated I/O devices need greater modeling support, with the limiting case being intelligent workstations modeled as autonomous computing systems.

3.1.5 SECURITY ATTRIBUTES

---

[9]Trusted Xenix" is a trademark of Trusted Information Systems. Trusted Xenix was formerly known as Secure Xenix, which is the name found in [GLlG86].

A security attribute is any piece of information that may be associated with a controlled entity or user for the purpose of implementing a security policy. The attributes of controlled entities may be implicit; they need not be directly implemented in data structures. Labels on a multilevel tape, for example, can be stored separately from the objects they label, provided there is an assured method of determining the level of each object on the tape. [cf NCSC88b, C1-C1-05-84]

Primary attributes of security policies that are usefully reflected in the security model include locus of policy enforcement, strength and purpose of the policy, granularity of user designations, and locus of administrative authority. These policy aspects lead to a taxonomy of policies and security attributes.

There are several types of security attributes of any given system: informational attributes, access control attributes, nondisclosure attributes, and integrity attributes. *Informational attributes* are maintained for use outside the given computing system, whereas *access control attributes* limit access to system resources and the information they contain. Purely informational attributes are somewhat uncommon; an informative example is given in Section 4.4. Access control attributes may be classified according to what they control. A *loose access control attribute* controls access to the entity it is associated with, whereas a *tight access control attribute* also controls access to information contained in that entity. Thus, access restrictions determined by tight attributes must propagate from one object to another when (or before) information is transferred, because control over the information must still be maintained after it leaves the original entity. *Nondisclosure attributes* are used to prevent unauthorized release of information, whereas *integrity attributes* are used to prevent unauthorized modification or destruction of information.

Attributes can also be classified by the granularity and authority of their control. The *user granularity* of an attribute may be "coarse," controlling access on the basis of broadly defined classes of users, or it can be "per-user," controlling access by individual users and processes acting on their behalf. Finally, *centralized authority* implies that policy for use of attributes is predefined and takes place under the control of a system security administrator, whereas *distributed authority* implies that attributes are set by individual users for entities under their control. This classification of security attributes is based partly on the work of Abrams. [ABRA90]

When applied to typical security policies for B2 systems, these distinctions among security attributes might take the form given in Figure 3.1. MAC policies must enforce nondisclosure constraints on information and may enforce integrity constraints as well. They must regulate access to information on the basis of user clearance and labeled sensitivity of data. The involvement of user clearances typically comes at the expense of per-user granularity because several users are likely to have the same clearance. Authority to assign security attributes is usually somewhat centralized. Users can create entities at various levels, but ability to change the levels of existing entities is usually restricted to authorized users.

DAC policies must enforce access constraints on both reading and writing. They must provide per-user granularity as criteria for access decisions, although they often include a group mechanism for coarse gained access decisions as well. DAC policies are normally used to enforce both nondisclosure and integrity, but constraints on access to named objects do not necessarily imply corresponding constraints on access to information in those objects. Authority to change discretionary attributes is usually distributed among users on the basis of ownership.

	MAC	*DAC*

Binding Strength:	Tight	Loose[10]
Purpose:	Nondisclosure[10]	Nondisclosure & Integrity
User Granularity:	Coarse	Per-user
Authority	Centralized	Distributed[10]

Figure 3.1. Typical Classification of Access Control Policies

3.1.6 PARTIALLY ORDERED SCURITY ATTRIBUTES

A security attribute belonging to a partially ordered set may be referred to as a level. The associated partial ordering may be referred to as the dominance relation; in symbols, *L_1 is dominated by L2* if and only if $L1 \leq L2$. The use of partially ordered levels is usually associated with tight access control policies and constraints on information flow. Constraints on information flow can arise for several different reasons. As a result a multifaceted policy might have a different partial ordering for each reason. Often, the combined effect of constraints associated with several partial orderings can be expressed in terms of a single composite ordering. In this case, facts about Cartesian products of partially ordered sets given in Appendix A may be used to simplify the formal model and, possibly, the system's implementation as well. The following paragraphs discuss the connection between levels and constraints on information flow, the use of these constraints for supporting nondisclosure and integrity objectives, and the tailoring of level-based policies to particular applications.

A dominance relation is often viewed as an abstraction of allowed information flows: information can flow from entity E^1 to entity E^2 only if the level of E^1 is dominated by the level of E^2. This rather general view, which is an analogue of the original *-property of Bell and La Padula, allows the illustration of some basic issues in the use of levels, but it is overly simple in some respects. It does not address whether information flows properly or whether the flow is direct or indirect. Moreover, this view does not contain any explicit assumption about why information may be prevented from flowing from one entity to another.

Partially ordered levels and related constraints on information flow have been suggested for use in enforcing both nondisclosure and integrity objectives. The motivation for these suggestions may be understood from the following considerations. Suppose that access controls could enforce the above information-flow constraints perfectly. Suppose that the two objects contain the same information, but one is labeled at a higher level than the other. In this situation, information is less likely to be disclosed from the higher-level object because fewer subjects have a sufficiently high level to receive this information. Conversely, if lower-level subjects can write higher-level objects, then inappropriate modification of the lower-level object is less likely because fewer subjects have a sufficiently low level to write to it. This dual relationship may cause the goals of nondisclosure and integrity to conflict, especially if the ordering on levels is strongly hierarchical. As explained in Section 3.5, this duality between nondisclosure and integrity has some significant limitations. A given set of security levels designed for enforcing nondisclosure may be in appropriate for

[10]The MAC policy enforced by a system may be useful for integrity as well as nondisclosure. DAC policies are not required to be either loose or distributed.

enforcing integrity, and not all integrity policies use partially ordered security attributes.

When a computing system is accredited for use at a particular site, it is assigned a *host accreditation range* — a set of levels at which the host may store, process, and transmit data.[11] A host range can prevent the aggregation of certain classes of data. If several objects have different levels and no level in the host range dominates all of them, then there is no legitimate way of concatenating these objects. This use of host ranges is discussed further in Section 4.4.

A desirable constraint holds between a host range and device ranges: if the host range contains a level that does not belong to any device range, a non-TCB subject running at that level can not communicate bidirectionally without using covert channels. By way of contrast, a device range may reasonably contain levels not in the host range. Suppose, for example, that A and B are incomparable levels dominated by a level C that does not belong to the host range. A device might reasonably use C as a maximum device level in order to allow use of the device at either level A or level B. But, in this case, the system would need to check that each object input from the device actually belonged to the host range.

3.1.7 NONDISCLOSURE LEVELS

The following paragraphs discuss the structure of nondisclosure levels as it relates to their abstract mathematical properties, to *TCSEC* requirements, and to their intended use. Analyses given in the following paragraphs suggest that the structure of nondisclosure levels that are used to enforce nondisclosure policies is often not needed for modeling purposes. If their structure plays no significant role in a given model, their inclusion is unnecessary and may limit the applicability of the model.

The *TCSEC* requires that nondisclosure levels contain a classification component and a category component. The hierarchical "classification" component is chosen from a linearly ordered set and the nonhierarchical "category" component must belong to a set of the form **(C)*, the set of all subsets of *C,* for some set C of categories. [cf NCSC85, Sec. 9.0, Sec. 3.1.1.4] In some applications, thousands of categories may be needed. In others, only the four clearance levels "unclassified", "confidential", "secret", and "top secret" are needed, as a result of Executive Order 12356. [REAG82] These facts illustrate the importance of configuration-dependent host accreditation ranges to remove unused security levels.

As explained in Appendix A, any partially ordered set may be fully embedded in one based on categories. As a result, *TCSEC* constraints on nondisclosure levels do not restrict the class of partially ordered sets that may be used for such levels (although these constraints do affect the use of human-readable names). The decomposition of levels into classification and category components can be configuration-dependent and may be given along with the host accreditation range. This fact is of some interest for computing systems with both commercial and military applications. [cf BELL90] In this case, the model should be general enough to embrace all intended configurations.

---

[11] Several different kinds of accreditation ranges may be associated with host on a network. For example, a network interface accreditation range is a device range which restricts the levels that may be used in connection with a given network interface.

Even if labels are explicitly assumed to contain classification and category components, nothing in the *TCSEC* prevents the addition of vendor-supplied components because such additions do not invalidate the mandated access checks. Thus, for example, each label could contain a release date after which the contained information will be valueless and, therefore, unclassified. If release dates are chronologically ordered, later dates would represent a higher nondisclosure level. The initial specification of release dates, like that of other nondisclosure attributes, needs to be handled by trusted path software. The regrading from classified -outdated to unclassified would be carried out by trusted software in conformance with accepted downgrading requirements.

A nondisclosure level is, by definition, commensurate with the level of harm that could result from unauthorized disclosure, and it should also be commensurate with the level of assurance against unauthorized disclosure that is to be found in the system's TCB and surrounding physical environment. Various rules of thumb have been worked out to correlate risk with levels of nondisclosure and assurance. [DOD88a, NCSC85a] While these relationships are not likely to show up in the security model itself, they may affect the kinds of security attributes that are included in the nondisclosure level.

3.1.8 UNLABELED ENTITIES AND THE TRUSTED COMPUTING BASE

In addition to controlled entities, there are system resources that either are not user accessible or are part of the mechanism by which user-accessible resources are controlled. These latter resources are the software, hardware, and internal data structures which make up the TCB. At higher levels of assurance, significant effort goes into minimizing the size and complexity of the TCB in order to reduce the overall effort needed to validate its correct operation.

The TCB typically consists of the implemented access control mechanism and other entities that must be protected in order to maintain the overall security of the system. A TCB process which is not part of the access control mechanism may be assigned security attributes and controlled as a subject in order to help enforce the principle of least privilege within the TCB. Conversely, certain subjects may need to be included within the TCB because they assign security levels to input, support trusted user roles, transform data in such a way as to legitimately alter the data security level, or perform some other security-critical function.[12] A data structure may also be security-critical, as when it contains user-authentication data, a portion of the audit trail, audit-control data, or security attributes of controlled entities. It is not always necessary to assign security attributes to TCB processes and *security-critical* data structures, but this is often done to enforce the principle of least privilege within the TCB and to regulate access by non-TCB subjects to security critical data structures.

If a piece of data can be accessed as a direct effect of a system call (i.e. access directly specified in a parameter) then it must be accounted for in the interpretation of controlled entities in such away as to satisfy MAC requirements. But some data structures may not be directly accessible. Possible examples include security labels, the current access matrix, internal object names that are not accessible to users of the system, and transient information related to access

---

[12]Subjects belonging to the TCB are often referred to "as trusted," but in some contexts, this word is used more narrowly, referring to TCB subjects that are exempt from certain constraints enforced by the reference monitor, or even more narrowly, for subjects that are exempt from the *-property. [cf SALT75, SCHE88, BELL76]

control, opening of files, and so forth. A data structure which is not directly accessible does not have to be labeled. A decision to supply a label may complicate the modeling process, whereas a decision not to supply a label may increase the difficulty of the covert channel analysis.

While there is no explicit requirement to model the TCB, the model must capture security requirements imposed by the *TCSEC*, including reference monitor requirements relating to non-TCB subjects and the entities they manipulate. [cf NCSC87, Appendices B.3.4, B.7.1] A possible approach to modeling these reference monitor requirements is discussed in Section 3.2.4. If significant aspects of the system security policy are embodied in TCB subjects that are exempt from modeled access constraints on non-TCB subjects, then exempt subject modeling is also needed. This topic is discussed further in Section 3.4.

3.2 NONDISCLOSURE AND MANDATORY ACCESS CONTROL

How can nondisclosure requirements be accommodated in a model's definition of security? To what extent can access control succeed in enforcing nondisclosure? What impact do nondisclosure and access control requirements have on trusted systems design? The first of these questions is customarily addressed by imposing access constraints on controlled entities (e.g., the *-property). But various research efforts have contributed additional approaches that provide a useful context in which to explain how access constraints can support nondisclosure objectives. Both traditional and newer research-related approaches are discussed below. The ordering is top-down, beginning with nondisclosure requirements on the external system interface, as in the modeling paradigm described in Section 2.3.

Section 3.2.1 shows how nondisclosure requirements can be formalized in an external-interface model. In Section 3.2.2, *external-interface requirements* are elaborated to obtain an *information-flow model,* in order to facilitate later analysis. Section 3.2.3 introduces the reference monitor interface and applies the information-flow requirements to individual subject instructions. Section 3.2.4 presents an access-constraint model that ensures satisfaction of the information-flow requirements at the reference monitor interface. Finally, Section 3.2.5 only briefly discusses rules of operation because of their system-specific nature.

Each of the three models presented in Sections 3.2.1, 3.2.2, and 3.2.4 provides adequate conceptual support for nondisclosure requirements found in the *TCSEC* mandatory security objective and could serve as a definition of mandatory security in a security policy model. Adequacy of the access-constraint model, in particular, is established by comparison with the previous two models. Comments regarding the impact of these models on overall system design are distributed among the various subsections, with the third access-constraint model providing the most explicit basis for designing rules of operation and judging correctness of implementation.

These models are very simple and need to be adjusted to accommodate policy variations found in particular trusted systems. These models do not address aggregation and inference. For sake of simplicity, no accommodation is made for trusted processes or discretionary access control. The presented access-constraint model is especially relevant to systems in which all user-initiated computation is performed by subjects outside of the TCB. It may not be adequate for systems whose TCB contains major trusted applications (e.g., a multilevel DBMS implemented as a TCB subject). Finally, the entire analysis assumes that the system to be modeled is *deterministic* in the sense that its behavior is completely determined by its combined input sequence and data initially in the system.

3.2.1 EXTERNAL-INTERFACE REQUIREMENTS AND MODEL

Consider a system where each input or output is labeled with a nondisclosure level. Users work with I/O streams containing items at a given level and provide inputs in such a way that the level of a given input stream accurately reflects the sensitivity of the informatIon it contains. The system creates each output item by extracting information from input items (possibly at several different levels) and affixing an appropriate label. A combination of automated and procedural methods is used to combine input streams into a single input sequence and to separate the resulting combined output sequence into separate output streams at different levels.

The actual nondisclosure requirement is this: outputs labeled with a given level must contain only information whose sensitivity is dominated by that of their label. This requirement is pictured in Figure 3.2, where lighter shadings represent higher information security levels: for level A dominates for level B, which dominates for level C.

This requirement is difficult to model (let alone implement) because it talks about the actual sensitivity of output information, whereas the system is only given the attributed sensitivity of its data. These difficulties can be avoided with the following alternate requirement: a given labeled output must not contain information derived from data whose attributed sensitivity fails to be dominated by the level of the output's label. This alternate requirement applies to both data supplied in the input streams, and data residing in the system itself. This alternate requirement is slightly stronger than the original provided that information at a given level cannot be synthesized by aggregation and inference from information at strictly lower levels. In this case, any classified information in the system either came from the input stream or was already contained in the system when it was installed level .

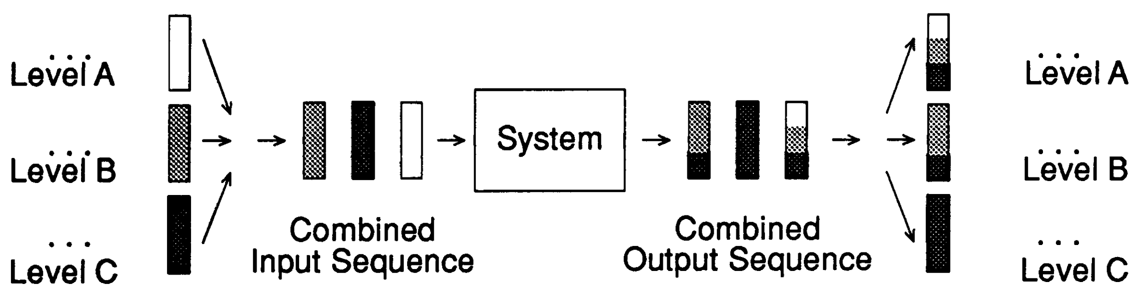

Figure 3.2 Intended Use of a Secure System

In the first external-interface model, each item in the system's input stream is taken to be a labeled value, as is each item in the output stream. There are two "nondisclosure security" requirements which are given relative to an arbitrary level L:

Noninterference:
Any output stream at level *L* remains unchanged when inputs at levels not dominated by *L* are altered or removed.

Nonobservability:
Any output stream at level *L* remains unchanged when data in the system whose sensitivity is not dominated by *L* is altered or removed.

These two terms and the requirements they name are similar; the only distinction is that noninterference discusses independence from high-level input whereas nonobservability discusses independence from high-level data in the system.

As with any mathematical modeling effort, there is a need to specify the physical interpretation of the model. There are also choices to be made as to which aspects of the system's observable behavior are actually addressed by this external-interface model. An "accurate" interpretation of this model is a physical system whose I/O streams and data are related according to the above two requirements. In a "complete" interpretation of this model, the input stream would contain all external influences on the system being modeled; the output stream would contain all observable effects of the system on its environment; and the data in the system would include all data that can influence the value of the output stream. An accurate interpretation can be "useful" without being complete if it includes all outputs associated with normal use as well as enough of the inputs and system state to predict the included outputs. It is standard engineering practice to avoid details that would make the modeling problem intractable and then consider them in a separate covert channel analysis.

The noninterference requirement is due to Goguen and Meseguer. (GOGU82] Under useful interpretations, this requirement rules out resource-exhaustion channels associated with those aspects of the system that are covered by the model's interpretation. Suppose that some system resource (such as memory) is used for processing both low-level and high-level inputs, and that the processing of low-level inputs cannot proceed when that resource is unavailable. In this case, the system may not emit a low-level diagnostic explaining the nature of the problem, since the diagnostic could reveal "interference" by high-level input. The low-level diagnostic, if allowed, would have provided a resource-exhaustion channel.

The nonobservability requirement was developed as part of the LOCK verification effort. It covers situations in which classified data is entered during system configuration using information paths not addressed in the security modeling process. A superficially stronger requirement is that the system initially contains no classified information. This stronger requirement occurs implicitly in the original noninterference models of Goguen and Meseguer. [GOGU82, GOGU84] With this stronger requirement, useful physical interpretations would need to include any classified inputs that occurred during the construction, installation, or booting of the system.

3.2.2 INFORMATION-FLOW MODEL

The information-flow model is discussed next because of its close relationship to noninterference. The requirements of this model are motivated by an informal view of how information flows through a deterministic state-machine system. The possible paths of information flow during a state transition are depicted as arrows in Figure 3.3, where *I, O,* and *S* abbreviate input, output, and state, respectively

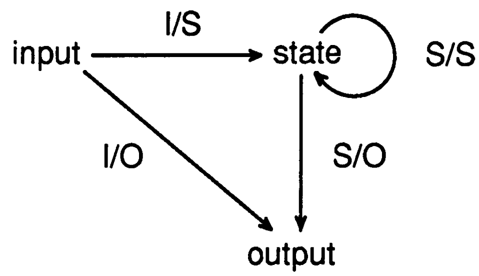

Figure 3.3. Information Flows in a Deterministic State Machine

Each input is assumed to induce a state transition and a (possibly empty) sequence of labeled outputs. As indicated in the above diagram, there are just four possible flows: directly from input to output, from an input to the next system state, from a given state to the next state, and from a given state to an output. Correspondingly, there are four flow-security requirements for the information-flow model that must hold for any nondisclosure level L. They refer to "portions" of the state, meaning collections of variables (i.e., state components):

I/O Security:
An output at level L can only be induced by an input whose level is dominated by L.

I/S Security:
An input at level L can affect only those portions of the system state whose levels dominate L.

S/O Security:
An output at level L can depend only on those portions of the system state whose levels are dominated by L.

S/S Security:
A portion of the state which is at level L can affect only those portions of the state whose levels dominate L.

To see the effect of these requirements, suppose, for example, that the current time is maintained as a state component that is implicitly incriminated by every input instruction according to the time needed for its execution. The I/S security property implies that the clock level must dominate every level in the input stream because every input affects this particular state component. This observation does not imply that all "system" clocks have to be system high, however. One possibility is to partition time into several disjoint "slices." This partitioning effectively creates a virtual clock for each time slice. Processes running in a given time slice affect only the virtual clock associated with that time slice, so that the level of the virtual clock need only dominate the levels of these processes. Consequently, a process that reads the virtual clock for its time slice must have a level that dominates those of other processes running in that time slice.

The four requirements of the information-flow model imply those of the nondisclosure-requirements model. This fact is an example of an "unwinding" theorem in that it shows how to recast nondisclosure in terms of individual inputs and state transitions. An informal justification of this fact can be given as follows. First, consider the nonobservability requirement: information at a given level L in the output stream cannot come directly from portions of the state not dominated by L (by the S/O security property), and it cannot come indirectly from a preceding state transition (by the S/S security property). Now consider noninterference: information in the output stream at a given level L cannot come directly from inputs not dominated by L (by the I/O security property), and it cannot come indirectly via an intermediate system state as a result of the I/S security and nonobservability properties.

All four of the flow-security requirements are necessary for noninterference. Moreover, there is a partial converse: in the case of systems that contain "print" commands capable of outputting arbitrary state components, the four flow-security requirements follow from the nondisclosure requirements.

This information-flow model is similar to those of Feiertag, Levitt, and Robinson [FEIE77] but it makes some minor improvements: an input that causes a state change may also induce output; a given input may induce outputs at several different levels; and it is not required that each input be associated with an identified user. The correctness of this model with respect to the nondisclosure-requirements model is formally proven in [WILL91]. The result is a variant of the unwinding theorem of Goguen and Meseguer. [GOGU84, RUSH85] An exposition of Rushby's treatment of the unwinding theorem can be found in the article "Models of Multi level Security." [MILL89]

3.2.3 APPLICATION TO THE REFERENCE MONITOR INTERFACE

Security policy models traditionally emphasize the reference monitor interface and cover the processing of subject instructions but ignore issues associated with the sequencing of subject instructions and their synchronization with external inputs. These ignored issues are handled separately via covert channel analysis. This is due to the lack of general, well-developed modeling techniques for dealing with time, concurrence, and synchronization. Subject-instruction processing, by way of contrast, is readily modeled. In particular, the above external-interface and information-flow models can be used for subject-instruction processing, just by giving them a new physical interpretation.

Under this new interpretation, the inputs of the state machine are the instructions that subjects execute, including system traps whose semantics are defined by the TCB's kernel-calI software. External system inputs are also included in the case of "input instructions." Each instruction is executed to produce a state change and zero or more outputs. The outputs include external system outputs and, possibly, feedback to the unmodeled instruction-sequencing mechanism.

Notice that subjects do not literally execute instructions (since they are untrusted). The TCB itself executes each subject instruction on behalf of an associated subject controlled by the TCB. In particular, each hardware instruction available to processes outside of the TCB (i.e., each "user-mode" instruction) is executed by the CPU, which is part of the TCB. To ensure nondisclosure security, all subject instructions, including user-mode hardware instructions, must satisfy the reinterpreted model requirements, either by virtue of the hardware design or by enforced restrictions on their use. For example, security may be violated if a subject can follow a kernel call with a "branch-to-previous C ontext" instruction that inadvertently restores all of the access privileges used during the processing of that kernel call. Instructions implemented by the hardware but not used by compilers should be modeled if the compilers are bypassable. In the unusual case where non-TCB subjects can directly execute microcode instructions, these too need to be modeled.

The actual accesses of a given subject to various objects in its environment are determined by the instructions it executes (or attempts to execute). Therefore, it is this stream of subject instructions that the reference validation mechanism must mediate in order to carry out the recommendations of the Anderson Report. [ANDE72] The application of noninterference and information flow to subject-instruction processing dates back to [FEIE77] where a form of noninterference is referred to as "property P1." The validation of noninterference as applied to LOCK subject-instruction processing is presented in [HAIG87a; FINE90]. Keefe and Tsai have adapted noninterference for use in modeling DBMS schedulers. [KEEF90] It can be shown that information flow for subject-instruction processing, together with a variant of noninterference for subject-instruction sequencing, implies noninterference and nonobservability for the entire system. [WlLL91]

3.2.4 ACCESS--CONSTRAINT MODEL

An access-constraint model can be obtained by expanding the information-flow model of instruction processing to include traditional notions of access control, including subjects, objects, a current-access matrix, and access constraints. This is not a complete access control model in the traditional sense because it lacks rules of operation. It is a definition of mandatory security for instruction processing; it does not show how access constraints are actually enforced.

The access-constraint model assumes that the instruction processing state is made up of labeled state components called objects. The model does not explicitly assume that subjects are controlled processes, but it does assume that every computation involving either access to objects or output has an associated subject. Each subject has a nondisclosure level and is assumed to include its local data space (including stack, program counter, and so forth). Consequently, each subject is also considered to be an object that could passively be acted on by other subjects. The system state, *st,* contains a "current-access matrix," A_{st} (s, o), that associates each subject-object pair with a set of "modes." For simplicity, the possible modes are taken to be just "observe" and "modify."

The requirements of the access-constraint model fall into three groups: traditional requirements, constraints on the semantics of observation and modification, and I/O requirements. These requirements are formulated for systems with file-like objects that are opened (in accordance with simple security and the *-property), accessed for a period of time (in accordance with the observe-semantics and modify-semantics requirements), and then closed. Each of the eight requirements must hold in every reachable state.

Simple Security:

A subject may have observe access to an object only if its level dominates that of the object.

***—Property:**

A subject may have modify access to an object only if its level is dominated by that of the object.

Tranquility:

The level of a given subject or object is the same in every reachable state.

Variants of simple security and the *-property are found in virtually all mandatory security models. The tranquility requirement can be weakened without compromising the mandatory security objective, but possibly at the expense of a more complicated model. [cf MCLE88]

The requirements which constrain the semantics of reading and writing are a major factor in deciding what checks must appear in the rules of operation. Other major factors include the actual system design and the degree of detail needed in the model.

Observe Semantics:

A subject that executes an instruction whose behavior depends on the value of some state component must have observe access to that state component.

Modify Semantics:

A subject that executes an instruction which modifies a given state component must have modify access to that state component.

The observe-semantics requirement is slightly stronger than necessary. A subject (e.g., a mail program) that knows the existence of two objects at a higher level might reasonably cause information to be transferred from one to the other by means of a "blind" copy instruction, but this is directly ruled out by the observe-semantics requirement. As noted by McLean [MCLE90, Sec. 4], Haigh's analysis contains a similar restriction. [HAIG84] A very careful treatment of what constitutes observation may be found in "A Semantics of Read." [MARC86] The use of a semantics of reading and writing may be found also in several other security modeling efforts, including those by Cohen, [COHE77] Popek, [POPE78] and Landwehr [LAND84].

The following requirements constrain allowed associations between subjects and I/O streams. They assume that each input is read on behalf of an associated subject referred to as its "reader."

Reader Identification:

The decision as to which subject is the reader for a given input must be based only on information whose level is dominated by that of the reader.

Reader Authorization:

The level of the data read must be dominated by that of the reader.

Writer Authorization:

A subject that executes an instruction which produces an output must have a level that is dominated by that of the output.

In some implementations, the associations between inputs and subjects are relatively static, and their validation is straightforward. In others, subjects are created dynamically to service inputs as they arrive, and explicit modeling may be useful.

If the eight requirements of this access-constraint model are satisfied, then so are the requirements of the information-flow model. [WILL91] This observation supports the thesis that access constraints can provide an adequate definition of mandatory security. Related comparisons of access control models and information-flow models may be found in the works by Gove and Taylor. [GOVE84, TAYL84] Unfortunately, this access-constraint model also shares some potential weaknesses of the previous models. In particular, use of the simple security and *-properties to enforce nondisclosure rests on the following implicit assumption: a non-TCB subject either outputs information at its own level or outputs information of an unknown lower level that must, therefore, be given worst-case protection. This assumption ignores the possibility that a process may be able to produce information which is more highly classified than its inputs through some form of aggregation or inference. Security models which address this possibility in the case of database management systems are discussed in Section 4.3.

3.2.5 TAILORING THE MODELS

The remaining tasks in modeling nondisclosure are to tailor the definition of mandatory security to meet specific system needs and to provide rules of operation describing the kinds of actions that will be exhibited by the system being modeled. The following paragraphs discuss adaptations relating to lack of current access and the desirability of modeling error diagnostics, trusted operations, and nondeterminacy.

In most systems there are some operations that access objects without being preceded by an operation that provides access permission. For these operations, authorization must be checked on every access, and either the model or its interpretation must treat the combined effects of the simple security and observe-semantics properties, and of the *-property and modify-semantics properties. If this is done in the model, the result is as follows:

Observe Security:
A subject may execute an instruction whose behavior depends on the value of an object only if its security level dominates that of the object.

Modify Security:
A subject may execute an instruction that modifies an object only if its level dominates that of the object.

These axioms omit reference to the traditional current-access matrix and are particularly well-suited to systems that do not have an explicit mechanism for granting access permissions.

Although it is necessary to model unsuccessful execution resulting from attempted security violations, it is not necessary to model resulting error diagnostics. If the model only covers normal use of the system, it is both acceptable and traditional to omit error diagnostics, as would typically be the case in an informal model of a B1 system. For higher evaluation classes, however, an available option is to give detailed rules of operation that explicitly model some or all error returns. Their inclusion in the model can provide an alternative to examining them by means of a more traditional covert channel analysis, as well as additional information for judging correctness of the system's design and implementation. Error diagnostics resulting from unsuccessful instruction executions can reasonably be modeled either as output or as information written into the subject's data space.

A variant of the above modeling strategy has been carried out for the LOCK system. The LOCK verification is based on noninterference and nonobservability applied to subject-instruction processing (as opposed to the entire system). The inputs consist of LOCK kernel requests and user-mode machine instructions. LOCK uses a "conditional" form of noninterference in which certain "trusted" inputs are explicitly exempted from the noninterference requirement. The LOCK model was elaborated by means of an unwinding theorem and then augmented to obtain an access control model. Technically, the LOCK noninterference verification is an extension of traditional access control verification because the first major step in proving noninterference was to verify the traditional Bell & La Padula properties for the LOCK access control model. This access-control verification represents about half of the LOCK noninterference verification evidence. The LOCK developers compared noninterference verification with a traditional covert channel analysis technique based on shared resource matrices. [HAIG87, FINE89] They have since concluded that the noninterference approach is preferable, especially if both nonobservability and noninterference are verified, because of the extensive hand analysis associated with the shared resource matrix approach.

Noninterference has been generalized to nondeterministic systems in several ways. A variety of nonprobabilistic generalizations has been proposed for dealing with nondeterminacy, but they do not provide a full explanation of nondisclosure because of the possibility of noisy information channels. [WITT90] Despite this limitation, nonprobabilistic generalizations of noninterference provide useful insight into the nature of security in distributed systems. An interesting result is that a security model can be adequate for sequential systems, but is not adequate for a distributed system. This is because the process of “hooking up” its sequential components introduces new illegal information channels that are not addressed by the model. [MCCU88a] A state-machine model that overcomes this lack of “hook-up” security has been provided by McCullough. [MCCU88] It relies on state-transition techniques and, like the original Goguen and Meseguer models, has a demonstrated compatibility with traditional design verification methodologies.

3.3 NEED-TO-KNOW AND DISCRETIONARY ACCESS CONTROL

Discretionary access control (DAC) mechanisms typically allow individual users to protect objects (and other entities) from unauthorized disclosure and modification. Many different DAC mechanisms are possible, and these mechanisms can be tailored to support a wide variety of user--controlled security policy objectives. Users may impose need-to-know constraints by restricting *read access* and may guard the integrity of their files by restricting *write access.* As explained below, the use of group names may also allow specific objects and processes to be associated with specific user roles in support of least privilege. Discretionary security mechanisms are more varied and tend to be more elaborate than mandatory mechanisms. The policy requirements for them are weaker in order to allow for this variation. As a result, a well-understood discretionary security model can play a larger role both in clarifying what is provided in a particular system and in encouraging an elegant security design.

Traditionally, systems have been built under the assumption that security objectives related to DAC are both user-enforced and user-supplied. A variety of well-known weaknesses are traceable to this assumption. By way of contrast, vendor cognizance of user security objectives allows the development of a DAC security model whose mechanisms correctly support higher-level, user-enforced security policies. Moreover, modeling of these higher-level policies would provide a suitable basis for validating correctly designed DAC mechanisms and for supplying guidance on their use for policy enforcement.

DAC mechanisms and requirements are summarized in Section 3.3.1. Group mechanisms and their use in supporting user roles are covered in Section 3.3.2. Section 3.3.3 discusses traditional weaknesses in meeting common user-enforced security objectives and Section 3.3.4 presents mechanisms that overcome some of these weaknesses. Further information on DAC mechanisms may be found in *A Guide to Understanding Discretionary Access Control in Trusted Systems*. [NCSC87a] The formalization of control objectives such as need-to-know and least privilege, as well as the subsequent verification of access control mechanisms with per-user granularity, are research topics that have yet to be adequately explored.

3.3.1 DAC REQUIREMENTS AND MECHANISMS

Separate DAC attributes for reading and writing are traditional but not required. DAC security attributes must be able to specify users both explicitly and implicitly (i.e., by specifying a user group whose membership might be controlled by another user or user group). For systems at B3 and above, DAC attributes must give explicit lists[13] of individuals and groups that are allowed or denied access. A wide variety of relationships among individual and group permissions and denials are possible. [cf LUNT88]

The assignment of DAC attributes may be carried out by direct user interaction with the TCB, by (non-TCB) subjects acting on behalf of the user [NCSC88b, C1-CI-01-86], or by default. This last alternative is needed in order to ensure that an object is protected until such time as its DAC attributes are set explicitly. [NCSC88b, C1-CI-03-86] The default attributes for an object may be inherited (as when a new object is created in UNIX by copying an object owned by the user) or they may be statically determined.

A user who has responsibility for assigning DAC attributes to an object may be regarded as an *owner* of the object. Typically, the user who creates an object has responsibility for assigning DAC attributes and is thus the initial owner in this sense, but this is not a requirement. [NCSC88b, C1-CI-03-85] In UNIX, each file has a unique explicit owner, but root can also modify DAC attributes and it is thus an implicit co-owner. Some systems provide for change of ownership and for multiuser ownership; others do not. A variety of issues arise in the case of multiple owners. May any owner grant and revoke access, or are some owners more privileged than others? How is coordination among owners achieved; is agreement among owners required? The answers can differ for read and write accesses and for granting and revoking. Analogous issues arise when transfer of ownership occurs. A traditional approach to these issues is to let any owner grant or revoke access or ownership; another is to adopt a principle of "least access," so that all owners must grant a particular kind of access in order for it to become available. [cf LUNT88]

By tradition, the entities controlled by the DAC mechanism must include all user-accessible data structures controlled by MAC. However, the explicitly controlled entities (i.e. named objects) may be different from MAC storage objects. Examples where storage objects and named objects differ are found in some database systems (see Section 4.3). Operating systems can also have this feature. For example, files might be named objects which are made up of individually labeled segments that are the storage objects.

3.3.2 USER GROUPS AND USER ROLES

Access control lists determine triples of the form [user/group, object, access-mode] and

---

[13]Lists, per se, are not required; any equivalent implementation mechanism for specifying access on a per-user basis suffice.

thereby provide a limited variety of triples of the sort used for role enforcement. In fact, generalizing the mechanism to allow arbitrary programs in place of access modes would provide a general mechanism of the sort used for role enforcement, as was discussed at the end of Section 3.1.3. In the case of a trusted role, all associated programs would belong to the TCB.

Some systems allow a user to participate in several groups or roles. Some of these systems require a user to set a “current group” variable for a given session. In this case, the user would run only programs with execute access in the current group, and they would access only files associated with that group. Again, the effect is to create a three-way constraint involving user groups, processes, and storage objects. In order for a user to participate in several groups or roles, it must be possible for some groups to be subgroups of others or for a user to be associated with several different groups. The possibility of subgroups involves some interesting implementation issues. [SAND88] In the SMMS [LAND84], groups represent user roles, each user has a unique “individual” role, there is a notion of current role, and a given user may have several current roles. This use of individual roles or groups is a way of tying individual accesses to group accesses so that the two kinds of access do not have to be discussed separately: individual access is implicitly given by the rules for group access applied to one-member groups.

The ability to define user groups may be distributed. It is usually assumed that the class authorized to define user groups is larger than the class of system security personnel but smaller than the entire user population. A group associated with a trusted user role (e.g., downgrader, security administrator) would necessarily be controlled by a system administrator.

The owner of a group and the users who specify access to their objects in terms of that group need to clearly understand both the criteria for group membership and the entitlements associated with membership in that group. Such understandings depend partly on the mechanics of the group mechanism, which may be clarified by including groups in the security model.

3.3.3 SOURCES OF COMPLEXITY AND WEAKNESS

In security policies and definitions of security, complexity tends to inhibit effective user understanding. As such, it is a weakness that, in some cases, may be offset by accompanying advantages. The following paragraphs discuss several sources of complexity and weakness in DAC mechanisms including objects that are not disjoint, the coexistence of two or more access control mechanisms, discrepancies between allowed and authorized accesses, and the use of “set-user-id” to allow object encapsulation. Finally, the most common weakness found in DAC mechanisms is discussed, namely, that they are loose; that is, they control access to objects without necessarily controlling access to the information they contain.

It can happen that named objects overlap so that a given piece of data can be associated with several different sets of security attributes. In this case, they can be called disjoint. This is essentially what happens if ownership is associated with file names rather than with files. In this case, a given piece of data can have multiple owners, each of whom can give or revoke access to it. As with mandatory access controls, a lack of disjointness tends to interfere with one's ability to determine allowed accesses. It usually implies that named objects are different from storage objects, a fact which is a source of complexity that may have some advantages. Discretionary access may be controlled to a finer or coarser level of object granularity than mandatory access. Aggregation problems can be addressed by allowing some objects to be subobjects of others and by imposing stricter access controls on an aggregate object than on its components. In the case of aggregate objects, there is the additional problem that a change in the permissions of the aggregate may logically entail a change in the permissions of its components.

When a C2 or B1 system is created by retrofitting security into a previously unevaluated system, it may happen that the new mechanism supports access control lists in addition to the old discretionary mechanism supported. In this case, the discretionary portion of the security model can play a useful role in showing how the two mechanisms interact. [BODE88]

If a process has its discretionary permission to access an object revoked while the process is using it, some implementations allow this usage to continue, thereby creating a distinction between authorized and allowed accesses. This distinction is both an added complexity and a weakness that needs to be modeled in order to accurately reflect the utility of the mechanism. The principal reason for allowing this distinction is efficiency of implementation. However, in virtual memory systems, immediate revocation can often be handled efficiently by deleting access information in an object's page table. [KARG89]

Although discretionary access is ordinarily thought of as relating to users and objects, processes also acquire discretionary permissions, either dynamically when they are executed, or statically from their executable code. With dynamic allocation, the permissions may be those associated with the user who invoked the process; the process's user id would be an example. In command and control systems, there is often a "turnover" mechanism in which the user id of a process can change in order to allow a smooth transfer of control when a user's shift ends. In UNIX, the user id of a shell program changes in response to a "switch user" command.

With static allocation, security attributes might be associated with a subject on the basis of its executable code. The UNIX "set user id" feature provides an example of this type of allocation. The owner of a program can specify that it is a "set-id" program, meaning that it will run with the owner's "effective user id" and thereby assume its owner's permissions when executed by other users. The purpose of this mechanism is to allow programmers to create "encapsulated" objects. Such an object is encapsulated by giving it owner-only access, so that it can be handled only by programs whose effective user id is that of the owner. Other users can access the object by only executing "encapsulating" set-user-id programs that have the same owner as the encapsulated object. The UNIX set-user-id option is a source of added complexity, and, as explained below, is vulnerable to a variety of misuses. Other methods of object encapsulation are thus well worth investigating. Suggested patches to the UNIX set-user-id mechanism have been considered in the design of Secure Xenix [GLIG86], and modeling of the set-user-id mechanism itself is illustrated in the TRUSIX work. [NCSC90b]

Discretionary access controls are inherently loose. This can cause information to be disclosed even though the owner has forbid it. For example, breaches of need-to-know security may occur when user *i,* who owns a file *f* gives permission to access information in *f* to user *j* but not to user *k,* and then *k* indirectly obtains access to *f.* This can happen in a variety of ways, including the following:

- *j* copies *f* to a file that *k* has read access to.
- a *Trojan* horse acting under authority of either *i* or *j* gives *k* a copy of *f.*
- a *Trojan* horse acting under authority of *i* gives *k* read access to *f.*
- a poorly designed set-user-id program created by *j* is run by *k* (with the discretionary

permissions of *j*) and is misused to give *k* a copy of *f*.

Each case involves an extension of access that is analogous to downgrading in MAC. If this extension occurs without the knowledge and permission of *f* owner, then the intent of the owner's protection is violated.

The first three of the above five weaknesses are specific to policies dealing with nondisclosure, and they need not carry over to other policies dealing with unauthorized modification. The fourth and fifth weaknesses, by way of contrast, involve tampering with DAC security attributes by non-TCB subjects, a feature that seriously affects any policy relying on DAC attributes.

The first of the above weaknesses appears to be an inherent aspect of traditional DAC mechanisms. An interesting fact about these mechanisms is that there is no general algorithm that can decide, for an arbitrary system and system state, whether a given user can ever obtain access to a given object. [HARR76] As explained below, such weaknesses are not forced by DAC requirements, despite their prevalence in existing systems.

3.3.4 TIGHT PER-USER ACCESS CONTROL

Tight controls on the distribution of information within a computing system originate from efforts to provide DAC-like mechanisms that have useful information-flow properties [MILL84] and from efforts to provide automated support for "ORCON" and similar release markings that are used in addition to security classifications. (ISRA87, GRAU89] Some typical release markings are:

ORCON—dissemination and extraction of information controlled by originator
NOFORN—not releasable to foreign nationals
NATO—releasable to NATO countries only
REL <countries/organization>—releasable to specified foreign countries only
EYES ONLY <groups/offices>—viewable by members of specified offices only
PERSONAL FOR <individuals>—releasable to specified individuals only.

Release markings are used by the originator(s) of a document for providing need-to-know information. Some release markings such as NOFORN, NATO, and REL <>, have coarse user granularity and, as explained in Appendix A.4, can be handled via nondisclosure categories. Others have per-user granularity but, unlike traditional DAC mechanisms, restrict access to both the document and the information it contains. A catalogue of release markings and an accompanying language syntax may be found in the article "Beyond the Pale of MAC and DAC — Defining New Forms of Access Control." [MCCO90]

Tight access control mechanisms designed to support need-to-know differ from traditional DAC mechanisms in several crucial respects. The explicitly controlled entities (i.e., "named objects") include processes as well as data structures. In addition, a user's ability to modify their security attributes is highly constrained. The first such mechanism was proposed by Millen [MILL84]; a minor variant of it follows.

Each controlled entity is associated with two sets of users, a "distribution" set and a "contribution" set. These are obtained by evaluating the entity's "access expression" whenever the object is involved in an access check. Access expressions are the DAC attributes of the policy; their semantics explain the interplay among individual and group authorizations and denials. For the sake of brevity these are not modeled. The distribution and contribution sets enforce nondisclosure and integrity constraints, respectively. Smaller distribution sets represent a higher level of nondisclosure, and smaller contribution sets represent a higher level of integrity. The empty set is both the highest nondisclosure level and the highest integrity level. Information may flow from entity *f* to entity *g,* provided the distribution set for *f* contains the distribution set for *g* and the contribution set for *f* is contained in the contribution set for *g*.

In Millen's model, devices have a controlling influence on user behavior. A terminal, for example, is modeled as two separate devices—a keyboard for input and a screen for output. When a user logs in, the distribution and contribution sets for both the keyboard and the screen are set (by a secure terminal server) to $\{i\}$. Communication from other users (or files that they own) is enabled by extending the contribution set for the screen. Communication to other users is enabled by extending the distribution set for the keyboard. The keyboard contribution set and the screen distribution set must always contain $\{i\}$ in order to reflect the actual presence of the person using the terminal.

The distribution sets and contribution sets of Millen's model may be viewed as levels whose partial ordering is directly tied to an information-flow policy. Consequently, this DAC mechanism is tight enough to control information flow, and covert storage channel analysis can be used to check the extent to which distribution sets control disclosure in an actual implementation. In Millen's model, DAC permissions for an object are set (by its creator or by default) when it is created and are never modified thereafter. As a result, DAC Trojan horses of the sort discussed in Section 3.3.3 are impossible. The ability to dynamically change discretionary attributes without losing tightness would require some nontrivial additions to the policy. Expansion must be done only by trusted path software in response to appropriate owner authorization. Ownership must propagate so that it is associated not only with controlled entities but also with the information they contain. One option is for ownership to propagate in the same way as contribution sets, with each owner supplying access constraints. A slightly different option is suggested in McCollum's work. [MCCO90]

Similarities between MAC and the tight DAC of Millen's model suggest the possibility of a single mechanism meeting both sets of requirements. Moreover, both sets of requirements stem from the same policy objective in *Security Requirements for Automated Information Systems (AISs).* [DOD88a] The consistency of the two sets can be informally justified as follows: assume the system supports co-ownership by maintaining separate distribution and contribution sets for each owner, taking intersections in the obvious way. Each input must be co-owned by a system security officer (SSO), and there is a trusted path mechanism that allows a user to select the SSO's distribution set from a collection of SSO-controlled groups. For example, the groups might be TS, S^+, and U^+ where, by definition, U^+ is the group of all users, S^+ is the union of TS and the secret users, and TS is the top secret-users. Users are instructed to select the SSO's distribution set according to the sensitivity of their data, and, as a result, the main MAC requirements for labeling and access control are satisfied.

A disadvantage of tight DAC is that there are many innocuous violations of the policy, as, for example, when a user creates a file with owner-only access and then mails it to a colleague. Pozzo has suggested that, if the user is authorized to extend access, a trusted path mechanism should interrupt the program causing the violation in order to obtain the user's permission, thereby minimizing the inconvenience associated with tight control. [POZZ86] Another strategy for minimizing unnecessary access violations, which has been suggested by Graubartis, is to allow DAC security attributes to float so that when a process reads a file, for example, the distribution list for the process would be intersected with that of the file. [GRAU89] A disadvantage of propagated ownership is that the set of owners tends to expand, and this is inconvenient if all owners must agree on access control decisions.

3.4 TCB SUBJECTS—PRIVILEGES AND RESPONSIBILITIES

TCB processes are often exempt[14] from some of the access constraints placed on non-TCB subjects and are, therefore, able to access data and perform actions that are not available to non-TCB subjects. The responsible use of such exemptions by the TCB is properly part of the system security policy. Exemptions in this sense are not a license to violate policy. If a TCB process is exempt from some of the constraints placed on non-TCB subjects but not others, it may be useful to treat it as a TCB subject so that the TCB can enforce those constraints from which it has not been exempt.

The trusted-role programs identified in Section 3.1.3 are usually exempt in this sense because they have access to security critical data structures and, in many cases, address role-related extensions of the basic system security policy. Device-related subjects are also likely to be exempt. A secure terminal server needs to handle inputs at a variety of security levels and may be implemented as a TCB subject exempt from some of the MAC constraints. A print server is also likely to be implemented as a multilevel TCB subject because of the need to save, label, and print files at several different security levels.

In some cases, exempt subjects conform to the same overall policy that the system enforces on non-TCB subjects. In others, exempt subjects provide limited but significant extensions to the basic system policy. As a result, the presence of unmodeled exempt subjects can make it difficult to determine the actual system security policy by looking at the security policy model. [cf LAND84] To the extent that the policy for exempt subjects differs from the policy described in the *system security model,* the validity of the model as a representation of the system is compromised as are assurances derived from an analysis of the model. The extent of the compromise tends to be influenced by the extent to which such exempt subjects interact with nonexempt subjects.

The actual rules enforced by the system include both what may be done by non-TCB subjects and what is done by TCB subjects. Unmodeled special cases can be avoided by directly addressing the policies associated with exempt subjects in the model. [cf ABRA90] The following paragraphs address the modeling of exemptions and their legitimate use by TCB processes. There is no explicit requirement to model TCB subjects and their exemptions from access control, but there may be implicit modeling requirements that apply to some subjects, especially those that are directly involved in the access control mechanism. In the case of subjects exempt from mandatory access checks, it is often appropriate to substitute covert channel analysis for explicit modeling.

3.4.1 IDENTIFYING PRIVILEGES AND EXEMPTIONS

[14] TCB subjects which are exempt from some or all access constraints enforced on non-TCB subjects are often referred to as "privileged," but this word has a variety of other uses, as in the phrase "principle of least privilege." As mentioned in Section 3.1.8, TCB subjects (in particular, exempt subjects) are often referred to as "trusted" subjects, but this term has other uses as well.

If the principle of least privilege is followed in allocating exemptions to TCB subjects, then the extent of a subject's exemptions are both a partial measure of the need to model its behavior and an indicator of what should be modeled. This information indicates the extent to which a TCB subject's privileges may be abused and, thus, the amount of assurance that must be provided regarding the subject's correctness. Information on the extent of a subject's exemptions can be provided by an explicit identification of exemptions, by its security attributes, and by specific knowledge of the resources accessible to it. These identification techniques provide a variety of techniques for modeling and implementing exemptions.

A useful first step in describing exemptions is to classify them according to various kinds of security requirements, such as mandatory access control, discretionary access control, auditing, and service assurance. The purpose of this classification is to guarantee that any process which fails to have a particular kind of exemption will not interfere with the corresponding kind of security requirement. Each named exemption or "privilege" is associated with a particular kind of functionality allowed by that exemption. Ideally, this functionality should be available only through possession of the named exemption. As with other security attributes, it is important to know how a process inherits its exemptions (e.g., dynamically from its parent or statically from its executable code). Whether this classification is an explicit part of the model will depend largely on whether it appears in the system design. Including exemptions explicitly in the design simplifies the analysis of exempt subjects but also complicates the access control mechanism.

In the Army Secure Operating System (ASOS), exemptions are an explicit part of the access control mechanism. [DIVI90] The association between exemptions and kinds of security allows the model to assert, for example, that any process (trusted or otherwise) satisfies the *-property unless it has the "security-star-exemption." To possess an exemption, a process must receive that exemption both statically during system generation and dynamically from its parent process. Thus, most ASOS subjects can never have exemptions. Those subjects that can have exemptions will only run with the exemptions when they are necessary. The TRUSIX work also illustrates the use of exemptions for the case of a trusted login server: two exemptions and associated transformations allow (a new invocation of) the server to change its real user id and raise its security level. [NCSC90b]

The DAC mechanism can be extended to identify TCB subjects and security-critical objects in the following way. The system has one or more distinguished pseudousers. Some or all security-critical data structures are owned by these pseudousers. A named object owned by a pseudouser has owner-only access, and DAC is designed in such a way that nothing can ever alter the access permissions of an object owned by a pseudouser. (This implies that the system's DAC mechanism is free from some of the traditional weaknesses mentioned in Section 3.3.3.) Only TCB subjects are allowed to act on behalf of pseudousers and, therefore, are the only subjects capable of accessing objects owned by their pseudousers. Thus, if the audit trail, for example, is owned by an "auditor" pseudouser, then editing of audit files can be performed only by specially designed TCB subjects acting on behalf of the auditor, if at all.

The MAC mechanism can also be extended to allow partial exemptions. Security-critical data structures can be treated as objects with special security levels possessed only by TCB subjects. More significantly, subjects can be <u>partially exempt</u> from the *-property. Each subject is provided with two separate security labels, an "alter-min" label that gives the minimum level to which a subject may write and a "view-max" label that gives the maximum level from which it may read.

A process is partially trusted to the extent that its alter-min level fails to dominate its view-max level. [SCHE85, BELL86, DION81] A straightforward application of partially trusted processes is found in the GEMSOS design. [SCHE85] Each process has two security labels and is classified as "single-level" or "multilevel," according to whether the two labels are equal or distinct. Each security label has separate nondisclosure and integrity components. The nondisclosure component of the view-max label must dominate the nondisclosure component of the alter-min label, whereas, the integrity component of the alter-min label dominates the integrity component of the view-max label because of the partial duality between nondisclosure and integrity mentioned in Section 3.1.6.

Finally, as an example of the above identification techniques, consider the requirement that only certain administrative processes can access user-authentication data. One possibility is to treat this independently of other policy requirements. User-authentication data can be stored in unlabeled TCB data structures that are not available to non-TCB subjects or even most TCB subjects. This nonavailability constraint (and exemption from it) might be implemented via hardware isolation or by an explicit software exemption mechanism. This approach could be explicitly modeled.

A second possibility is to permanently place user-authentication data in named objects owned by an "administrator" pseudouser and allow only certain programs to run on behalf of this pseudouser. A (fixable) drawback of this approach is that such programs may need to run on behalf of actual users for auditing purposes.

A third possibility is to place user-authentication data in a storage object which has a unique security level that is incomparable with any level available to users of the system. Only certain administrative programs are allowed to run at this level; such programs may need to be partially trusted in order to access data at other security levels. Notice that other TCB-only security levels may be needed to ensure that these administrative programs do not have access to other security-critical data structures which are also being protected by the MAC mechanism.

3.4.2 RESPONSIBLE USE OF EXEMPTIONS

In modeling an exempt subject, the goal is to prohibit abuses of privilege that might result from exemptions by placing explicit constraints on that subject's behavior. The main challenge in formulating these constraints is to achieve an appropriate level of abstraction. In most cases, the requisite security requirement is considerably weaker than a detailed statement of functional correctness. The following paragraphs first discuss subjects that conform to the overall policy enforced for non-TCB subjects and then discuss those that do not. These latter subjects include, primarily, the trusted-role programs identified in Section 3.1.3.

An exempt subject that conforms to the basic policy illustrated by the modeling of non-TCB subjects usually does not require separate modeling unless it supports a significant extension of the TCB interface that is not covered by the general model. This is the case of a subject which is also a trusted DBMS, for example. However, the modeling of a policy-conforming exempt subject can provide additional insight into its proper design and is particularly valuable if the subject is visible at the user interface. For example, a scheduler is user-visible, should be policy-conforming, and could be treated as an exempt subject whose behavior is justified either by modeling noninterference-like requirements [cf KEEF90; MAIM90] or by performing a covert channel analysis.

A secure terminal server is another example of a user-visible, policy conforming, exempt subject. The main security-critical requirements for a secure terminal server are that each user and terminal have a trusted path (or paths) for exchanging security-critical information with the TCB and that there be no "cross talk" between a given trusted path and any other I/O channel. The issue of how logical channels are multiplexed onto a given terminal is not security-relevant, as long as it is correct. The fact that the multiplexing includes an Identification and Authentication (I&A) protocol is security-relevant, but modeling details of the I&A mechanism is unlikely to add much assurance unless it is accompanied by an analysis of subvertability.

In the case of a trusted-role program, misuse and resulting breaches of security can be prevented through a combination of software checks and procedural constraints on correct use of the program. If the program and the trusted role it supports are designed together, then the design analysis can identify errors of use and can determine whether they are easily detected through automated checks. The automated checks are appropriately covered in a trusted-process security model. Knowledge of errors that are not caught by the automated checks can be recast as informal policies associated with the trusted role itself. These procedural policies are reasonably included in the system's Trusted Facility Manual. A reader of the high-level system documentation should be able to see how a given misuse of a trusted role is inhibited through an appropriate combination of software mechanisms required by the model and procedural constraints found in the definition of the trusted role. One should be able to see, for example, how a system administrator is inhibited from falsifying evidence about the behavior of other users by editing the audit trail.

Security properties for trusted-role processes often involve constraints that hold over a sequence of events, as opposed to constraints on individual state transitions. The use of locks to ensure proper sequencing of events may be needed, at least in some cases. [cf LAND89] Modeling of trusted-role processes is often omitted on grounds of restricted use and lack of accepted examples, but this argument is weak because the higher level of user trust is offset by a greater potential for abuse.

3.5 INTEGRITY MODELING

Although both integrity and nondisclosure are important for both commercial and military computing systems, commercial systems have historically placed greater emphasis on integrity, and military systems more emphasis on nondisclosure. Accordingly, guidelines on integrity policy are under development by the National Institute of Science and Technology (NIST).

Although the *TCSEC* does not impose specific requirements for integrity policy modeling, it does provide for vendor-supplied policies and models. As mentioned in Section 1.3.2, DoD Directive 5200.28 includes both integrity and nondisclosure requirements. In addition, the NCSC evaluation process accommodates any security policy that meets minimum *TCSEC* requirements and is of interest to the Department of Defense.

Although the following paragraphs do not offer guidance on the formulation of system integrity policies, they do consider relevant modeling techniques and *TCSEC* requirements. A brief discussion of integrity objectives, policies, and models is given in order to provide an overall picture of the field of security policy modeling. This is followed by a brief taxonomy of integrity-related concepts and their relationship to security modeling. Topics covered in the taxonomy fall into two broad areas: error handling and integrity-oriented access control. Examples relating to TCB integrity are included with the taxonomy as indications of possible relationships between integrity modeling and related assurance issues for the TCB. The presented taxonomy is based largely on the one found in "A Taxonomy of Integrity Models, Implementations and Mechanisms." [ROSK90]

3.5.1 OBJECTIVES, POLICIES, AND MODELS

The term integrity has a variety of uses in connection with computer security [cf RUTH89], all stemming from a need for information that meets known standards of correctness or acceptability with respect to externally supplied real-world or theoretical situations. A closely related need is the ability to detect and recover from occasional failures to meet these standards. These needs lead to derived objectives for user integrity, data integrity, and process integrity in order to maintain and track the acceptability of information as it is input, stored, and processed by a computing system.

Commercial experience as a source of integrity objectives, policies, and mechanisms is covered in the landmark paper by Clark and Wilson. [CLAR87; KATZ89] Their paper includes a separation-of-duty objective for promoting user integrity; application-dependent, integrity-validation processes for ensuring data integrity; the application-dependent certification[15] of "transformation procedures" for establishing process integrity; and a ternary access control mechanism for constraining associations among users, processes, and data. A thorough discussion of their work is given in the *Report on the Invitational Workshop on Integrity Policy in Computer Information Systems (*WIPCIS). [KATZ89]

Their requirement for well-formed transactions suggests that for high-integrity processes, application-dependent software modeling may be needed as input to the certification process. As discussed in Section 3.5.3, the access control mechanism of Clark and Wilson determines which procedures may act on a given object and thereby provides an encapsulation mechanism similar to those associated with data abstraction in modern programming and specification languages.

3.5.2 ERROR DETECTION AND RECOVERY

In the following paragraphs, general observations on the nature of error handling are followed by a variety of examples that arise in connection with integrity policies.

An error can only be recognized if an unexpected value occurs or if an unexpected relationship among two or more values occurs. The possibility of anomalous values and relationships may be built in as part of the system design, as in the case of audit records and message acknowledgments. It may be added by an application, be of external origin resulting from syntactic and semantic constraints on the structure of the application data, or be external in the form of information held by multiple users (in addition to the computing system).

The detection of anomalous values and relationships is only partially automated in most cases. Typically, an initial error or anomalous situation is detected, perhaps automatically. This discovery

---

[15] The term "certification," is sometimes confused with the similar terms, evaluation" and accreditation." Certifications, like evaluations, are carried out with respect to security-relevant criteria and, like accreditations, are with respect to a particular application and site. However, an evaluation applies to a product whereas a certification applies to its use in a particular site or application. An accreditation is an authorization for use, whereas a certification is a decision that relevant technical requirements are met.

may be followed by further automated or manual investigation to find related errors and, perhaps, the root cause of these errors. Finally, corrective action removes the errors and/or inhibits the creation of new errors.

The automated portions of this three-part, error-handling process are more likely to be suitable for security modeling. In the case of a partially automated error-handling mechanism, modeling can help clarify which portions of the mechanism are automated (namely, those that are modeled). If detection is automated and recovery is manual, there may be additional issues associated with the design of an alarm mechanism (e.g., timeliness, avoidance of masking high-priority alarms with low-priority alarms).

As already mentioned, the TCB audit mechanism required for systems at classes C2 and above is a built-in, error-rejection mechanism. It is usually not modeled but could be; Bishop has provided an example. [BISH90] Recent integrity articles have suggested that audit records should allow for a complete reconstruction of events so that errors can be traced to their source. [SAND90, CLAR89] This integrity requirement on audit trails is potentially amenable to formal analysis.

"Checkpoint and restore" is another built-in mechanism found in many systems. Modeling of this mechanism may be no more interesting than for auditing. However, systems at classes B3 and above have a trusted-recovery requirement. The state invariants that come out of the formal modeling and verification process effectively define exceptional values of the system state that must not exist in a newly restored state. Moreover, run-time checking of these state invariants may be necessary after an unexplained system failure.

A variety of integrity mechanisms promote user integrity by collecting the same or related information from several different users. Supervisory control and N-person control allow two or more users to affirm the validity of the same information. Supervisory control involves sequential production and review of an action, whereas N-person control refers to simultaneous or independent agreement on taking action. A possible approach to modeling N-person control is given by McLean. [MCLE88, Sec. 3] The subjects of the model include composite subjects made up of N ordinary subjects acting on behalf of different users. The explicit inclusion of these N-person subjects invites questions about their security attributes; for example, what is their security level? In both supervisory and N-person control, lack of agreement is an error condition, and the system performs error handling by blocking the action being controlled.

Certain access controls may be suspended by any user in an emergency, but the system may require that a state of emergency be explicitly declared before suspending these access controls. The explicit declaration together with the actual violation of the controls provides two different indications of the need for a typical action. The decision to constrain a particular action via supervisory, N-person, or emergency-override control might be made by an appropriately authorized user, as opposed to the system's designers. In all three of these mechanisms, the actions to be controlled need not be vendor-defined.

A related strategy for promoting user integrity is separation of duty. This involves defining user roles in such a way that no one user can commit a significant error without detection. Instead, there would have to be collusion among several users. For example, if no one person can order goods, accept delivery, and provide payment, then it is difficult for a user to buy nonexistent items from himself. A more generic example recommended by Clark and Wilson is that no person who is authorized to use a transformation procedure has participated in its certification. Separation of duty promotes user integrity, if roles are designed in such a way that several different users contribute overlapping, partially redundant views of an external situation modeled in the computing system. In most cases, design of the roles is application-dependent. Provisions for role enforcement are part of the system design. Detection of errors can be either manual or automated (but application-dependent). Error recovery is largely manual. In the Clark-Wilson model, separation of duty is enforced via access control triples, but is itself not formally modeled.

According to Clark and Wilson, consistency checks on the structure of user-supplied data are needed initially to guarantee that data has been properly entered [CLAR87] and can be run later as a check against inappropriate modification [CLAR89]. Typically, these Integrity Validation Procedures (IVPs) compare new information against previously computed information, as in a program that compares actual inventory data against previously computed inventory. The use of IVPs is explicitly modeled by Clark and Wilson, but the fact that they reject anomalous data is not.

Redundancy to improve process integrity is used in high-availability systems. Two or more processes with different hardware and/or algorithms are run with the same expected result, and a voting algorithm combines the results. An interesting feature of these data- and process-integrity promoting algorithms is that they apparently increase integrity through aggregation and inference. In this respect, integrity is similar to nondisclosure, not dual to it. Interesting formal models of voting algorithms include the “Byzantine Generals Problem”, in which a process can give inconsistent results [LAMP82], and clock synchronization algorithms. [LAMP87, RUSH89]

In general, all error-handling mechanisms exploit redundancy of the sort discussed in information theory and conform to the same general principles that provide the basis for error-correcting codes [cf HAMM80] used to suppress noise. What sets integrity-related mechanisms apart is their emphasis on user-induced errors.

3.5.3 ENCAPSULATION AND LEVEL-BASED ACCESS CONTROL

The following paragraphs discuss encapsulation mechanisms, the use of level-based access control and integrity hierarchies, and the use of partially trusted subjects to achieve encapsulation.

Encapsulation mechanisms ensure what Clark and Wilson refer to as “internal consistency.” They provide a limited set of high-level operations that can be certified to maintain a known structure on the data that they encapsulate. Several mechanisms suitable for performing encapsulation have been discussed in Sections 3.1.3, 3.3.2, and 3.3.3. Another encapsulation mechanism is message passing, as illustrated in the design of Smalltalk. [GOLD80] As discussed below, level-based access control with partially trusted subjects also provides an encapsulation capability similar to type enforcement.

If an encapsulation mechanism is supported, it may be used to provide tamper proofing for the TCB and to enforce the principle of least privilege within the TCB. Type enforcement has been used for this purpose. [BOEB85] In the LOCK system, each security-critical entity (for example, the password file) is of a type that can be accessed only by the appropriate TCB subjects. This typing information is preset and, in the case of TCB entities, cannot be modified, even by the system security officer. Since the type enforcement mechanism is formally verified, this verification provides a partial verification of TCB integrity as a special case. Type enforcement has also been used to extend the LOCK TCB for a trusted DBMS application. [STAC90] In the extension, new TCB subjects are straightforwardly prevented from interfering with the old part of the TCB through static access restrictions in the type-enforcement table.

The crucial idea behind access control based on integrity levels was alluded to in Section 3.1.6. That information may flow from one entity to another only if the latter's integrity level is at or below that of the former, thereby preventing the latter from being contaminated with low-integrity information. Thus, if the integrity ordering is inverted for purposes of access control, then the *-property will automatically enforce the desired property. As observed by Roskos, [ROSK90] this duality applies not only to access control, but to higher-level nondisclosure policies as well. In the case of noninterference, for example, if inputs from user A cannot "interfere" with outputs to user "B," then A cannot compromise the integrity of these outputs. [PITT88] The practical utility of these observations is undetermined.[16]

For level-based integrity to be useful, there must be some convention for assigning integrity levels to controlled entities. Biba [BIBA77] suggested a hierarchy dual to that given by Executive Order 12356, which, as mentioned in Section 3.1.7, defines the terms "confidential," "secret," and "top secret." Thus, integrity levels would be classified according to the level of harm which could result from unauthorized or inappropriate modification of information. The dual of "secret," for example, would be a level indicating that inappropriate modification could result in serious, but not exceptionally grave, damage to national security. While this suggestion has not found wide acceptance, there are clear-cut examples of integrity orderings associated with the possibility of serious harm. The inadvertent substitution of "exercise" data for "live" data is one such example; effectively, "exercise" is a lower integrity level than "live." What Biba's suggestion lacks (in contrast to the work of Clark and Wilson) is a convention for controlling not only who may modify data but also how they do it.

There are several other integrity measures not based on level of harm. [cf PORT85] User contribution sets form an integrity ordering under the dual of the subset relation, as was noted in Section 3.3.4. A careful analysis of user roles can also give indications of needed integrity and nondisclosure levels. [LIPN82] The results of trusted error-detection and integrity-validation procedures can be stored as (components of) integrity levels in order to disable inappropriate uses of flawed data and to enable appropriate error-recovery procedures.

Integrity levels (or components) that are based on freedom from various kinds of error cannot be preserved during processing unless the procedures involved do not introduce these kinds of errors. This motivates the use of software assurance hierarchies. The necessary level of software assurance is found by working backwards from a level of acceptable risk, which depends both on the expected level of software errors and the level of harm which may result from these errors in a particular operating environment (the latter being bounded by the relevant Biba integrity levels). If software at several levels of assurance is needed on the same system, then an access control mechanism is needed to be sure that software used in a given application is backed up by adequate assurance. A technique that suggests itself is to include assurance levels as components of integrity levels.

A well-defined hierarchy of assurance levels is given in the *TCSEC* for evaluating TCBs. It has been adapted to subsystems [cf NCSC88a] and is readily adapted to individual processes; the

---

[16] Yet another relationship between nondisclosure and integrity is found in the "Chinese Wall" policy discussed in Section 4.4.4 - a nondisclosure policy can promote user integrity by preventing conflicts of interest.

hierarchy itself is summarized in Appendix D of the *TCSEC*. An easily applied assurance hierarchy based on credibility of origin has also been suggested by Pozzo: [POZZ86] software supplied by a system's vendor is regarded as having higher assurance than software developed by a company's support staff, and production software typically has higher assurance than experimental software. Software of unknown origin has lowest assurance. Such a hierarchy presupposes an ability to track data sources, as described in DoD 5200.28-R. [DODS8a, Enclosure2, Req. 7]

The use of level-based integrity for TCB encapsulation is suggested by Millen. [MILL84] One or more integrity categories can be reserved for TCB entities, as a means of protecting them from non-TCB entities. An application of this idea is found in the network described by Fellows. [FELL87] In this network, each communication process in the TCB is a partially trusted subject that sends messages labeled with its own unique integrity category, thereby encoding authorship of the message in the security label itself.

Lee and Shockley have argued that level-based integrity with partially trusted[17] subjects can partially fulfill the requirements of Clark and Wilson. [LEE88, SHOC88] The ASOS integrity policy may be viewed as a special case of this general partially trusted subject mechanism. Access checking in ASOS allows any process to read below its own integrity level; the implicit assumption is that any subject with reasonable integrity can be relied on to make its own integrity checks when needed. In effect, each subject in ASOS is treated as if it had a separate view-max security label obtained by setting the integrity component of its security label to integrity-low. [DIVI90]

4. TECHNIQUES FOR SPECIFIC KINDS OF SYSTEMS

Though relevant to many systems, some security modeling techniques are best illustrated by a particular kind of system. While traditional access control models are applicable to a wide class of systems, they are nicely illustrated by operating systems because they both contain file-like objects that are opened, accessed for a period of time, and then closed. The issue of selecting an underlying *model of computation* occurs in any modeling effort, but models not based on state machines have occurred most frequently in *networks*. Trusted application models often take the form of models for database systems. Issues having to do with label accuracy are illustrated in Compartmented Mode Workstations (CMWs) and the Secure Military Message System (SMMS). The treatment of these topics is largely introductory because of the breadth (and, in some cases, relative newness) of the material to be covered.

4.1 OPERATING SYSTEMS

In modeling operating systems, there is especially rich literature to draw on. The following paragraphs discuss traditional access control models and the models of Bell and La Padula in particular.

4.1.1 TRADITIONAL ACCESS CONTROL MODELS

An access control model traditionally involves a set of states, together with a set of primitive operations on states. Each *state* contains a set S of "subjects," a set O of "objects," and an "access" matrix A. For each subject s and object o, A $[s, o]$ is a set of access rights, such as "read," "write,"

[17] Subjects that are partially trusted with respect to integrity labels do not necessarily belong to the TCB.

"execute," and "own." In the context of an access control model, rules of operation are axioms or definitions describing the primitive operations.

The simplest useful access control model is perhaps the HRU model of Harrison, Ruzzo and Ullman. [HARR76] The HRU model assumes that every subject is also an object, the rationale being that (almost) every subject has local memory that it can access. In the HRU model, the primitive operations are, create *s*, create *o,* destroy *s,* destroy *o*, enter *r* into *A* [*s, o*], and delete *r* from *A* [*s, o*], where *r* varies over all supported access rights.

To use an access control model for security modeling, one needs to add functions that extract security labels from subjects and objects and to explain how operations are affected by security labels. In some cases, it may also be appropriate to model other attributes of objects, such as directory hierarchies. These additions to the basic access control model facilitate a clear presentation, but are not theoretically necessary for discussing security, as can be seen from Pittelli's translation of a Bell and La Padula model into the HRU model. [PITT87] One can model the addition of new subjects or objects by allowing *s* and *o* to change in response to the create and delete operations, or one can use fixed sets of subjects and objects together with an indication of which subjects are active. Finally, if an access control model is to discuss information flow, it needs to discuss which objects are actually written as a result of a given operation. This can be done by adding functions that extract object content. A complete information-flow analysis would also need to address flows to and from non-object entities. Flows to and from the current-access matrix, for example, could be handled either by treating rows of the matrix as objects or by treating these flows in the covert channel analysis.

The assumption that subjects are objects is associated with several interesting observations that need to be modeled whether or not subjects are regarded as objects. Typically, every subject has read and write access to itself (that is, to its stack and local data area). This fact can be modeled as the invariant (read, write) $\in A[s, s]$. Consequently, if a subject outside the TCB is allowed to change its security level, the new level should dominate the old one. This is because information at the old level in its local memory can be transferred only to entities whose level dominates the subject's new and (hence) old security level. The assumption that subjects are objects also provides a means of modeling interprocess communication: a process P_1 may send a message or signal to process P_2 if and only if P_1 has write access to P_2.

Hierarchical file systems have a potential for introducing various covert storage channel problems. In many cases, these channels can be closed through the use of additional access control checks that depend on the structure of the directory hierarchy. It is acceptable to include such access control checks (and hence, directory structures) in the model. One problem is that if a directory contains a file at a lower security level, then it is difficult to delete the directory without users at the lower security level finding out, since they can no longer access the file after it is deleted. A widely imitated solution proposed by Walter is to require a *compatibility property* for directories, namely that the level of each file dominate that of its parent directory. [WALT74] Alternatively, the system could just remove access and leave actual deletion of files and directories to a trusted garbage collector. A similar problem that is not ruled out by compatibility arises when a lower-level process creates a directory at a higher level. If a higher-level process then adds a (higher-level) file to the directory, a UNIX-like operating system would refuse a later request by the lower-level process to delete the directory, thereby signaling the existence of the higher-level file. [GLIG86]

The compatibility principle was regarded by the TRUSIX working group as being a generally useful property of directories, independently of its use to discourage covert channels. They considered four possible approaches to modeling directories and, for sake of generality, decided that security labels would be assigned to individual directory entries as well as directories. [NCSC90b, Sec. 6.10, 6.11]

Kernel call based implementations of operating systems invariably rely on some form of exception mechanism. At higher assurance levels, it is likely to be a matter of policy that exceptions not return information about higher-level entities. In this case, kernel exceptions can be included in the security model in order to represent this policy, either directly or in summary form. One might, for example, use the three exception classes "access-denied," "user-error," and "system error". [BELL76]

For further information on access control models, the reader is referred to *Cryptography and Data Security* [DENN82, Ch.4] and "Models of Multilevel Security" [MILL89].

4.1.2 THE MODELS OF BELL AND LA PADULA

As mentioned in Section 1.4, the work of Bell and La Padula identified the main steps in the security modeling process and provided the first real examples of security verification by proving that their rules of operation preserved necessary state invariants. The identified invariants codified important mandatory access control requirements and provided guidance for their implementation.

Not surprisingly, fifteen years of close scrutiny has produced an understanding of areas where refinement of the original approach is desirable in future efforts. The question of correspondence to externally imposed MAC security policy was handled quite informally. As indicated in Section 3.2 (and in the LOCK verification effort), a closer correspondence is possible with the use of external-interface models. In fact, the Bell and La Padula models contain some well-known weaknesses. A lack of attention to local process memory is related in part to the fact that subjects need not be objects. A subject may, in the absence of the tranquility principle [LAPA73, p. 19], lower its security level. As a result, the "change subject current security level" and "change object level" rules provide a variety of opportunities for untrusted downgrading of information. The significance of these and similar rules was not well understood until much later. [MILL84, MCLE85, MCLE87, BELL88, LAPA89]

The Bell and La Padula models have a relatively narrow focus. None of the models explicitly mentions multilevel devices, external interfaces, or user identities; and there is no modeling of integrity. With the exception of the *-property, exemptions from access control are not modeled, so that there is no basis for relating exemptions to use of privilege, subject integrity, or trusted user roles. The Multics model interpretation was necessarily incomplete because design work on Secure Multics was still in progress in 1976. Some of the notational conventions used in the models (e.g., overuse of subscripts and lack of mnemonic naming conventions) are avoided in many of the more recent security modeling efforts.

4.2 NETWORKS AND OTHER DISTRIBUTED SYSTEMS

Which network decomposition techniques lend themselves to security analysis? How should this decomposition be reflected in the model? How should the overall network security policy be reflected in the model? What is the role of *individual components* in enforcing the overall policy?

Should nondeterminacy and the distributed nature of a network be reflected in the model's underlying model of computation; that is, should the network be treated as a state machine or as something else? The following subsections discuss these questions as they relate to network security objectives and the requirements of the *Trusted Network Interpretation (TNI).* [NCSC87]

4.2.1 SYSTEMS AND THEIR COMPONENTS

A *network system* is an automated information system that typically consists of a communications subnet and its clients. Clients are entities that the subnet interacts with (e.g., host computers, other networks, users, electronic censors), as in Figure 4.1. Some clients may play a distinguished role in the network, performing key distribution or network administration services. The communications subnet might consist of message switches and transmission lines.

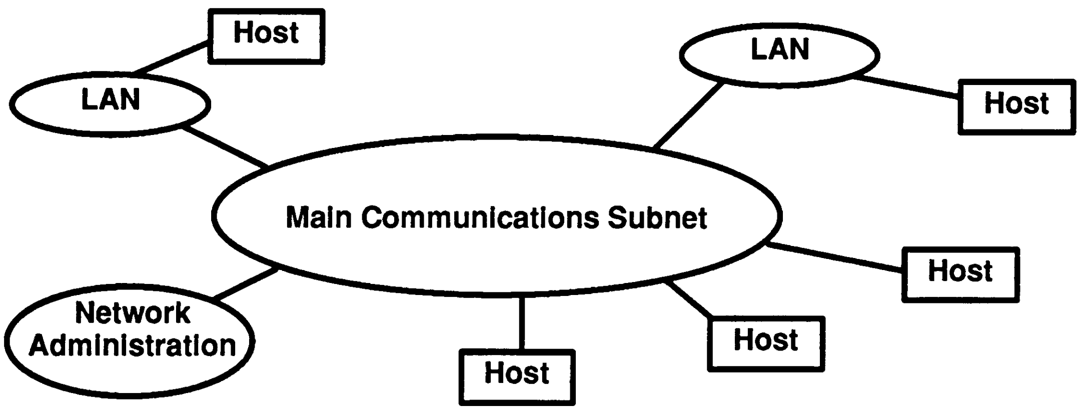

Figure 4.1. A Physical Network Decomposition

In addition to a physical decomposition of a network, there is also the possibility of decomposing along protocol layers, so that a network handling a layer *n* protocol decomposes into abstract protocol machines and network(s) handling a more primitive layer *n*-1 protocol. [cf ISO84, TANE88] In this second approach, it is possible to reassemble the protocol machines at various layers into physical components, so that the second approach is compatible with the first. In Figure 4.2, abstract protocol machines are collected to form interface units that may either be part of or separate from corresponding hosts. Each interface unit includes protocol handlers for all layers.

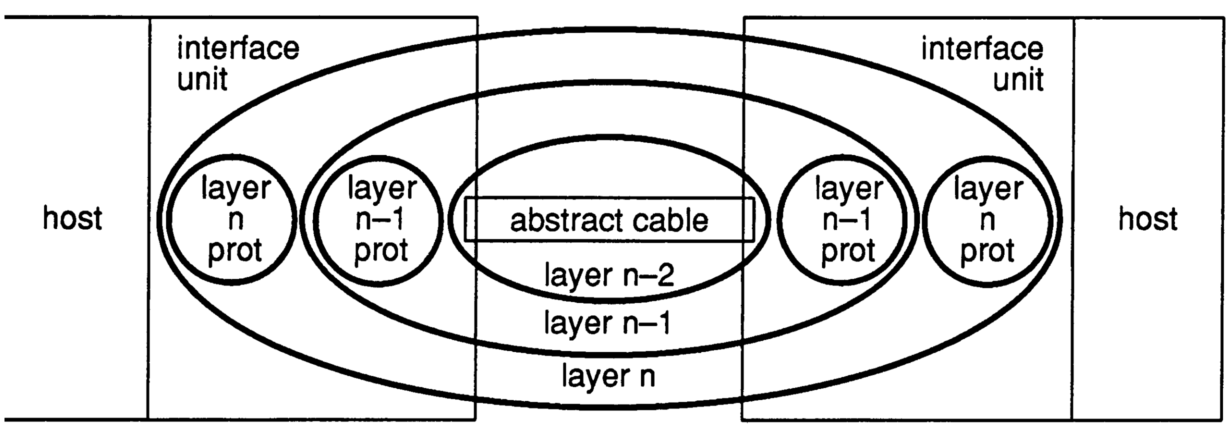

Figure 4.2. A Network Protocol Decomposition

The partitioning of a network system results in optional subsystem components, such as a communications subnet, as well as individual components whose further subdivision is not useful for modeling purposes. [NCSC87, Sec. I.3] In the following paragraphs, network systems and subsystems are both referred to as networks. Individual and subsystem components are both referred to as *components*. Network partitioning has a variety of security-relevant ramifications. There is a need for both system security models and component models as well as the need to consider TCB interfaces both in components and in the entire network system. In contrast to operating systems, there is little point in assuming that a network is either "up" or "down." Rather, it is necessary to model the network across a range of partially functioning states. [FELL87] The principle of "mutual suspicion" can be applied to components in order to minimize security violations in case an individual component is subverted. [SCHA85] In this case, each component enforces a component policy that is strong enough not only to contribute to the overall system policy but also to detect suspicious behavior in other components.

Networks are usually extensible, and, as a result, analysis of their security cannot depend on configuration details that change as the network expands. A similar problem is posed by the need for fault tolerance. Component failures and subsequent restarts should not lead to violations of network security policy. [FELL87]

Some networks have covert channel problems related to indeterminate information in message headers. In some cases, these "header" channels are visible in the model itself. Other covert channels result from interconnection subtleties and from nondeterminacy caused by channel noise and by race conditions associated with distributed computation. Similar covert-channel problems can arise in other nondeterministic systems, and, as mentioned in Section 3.2, a probabilistic analysis may be needed in order to rule out certain kinds of covert channels. [WITT9O] However, nonprobabilistic analyses have also provided some useful insights. [MCCU88, MCCU88a, JOHN88]

4.2.2 MODEL STRUCTURE AND CONTENT

A crucial modeling requirement for networks is that "the overall network policy must be decomposed into policy elements that are allocated to appropriate components and used as the basis for the security policy model for those components". [NCSC87, Sec. 3.1.3.2.2] One way of meeting this requirement is to provide a security policy model consisting of several submodels. A *network security model* can give an overall definition of security for the network in order to clarify how the network supports higher-level security objectives. A structural model can explain how the network is partitioned. This is particularly useful if the component structure is directly involved in the network's policy enforcement strategy. Finally, each component is given a security policy model. Correctness is shown by demonstrating that the component model properties together with the structural model properties imply the requirements in the network security model. The given demonstrations may rely either on external interface properties of the components [cf MOORE90, BRIT84, FREE88] or on modeled internal properties [cf GLAS87].

For a component that is part of a larger network, external-interface requirements will often be apparent from the network modeling effort. For a separately evaluated component, rated B2 or higher, external-interface requirements will usually be given in an accompanying network security architecture description. [NCSC87, Sec. 3.2.4.4, A.3] The NCSC Verification Working Group has concluded that access control requirements imposed by the network security architecture should be modeled. Consequently, the definition of security for the component model needs to contain or comply with such external-interface requirements. It is easier to demonstrate conformance if these requirements are directly included. This “external” portion of a component model is essentially new and is distinct from internal requirements or rules of operation. In the case of MAC requirements, this external interface portion may be regarded as a generalization of *TCSEC* device labeling requirements. [NCSC87; Sec. 3.1.1.3.2, 3.2.1.3.4] A model for a network system, in contrast to a component, may not need such external-interface requirements because the environment of the network system need not have a well-defined security architecture and is likely to contain only trusted entities such as users and other evaluated network systems.

Security-critical entities in the network or its containing network system are collectively referred to as the *network TCB (NTCB).* A second network modeling requirement is that a component model should cover all interfaces between security-critical network entities and other kinds of entities (see Figure 4.3). Collectively, these interfaces contain the reference monitor interface for the NTCB. Relevant non-NTCB entities for a network might belong either to the network or to its environment. Whole components of a network may lie outside of the NTCB. Individual components may also contain non-NTCB entities (as in the case of an ordinary host, for example).

When modeling a component, it may be desirable to model not only the interface between its TCB and non-TCB entities but also the interface between its TCB and other NTCB entities in order to give a better description of the security services provided by that component. For an A1system, this additional modeling information can provide correctness criteria for the FTLS, since each component FTLS must describe the interface between that component’s TCB and the TCB portions of other components. [NCSC87, Sec. 4.1.3.2.2] These additional interfaces are indicated by dashed lines in Figure 4.3.

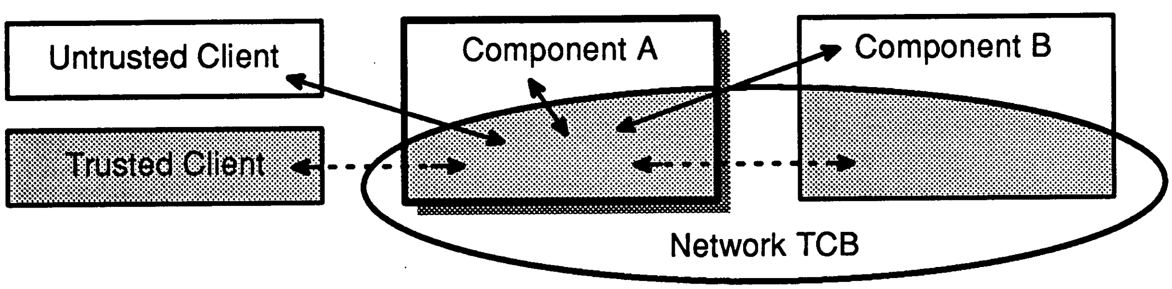

Figure 4.3. Modeled Interfaces to the TCB Portion of Component A

4.2.3 NETWORK SECURITY MODELS

The security policy model for a network often contains a network security portion, although this is not required by the *TNI.* Overall network security modeling is especially useful for a network that is too large for effective system testing because, in this case, overall security assurance can be achieved only by some form of static analysis. A network security model can give a precise description of security services provided by the network and can help to identify the boundaries of the system being modeled. It can identify security-relevant aspects of communication among major components and can describe associated security requirements. The entities described in this model need not be local to individual physical components. This model partially determines and may share the external-interface requirements found in the various component models.

For network systems rated B1 and above, mandatory access control plays a major role in modeling user-related communication. [SCHN85] Other useful security requirements include correct delivery [GLASS87]; discretionary access control; label, header, or message integrity [GLAS87, FREE88]; and encryption [BRIT84]; among others. [ISO89; NCSC87, Sec.9]

In many networks, there is a significant amount of TCB software devoted to security-critical communications protocols. A clear understanding of the basis for these protocols is required in order to completely assess security assurance. A network security model can provide this understanding by providing requirements that serve as correctness criteria for security-critical protocols found in the NTCB portion of the network. Label association protocols and label integrity requirements are of particular relevance in the case of MAC policy modeling.

Access control modeling may be influenced by the network topology. For example, access enforcement may be carried out by a communications subnet, while access decisions are performed by a designated host that serves as an access decision facility. [BRIT84]

One approach to specifying mandatory access control is to regard hosts as subjects (partially trusted subjects, in the case of multilevel secure hosts) and host "liaisons" as "communication" objects.[18] [BELL86, BELL88a] Host liaisons are temporary connections established between pairs of hosts for the purpose of exchanging messages at a given security level, such as transport layer connections. This approach is similar to that taken in several real systems. In the Boeing local area network (LAN) model the subjects are arbitrary LAN clients. [SCHN85] In the model described by Fellows subjects are hosts. [FELL87] In this latter modeling effort, the host liaison approach was found to be satisfactory for expressing system security requirements with the possible exception of what came to be known as the "entelechy"[19] property. This property means that a host may send or receive messages over a connection if and only if it has current access to that connection. These models are internal-requirements models for network systems. Because these models treat hosts very simply, they also serve as external-interface specifications of their communication subnets.

A more explicit external-interface model was given for the Multinet Gateway System (MGS). The clients of the MGS are "external systems" that communicate with the MGS over message ports. [FREE88] Messages have security labels, message ports are multilevel devices, and the expected access constraints are enforced between messages and message ports. In addition, the model contains limited information about the structure of messages and declares that any delivered message must be legitimately "derived from" messages received by the system at the same security level. In other words, no spoofing or regrading is allowed. The MGS model and FTLS are, in part, partitioned along protocol layers.

---

[18]The words "subject" and "object" are used somewhat differently in these presentations than in the *TSEC,* in order to convey an analogy between the presented network models and more traditional access control models, TSEC requirements pertinenting to controlled process subjects and storage objects do not always carry over in the manner suggested by the analogy.

[19]*en-tel'-e-key- the* realization of form-giving cause (as contrasted with potential existence).

In some cases lower-level component models involve system-only messages which are not visible at the system interface. [FELL87, FREE88] In these cases special security labels are used to separate system-only messages from normal client messages so that privileged information in system-only messages cannot accidentally leave the system. Lower level component specifications had to distinguish among several kinds of messages, such as acknowledgments and other "protocol" messages as well as between encrypted and plain text messages. Messages are split into header and data components. In networks, some understanding of object structure is essential to enforcing security, and this understanding necessarily turns up in derived security requirements for network components.

In some cases, the network model may be trivial and therefore unneeded. A network system consisting of single-level hosts connected by one-way links, for example, might have only the obvious requirement that the level of a destination host dominate that of a receiving host. Even this simple case may be worth modeling, if the cascading problem is considered. Considering the cascading problem may determine when network connections increase the risk of compromise. [see MILL88]

4.2.4 INDIVIDUAL COMPONENT SECURITY MODELS

Like other component models, individual component security models should accommodate relevant external-interface requirements inherited from their containing networks. An individual component which contains entities that are not security critical must contain a reference monitor, and its model should constrain the reference monitor interface. In this case, the model will include a structure analogous to that presented in Section 3.2, including internal requirements and rules of operation. Correctness of the model can be proved by showing that the rules of operation imply both internal- and external-interface requirements. For some components, this proof may involve showing that the internal requirements already imply the external-interface requirements. For others, the internal requirements may apply only to the internal reference monitor interface, so that they are unrelated to the external-interface requirements and have properties that are enforced by different software.

As with other kinds of computing systems, an individual component may contain subjects that do not act on behalf of particular users; the *TNI* refers to these as *internal subjects*. In the case of networks, it is common for internal subjects to exist outside the TCB, as in the case of routing protocols which are not involved in security enforcement. In some components, all of the subjects are internal, and the isolation of untrusted processes is so simple and straight-forward that modeling of the reference monitor interface is unnecessary.

An individual component may lie entirely within the NTCB. In this case, its external-interface requirements may already be sufficient as a description of component policy enforcement because an *internal-security model* offers no essentially new insights. Alternatively, a simplified functional description, in the form of an FTLS or rules of operation, may be useful in showing how the component goes about meeting its external-interface requirements. In this case, the functional description should provably imply the external-interface requirements.

4.2.5 UNDERLYING MODELS OF COMPUTATION

State machines have been successfully used as a basis for security modeling in several network evaluations. However, concerns about their convenience for partitioning a network TCB into components have been expressed by some researchers and system developers. State invariants may not be testable in practice because they can discuss widely separated state components that cannot be accessed in a timely fashion. Two elementary state transitions may be concurrent, leading to a partially ordered view of event histories in which two events can occur without either fully preceding the other. Parts of the secure state may be replicated in different locations, which imposes consistency requirements as well as forced discrepancies between the system state and the Cartesian product of the component states. [FELL87] Because of these perceived inconveniences, various history mechanisms have been used as an alternative to state machines, including I/O histories [GLASS87; FREE88], time-stamped buffer histories [BRIT84], and event histories [MCCU88]. Event histories are essentially the "trace" semantics for Hoare's communicating sequential processes. [HOARE85]

The differing approaches used in the above security modeling efforts underscore the fact that a formal security model depends on an underlying model of computation. Models of computation that could reasonably be used for networks include modal logic with communication based on shared variables [MANN82, PNUE81], axiomatically defined process algebras [MILN83, HOAR85], and Petri nets augmented with data flow primitives [AGER82, CHIS85]. Specification languages based on process algebras [EIJK89] and on communicating state machines [DIAZ89] have been developed for describing communications protocols and may be adaptable for use in writing network security models.

In general, the underlying model of computation for a security model should be chosen to facilitate the statement and proof of the key security properties that will be studied. In the case of networks, several different options have been tried and many more are plausible, but no preferred approach has yet been established.

4.3 DATABASE MANAGEMENT SYSTEMS

The field of database security is relatively new and has provided a large variety of questions and issues, some of which are outlined in the following paragraphs. As a result, there are many unresolved issues, and significant effort may be needed to obtain a good security design. This section is based in part on observations drawn from works by Hubbard and Hinke [HUBB86, HINK90], from the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria (TDI),* [NCSC91] and from similar security modeling issues for other trusted applications.

Perhaps the most obvious modeling issue is how the DBMS security model should be related to the DBMS data model, that is, to the high-level user's model of how the DBMS manages data. Can the security model serve as a simplified data model? If so, is it consistent with the data model supplied to users of the system? Are all of the major entities in the data model covered in the security model? In the case of a relational DBMS, for example, does the model and/or accompanying documentation explain how relations, views, records, fields, and view definitions relate to the appropriate entities (e.g.,named objects) of the security model?

Commercial database systems are commonly designed to run on a variety of different operating systems. This suggests that a database system might be modeled separately from its underlying operating systems, if there is a common explanation of how its security features interact with those of the underlying operating systems. Database systems raise a variety of security policy issues that are not covered in the *TCSEC.* The large quantity of user-supplied information found in typical databases increases the potential for erroneous data (and data classifications) as well as the potential for the unauthorized release of information due to aggregation and inference. However, database systems can partially automate the labeling of data through the use of content-dependent classification rules, thereby eliminating some kinds of user errors.

4.3.1 SYSTEM STRUCTURE

Many of the structural considerations regarding database systems apply to application systems in general. [cf PAYN90] A discussion of these considerations is followed by DBMS-specific observations about the design of objects for mandatory and discretionary access control.

A DBMS (or similar application) can be built on an existing trusted system or can be built from scratch. A third alternative is to build on a modified system, but, in terms of modeling, this is similar to starting from scratch. Building on a trusted operating system offers a well-defined and well-documented basis that provides process isolation and user authentication. Depending on the DBMS design, the operating system may also provide mandatory or discretionary access control. However, it may be necessary to hide or "turn off" some of the original security mechanisms, if they are inconsistent with and impact the operation of the DBMS security policy. The DBMS may also need to hide the original TCB/user interface, if the DBMS provides a significantly different user interface than the underlying operating system. Starting from scratch can give a more unified design, may simplify the security modeling effort, and is particularly appropriate for database machines with simple, built-in operating systems.

The DBMS-supplied notion of security can be presented in an external-interface model that gives the intended user-view of security. If the DBMS is built on a trusted operating system, the external model can be followed with an internal requirements model that identifies the role of the underlying operating system and its security policy model. The external-interface model itself will be more relevant to users of the application if it emphasizes concepts associated with the DBMS. Terminology found in the OS security model, by way of contrast, may be inappropriate either because it refers to entities hidden by the application or is very abstract in order to apply to a wide range of product applications.

The security models associated with a trusted DBMS and its underlying operating system illustrate several issues that arise in other large applications. To ensure that the system design is consistent with the security policy, it is necessary to provide evidence that the model (or models) of underlying subsystems support the application policy. The decomposition of an overall security policy model into submodels for subsystems is particularly advantageous when subsystems are developed by different groups over a protracted period of time. Despite efforts to keep the system model abstract, there may be inconsistencies between it and off-the-shelf subsystems used in the development. As a result, the system security model will need to be maintained in an interactive process as the design and product selection proceed.

The granularity of storage objects for database systems is typically finer han that of the underlying operating system. The distinction between active and passive entities tends to be somewhat blurred in the case of object-oriented systems. Existing examples suggest that the data model has a strong influence on the mandatory security policy, as can be seen by comparing relational [e.g., DENN88], functional [THUR89], entity/relationship [GAJN88], and object-oriented security models [e.g., JAJO90; KEEF89b; LUNT90; LARR90]. These observations suggest that the underlying OS MAC facility may well need to be supplanted. Database entities may be viewed as multilevel structures that are divided up and stored in single-level files. In this case, the DBMS might contain non-TCB subjects operating at all security levels used by the DBMS. This approach is illustrated by the LOCK Data Views [HAIG90] and SeaViews [LUNT88b] systems. Alternatively, the DBMS itself may take responsibility for the assignment of security labels so that entities which are single-level objects to the operating system are multilevel data structures to the DBMS, as in the ASD [GARV90] and MITRE prototypes [DAVI88]. In either case, it is important to understand the relationship between database entities and the storage objects of the underlying operating system, and it is appropriate to explain and perhaps formally model the protocol for passing objects between the DBMS and the underlying operating system.

The named objects in a DBMS may differ not only from those of the underlying operating system but also from the storage objects of the DBMS itself. User views are traditionally used as a means of controlling access in relational database systems [KORT86, Ch.13.2]; and it is reasonable to use views as named objects.[20] In this case a user may have access to several overlapping views so that the ability to access a given piece of information would depend on the user's view of it. Consequently, the user's ability to access that information might not be immediately apparent. In degenerate cases, two named objects might coincide or, similarly, a given object might have several names.

4.3.2 INTEGRITY MECHANISMS AND POLICIES

Since pragmatic approaches to data integrity are, in varying degrees, part of most database systems, [KORT86, Ch 13.3; FERN81, Ch.8] the modeling of DBMS integrity policies may lead to a better formal understanding of integrity that can be applied much more generally. Issues of interest include whether integrity constraints obscure other critical security concerns, whether there are built-in mechanisms for monitoring or controlling the propagation of inconsistencies, and whether users are able to learn which integrity checks succeed or fail for a given piece of information.

Simple data integrity checks such as “A record field of type ‘month’ should be an integer between 1 and 12” guarantee structural integrity. If such checks are enforced on input, a corresponding integrity failure is unlikely, and therefore significant. Labeling constraints of the sort discussed in Section 4.3.5 can be modeled in much the same way as simple integrity checks; they enforce a form of label integrity.

A more complicated check such as “A record field of type ‘department’ should only be populated by department names, of which there are currently 93,” however, can be invalidated in at least two ways, by adding a misspelled department and by altering the set of official department names. In more complicated cases, it may not be possible to guarantee data integrity because inconsistencies do not uniquely identify an erroneous entry, so that inconsistencies in the database must be allowed for.

A model of a data integrity policy can show how information about integrity checks is associated with data and how it propagates during data processing. For example, indication of

[20] With sufficient restrictions, it appears tat view instances can be used successfully for storage objects in order to maintain some compatibility between MAC and DAC entities. [cf KNOD88]

whether a given record satisfies a particular integrity check is a security attribute that might be modeled as a category in an integrity label. The model could explain how this attribute is handled during data processing.

Finally, some system integrity issues appear to be unique to secure database systems. The scheduling of single-level updates associated with a series of multilevel updates has to be both correct and efficient, and it must allow secure recovery after system failure. [KEEF90]

4.3.3 AGGREGATlON

The aggregation of less classified information to form a more highly classified body of information is illustrated by the following example: locations of individuals are unclassified, but total troop strength in a given area is classified, suggesting that a comprehensive database of troop locations should itself be classified. In some cases, aggregation problems can be handled by separating individual pieces of information into several or many categories. Once this is done, aggregate levels are automatically represented by a set of categories and thus by a higher security level. With this approach, no explicit modeling of aggregation control is needed.

An alternate approach is to place objects in "containers" that are classified at a higher level and allow them to be read only by subjects that are at the higher level and by TCB subjects that are authorized to extract individual objects as a form of controlled downgrading, as in the SMMS model. [LANDS4] Methods for simulating containers in a relational database system have been shown by Meadows. [MEAD90]

Yet another approach to controlling aggregation is to keep track of user queries and provide context dependent classification of query results. In this case, the first N items from an aggregate might be automatically downgraded, but the remaining items could only be obtained by users acting at the level of the aggregate. [HAlG90; cf LUNT89] This approach can be formalized by modeling the access histories of containers.

A rather general approach to aggregation is to introduce an aggregate level function, g, that associates each set of data items with a security level. One expects that if H is a subset of K, then $g(H) \leq g(K)$. Additionally, one expects that, for any A, B, and C, if $g(A) \leq g(B)$, then $g(A \cup C) \leq g(B \cup C)$, since the information in $A \cup C$ is obtainable from information at levels dominated by $g(B \cup C)$, since $g(A) \leq g(B) \leq g(B \cup C)$ and $g(C) \leq g(B \cup C)$. Meadows provides further information regarding the construction of aggregate-level functions. [MEAD90b]

4.3.4 INFERENCE

In contrast to aggregation, *inference* allows new information to combine with preexisting knowledge in a given environment, obtaining results that are not contained in the original aggregate. The inference of restricted information from authorized queries and database information is illustrated by the following examples:

1. The total.quantity of an item is classified, but its total cost and cost per item are not; two unclassified queries suffice to retrieve a piece of classified information. [DENN82]
2. An uncleared user query of the form, "Give me the contents of all containers which contain the fact that Flight 127 is carrying bombs to the front," might elicit a response of "access denied" because one or more of the requested records is classified, thereby indicating that

Flight 127 is secretly carrying bombs to the front. [WISE90]

3. A customer's bank balance is restricted information. A user of a statistical database obtains the total of all bank balances for bank customers in Smalltown, along with a corresponding list of all customer names (of which there is only one). [KORT86]

The first inference example can be addressed by classifying one of the facts which led to the inference, by enforcing separation of duty so that no uncleared user is authorized to get both facts, or by maintaining user access histories and providing context dependent classification of query results. [HAIG90] A possible objective for the modeling effort is to provide enough information to see which options are available in the modeled system.

The second example looks superficially like a covert channel associated with error returns but is potentially more dangerous because it is so easy to use. It can be avoided by redefining the query semantics so that the phrase "all containers" implicitly refers to all containers at or below the user's level. With this change, the appropriate query response is no such records exist." A strong model and model interpretation would simply rule out this example.

The third example underscores the fact that inference problems also arise in commercial systems. [cf KORT86, Ch. 13.5; FERN81, Ch. 13] The need to model inference control thus extends beyond traditional multilevel security.

An extensive quantitative analysis of the inference problem may be found in *Crytography and Data Security* [DENN82] In the LOCK Data Views effort, much of the inference problem is delegated to a database system security officer who can specify content- and context- dependent classification rules. The classification of a field in a record may depend on its value, on other fields in the record it is displayed with, or on previous queries made by the user. [HAIG90] An interesting consequence of inference problems is that there may be no lowest, safe security level for a given piece of information: an otherwise unclassified piece of information might allow users at level {a} to infer information at level {a, b} and users at level {p} to infer information at level {p, q}. Thus, {a, b}, {p, q}, and {a, b, p, q} would be admissible levels for this piece of information, whereas {a} and {p} would not. These and other considerations have led to an explicit axiomatization of inference in terms of an "infer" function [HAIG90] that could be included in a model's definition of security in order to explicitly capture an inference control policy.

Inference constraints may be enforced during database design, updating, and query processing. In principle, enforcement need only occur during query processing, but this approach may also be the least efficient. [cf KEEF89, HINK88, ROWE89, FORD90] Thus, locus of policy enforcement may be a significant factor in modeling inference control.

4.3.5 PARTIALLY AUTOMATED LABELING

As discussed in Section 3.2, the usual labeling paradigm is that users know the sensitivities of their inputs and supply these sensitivities when the information is being entered. In the case of databases, however, information about object classifications may not be available when a database schema is defined. Including this classification information as part of the schema definition not only provides a uniform approach to classification, but also improves labeling accuracy by allowing specially authorized users to assign or constrain object sensitivity levels.

In actuality, information rather than data or containers for data is classified.[21] Consequently, a rule for classifying storage objects in a database may depend not only on the particular object but also on the data it contains and on how the data is used, as is illustrated by the following examples:

1. A ship's location might be unclassified when it was in a U.S. port but classified if it were fifty miles from Tripoli. [GRAU90]
2. Names of manufacturing companies are unclassified but most information about a particular spy plane, including its manufacturer, is classified.

In the first example, classification is content-dependent. As noted by Graubart, content-dependent classification is antithetical to the tranquility principle. [GRAU90] In the second example, whether or not the name of the plane's manufacturer is classified depends on its use in a given context. Lunt has argued that context-sensitive classifications are easily confused with inference problems [LUNT89] and that object-oriented systems are especially appropriate for handling context dependencies. [LUNT90]

Lack of tranquility exposes what has been called the "multi party update conflict" problem. Two users working at different classifications attempt to store conflicting reports, leading to questions about which report and which classification is correct, and whether or not the mistaken user should be so informed. One possibility is that the less -classified user has been given a "cover story" to hide publicly observable aspects of a classified situation. In this case, a possible approach to the update conflict is to store both the cover story and the truth in the same spot in the database with the two entries being distinguished only by their classification. This approach, known as "polyinstantiation," is not the only approach to multiparty update conflicts [KEEF90b], and it may well be inappropriate in situations where cover stories are not involved. [SMIT90]

A fairly wide variety of content-and/or context-dependent classification mechanisms have been investigated, but a general approach to the problem has yet to be developed. [SMIT88] As indicated in Section 3.2.1, traditional approaches to MAC modeling begin with the assumption that data sensitivity is user-supplied. As a result, some aspects may need to be rethought when modeling policies for automated data labeling.

4.4 SYSTEMS WITH EXTENDED SUPPORT FOR LABEL ACCURACY

As mentioned in Section 3.1.5, security labels are used both to control information flow and to provide accurate s*ecurity markings* for data. These goals may well conflict, and this conflict may be theoretically unavoidable in some situations. [JONE75; DENN82, § 5.2.1] Policies and coping strategies for maintaining label accuracy have turned up in security models, whereas label accuracy itself has not yet been explicitly modeled.

The following paragraphs discuss pragmatic factors that inhibit accuracy, some coping strategies suggested by these factors, and the realization of these strategies in Compartmented Mode Workstations (CMWs). As a final example, the "Chinese wall" policy for stock market analysts is introduced and shown to be a variant of the workstation security policy.

4.4.1 FACTORS INHIBITING ACCURACY IN LABELING

---

[21] In some cases data itself is classified, as in the case of a secret code name for a classified project; but even here the intent is to classify all information associated with the named project.

In practice, several factors may interfere with accuracy in labeling, including the need to maintain labels across different working environments, the use of complex labeling schemes, and data fusion.

In many applications, the collection, processing, and distribution of information all take place in different environments. It can happen that labels are assigned during collection and used to control the ultimate distribution, but the majority of processing takes place in a system-high environment where automated access control is not supported or in a multilevel environment where not all security markings are used to control access by users. In these cases, it is necessary to accurately maintain security markings even when they are not needed to enforce access control in the information processing environment.

Some applications may involve the use of literally thousands of categories. Moreover, the required security markings may include not only a DoD classification but also code words, handling caveats, and/or release markings as well. Such markings usually form a lattice, but the rules for combining markings may be complex. [WOOD87]

The fusion of data at several security levels requires a convention for labeling fused data. Even if aggregation and inference issues are ignored, it is still necessary to find a level which dominates that of every data source. If the level of fused data can be predicted in advance, then aggregate data can be maintained in multilevel data structures, as in the SMMS. [LAND84] However, finding an appropriate aggregate level may be hindered by difficulty in combining labels or by the fact that the sources cannot be predicted in advance, either because they arrive in real time or because the avoidance of inappropriate sources is part of the fusion task itself.

4.4.2 FLOATING SENSITIVITY LABELS

If the security levels form a semi-lattice, then some form of "high water mark" policy may be used to maintain an upper bound on fused data. In such a policy, the TCB maintains the least upper bound of all data levels associated with an entity in a "floating" label. While the *TCSEC* requirements do not rule out floating labels, the MAC requirements do imply that the level of a subject must never float above the clearance of its user. At B2 and above, the covert channel requirements suggest that label data should not be exploitable as a covert channel. Unfortunately, most high watermark policies not only allow, but actually force, the existence of covert channels based on the use of floating labels. [DENN82, Ch.5.3]

The level of a process can be kept from floating too high by going to a dual-label mechanism in which a fixed label contains the subject's maximum security level, which dominates that of the floating label. In this situation, the fixed label is used only for access control and need only include those security attributes that are actually used for access control. Factors which favor acceptability of a covert channel include low bandwidth and the absence of application software that can exploit covert channels.

4.4.3 COMPARTMENTED MODE WORKSTATIONS (CMW)

The CMW design described by Woodward [WOOD87] is for a dual label system. The fixed label contains a "sensitivity" level and is the only label used for access control. The floating label contains an "information" level that consists of a second sensitivity level and additional security markings. The sensitivity levels are used to control nondisclosure and consist of DoD clearance and authorization components. The security markings form a lattice; hence, so do the information levels. The intended use of the two labels is that the fixed label enforces an upper bound on the sensitivity of information held by an entity, while the information level describes the current security level of information held. The covert channel problem resulting from the use of floating labels can lead to erroneous information labels but cannot be used to violate the access control policy enforced by the fixed labels.

A CMW security model has been developed by Bodeau and Millen. [MILL90] Its MAC policy may briefly be summarized as follows: when a user creates a new file or process, it receives a sensitivity level equal to that of the user's login level. By way of contrast, the information label for a newly created file contains the lowest possible information level. In particular, its sensitivity component is "unclassified." A *sensitivity label* never changes throughout the life of an entity, but the information label is allowed to float upwards in such a way that its sensitivity component is always dominated by the level of that entity's sensitivity label.

The rules of operation are such that, whenever an operation is invoked that involves an (intended) information flow from an entity E_1 to an entity E_2, the sensitivity labels are checked to be sure that the sensitivity of E_2 dominates that of E_1. The maximum of the information levels for E_1 and E_2 is then computed, and it becomes the new value of the information label for E_2. In addition to the properties just described, the CMW model also discusses privilege and DAC. The model does not address direct process-to-process communication, but does treat pipes as objects. As is appropriate when modeling a policy that is subject to misuse, the existence of a channel based on information labels is derivable from the model itself.

4.4.4 THE CHINESE WALL SECURITY POLICY

As presented by Brewer and Nash, the "Chinese Wall" Policy is a mandatory access control policy for stock market analysts. [BREW89] This organizational policy is legally binding in the United Kingdom stock exchange. According to the policy, a market analyst may do business with any company. However, every time the analyst receives sensitive "inside" information from a new company, the policy prevents him from doing business with any other company in the same industry because that would involve him in a conflict of interest. In other words, collaboration with one company places a "Chinese wall" between him and all other companies in the same industry. Same ness of industry might be judged according to business sector headings found in listings of the stock exchange, for example. Notice that this policy does not, by itself, prohibit conspiracies. For example, one market analyst can give information about a company in industry I to a company in industry J. Subsequently, another analyst can transfer this information from the company in industry J back to some other company in industry I. Analogous conspiracies between colliding processes, however, are explicitly ruled out in the corresponding system security policy and its model. [cf BREW89, Axiom 6]

Brewer and Nash argue that this policy cannot be modeled using models in the style of Bell and La Padula that obey tranquility. However, this policy can easily be modeled as a variant of the CMW model. Information and sensitivity levels are both sets of categories, where each category represents a different company. There is an accreditation range for information levels. A level belongs to the accreditation range if and only if it does not contain two or more companies from the same industry. The model must be extended slightly by adding information labels for users.

Every user and every controlled entity has a system-high sensitivity level, reflecting the fact that an analyst may work with any particular company. Every user starts out with an empty (system-low) information label. Each time an analyst attempts to read information in a new category (i.e., company), his information level floats up, unless it goes outside of the accreditation range, in which case his attempt is rejected. Notice that, as in the CMW example, there are thousands of categories. While the rule for combining categories is straightforward, the accreditation range is not. Additional thoughts on the use of accreditation ranges in controlling aggregation may be found in the paper "Extending the Brewer-Nash Model to a Multilevel Context." [MEAD90b]

5. MEETING THE REQUIREMENTS

At evaluation class C1 and above, a "description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB" is required. For evaluation classes B1 and above, this requirement is supplanted with explicit security modeling requirements. This section contains a listing of these requirements, followed by a discussion of how a security policy model can meet these requirements, as well as satisfy related assurance and architectural requirements.

5.1 STATED REQUIREMENTS ON THE SECURITY MODEL

Requirements that are new at a given evaluation level are presented in bold face.

5.1.1 B1 REQUIREMENTS

1. **An informal or formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system. An informal or formal description of the security policy model enforced by the TCB shall be available.**
2. **[The security model] shall be demonstrated to be consistent with its axioms.**
3. **An explanation [shall be] provided to show that it is sufficient to enforce the security policy.**
4. **The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.**

5.1.2 B2 REQUIREMENTS

1. **A formal** model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system. **A formal** description of the security policy model enforced by the TCB shall be available.
2. [The model] shall be **proven** consistent with its axioms.
3. [The model shall be] **proven** sufficient to enforce the security policy.
4. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.

5.1.3 B3 REQUIREMENTS

1. A formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system. A formal description of the security policy model enforced by the TCB shall be available.
2. [The model] shall be proven consistent with its axioms.

3. [The model shall be] proven sufficient to enforce the security policy.
4. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.
5. *A convincing argument shall be given that the DTLS is consistent with the model.*

5.1.4 A1 REQUIREMENTS

1. A formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system. A formal description of the security policy model enforced by the TCB shall be available.
2. [The model] shall be proven consistent with its axioms.
3. [The model shall be] proven sufficient to enforce the security policy.
4. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.
5. A convincing argument shall be given that the DTLS is consistent with the model.
6. **A combination of formal and informal techniques shall be used to show that the FTLS is consistent with the model. This verification evidence shall be consistent with that provided within the state-of-the-art of the particular National Computer Security Center endorsed formal specification and verification system used**.
7. During **the entire life cycle, i.e., during the design,** development and maintenance of the TCB, a configuration management system shall be in place... that maintains control of changes to the formal model....

5.2 DISCUSSION OF THE B1 REQUIREMENTS

1. Provided Model of the Security Policy Enforced by the TCB. The decision of whether to use a formal or informal model is a matter of choice, as can be seen from the *TCSEC* Glossary. An informal security policy model should be presented with enough precision that a formal mathematical model could be constructed if needed. A formal security model must conform to accepted standards of mathematical rigor.

The security model must present a definition of security and an enforcement policy (i.e., rules of operation) for controlled system entities. The controlled entities may be correctly interpreted as storage objects, devices, processes, and other controlled system resources. The model must present the system's notion of access and explain how access checks (or constraints) succeed in preventing unauthorized access. Thus, the model must consist of more than just a high-level description of security requirements.

Since formal proof is not required, a firm distinction between abstract security requirements and concrete enforcement mechanisms may be unnecessary. It is absent from the [NCSC88] definition of a security policy model. A distinction between the reference monitor and other portions of the TCB is also not required, but the role of TCB subjects in the enforcement of the security policy must be taken into account. Moreover, in the case of a retrofitted system that did not originally support trusted user roles, the interface between TCB subjects and the basic system kernel may be too complicated for TCB-subject modeling to be valuable.

2. Demonstration of Internal Consistency. The second modeling requirement covers two related concerns. The model must not contain inconsistencies. In other words, it must not contradict itself. Moreover, a formal or informal demonstration must show that the modeled enforcement policy is sufficient to achieve any modeled security requirements ("axioms"). In other words, the rules of operation must imply the definition of security.

3. Explanation of Sufficiency to Enforce the Security Policy. An explanation must be provided to show that the model's description of the security policy is adequate-that the model leads to a system whose TCB enforces the advertised system security policy. For the model to be sufficient or adequate in this sense, it must include key ideas used in the design of the security enforcement mechanisms. The resulting rules of operation should provide a basis for understanding how the system's main security enforcement mechanisms enforce the security policy. In the case of networks and other complex systems, models of key subsystems are needed in addition to a model of the entire system. In particular, "the overall network policy must be decomposed into policy elements that are allocated to appropriate components and used as the basis for the security policy model for those components". [NCSC87, Sec. 3.1.3.2.2]

The system security policy itself must include the minimum requirements of Section 3.1.1 of the *TCSEC,* and the model should tailor Section 3.1.1 to the particular system at hand. The access control requirements in Section 3.1.1 must always be modeled, whereas the need to model the labeling and object reuse requirements will vary from one system to the next. [cf NCSC87, Sec. 3.1.4.4]

The security policy model must include a description of DAC. An explanation of how DAC interacts with MAC is encouraged but not explicitly required. In particular, there is no *a priori* requirement for DAC objects to coincide with MAC objects.

The security policy model must include a description of MAC. The decomposition of sensitivity levels into clearance, nondisclosure category, or other components need not be explicitly included unless such decomposition is essential to an understanding of the model. Flagrant examples of illegal information channels may be regarded as MAC policy violations, even though a covert channel analysis is not required. More specifically, a documented (or trivially inferred) use of a system function must not result in an illegal transfer of information.

Modeling of the following security policy requirements may be useful, but is traditionally not required:

- A process acting on behalf of a user must have a label that is dominated by the clearance and authorization of that user.
- Information flowing across a device must have an implicitly or explicitly associated sensitivity level, according to whether the device is classified as single-level or multi level.
- Object reuse and process initialization do not happen in such a way as to allow unauthorized disclosure.
- In a network, the overall network security policy is enforced by the NTCB.
- Additional vendor supplied security requirements, whether derived from governing regulations or customer needs, may be enforced by the TCB.

4. TCB Protection Mechanisms and Correspondence to Model. The *TCSEC* modeling requirements must be met in away that is consistent with the needs of the system development process and with actual TCB protection mechanisms. The vendor must supply a model interpretation showing how the rules of operation in the model relate to the actions of the TCB. The model interpretation must address the reference monitor interface and show how subject instructions are accounted for.

The following aspects of a system do not have to be modeled although their representation in the model may be useful:

- Error diagnostics,
- The contents of storage objects and other controlled entities,
- The scheduling of subjects and their synchronization with external inputs,
- Internal TCB structure,
- Portions of the TCB that do not support user-requested computation (e.g., security-administrator functions), and
- An individual network component that has a very simple (or nonexistent) reference monitor and contains no subjects which act on behalf of users.

5.3 DISCUSSION OF THE B2 REQUIREMENTS

1. Provided Model of the Security Policy Enforced by the TCB. The model must be written in a formal mathematical notation; either mathematical English or a well-defined formal specification language is acceptable. From the *TCSEC* Glossary definition of a formal security policy model, it is clear that the model must describe both what security is and how it is enforced. As discussed in Section 3.2, the definition of security can be formalized using either external-interface requirements or internal requirements on controlled entities. The explanation of how security is enforced typically takes the form of rules of operation.

2. Demonstration of Internal Consistency. A mathematical proof must be given showing that the rules of operation ensure satisfaction of the modeled security requirements.

3. Explanation of Sufficiency to Enforce the Security Policy. The proof referred to in this requirement is an informal, rather than a mathematical proof, because sufficiency depends on the typically informal security policy. The intent of this documentation requirement is that stronger evidence of sufficiency be provided at B2 than at B1. If the system has a novel approach to mandatory or discretionary access control, it may be necessary to model both what the system does and what Section 3.2.1 of the *TCSEC* requires and then to prove that the system does what is required. Stronger evidence of sufficiency is possible, if the model is compatible with other B2 security, accountability, and assurance requirements. For this reason, explicit inclusion of the following requirements may be useful:

- The TCB shall support the assignment of minimum and maximum security levels to all attached physical devices. These security levels shall be used by the TCB to enforce constraints imposed by the physical environments in which the devices are located.
- [The TCB design shall] separate those elements [of the TCB] that are protection-critical from those that are not.
- The TCB modules shall be designed such that the principle of least privilege is enforced.
- The user interface to the TCB shall be completely defined and all elements of the TCB identified.

- The TCB shall support separate operator and administrator functions.
- The system developer shall conduct a thorough search for covert storage channels.
- The TCB shall support a trusted communication path between itself and the user for initial login and authentication. Communications via this path shall be initiated exclusively by the user.

If separate security models are given for the security kernel and other components of the TCB, then the pieces must fit together correctly, and arguments should be given to the effect that overall system security is achieved and necessary checks are not inadvertently omitted. As indicated in Section 3.2, the notion of which storage channels are covert is partially reflected in the security model, and the inclusion of information flow requirements in the model's definition of security can reduce the effort needed to perform covert channel analysis.

4. TCB Protection Mechanisms and Correspondence to Model. The B2 requirements for validation of TCB protection mechanisms are basically the same as those discussed at the end of Section 5.2, except for an implicit completeness requirement. This requirement is that "the TCB shall enforce a mandatory access control policy over all resources (i.e., subjects, objects, and I/O devices) that are directly or indirectly accessible by subjects external to the TCB." This requirement should be reflected in the model interpretation, and its satisfaction may require extensions to the model in order to account for all accessible resources in the implementation. The model interpretation itself may be accomplished in two steps, by first mapping the model to the DTLS and then the DTLS to the TCB implementation.

5.4 Discussion of the B3 Requirements

The B3 requirements include the B2 requirements as well as several new requirements.

1, 2. Provided Model and Internal Consistency. The basic structure of the security model is the same at B3 as at B2.

3. Explanation of Sufficiency to Enforce the Security Policy. There is one additional security policy requirement at B3, namely that access controls "shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object. For each named object, it shall be possible to specify a list of named individuals and a List of groups of named individuals for which no access to the object is to be given."

In addition, the model may usefully reflect, and must be consistent with, the following B3 architectural assurance requirement: "The TCB shall be designed and structured to use a complete, conceptually simple protection mechanism with precisely defined semantics. This mechanism shall play a central role in enforcing the internal structuring of the TCB and the system. Significant system engineering shall be directed toward minimizing the complexity of the TCB and excluding from the TCB modules that are not protection-critical."

4, 5. TCB Protection Mechanisms and DTLS Correspondence to Model. The validation of TCB protection mechanisms is normally performed by showing that the model is an abstraction of the DTLS and then mapping the DTLS to the actual TCB implementation. (See Section 2.3.6.)

5.5 DISCUSSION OF THE A1 REQUIREMENTS

1-5. Previously Introduced Requirements. The first five requirements are the same at A1 as at B3 (except for the fact that the FTLS rather than the DTLS is mapped to the implementation). The requirements for a formal covert channel analysis invite, but do not require, the use of information flow or similar external-interface models.

6. FTLS Correspondence to Model. The main new requirement at this level is that a "formal" mathematical proof be given showing that the security model is an abstraction of the FTLS. This proof may be carried out either by hand or with the use of an endorsed formal verification system. In either case, the vendor is responsible for providing an understandable, logically correct justification of correspondence. This justification normally begins with the formulation of a rigorous conjecture to the effect that the model is an abstraction of the FTLS. If part or all of the proof is presented to a verification system, then the vendor must demonstrate that the presented portion is correctly codified in the formal language of the verification system. This demonstration may require informal argument and an appeal to the semantics of the system's formal specification language.

There are two common ways of showing that the FTLS is consistent with the model: comparing the FTLS with the rules of operation and comparing the FTLS directly with the model's definition of security. In the latter case, the rules of operation and model interpretation are similar in purpose to the FTLS and its implementation correspondence, respectively. This parallelism of requirements does not necessarily imply duplication of effort, however, because there are no explicit requirements which force a distinction between the FTLS and the rules of operation.

7. Configuration Management for the Model. The requirement that the model and related documentation be maintained under configuration management is not a modeling requirement, perse. However, because of this requirement, extra care is needed to ensure that the model is given in a form that contributes to its maintainability.

APPENDIX A. SECURITY LEVELS AND PARTIALLY ORDERED SETS

Policies that control information flow often rely on partially ordered sets of security attributes. This appendix introduces partial orderings as they relate to information flow and presents basic facts about partial orderings that are relevant to the modeling and implementation of label-based security policies. Many of these facts can also be found in the undergraduate text [ABBO69].

Most processes satisfy two basic information flow properties:

reflexivity: A process can access any information it possesses. That is, information can always flow from a process to itself.

transitivity: If information can flow from process P_1 to process P_2 and can flow from P_2 to P_3, then information can flow from P_1 to P3.1[22]

These two properties of information flow determine a preordering relation between processes. If processes are labeled in such a way as to make economical use of label values, then the following property may also hold as well:

antisymmetry: If information can flow from a process with label L_1 to a process with label L_2, and conversely, then $L_1 = L_2$.

Thus, the intended relationship between labels and information flow leads to consideration of a reflexive, transitive, antisymmetric relation on label values, that is, a partial ordering. The intended relationship between information flow and this partial ordering can be expressed by saying that $L_1 \leq L_2$, if information is allowed to flow from controlled processes with label L_1 to controlled processes with label L_2. This relation is traditionally referred to as dominance: L_1, is dominated by L_2 if and only if $L_1, \leq L_2$.

The following sections introduce basic terminology and show how partial orderings can be constructed and manipulated through the use of embeddings, Cartesian products, and dual orderings.

A.l TERMINOLOGY

Formally, $\leq$ is taken to be a partial ordering on a set $\wp$ whose elements are referred to as levels. A pair of the form $(\wp, \leq)$ is a partially ordered set. For any levels L_1 and L_2, $L_1 < L_2$ if and only if $L_1 \leq L_2$ and $L, \neq L_2$. The lowest, or minimum, level of $\wp$, if such exists, is that level which is dominated by all other levels in $\wp$. Similarly, the highest, or maximum, level dominates all other levels. A minimal level, by way of contrast, is one that fails to dominate any level other than itself. Similarly, a maximal level is not dominated by any other level. If there are several maximal levels then there is no maximum level.

Traditionally, the lowest level among all levels in a given system is referred to as system-low. Similarly, the highest level is system-high. However, these levels do not always exist. In Figure A.1, the highest level is *H*. There are two minimal levels, *L* and *L'*, and, therefore, no lowest level.

---

[22] This is a worst-case assumption; P_2 might be constructed so that information from P_1 was never passed along to P_3.

(The intended partial ordering here is the transitive, reflexive closure of the relation actually pictured, so that $L \leq H$, by transitivity, and $H \leq H$, by reflexivity.

)

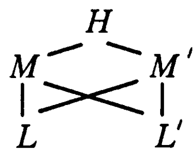

Figure A.1. A Partial Ordering

The greatest lower bound of two levels, L_1 and L_2, is the highest level that is dominated by both L_1 and L_2, provided such a level exists. Similarly, the least upper bound of L_1 and L_2, is the lowest level that dominates both L_1 and L_2, provided such exists. In the above diagram, H is the least upper bound of M and M' . $H$is also an upper bound of L and L_1, but not the least such, because M and M' are both lower. Moreover, L and L_1 do not have a least upper bound, because the set of levels greater than both L and L_1 contains two minimal elements, namely M and M'. A partial ordering in which all pairs of levels have a least upper bound is a semilattice. A semilattice in which all pairs of levels have a greatest lower bound is a lattice.

Two levels L_1 and L_2 are incomparable if and only if L_1 neither dominates nor is dominated by L_2. A set of * levels is linearly ordered if and only if no two elements of * are incomparable. In the above diagram, M and M' are incomparable. The set $\{L, M, H\}$ is Iinearly ordered (with $L < H$, by transitivity).

If C is any finite set, then the set of all subsets of C,*(C), is partially ordered by the set-inclusion relation. In other words, L_1 is dominated by L_2 if and only if L_1 is a subset of L_2; in symbols, $L_1, \subseteq L_2$. This gives a lattice ordering on $P(C)$[23]. The lowest level of? (C) is the empty set; the minimum levels above the empty set are the singleton sets of the form {c}, where c belongs to C. These singleton levels are variously referred to as categories or atoms. The highest level is C itself.

A.2 EMBEDDINGS

When implementing a partially ordered set of levels, it is always possible to choose a larger implementation set whose additional elements are simply not used. In particular, any partially ordered set can be fully embedded in one of the form (P(C), $\subseteq$); that is, given any partial ordering $\leq$ on a set ∗, there is a one-to-one mapping e into a set of the form P(C) such that, for any L_1, L_2 in ∗, $L_1 \leq L_2$ if and only if $e(L_1) \subseteq e(L_2)$. In fact, one may take $e(L) = \{L' / L' \leq L\}$. In the case of linearly ordered sets, the embedding $f(L) = \{L', /L' < L\}$ also works. For example, if ∗ is a linearly ordered set of 16 clearance levels, then * may be fully embedded in a set of fifteen categories. A word of caution is in order regarding full embeddings: they need not preserve least upper bounds. If L is the least upper bound of L_1 and L_2, then $e(L)$ is an upper bound of $e(L_1)$ and $e(L_2)$, but need not be the least upper bound. Greatest lower bounds can also fail to be preserved under full

[23]P(C) is traditionally referred to as the "power set of C."

embeddings.

As a further application of embeddings, consider the problem of specifying a set $\Re$ of levels at which a given device may pass information. Assume $\Re$ is convex in the sense that L $\in \Re$ whenever $L_1, L_2 \in \Re$ and $L_1, \leq L \leq L_2.$ Assume also, that $\Re$ is a subset of the system's accreditation range A. The partially ordered set A can always be embedded in a lattice $*$ in such a way that, for some device minimum, *min*, and some device maximum, *max,* $\Re = \{L \in A \mid min \leq L \leq max\}$. Thus, arbitrary *convex sets* may be specified as device ranges, at least if one allows unaccredited levels in the specification of the device range.

A.3 CARTESIAN PRODUCTS

The effect of simultaneously applying two label-based policies to a set of entities is the same as applying a corresponding composite policy to entities, each of which has a single composite label. If the partially ordered sets for the two policies are $(*, \leq)$ and $(*', \leq')$ then the composite label has values in the Cartesian product, $(*,\leq) \times (*', \leq') = (* \times *', \leq'')$, where the new partial order, $\leq''$, is defined by $\langle L_1, L_1', \rangle \leq'' \langle L_2, L_2' \rangle$ if and only if $L_1 \leq L_2$ and $L_1' \leq L_2'$. The sets $*$ and $*'$ are referred to as components of $* \times *'$. A sample Cartesian product ordering is illustrated in Figure A.2. Notice that, in the Cartesian product, the level $\langle S, \emptyset \rangle$ is incomparable to the level $\langle U, \{d\} \rangle$. (Some punctuation has been omitted in the third lattice diagram.)

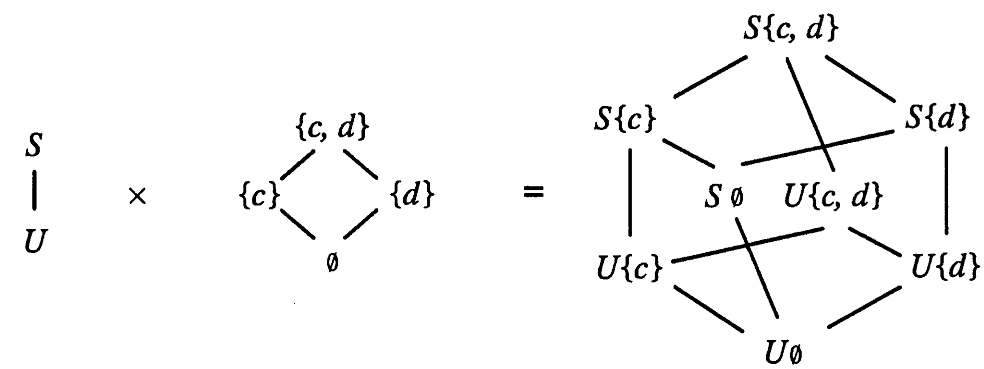

Figure A.2. A Cartesian Product Ordering

If C and D are disjoint, then the partially ordered sets $(P(C), \subseteq) \times (P(D), \subseteq)$ and $(P(C \cup D), \subseteq)$ are isomorphic, meaning that there is a full embedding which maps $P(C) \times P(D)$ on to $P(C \cup D)$. In particular, traditional nondisclosure labels consisting of clearances and category sets can be fully embedded into a set of the form P(A), with $A = C \cup D$.

Usually, there are several nonequivalent ways to decompose a partially ordered set into two components. For a partially ordered set of the form $(P(A), \subseteq)$; there are $n + 1$ meaningfully different ways, where n is the number of categories in A (since the first component can have any number of categories between 0 and n). In particular, the decomposition into clearance and category components could be configuration dependent, even if the labels themselves were not.

A.4 DUALITY

The notion of duality between nondisclosure and integrity that is evident in some security policies has a formal analogue for partial orderings. For a given set $*$ and associated partial ordering *, the dual ordering for (*,*); is the relation * defined by L_1 * L_2 if and only if L_2 * L_1. A Biba policy (as described in Section 3.5.3) that is based on (*, *) can be rewritten as a nondisclosure policy based on the dual ordering (*, *), so that information is allowed to flow from L_1, *to* L_2, provided L_1, * L_2. Usually, to avoid confusion, the elements of the dual ordering would be renamed so that the intended ordering is obvious from the level itself.

The dual ordering for (*(C), ⊆) is the set-containment relation, ⊇, and the lattice (*(C), ⊆) is isomorphic to (*(C), ⊇) under the mapping *e* given by *e(A)* = C| *A,* for each *A* in *(C), where C| A is the *complement set* containing those elements of *C* not in *A*. In this case, the categories of (*(C), ⊇) are the dual categories of (*(C), ⊆), that is, they are sets of the form *C*| {*c*}. In Figure A.2, {*c*} = *C* \ {*d*}), so that {*c*} is both a category and a dual category.

The ⊇ relation is the ordering used to ensure nondisclosure for "distribution" sets in Section 3.3.4. This observation suggests a method for accommodating release markings of the form "REL <countries>" through the use of MAC categories. Each country *c* is associated with the dual category C| {*c*}; a message with category set *B* is releasable to country *c*, if $B \subseteq C \setminus \{c\}$, that is, if $c \notin B$. The most sensitive of these release markings, namely "NOFORN," is represented by the largest category set, the set *C* containing all relevant countries.

In a system with a nondisclosure policy based on (*,≤) and a Biba integrity policy based on (*,*), the partial orderings that control information flow would be ≤ and *, so that the two policies can be combined to obtain a single policy pertaining to information flow that is based on composite levels taken from the partially ordered set (*, ≤)x(*, *). Suppose each of the original orderings have minimum and maximum levels. If the minimum integrity level is integrity-low, then integrity-low is the maximum level with respect to the dual ordering. Consequently, the composite ordering contains the four levels indicated in Figure A.3. Notice that the composite level which represents both system-high nondisclosure and system-high integrity is not the maximum level but the one on the right.

(nondisclosure–high, integrity–low)

(nondisclosure–low, integrity–low) **(nondisclosure–high, integrity–high)**

(nondisclosure–low, integrity–high)

Figure A.3. Extreme Points in a Product Ordering

Finally, if nondisclosure and integrity are handled similarly, the decomposition of combined levels into nondisclosure and integrity components may be treated as a system configuration decision in order to handle varying emphasis on nondisclosure and integrity.

APPENDIX B. AVAILABLE SUPPORT TOOLS

This appendix discusses the use of NCSC-endorsed formal *verification systems* for security modeling. A list of endorsed verification systems called the *Endorsed Tools List (ETL)* is maintained by the NCSC. These tools define the level of rigor which must be met by formal assurance provided for systems receiving an Al evaluation. The following paragraphs discuss verification systems with emphasis on the two currently endorsed systems, GVE ("Gypsy Verification Environment") and FDM ("Formal Development Methodology").

In general, a verification system is defined by its specification language and its reasoning mechanism. Its implementation includes tools for parsing specifications, verifying their legality, and outputting conclusions which have been verified by the system. A verification system can provide a codification of a security model and can help demonstrate that its rules of operation are consistent with its security requirements. The utility of a verification system depends on the expressiveness of its specification language, the soundness of its reasoning mechanism, the correctness and utility of its implementation, and the quality of its documentation. [cf NCSC89]

Both GVE and FDM provide a rich variety of data types and associated operations from mathematics and computer science, (including scalar types, finite sets, sequences, and records.) Both systems have interactive theorem provers that may be used to verify specifications; they accept theorem-proving commands and include facilities for printing completed proofs. Both theorem provers use specially designed systems of logic that extend many-sorted, first-order logic. Both systems come with special-purpose tools for performing covert channel analysis via shared resource matrices. These systems are suitable for writing formal state-machine models, but they require specialized training.

The writing of formal models and specifications in a formalized language has been found to be the most fruitful aspect of using verification tools. This is the conclusion of several major verification efforts, including BLACKER, the Boeing LAN, LOCK, and the Multi net Gateway. The writing of formal descriptions encourages a more precise and abstract understanding of security, forces the resolution of design issues needed to complete the model or specification, and provides a clear basis for implementation analysis. In general, both manual proofs and machine proofs provide more assurance than specifications alone. Experience has shown that machine proofs are more reliable than hand proofs but that they are also more time consuming and are still not foolproof. There are still many sources of possible error. The *TCSEC* itself is based on policy directives rather than mathematical formalisms. Rigorous, general frameworks for computer security have not yet been developed.

B.1 FDM: THE FORMAL DEVELOPMENT METHODOLOGY

FDM supports multilevel specifications at a series of abstraction levels. Each level refines the prior level by adding more detail and more functions. The FDM specification language, Ina Jo, provides a formalized notation for describing state machines that includes provisions for declaring state variables and for specifying initial states, state invariants, state transition constraints, and state transformations. Its type mechanism includes a general facility for declaring subtypes that is useful for classifying controlled entities in a security model, for example. The Ina Jo specification processor automatically generates correctness assertions to the effect that every reachable state satisfies the provided state invariants and state transition constraints.

FDM began in 1974 as an internal research project at the System Development Corporation (now UNlSYS Corporation). FDM is currently owned by PARAMAX Systems Corporation, a subsidiary of UNlSYS Corporation. Its early development was driven by specific problems in formal modeling and formal specification and verification with the goal of complementing system testing as a means of validating correctness. Since then, a series of incremental enhancements has increased its power, efficiency and portability, added new capabilities, and improved its documentation. FDM Release 12.4, the most recently endorsed version, contains two new flow tools: a shared resource matrix tool that assists in the manual analysis of potential covert channels and an automated flow tool for detecting and analyzing information flows in specifications. FDM Beta Release 12.5 contains a much improved flow tool and a new interactive tool for executing in a Jo specifications against user-supplied test cases. The FDM tools run on Sun Workstations, on DEC VAXes running Berkeley UNIX, and on Multics.

The available documentation on FDM includes the *FDM User Guide [EGGE89]*, the *Ina Jo Specification language Reference Manual* [SCHE89], and the *Interactive Theorem Prover (ITP) Reference Manual* [SCHO88]. Examples of Ina Jo based security modeling efforts may be found in [NCSC90b, CHEN90, FELL87, CHEH81].

B.2 GVE: THE GYPSY VERIFICATION ENVIRONMENT

The GVE is an interactive system that makes extensive checks to ensure legality of specifications. In particular, recursive functions must terminate, partial functions must not be applied outside their domains, and theorems may be proved. The GVE maintains user databases and supports incremental program development through the use of “header” files.

The GVE specification language, Gypsy, has been used for a variety of security models, including several that are not based on state invariants and state transition constraints. Gypsy supports the use of local name spaces called “scopes” and includes “mapping” types that can be used as abstractions of hash tables and other complex data structures. In Gypsy, states can be modeled as records whose components represent state variables. Initial states and state transitions are described with the help of a binary “with” operator, and theorems about reachable states are stated directly using the “lemma” construct. Gypsy contains a programming language portion that has a Pascal-like syntax, includes condition handling facilities and concurrent processes, and is supported by a “verification condition generator” that allows proofs of program correctness. An operational semantics has been given for a portion of Gypsy. [GOOD90]

The development of the GVE began in 1974 at the University of Texas at Austin with the goal of verifying communications processing systems. Initial work with the first version of Gypsy led to significant language simplifications and some extensions in Gypsy 2.0. In 1986, Computational Logic, Inc., assumed responsibility for GVE and completed work on Gypsy 2.1. A subset of this has been implemented as Gypsy 2.05, the current language for the Gypsy methodology. The currently endorsed implementation is GVE 13.16. Recent work has been directed towards improved documentation, performance analysis and improvement, and the development of a better configuration management system. Version 20.70 of the GVE is currently being considered for NCSC endorsement.

Available documentation on GVE includes the *Report on Gypsy 2.05* [GOOD89], *Using the Gypsy Methodology* [GOOD88], and the *Gypsy Verification Environment User’s Manual* [AKER90]. Examples of Gypsy-based security modeling efforts may be found in [DIVI90,

FINE90, FREE88, CHEH81].

APPENDIX C. PHILOSOPHY OF PROTECTION OUTLINE

This appendix outlines material suitable for a Philosophy of Protection (POP) document. Its purpose is to provide a template for vendors who desire additional guidance in producing a POP. Unless otherwise noted, required portions of this and the following appendix are those which are accompanied by an explicit reference to the *TCSEC.*

There are no *TCSEC* requirements on the organization of this material or on how it is packaged in vendor-supplied documents. If the information outlined in this and the following appendix appears in two or more separate documents, careful cross referencing will help the reader assemble a complete picture.

1. INTRODUCTION

Present "... a description of the manufacturer's philosophy of protection." [2.1.4.4]

A higher-level policy on the design and use of a trusted computing system is normally implemented by a combination of automated, procedural, and physical safeguards. The POP can show that these safeguards fit together in such away as to achieve stated security objectives. The POP is an open-ended document of varying length and content, and any relevant system-related activity may be used to substantiate the vendor's philosophy of protection; including requirements analysis, design, implementation, testing, marketing, and method of delivery. For systems in higher evaluation divisions, the POP is likely to contain several staff years of accumulated wisdom.

2. CONTROLLING SECURITY OBJECTIVES

Describe the anticipated policy for use of the computer and the effect this policy has had on the security design. In other words, describe the security objectives which guide the design and use of the system being evaluated. Alternatively, give a summary with references to a separate security policy document (see Appendix D.2).

There are several questions that should be answered when preparing this section of the POP. In general terms, what is the system to which these objectives are applied, and how does the system support these objectives? What is the system interface, and what is the security *perimeter,* that is, the portion of the system and its environment where security objectives are actively addressed? (By definition, the security perimeter contains the TCB and associated controlled entities.)

3. AUTOMATED PROTECTION

The POP should address how the implemented policy compares with the higher-level policy objectives. Is it more or less restrictive? Where, how, and why?

3.1 SYSTEM SECURITY REQUIREMENTS

The POP gives security requirements imposed on the system in order to meet the above security objectives. It indicates which requirements are refinements or extensions of *TCSEC* requirements. Distinguish between "policy" requirements that are modeled with the assurance required for the system's evaluation class and other security-relevant functional requirements.

At B1 and above, this distinction can be emphasized by merely summarizing policy requirements in the philosophy of protection and referencing their full presentation in a separate policy model document (see appendix D.4).

3.2 TCB STRUCTURE

A description of the TCB provides needed context for an explanation of how the vendor's philosophy of protection is reflected in the TCB. Moreover, "if the TCB is composed of distinct modules, the interfaces between these modules shall be described." [2.1.4.4] Relevant aspects of the TCB hardware/software architecture include the TCB interface to untrusted resources, as well as software, firmware, and hardware protection mechanisms. For classes B2 and above, much of the needed information will be contained in the DTLS and need only be summarized here.

Brief answers and/or references are recommended for the following sorts of questions. What are the protected resources? How is subject activation and deactivation accomplished? How is I/O handled? How is output labeled and what security attributes are associated with "named users"? What are the named objects used for DAC, and what are the storage objects used for MAC? How are they created and destroyed? Are there "public" objects accessible to all users?

What are the main TCB software modules? What are their interfaces to each other, to the hardware, and to untrusted resources? What is the virtual machine architecture and its relationship to the underlying hardware? What is the hardware architecture? How does it perform task management, storage management, and device control?

3.3 VALIDATION OF THE TCB WITH RESPECT TO THE SECURITY REQUIREMENTS

"Documentation shall be available that provides a description of. how this philosophy [of protection] is translated into the TCB." [2.1.4.4] This requirement may be satisfied by directly relating the avowed philosophy to the TCB protection mechanisms.

At higher evaluation levels, this requirement is largely supplanted by other similar requirements. At B1 and above, it is supported by explanations which show that the TCB satisfies the security model and, therefore, enforces the modeled portion of the system security policy. At B2 and above, it is supported by the DTLS and by documentation showing that the DTLS accurately describes the TCB interface. At Al, it is supported by a proof showing that the FTLS satisfies the model and by informal testing and analysis showing that the FTLS accurately describes the TCB interface.

4. PROCEDURAL AND PHYSICAL SECURITY MECHANISMS

In general, threats to the secure use of a system are thwarted by a combination of automated mechanisms, procedural methods, and physical constraints on the computing environment. The effectiveness of these combined safeguards can be argued by giving a taxonomy of potential threats against the system and by showing how each kind of threat is countered by a combination of TCB security requirements and related measures described in the Security Feature User's Guide and/or the Trusted Facility Manual.

5. NOTES AND CONCLUSIONS

The POP should present any other significant aspects of the vendor's philosophy of protection. It should explain what conclusions are to be drawn from the empirical and analytical evidence presented.

6. GLOSSARY

The POP should list technical terms and give definitions. It should include all terms whose usage differs with either *TCSEC* definitions or common usage. Include all terms for which multiple definitions are in common use (e. g., user, subject, object, trusted subject).

7. REFERENCES

The POP should include references to relevant design documentation.

8. APPENDIX ON SATISFACTION OF THE (e.g., CLASS B2) CRITERIA

It may be helpful to list each *TCSEC* requirement for the candidate evaluation class and show how it is met. This step is definitely optional, but it can help resolve questions about the meaning of particular requirements as they apply to particular systems.

APPENDIX D. SECURITY MODEL OUTLINE

The following outline contains a suggested organization of material for a *Security Policy Model* Document. Its purpose is to provide a template for vendors who desire additional guidance in producing a security policy model. Any or all of this material could appropriately be included in the Philosophy of Protection.

1. INTRODUCTION

The purpose of this document is to present “an informal or formal model of the security policy supported by the TCB.” [3.1.3.2.2] By definition, the security model includes a definition of security describing the policy enforced by the system as well as rules of operation giving design guidance on how to enforce the requirements in the definition of security.

A summary of the manufacturer’s philosophy of protection [cf 2.1.4.4] may help establish an appropriate context for the presentation of the security policy and model. In particular, a summary of the TCB protection mechanisms, their support for the security policy, and their role in implementing the model will help establish the relevance of the model to the system being developed.

2. THE SECURITY POLICY

“A statement of intent with regard to control over access to and dissemination of information, to be known as the security policy, must be precisely defined and implemented for each system that is used to process sensitive information.” [5.3.1] It will help avoid confusion if high-level security policy objectives are carefully distinguished from, and related to, derived policies that directly impact the design and use of the system. The policies that most need to be modeled are those giving the actual security requirements to be enforced by the system and the rules of operation showing how these requirements are enforced.

The security policy statement should answer several questions. Is the security policy composed of several different policy elements? Is the system composed of several subsets, subsystems, components, or layers? If so, for each identified portion, what security services are provided, and what security services are relied on as provided by other portions? What are the interfaces between the various portions to be modeled? How do they combine to form the complete system?

2.1 MARKING OF INFORMATION

“...it is necessary that the system mark information with appropriate classification or sensitivity labels and maintain these markings as the information moves through [and is exported from] the system.” [5.3.1.3] A marking policy is required in relation to MAC and may be appropriate for any policy based on the use of associated security attributes.

Several questions concerning the marking of information should be answered within the security model. How are security attributes assigned to controlled entities and under what authorization? To what extent are marking decisions made, remembered, and then repeatedly applied fora period of time? How is object reuse accomplished? How do security attributes propagate during the processing of information (e.g., when a new storage object is created)?

How (and under what circumstances) are security attributes presented "`then information leaves the system? What is the precise marking policy for paged hardcopy output? Are there additional marking requirements for this system that are not covered in the *TCSEC*? How is the level of a subject tied to the clearance and authorization of its user? Is a given I/O action taken on behalf of a particular user? If so, is it consistent with the clearance and authorization of that user?

The following questions are relevant at B2 and above. How are minimum and maximum device levels used? What is the policy for changing device levels? How is the marking policy allocated to different portions of the TCB? How does the marking policy interact with special user roles associated with security administration and system operation?

2.2 MANDATORY SECURITY POLICY

The security model should "... include a set of rules for controlling access based directly on a comparison of the individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought, and indirectly on considerations of physical and other environmental factors of control." [5.3.1.1]

2.3 POLICY REGARDING NEED-TO-KNOW

The security model should "... include a consistent set of rules for controlling and limiting access based on identified individuals who have been determined to have a need-to-know for the information." [5.3.1.2] "The AIS shall function so that each user has access to all of the information to which the user is entitled... but no more." [DOD88a, Enclosure 3] The model should answer the following questions: What system policy and mechanisms are offered in support of this objective? And to what extent do they support it?

2.4 ADDITIONAL SECURITY POLICY ELEMENTS

"The security policy must accurately reflect the laws, regulations, and general policies from which it is derived." [5.3.1] This portion is dictated by specific customer requirements, of which there could be many. The following list presents a typical slice through four government policy-making levels. Each level is followed by a list of examples. At each level, the boldface example is the one chosen for elaboration at lower levels:

National Policy on Secrecy, Integrity, and Availability
(Executive Order 12356 covers secrecy policy [REAG82]);

National Department Policy
(e.g., DOE, **DOD**, NIST HEW)
(Dir. 5200.28 [DOD88a], *TCSEC* [NCSC85] refine [REAG82]);

Branch Policy
(e.g., **USAF**, USN, NSA, DIA);

Command Center Policy
(e.g., Strategic Air Command, Electronic Systems Command).

3. ADEQUACY OF THE MODELING PARADIGM

The security model should explain how the security model's definition of security manages to capture essential notions of computer security and how the modeled policy relates to the intended policy and to the implemented policy. It should indicate the overall abstraction level used in the model. At B2 and above, it should explain any formalization techniques that might interfere with a correct understanding of the model.

3.1 HERITAGE

The security model should:

a. Identify any previous models and mathematical formalisms upon which the model is based,
b. Briefly describe these previous model(s); identify (and emphasize) those portions from which the model is derived,
c. Identify areas where the model has extended the base model(s) in order to provide better correspondence to policy or to model new policy elements.
d. If the model is not based on a state-machine paradigm, motivate and explain the paradigm.

3.2 SUFFICIENCY TO ENFORCE THE POLICY

The model should "show that it [the model] is sufficient to enforce the security policy." [3.1.4.4] It should also explain why a system based on the model will adequately support the security policy identified in Section 2. If the model is based on a previous security model, it should explain how differences enumerated in parts 3.1(b) and (c) above maintain or enhance the adequacy of the original approach.

3.3 RELEVANCE TO THE ACTUAL SYSTEM

The security model should briefly describe the intended interpretations of the various constructs found in the model. At B2 and above, it should also discuss whether all system resources are accounted for in the model.

4. THE SECURITY POLICY MODEL

The presentation of the model might be loosely divided into four sections: basic concepts, a definition of security, other requirements reflected in the model, and rules of operation. Some systems will have a more elaborate structure, due to distinctions between system and subsystem models or between external and internal requirements. In the case of security properties enforced by TCB subjects, the distinction between requirements and rules of operation may be moot. An informal model may legitimately be presented as part of the security policy.

4.1 BASIC CONCEPTS

This section should introduce the basic data types, constants, and operations that will be used to build the model. It also presents the underlying model of computation; specifies the various kinds of controlled entities and security attributes that will occur in the model; and identifies particular security-critical subjects, objects, or unlabeled TCB entities that will play a distinguished role in the model. It should also answer several general questions. Are there special objects that are not accessible by non-TCB subjects? If so, what is their role, and how are they protected from access by subjects outside of the TCB? Which subjects are inside the TCB; why are they considered part of the TCB? How are these TCB subjects created (i.e., statically during system initialization or dynamically by other TCB subjects)? What portions of the system policy are implemented by TCB subjects exempt from one or more constraints enforced by the reference monitor? How are devices modeled? What special properties must hold at system start-up (e.g., initial-state requirements in the case of a state machine model)? What are the security attributes of each predefined security-critical entity (e.g., security level, owner or user, access control lists, associated user roles, exemptions from access control)?

In a system with multiple subpolicies or a layered design with differing security services provided by different layers, some or all of the above questions may have multiple answers, depending on the policy or layer.

4.2 SECURITY POLICY REQUIREMENTS

In the case of an access control model in the tradition of Bell and La Padula, the security policy requirements include the following kinds of assertions or "axioms":

secure state invariants
- MAC invariants (e.g., simple security, *-property)
- DAC invariants (may legitimately be empty for some policies)
- Usage invariants (e.g., no subject can access an unused entity)
- Other policy-specific state invariants;

secure state transition constraints
- MAC constraints (e.g., tranquility)
- DAC constraints (e.g., getting access implies authorization)
- Creation constraints (e.g., assignment of security attributes)
- Other policy-specific state transition constraints.

In the case of a model in the tradition of Goguen and Meseguer, support for a mandatory access control objective would involve a statement of noninterference. It would be a statement to the effect that the system view available to a subject (or user) cannot be influenced by the behavior of other subjects (users) whose security level fails to be dominated by that of the given subject.

4.3 OTHER SECURITY REQUIREMENTS REFLECTED IN THE MODEL

The following questions are relevant at B2 and above; they may, optionally, be addressed in the model. Which objects can be viewed by someone acting in a special role associated with system operation or security administration? What activities can be accomplished only by invoking such a role (e.g., shutdown and restart the system, make backup tapes, restore files, set the clock, take devices off line, collect performance statistics, kill runaway programs, manipulate printer queues)? Which objects can be viewed or modified by the security administrator? In how many ways can the security administrator perform downgrading? What activities can be accomplished only by invoking this role (e.g., suspend and restore auditing, save audit data on tape, change object ownership, view or modify security attributes of arbitrary controlled entities, or view or modify user authentication data)?

The following questions are relevant at B3 and above and may optionally be addressed in the model. What is the purpose of each exemption allowed for subjects inside the TCB? Which TCB subjects are involved in identification and authentication and therefore need access to user authentication data? Which TCB subjects are involved with trusted path and related activities (e.g., set password, Iogout)? Does modification of discretionary access require use of trusted path? What privileges are afforded to TCB software supporting special user roles, and to what extent can these privileges be passed on to the system operator and security administrator?

4.4 RULES OF SAFE OPERATION

The model should present key ideas needed to understand the design of the policy enforcement mechanism, identify basic kinds of interactions with the TCB and explain what constraints are enforced in order to satisfy the model's definition of security.

5. VALIDATION OF CONSISTENCY

"[The model] shall be... demonstrated/proven consistent with its axioms." [3.1.3.2.2, 3.2.3.2.2] The model should demonstrate that any system which obeys the identified rules of operation also satisfies the model's definition of security. Depending on how the rules of operation are specified, it may also be necessary to justify internal consistency among the rules of operation themselves.

6. NOTES A ND CONCLUSIONS

There should be a section in the model that explains what conclusions are to be drawn from the empirical and analytical evidence presented. Significant ramifications of the security model should be mentioned.

7. GLOSSARY

The model should list technical terms and give definitions; including all terms whose usage differs with either *TCSEC* definitions or common usage. It should also include all terms for which multiple definitions are in common use (e.g., user, subject, object, trusted subject).

8. REFERENCES

The model should include references to design documentation and previous security models on which it was based.

APPENDIX E. GLOSSARY

This Glossary contains definitions of terms involved in security modeling. When possible, definitions are based primarily on the NCSC Charter [DOD88a, Definitions], the TCSEC [NCSC85, Glossary], the TNI [NCSC87, Glossary], the *Computer Security Subsystem Interpretation* [NCSC88a, Glossary], Criteria Interpretation Reports [NCSC88b], and the NCSC *Glossary of Computer Security Terms* [NCSC88]. In some cases, additional information has been added to previously given definitions in order to provide further clarification; this additional information is distinguished by its enclosure in doublebrackets ([.....])

Access *(to Information)*
: The ability and opportunity to obtain knowledge of classified, sensitive unclassified, or unclassified information. [DOD88a]

Access *(to a Resource)*
: (1) A specific type of interaction between a subject and an object that results in the flow of information from one to the other. [NCSC85; NCSC87]
(2) The ability and the means necessary to store or retrieve data, to communicate with, or to make use of any resource of an ADP system. [NCSC87]

Access Control
: (1) Restrictions controlling a subject's access to an object. [ANDE72; NCSC87]
(2) The limiting of rights or capabilities of a subject to communicate with other subjects or to use functions or services in a computer system or network. [NCSC87]

Access Control Attribute
: Security attribute used for access control.

Access Control Model
: A model that gives rules of operation showing how access decisions are made. Traditionally, an access control model involves a set of states together with a set of primitive operations on states whose behavior is defined by rules of operation. Typically, each *state* contains a set S of “subjects,” a set *O* of “objects,” and an access matrix A. For each subject sand object *o, A [s, o]* is a set of access *rights,* such as *read, write, execute,* and *own.*

Accreditation (of an AIS)
: A formal declaration by the *designated approving authority* that the AIS is approved to operate in a particular security mode using a prescribed set of safeguards. [DOD88a]

Accreditation Range (of a Net’ork Host)
: A set of mandatory access control levels for data storage, processing, and transmission. The accreditation range will generally reflect the sensitivity levels of data that the accreditation authority believes the host can reliably keep segregated with an acceptable level of risk in the context of the particular network for which the accreditation range is given. [NCSC87] in practice, several different accreditation ranges may be associated with a host. A *network* accreditation range is the set of levels that are permitted for transmission of data on the

network. The host's *network* interface accreditation range is given by minimum and maximum levels for the associated/O device. The *host* accreditation range is the set of all levels that may be used within the host itself. Finally, the host's user accreditation range is the subset of the host accreditation range obtained by excluding levels that are associated exclusively with TCB subjects, trusted user roles, or related security critical objects.i

Aggregation Problem

An occurrence when a user's right to several pieces of information results in knowledge they do not have a right to. It can happen that a user is not allowed access to a collection of data items, but is allowed access to any given item in the collection. In this case, the aggregation problem is to prevent the user (or a subject acting on their behalf) from gaining access to the whole collection through repeated accesses to items in the collection.

Antisymmetry

A relation in which no two elements are equivalent. More precisely, a relation R is antisymmetric if and only if, for all $x, y, x\,R\,y$ and $y\,R\,x$ implies $x = y$

Application-Dependent Security Model

A security model that includes security-relevant information about the semantics of a particular application; an application dependent model contrasts with a security model for a general-purpose computing system that supports a variety of differing applications. Database security models are application security models in this sense.

Assurance (Activity)

Activity aimed at achieving a level of assurance, including informal argument, mathematical proof, the performance of dynamic checks on the behavior of an AlS, and the performance of static checks on AS hardware or software.

Assurance (Measure)

A measure of confidence that the security features and architecture of an AIS accurately mediate and enforce the security policy. [NCSC88;cf DOD88a]

Attribute

See security attribute.

Automated Information System (AIS)

An assembly of computer hardware, software, and/or firmware configured to collect, create, communicate, compute, disseminate, process, store, and/or control data or information. [DOD88a]

AIS Security

Measures and controls that protect an AIS against denial of service and unauthorized (accidental or intentional) disclosure, modification, or destruction of AISs and data. AIS security includes consideration of all hardware and/or software functions, characteristics and/or features; operational procedures, accountability procedures, and access controls at the central computer facility, remote computer, and terminal facilities; management constraints;

physical structures and devices; and personnel and communication controls needed to provide an acceptable level of risk for the AIS and for the data and information contained in the AIS. It includes the totality of security safeguards needed to provide an acceptable protection level for an AIS and for data handled by an A'S. [NCSC88]

Centralized Authority
Authority to modify security attributes that is limited to the system security administrator acting on behalf of the system's owner (e.g., DOD).

Clients
People or processes accessing an AIS either by direct connections (i.e., via terminals) or indirect connections (i.e., prepare input data or receive output that is not reviewed for content or classification by a responsible individual). [cf DOD88a, definition of user]

Compatibility Property (on Directories)
The requirement that any file in a directory have a security level which dominates that of the parent directory. [WALT74]

Component (of a Network)
A device or set of devices, consisting of hardware, firmware, and/or software that performs a specific function on a computer communications network. A component is a part of the larger system and may itself consist of other components. Examples include modems, telecommunications controllers, message switches, technical control devices, host computers, gateways, communications subnets, etc. [NCSC87]

Component Security Model
A subsystem security model for a system that is the union of its modeled component subsystems.

Control Objectives
Higher-level policy objectives that constrain the design and use of a trusted computing system; specific control objectives are listed in Section 5 of the *TCSEC*.

Controlled Entity
Any system resource (e.g., process, storage object, I/O device) for which the TCB controls access by users. (See also *System Entity, Explicitly controlled Entity.*)

Convex Set (of Levels)
A set of levels that does not contain any "holes" with respect to the dominance relation. Formally, a partially ordered set * of levels is convex if and only if $L \in *$ whenever $L_1, L_2 \in *$ and $L_1 \leq L \leq L_2$

Covert Channel
A communication channel that allows a process to transfer information in a manner that violates the [intent of the system] security policy. [NCSC85] [In some cases, the offending process may be external to the system that has the covert channel.] A covert channel typically

communicates by exploiting a mechanism not intended to be used for communication. [NCSC87]

Covert Channel Analysis

Determination of the extent to which the security policy model and subsequent lower-level program descriptions may allow unauthorized access to information. Covert channel analysis properly includes all forms of covert channels, external as well as internal, and timing as well as storage channels.

Covert Storage Channel

A covert channel that involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process. [NCSC87]

Covert Timing Channel

A covert channel in which one process signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process. [NCSC87]

Denial of Service

Any action or series of actions that prevent any part of a system from functioning in accordance with its intended purpose. This includes any action that causes unauthorized destruction, modification, or delay of service. [NCSC88]

Descriptive Top-Level Specification (DTLS)

A top Level specification that is written in a natural language (e.g., English), an informal design notation, or a combination of the two. [NCSC85]

Designated Approving Authority (DAA)

The official who has the authority to decide on accepting the security safeguards prescribed for an AlS. [DOD88a]

Deterministic

An AIS characterized by having its behavior, in principle, entirely determined by its inputs.

Discretionary Access Control (DAC)

A means of restricting access [to named] objects based on the identity of [named users or] subjects and/or groups to which they belong. [NCSC85] It is not necessary for the creator of an object to control access to that object. [NCSC88b] At levels C2 and higher, a system must protect objects by default at creation time; the default protection may be changed only by authorized individuals and processes acting on their behalf. [NCSC88b] DAC is often employed to enforce need-to-know. [NCSC87] The controls are [often] discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control). [NCSC85]

Disjoint (Objects)

A relationship between objects. Two objects are disjoint if and only if writing to either object cannot affect the value of the other.

Distributed Authority

An AIS characterized by having authority to modify security attributes given to arbitrary users for entities under their control.

Dominate

A relationship in which security level S_1 is said to dominate security level S_2 if the hierarchical classification of S_1 is greater than or equal to that of S_2 and the non hierarchical categories of S_1 include all those of S_2 as a subset. [NCSC85] [More generally, if S_1 and S_2 are levels from any partially ordered set of security attributes, then S_1 dominates S_2 if and only if $S_2 \leq S_1$.]

Encapsulated Device

An abstract representation of an I/O device, its associated device driver, and other related entities such as attached hardware and dedicated device buffers. An encapsulated device is an I/O port for an AIS.

Encapsulation Mechanism

A mechanism that provides restricted access to a data structure or other entity. Typically, the encapsulation mechanism supports a set of "abstract operations" on a data structure; the operations together with the data structure form a "data abstraction." The abstract operations are usually defined in terms of more elementary operations that are not available to users of the data abstraction, and TCB support is needed to prevent users of the data abstraction from accessing the data structure by means of more elementary operations.

Endorsed Tools List

The list of formal verification tools endorsed by the NCSC for the development of systems with high levels of trust. [NCSC88]

Entity

See Controlled Entity

Evaluation

The process of determining whether a computing system meets given requirements.

Exempt Subject

A TCB subject that is exempt from some of the constraints imposed on non-TCB subjects and is thus able to perform actions that are not available to non-TCB subjects.

Explicitly Controlled Entity

A controlled entity for which there are explicitly associated security attributes, such as subjects and storage objects, in the case of a MAC policy, or named objects, in the case of a DAC policy. In addition to explicitly controlled entities, there may be implicitly controlled entities, including fragments of controlled entities and composite entities.

External-Interface Model

A model whose definition of security is cast in terms of external-interface requirements. The purpose of an external-interface model is to present system requirements while avoiding unnecessary constraints on internal structure.

External-Interface Requirement

A requirement that must hold for the interface between a system and its environment.

Floating Label Policy

A policy based on security levels in which the level of an entity can increase as a result of receiving information from a higher-level entity.

Formal Proof

(1) [A mathematical proof:] A complete and convincing mathematical argument, presenting the full logical justification for each proof step for the truth of a theorem or set of theorems. [NCSC85]

(2) A machine checked proof: Text that a *proof checker* has accepted as evidence showing that a conjecture is a valid consequence of its axioms.

(3) A Hilbert proof in a theory T: a sequence of formulas, each of which is either an axiom of T or is a direct consequence of preceding formulas in the sequence by virtue of a rule of inference associated with the underlying formal system. [cfMEND79]

Formal Security Policy Model

A mathematically precise statement of a [system] security policy. [NCSC85] Some formal modeling techniques include: state-transition models, temporal-logic models, denotational-semantics models, and algebraic-specification models. An example is the model described by Bell and La Padula in *Secure Computer Systems*: *Unified Exposition and Multics Interpretation.* [NCSC85] To be adequately precise, a model [in the tradition of Bell and LaPadula] must represent the initial state of a system, the way in which the system progresses from one state to another, and a definition of a "secure" state of the system. To be acceptable as a basis for a TCB, the model must be supported by a formal proof that, if the initial state of the system satisfies the definition of a "secure state" and if all assumptions required by the model hold, then all future states of the system will be secure. [NCSC85] [More generally, the model should contain a *definition of security* that regulates how a system manages, protects, and distributes sensitive information as well as *rules of operation* that show how the definition of security is to be enforced. At levels B2 and above, it must be supported by a formal proof showing that the rules of operation guarantee satisfaction of the definition of security.]

Formal (Security) Verification

The process of using formal proofs to demonstrate the consistency between Ball [all valid interpretations of] a formal specification of a system and a formal security policy model (design verification) or between the formal specification and [all valid interpretations of] its high-level program [i.e., software] implementation (implementation verification). [NCSC85] [In general, programs, like specifications, are subject to multiple interpretations because programming languages allow compilers to take minor liberties in order to produce optimal code.]

Formal Top-Level Specification (FTLS)

A top-level specification that is written in a formal mathematical language to allow theorems showing the correspondence of the system specification to its formal requirements to be hypothesized and formally proven. [NCSC85]

Host (on a Network)

Any computer-based system connected to the network and containing the necessary protocol interpreter software to initiate network access and carry out information exchange across the communications network. [NCSC87]

I*ndividual Component (of a Network)*

A component of a network that is not subdivided into smaller components for purposes of security analysis.

Individual Component Model

A security policy model for an individual component of a network.

Inference Problem

The occurrence when a user is able to deduce information to which they do not have privilege from information to which they do have privilege. It can happen that a user is not allowed access to a piece of information that is logically inferrible from known information and pieces of information that the user does have access to. In this case, the inference problem is to prevent the user (or a subject acting on their behalf) from indirectly gaining access to the inferrible information.

Informal Security Policy Model

A precise description of the security policy enforced by the system. It must identify the rules and practices that regulate how a system manages, protects, and distributes sensitive information. [cf NCSC88, *Security Policy Model*]

Information-Flow Model

A definition of security (typically a state-machine model) which depicts allowed information flows that occur in response to individual user inputs and changes of state.

Informational Attribute

A security attribute that is used for some other purpose than access control within a given computing system; the association may be for later use outside the system, for example.

Integrity (of Data)

(1) The property that data meet an a priori expectation of quality. [NCSC88]

(2) The state that exists when the quality of stored information is protected from contamination or degradation by information of lower quality. [COUR89]

Integrity (of a System or Process)

The quality that a system or process: has when it performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation. [NCSC88]

Integrity Attribute, Integrity Level
A security attribute used to prevent unauthorized or inappropriate modification or destruction of information.

Internal-Security Model
A security model whose definition of security consists of internal security requirements.

Internal-Security Requirement
A system requirement that is stated in terms of desired relationships among controlled entities, constraints on their interaction, or in terms of allowed forms of interaction.

Internal Subject
A subject that is not acting as direct surrogate for a user. A process which is not associated with any user but performs system-wide functions such as packet switching, line printer spooling, and so on. [NCSC87]

In variant
An assertion that is true in every reachable state.

Label
See Security Label.

Level
A security attribute chosen from a partially ordered set of security attributes. For a given system configuration, the accreditation process should ensure that levels are limited to an appropriate accreditation range and that the system never assumes one level dominates another when, in reality, it does not. (See also, Security Level.)

Least Privilege
A principle which requires that each subject in a system be granted the most restrictive set of privileges (or lowest clearance) needed for the performance of authorized tasks. The application of this principle limits the damage that can result from accident, error, or unauthorized use. [NCSC85]

Loose Access Control Attribute
A security attribute that controls access to system resources without controlling access to the information they contain in some cases.

Mandatory Access Control (MAC)
A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity. [NCSC85]

Mandatory Security Policy
A policy that is based on constraints imposed by a recognized authority for the protection of sensitive information and applied uniformly to all users of a computing system.

Mealy Machine

A model of computation consisting of inputs, outputs, states, an initial state, a state-transformation function that shows how a given input and state induce a new state, and an output function that shows the output which results from a given input in a given state.

Model

An abstraction or simplification of reality, the purpose of which is to capture key aspects of the behavior of that reality.

Model Interpretation

An intended association between the constructs of a model and identified aspects of the system being modeled; this association provides a means of judging whether assertions in the model are true of the modeled system. If they are, then the system is said to satisfy the model with respect to the given interpretation, and the interpretation is said to be accurate.

Model of Computation

A general model of a potentially large class of computing systems.

Moore Machine

A model of computation consisting of inputs, outputs, states, an initial state, a state-transformation function that shows how a given input and state induce a new state, and an output function that maps each state to an associated output.

Multilevel Device

A device that is used in a manner that permits it to simultaneously process data of two or more levels without risk of compromise. To accomplish this, sensitivity labels are normally stored on the same physical medium and in the same form (i.e., machine-readable or human-readable) as the data being processed. [NCSC85]

Multilevel Data Structure

A data structure that contains (or overlaps) several objects whose security levels are not necessarily the same.

Named Object

An object which is directly manipulable at the TCB interface. The object must have meaning to more than one process. [NCSC88a] [In other words, a named object is a data structure or other controlled entity to which discretionary access controls can be directly applied. DAC attributes need only be associated with a data structure if it is directly visible at the TCB interface and has meaning to more than one process.]

Need-to-Know

[Determination by an authorized holder of sensitive information of] the necessity for [another person to have] access to, knowledge of, or possession of [specific] information required to carry out ~specific: official duties. [NCSC88]

Network

See *network system or network subsystem.*

Network Security Model

That portion of a security policy model for a network that describes overall network security policy as opposed to security services provided by particular components.

Network Subsystem

A component of a network that might itself be partitioned into smaller components.

Network System

A system which is implemented with a collection of interconnected network components. A network system is based on a coherent security architecture and design. [NCSC87]

Network Trusted Computing Base (NTCB)

The totality of protection mechanisms within a network system—including hardware, firmware, and software ... the combination of which is responsible for enforcing a security policy. [NCSC87]

Nondeterministic

A characteristic of an AlS where its behavior is not entirely determined by its inputs.

Nondisclosure Attribute

A security attribute used to prevent the unauthorized release of information. Usually, nondisclosure attributes belong to a partially ordered set of nondisclosure levels.

Nondisclosure Controls

Automated controls used to prevent the unauthorized release of information.

Object

A passive entity that contains or receives information. Access to an object potentially implies access to the information it contains. Examples of objects are: records, blocks, pages, segments, files, directories, directory trees, and programs, as well as bits, bytes, words, fields, processors, video displays, keyboards, clocks, printers, network nodes, etc. [NCSC85]

Owner (of an Entity)

The user (or users) responsible for controlling the use of a controlled entity.

Partial Ordering

A relation that is transitive, reflexive, and antisymmetric.

Partially Trusted Subject

A subject, typically a TCB subject, that has two security levels, “alter-min” and “view-max,” and is constrained in such a way that it may only write at levels dominating alter-min and read at levels dominated by view-max.

Philosophy Of Protection (Protection Philosophy)

An informal description of the overall design of a system that delineates each of the protection mechanisms employed. A combination (appropriate to the evaluation class) of formal and informal techniques is used to show that these mechanisms are adequate to enforce the security policy. [NCSC85]

Policy
A high-level overall plan embracing the general goals and acceptable procedures, especially of a governmental body.

Process (Single-threaded).
A sequential program in execution. It is completely characterized by a single execution point (represented by the machine state) and an address space. [NCSC85]

Process Family (Multithreaded Process)
A program in execution, especially one with multiple points of control.

Proof Checker
A tool that (1) accepts as input an assertion (called a conjecture), a set of assertions (called assumptions), and a proof; (2) terminates and outputs either success or failure; and (3) if it succeeds, then the conjecture is a valid consequence of the assumptions. [NCSC89] [A proof checker is typically supported by a formal semantics that determines interpretations of formula S; in this case, a formula * is a valid consequence of a set Φ of formulas if and only if every interpretation which satisfies Φ also satisfies *.]

Protection State
That portion of the system state which is crucial to understanding an access control policy and is therefore not abstracted away in an access control model.

Reachable State
Any state that can be obtained from an initial state via inputs and state transformations.

Read Access (to an Entity)
Permission to read information. [NCSC87]

Reference Monitor Concept
An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects. [NCSC85]

Reflexivity
Property of a binary relation R which says that every element is related to itself, that is, $x\ R\ x$, for all x.

Release Markings
Authorized markings placed on a document by its originator for the purpose of imposing restrictions on dissemination of the document and the information it contains.

Role Support Program
A program that is executed in support of an associated user role.

Rules of Operation
Descriptions of key ideas associated with the design of the security-enforcement mechanisms in a trusted computing system. Rules of operation typically describe basic state transformations that accomplish necessary access control checks.

Secure System
An AlS that satisfies an associated system security policy.

Security Administrator
A user responsible for the security of an AlS and having some authority to enforce security safeguards on other users of the AlS.

Security Attribute
Any piece of information that may be associated with a controlled entity or user for the purpose of implementing a security policy.

Security-Critical Data
User authentication data, audit data, audit control data, security attributes of controlled entities, or other data that are necessary for the correct functioning of the TCB.

Security Label
A container for associated security attributes of a controlled entity, especially attributes in a partially ordered set of security attributes related to information flow.

Security Level
The combination of a hierarchical classification and a set of nonhierarchical categories that represents the sensitivity of information. [NCSC85] [in some contexts, this term is used more generally to mean, simply, a level in a partially ordered set of security attributes.]

Security Markings
Security attributes not used for mandatory access control.

Security Mechanism
A concretely given security requirement, especially one that is not directly tied to a controlling security objective.

Security Perimeter
The portion of a system and its environment where security objectives are actively addressed. The security perimeter contains the TCB and associated controlled entities.

Security Policy (Automated Information System)
A set of restrictions and properties that specify how an AIS prevents information and computing resources from being used to violate an organizational security policy. It should be

accompanied by a persuasive set of engineering arguments showing that these restrictions and properties play a key role in the enforcement of the organizational security policy. [cf STER91]

Security Policy (Organizational)

A set of laws, rules, and practices that regulates how an organization manages, protects, and distributes sensitive information. [NCSC85]

Security Policy Model

(1) An informal presentation of a *formal security policy model*. [NCSC85]

(2) A [precise, if not] formal presentation of the security policy enforced by the system. It must identify the set of rules and practices that regulates how a system manages, protects, and distributes sensitive information. [NCSC88]

Security Requirements

Types and levels of protection necessary for equipment, data, information, applications, and facilities to meet [a given] security policy. [NCSC88]

Sensitive Information

Information that, as determined by a competent authority, must be protected because its unauthorized disclosure, alteration, loss, or destruction is deemed to at least cause perceivable damage to someone or something. [NCSC85] [Sensitive information includes both classified information and unclassified sensitive information. [cf DOD88a]]

Sensitivity Label

A piece of information that represents the security level of an object and that describes the sensitivity (e.g., classification) of data in the object. Sensitivity labels are used by the TCB as the basis for mandatory access control decisions. [NCSC85]

Separation of Duty

A design principle in which user roles are defined so that privileges are divided among several roles in such a way as to inhibit the abuse of any given role.

Simple Security Condition (i. e., Simple Security Property)

(1) A Bell-La Padula security model rule [i.e.,state in varian]: allowing a subject read access to an object only if the security level of the subject dominates the security level of the object. [NCSC85]

(2) A state invariant to the effect that a subject may have read access to an object only if its maximum security level dominates the security level of the object. [BELL76]

Single-Level Device

A device that is used to process data of a single security level at any onetime. Since the device need not be trusted to separate data of different security levels, sensitivity labels do not have to be stored with the data being processed. [NCSC85]

** properly (Star Property)*

(1) A Bell-LaPadula security model rule [i.e.,state invariant] allowing a subject write access to an object only if the security level of the subject is dominated by the security level of the object [NCSC85].
(2) A state invariant to the effect that an untrusted subject may write only objects at or above and may read only objects at or below its current security level. [BELL76]

State
That component of a state machine model which holds the current (abstracted) state of the system being modeled.

Storage Object
An object that supports both read and write accesses. [NCSC85]

State Invariant
A property that is true of all reachable states.

State Machine
A model of computation involving inputs, outputs, states, and state transition functions, for example, a Mealy machine or a Moore machine.

State Transition Constraint
A relationship between states that must hold for every state transition; the tranquility property is a simple state transition constraint.

Subject
An active entity, generally in the form of a person, process [, process family], or device, that causes information to flow among objects or changes the system state. Technically, [in Multics, it is] a process/domain pair, where a domain is the set of objects that a [process or] subject has the [potential or actual] ability to access. (NCSC85]

Submodel
A portion of a security policy model that deals with a particular policy objective, a particular system component, requirements that relate internal components, or rules of operation showing how security is enforced.

Subsystem Security Model
A security model for a subsystem of a larger system.

System Entity
Any system resource (e.g., process, storage object, I/O device) that is directly or indirectly accessible by users.

System Operator
A user responsible for the routine operation and maintenance of the system.

System Security Model

A security model for an entire system, as opposed to a subsystem model.

TCB Subject

A subject internal to the TCB. The two main kinds of TCB subjects are multilevel subjects (ones that compute the security levels or security attributes of their outputs) and trusted-role subjects.

Tight Access Control Attribute

A security attribute that controls access to system entities and the information they contain.

Top-Level Specification (TLS)

A nonprocedural description of system behavior at the most abstract level. Typically, a functional specification that omits all implementation details. [NCSC85] [A TLS discusses what a system does as opposed to how; the requirement that the specification be “nonprocedural” applies to the content rather than the form of the specification.]

Tranquility

A property applied to a set of (typically untrusted) controlled entities saying that their security level may not change (except possibly at the instigation of trusted processes).

Transitivity

Property of a binary relation R which says that if $x\ R\ y$ and $y\ R\ z$, then $x\ R\ z$.

Trojan Horse

A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security. For example, making a “blind copy” of a sensitive file for the creator of the Trojan horse. [NCSC85]

Trusted Computer System

A system that employs [i.e, whose design employs] sufficient hardware and software integrity measures to allow its use for [enforcing a system security policy, such as a nondisclosure policy, that allows] processing simultaneously a range of sensitive or classified information. [NCSC87]

Trusted Computing Base (TCB)

The totality of protection mechanisms within a computer system, including hardware, firmware, software, [and data], the combination of which is responsible for enforcing a [system] security policy. [NCSC85] It creates a basic protection environment and provides additional user services required for a trusted computer system. [NCSC87] A TCB [thus] consists of one or more components that together enforce a unified [system] security policy over a product or system. The ability of a TCB to correctly enforce an [organizational] security policy depends solely [i.e., jointly] on the mechanisms within the TCB, on the correct input by system administrative personnel of parameters (e.g.,a user’s clearance) related to the security policy [,and on the proper actions of its users (e.g., proper labeling of input, password secrecy]:. [NCSC85]

Trusted Role Process
A process that supports a trusted user role by manipulating security-criticaI data.

Trusted User Role
A user role that involves handling security-critical information maintained by the system.

Type Enforcement
A form of mandatory access control in which objects and subjects are assigned types, and access by subjects to objects is restricted by looking up allowed accesses in a "type" table.

User
Any person who interacts directly with a computing system; [NCSC85] see also *clients*.

User Granularity
A property of access control attributes which determines whether they are "coarse," controlling access on the basis of broadly defined classes of users, or "fine," with the ability to control access by individual users and processes acting on their behalf.

User Role
A prescribed use of an AIS as defined by a combination of procedural and software constraints.

Verification
The process of comparing two levels of system specification for proper correspondence (e.g., security policy model with top-level specification, top-level specification with source code, or source code with object code). This process may or may not be automated. [NCSC89]

Verification System
An integrated set of tools and techniques for performing verification. [NCSC89]

Write Access
Permission to write information. [NCSC87]

REFERENCES

ABBO69 Abbot, J. C., *Sets, Lattices, and Boolean Algebras*, Chapter 4, Allyn and Bacon, 1969.

ABRA90 Abrahms, M. D., et al., "A Generalized Framework for Access Control: An Informal Description," *13th National Computer Security Conference*, pp. 135-143, NCSC/NCSL, October 1990.

AGER82 Agerwala, T., and N. I. Arvind, "Data Flow Systems," Computer Vol. 15, No. 2, pp. 10-14, IEEE, February 1982.

AKER90 Akers, R. L. et al., *Gypsy Verification Environment User's Manual,* Computational Logic Inc., Austin TX 78703, July 1990.

ANDE72 Anderson, J. P., *Computer Security Technology Planning Study Vol I,* ESD-TR-73-51, NTIS# AD-758206, Electronic Systems Division, Air Force Systems Command, October 1972.

ANDR80 Andrews, G. R., and R. P. Reitman, “An Axiomatic Approach to Information Flow in Programs,” *ACM Transactions on Programming Languages and Systems,* Vol. 2, No. 1, pp. 56-76, ACM, January 1980.

BELL73 Bell, D. E., and L. J. La Padula, *Secure Computer Systems, VoI. I: Mathematical Foundations,* ESD-TR 73-278, NTlS# AD-770768, Electronic Systems Division, Air Force Systems Command, November 1973.

BELL74 Bell, D. E., *Secure Computer Systems, Vol III: A Refinement of the Mathematical Model,* ESD-TR-73-278, NTIS# AD-780528, Electronic Systems Division, Air Force Systems Command, April 1974.

BELL74a Bell, D. E., and L. J. La Padula, “Secure Computer Systems: Mathematical Foundations and Model,” M74-244, The MITRE Corporation, Bedford MA, October 1974.

BELL76 ________, Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306, NTIS# AD-A023588, Electronic Systems Division, Air Force Systems Command, March 1976.

BELL86 Bell, D. E., “Secure Computer Systems: A Network Interpretation,” *Second Aerospace Computer Security Conference: Protecting Intellectual Property,* pp 32-39, IEEE, December 1986.

BELL88 ________ “Concerning ‘Modeling’ of Computer Security,” *1988 Symposium on Security and Privacy,* pp. 8-13, IEEE, April 1988.

BELL88a ________ “Security Policy Modeling for the Next-Generation Packet Switch,” *1988 Symposium on Security and Privacy,* pp. 212-216, lEEE, April 1988.

BELL90 ________,“Lattices, Policies, and Implementations,” *13th National Computer Security Conference,* pp. 165-171, NIST/NCSC, October 1990.

BIBA77 Biba, K. J., “Integrity Considerations for Secure Computer Systems,” ESD-TR-76-372, NTlS# AD-A039324, Electronic Systems Division, Air Force Systems Command, April 1977.

BISH90 Bishop, M., “A Model of Security Monitoring,” *Fifth Computer Security Applications Conference,* pp. 46-52, held December 1989, lEEE, 1990.

BODE88a Bodeau, D. J., “Some Observations on B1 Informal Models,” MTP 278, The MITRE Corporation, Bedford MA, September 1988.

BOEB85 Boebert, W. E., and R. Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," *8th National Computer Security Conference,* pp. 18-29, NBS/NCSC, October 1985.

BREW89 Brewer, D. F. C. and M. J. Nash, "The Chinese Wall Security Policy," *1989 Symposium on Security and Privacy,* pp. 206-214, IEEE, May 1989.

BRIT84 Britton, D. E., "Formal Verification of a Secure Network with End-to-End Encryption," *1984 Symposium on Security and Privacy,* pp. 154-166, IEEE, April 1984.

CHEH81 Cheheyl, M. H., et al., "Verifying Security," *Computing Surveys*, Vol. 13, No. 3, pp. 279-339, ACM, September 1981.

CHEN90 Cheng, P.-C., and V. D. Gligor, "On the Formal Specification and Verification of a Multiparty Session Protocol," *1990 Symposium on Research in Security and Privacy,* pp. 216-233, IEEE, May 1990.

CHIS85 Chisholm, G. H. et al., "Preliminary Report on the Formal Analysis of the Draper FTP Hardware and Software Using ITP, "ANL/MCS-TM-59, NTIS# DE86003985, Argonne National Laboratory, Argonne, IL 60439, September 1985.

CLAR87 Clark, D. D. and D. R. Wilson, "Comparison of Commercial and Military Computer Security Policies," *1987 Symposium on Security and Privacy,* pp. 184-194, IEEE, April 1987.

CLAR89 ________, "Evolution of a Model for Computer Integrity," in *Report of the* I*nvitational Workshop on Data Integrity Special Publication* 500-168, pp. A.2.1-A.2.13, NIST, January 1989.

COHE77 Cohen, E., "Information Transmission in Computational Systems," *6th ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review* Vol. 11, No. 5, pp. 133-139, ACM, November 1977.

CONG87 Congress, 100th, *Computer Security Act of 1987*, Public Law 100-235, January 1988.

CONG88 ________, *Computer Matching and Privacy Protection Act of 1988,* Public Law 100-503, October 1988.

COUR89 ________ Courtney, R. H., "Some Informal Comments About Integrity and the Integrity Workshop," *Report of the Invitational Workshop on Data Integrity,* NIST Special Publication 500-168, pp. A.1.1-A.1.1 8, NIST, January 1989.

DAVI88 Davison, Jay W., "Implementation Design for a Kernelized Trusted DBMS," *Fourth Aerospace Computer Security Applications Conference,* pp. 91-98, IEEE, December 1988.

DENN76 Denning, D. E., "A Lattice Model of Secure Information Flow," *Communications of the ACM,* Vol. 19, No. 5, pp. 236-243, ACM, May 1976.

DENN77 ________, and P. J. Denning, "Certification of Programs for Secure Information Flow," *Communications of the ACM,* Vol. 20, No. 7, pp. 504-512, ACM, July 1977.

DENN82 ________, *Cryptography and Data Security*, Addison-Wesley, 1982.

DENN88 _______ et al., "The Sea View Security Model," *1988 Symposium on Security and Privacy,* pp. 218-233, lEEE, April 1988.

DlAZ89 Diaz, M., et al., *The Formal Description Technique Estelle*, North-Holland, 1989.

DION81 Dion, L. C., "A Complete Protection Model," 1981 *Symposium on Security and Privacy*, pp. 49-55, IEEE, April 981.

D1V190 DiVito, B. L., et al., "Specification and Verification of the ASOS Kernel," *1990 Symposium on Research in Security and Privacy,* pp. 61-74, IEEE, May 1990.

DOD82 Department of Defense, *Department of Defense Privacy Program,* DoD Directive 5400.11, June 1982.

DOD86 ________, Information Security Program Regulation, DoD Regulation 5200.1, May 1986.

DOD88 ______, *Defense System Software Development,* DOD 2167A-STD, June 1985.

DOD88a ________, *Security Requirements for Automated Information Systems* (AISs), DoD Directive 5200.28, March 1988.

EGGE89 Eggert, P. et al., *FDM User Guide,* Unisys Corporation, Culver City, CA 90230, September 1989.

EIJK89 Eijk, P. H. J. van, C. A. Vissers, and M. Diaz, *The Formal Description Technique LOTOS,* North-Holland, 1989.

FARM86 Farmer, W. M., D. M. Johnson, and F. J. Thayer, "Towards a Discipline for Developing Verified Software," *9th National Computer Security Conference,* pp. 91-98, NBS/NCSC, September 1986.

FEIE77 Feiertag, R. J., K. N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design," *6th Symposium on Operating Systems Principles, ACM Operating Systems Review*, Vol. 11 No.5, pp. 57-65, November 1977.

FEl

FEIE80 Feiertag, R. J., “A Technique for Proving Specifications are Multilevel Secure,” CSL-1 09, SRI International, Menlo Park, January 1980.

FELL87 Fellows, J. et al, “The Architecture of a Distributed Trusted Computing Base,” *10th National Computer Security Conference,* pp. 68-77, NBS/NCSC, September 1987.

FERN81 Fernandez, E. B., R. C. Summers, and C. Wood, *Database Security and Integrity,* Addison-Wesley, 1981.

FINE89 Fine, T., et al., “Noninterference and Unwinding for LOCK,” *The Computer Security Foundations Workshop II,* IEEE-CS Order No. 1955, pp. 22-30, IEEE, June 1989.

FINE90 Fine, T, “Constructively Using Noninterference, *1990 Symposium on Research in Security and Privacy,* pp. 162-169, IEEE, May 1990.

FORD90 Ford, W. R., J. P. O’Keeffe, and B. M. Thuraisingham, “Database Inference Controller ... An Overview,” MTR 10963, Volume 1, The MITRE Corporation, Bedford MA, August 1990.

FREE88 Freeman, J. W., R. B. Neely, and G. W. Dinolt, “An Internet Security Policy Model,” *11th National Computer Security Conference,* pp. 10-19, NBS/NCSC, October 1988.

GAJN88 Gajnak, G. E., “Some Results from the Entity/Relationship Multilevel Secure DBMS Project,” *Fourth Aerospace Computer Security Applications Conference,* pp. 66-71, IEEE, December 1988.

GALO89 Galovich, S., *Introduction to Mathematical Structures*, Harcourt, Brace & Jovanovich, 1989.

GARV90 “A Layered TCB Implementation Versus the Hinke-Schaefer Approach,” in *Database Security III:* Status and Prospects, pp. 151-165, edited by D.L. Spooner and C. Landwehr, North-Holland/IFIP, 1990.

GASS87 Gasser, M., *Building a Secure Computer System,* Van Nostrand, 1987.

GLAS87 Glasgow, J. I., and G. H. MacEwen, “The Development and Proof of a Formal Specification for a Multilevel Secure System,” *ACM Transactions on Computing Systems,* Vol. 5, No. 2, pp. 151-184, ACM, May 1987.

GLIG86 Gligor, V. D. et al., “On the Design and the Implementation of Secure Xenix Workstations,” *1986 Symposium on Security and Privacy* pp. 102-117, IEEE, April 1986.

GOGU82 Goguen, J. A., and J. Meseguer, “Security Policies and Security Models,” *1982 Symposium on Security and Privacy* pp. 11-20, IEEE, April 1982.

GOGU84 ________, “Unwinding and Inference Control,” *1984 Symposium on Security and Privacy* pp. 75-85, IEEE, May 1984.

GOLD80 Goldberg, A., *Smalltalk-80: The language and its Implementation,* Addison-Wesley, Reading, MA, 1983.

GOOD88 Good, D. l., B. L. DiVito, and M. K. Smith, *Using the Gypsy Methodology* Computational Logic, Inc., Austin TX 78703, January 1988.

GOOD89 Good, D. l., R. L. Akers, and L. M. Smith, *Report on Gypsy 2.05,* Computational Logic, Inc., Austin TX 78703, August 1989.

GOOD90 Good, D. l., A. E. Siebert, and W. D. Young, *Middle Gypsy 2.05 Definition,* Technical Report 59, Computational Logic Inc., Austin TX 78703, May 1990.

GOVE84 Gove, R. A., “Extending the Bell & La Padula Security Model,” *7th DoD/NBS Computer Security Conference*, pp. 112-119, NBS/NCSC, September 1984.

GRAH72 Graham, G. S., and P. J. Denning, Protection-Principles and Practice,” *Proceedings of the 1972 AFIPS Spring Joint Computer Conference,* Vol. 40, pp. 417-429, AFIPS Press, 1972.

GRAU89 Graubart, R. G., “On the Need for a Third Form of Access Control,” *12th National Computer Security Conference,* pp. 296-304., NIST/NCSC, October 1989.

GRAU90 ________, “Comparing DBMS and Operating System Security Requirements: The Need for a Separate DBMS Security Criteria,” in *Database Security III: Status and Prospects,* pp. 109-114, edited by D. L. Spooner and C. Landwehr, North-Holland/IFIP, 1990.

GRAY91 Gray, J. W., III, “Toward a Mathematical Foundation for Information Flow Security,” *IEEE Symposium on Research in Security and Privacy,* pp. 21-34, lEEE, May 1991.

HAIG84 Haigh, J. T., “A Comparison of Formal Security Policy Models,” *7th DoD/NBS Computer Security Conference,* pp. 88-111, NBS/NCSC, September 1984.

HAIG87 ________, “An Experience Using Two Covert Channel Analysis Techniques on a Real System Design,” *lEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, pp. 157-168, IEEE, February 1987.

HAIG87a ________, and W. D. Young, “Extending the Noninterference Version of MLS for SAT,” *IEEE Transactions on Software Engineering,* Vol. SE-13, No. 2, pp. 141-150, IEEE, February 1987.

HAIG90 ________, et al., “The LDV Approach to Database Security, “*Database Security III: Status and Prospects*, pp. 323-339, edited by D. L. Spooner and C. Landwehr, North-

Holland/IFIP, 1990.

HAMM80 Hamming, R. W., *Coding and Information Theory,* Prentice-Hall, Englewood Cliffs, NJ, 1980.

HARR76 Harrison, M. A., W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," *Communications of the ACM,* Vol. 19, No. 8, August 1976.

HINK88 Hinke, T. H., "Inference Aggregation Detection in Database Management Systems," *1988 Symposium on Security and Privacy*, pp. 96-106, IEEE, April 1988.

HINK90 ________, "DBMS Trusted Computing Base Taxonomy, in *Database Security III: Status and Prospects,* pp. 97-108, edited by D. L. Spooner and C. Landwehr, North-Holland/IFlP, 1990.

HOAR85 Hoare, C. A. R., *Communicating Sequential Processes*, Prentice-Hall, 1985.

HONE85 Honeywell Information Systems, Inc., *SCOMP Interpretation of the Bell-LaPadula Model*, January 1985.

HUBB86 Hubbard, B. S., S. A. Walker, and R. R. Henning, "Database Systems and the Criteria: Do They Relate?," *9th National Computer Security Conference*, pp. 21-24, NBS/NCSC, September 1986.

ISO84 International Standards Organization, *Information processing systems Open Systems Interconnection Basic Reference Model,* ISO7498-1984 (E), (available from) American National Standards Association, New York, 1984.

ISO89 ________, *Information Processing Systems Open Systems Interconnection Basic Reference Model Part 2: Security Architecture,* ISO7498-2:1984 (E), (available from) American National Standards Association, New York, 1989.

ISRA87 Israel, H., "Design of Originator Controls in a Computer System: a Trusted Discretionary Access Control Mechanism," *3rd Symposium on Physical"/Electronic Security,* pp. 3-1-3-6, Armed Forces Communications and Electronics Association, Philadelphia, August 1987.

JAJO90 Jajodia, S., and B. Kogan, "Integrating an Object-Oriented Data Model with Multilevel Security," *1990 Symposium on Research in Security and Privacy,* pp. 76-85, IEEE, May 1990.

JOHN88 Johnson, D. M., and F. J. Thayer, "Security and the Composition of Machines," *The Computer Security Foundations Workshop,* pp. 72-89, the MITRE Corporation, Bedford MA, June 1988.

JONE75 Jones, A. K., and R. J. Lipton, "The Enforcement of Security Policies for

Computation," *ACM Operating Systems Review* Vol. 9, No. 5, pp. 197-206, November 1975.

KARG89 Karger, P. A., "New Methods for Immediate Revocation," *1989 Symposium on Security and Privacy* pp. 48-55, IEEE, May 1989.

KATZ89 Katzke, S. W., and Z. G. Ruthberg, editors, *Report on the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS),* Special Publication 500-160, NIST, January 1989.

KEEF89 Keefe, T. F., M. B. Thuraisingham, and W. T. Tsai, "Secure Query-Processing Strategies," *IEEE Computer,* Vol. 22, No.3, pp. 63-70, March 1989.

KEEF89b Keefe, T. F., W. T. Tsai, and M. B. Thuraisingham, "SODA: A Secure Object-Oriented Database System," *Computers and Security* Vol. 8, No. 6, pp. 517-533, Elsevier Science, October 1989.

KEEF90 Keefe, T F., and W. T. Tsai, "Multiversion Concurrency Control for Multilevel Secure Database Systems," *1990 Symposium on Research in Security and Privacy*, pp. 369-383, IEEE, May 1990.

KEEF90b ________, et al., "Multi-Party Update Conflict: The Problem and Its Solutions," *Filth Computer Security Applications Conference*, pp. 222-231, held December 1989, lEEE, 1990.

KNOD88 Knode, R. B., and R. A. Hunt, "Making Databases Secure with TRUDATA Technology," *Fourth Aerospace Computer Security Applications Conference,* pp. 82-90, December 1988, IEEE.

KORT86 Korth, H. F., and A. Silberschatz, *Database System Concepts,* McGraw Hill, 1986.

LAMP71 Lampson, B. W., "Protection," Proceedings, *5th Princeton Symposium on Information Sciences and Systems,* March 1971, and SIGOPS Operating Systems Review, Vol. 8, No. 1, pp. 18-24, ACM, January 1974.

LAMP73 ________, "A Note on the Confinement Problem," *Communications of the ACM,* VoI. 16, No. 10, pp. 613-615, ACM, October 1973.

LAMP82 Lamport, L., R. Shostak, and M. Pease, "The Byzantine Generals' Problem," *ACM Transactions on Programming Languages and Systems,* Vol. 4, No. 3, pp. 382-401, ACM, July 1982.

LAMP85 Lamport, L., and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM,* Vol. 32, No. 1, pp. 52-78, ACM, January 1985.

LAND81 Landwehr, C. E., "Formal Models for Computer Security," *Computing Surveys*, Vol.

13, No. 3, pp. 247-278, ACM, September 1981.

LAND84 ________, C. L., Heitmeyer, and J. McLean, “A Security Model for Military Message Systems,” *ACM Transactions on Computer Systems*, Vol. 2, No. 3, pp. 198-221, ACM, August 1984.

LAND89 Landauer, J., T Redmond, and T Benzel, “Formal Policies for Trusted Processes,” *The Computer Security Foundations Workshop II,* pp. 31-40, IEEE #1955, June 1989.

LAPA73 La Padula, L. J., and D. E. Bell, *Secure Computer Systems, VoI. Il: A Mathematical Model,* ESD-TR-73-278, NTlS #AD-771 543, Electronic Systems Division, Air Force Systems Command, November 1973.

LAPA89 La Padula, L. J., “The ‘Basic Security Theorem' of Bell and LaPadula Revisited,” *Cipher* January 1989.

LAPA90 ________, “Formal Modeling in a generalized Framework for Access Control,” *The Computer Security Foundations Workshop III*, pp. 100-109, IEEE, June 1990.

LARR90 Larrondo-Petrie, M. M., et al., “Security Policies in Object-Oriented Databases,” in *Database Security III: Status and Prospects,* pp. 257-268, edited by D. L. Spooner and C. Landwehr, North-Holland/IFIP, 1990.

LEE88 Lee, T. M. P., “Using Mandatory Integrity to Enforce ‘Commercial’ Security,” *1988 Symposium on Security and Privacy,* pp. 140-146, IEEE, April 1988.

LIPN82 Lipner, S. B., “Nondiscretionary Controls for Commercial Applications,” *1982 Symposium on Security and Privacy,* pp. 2-10, IEEE, April 1982.

LUNT88 Lunt, T. F., “Access Control Policies: Some Unanswered Questions,” *The Computer Security Foundations Workshop*, pp. 227-245, M88-37, The MITRE Corporation, Bedford MA, June 1988.

LUNT88b ________, et al., “A Near-Term Design for the Sea ViewMultilevel Database System,” *1988 Symposium on Security and Privacy,* pp. 234-244, IEEE, April 1988.

LUNT89 ________, “Aggregation and Inference: Facts and Fallacies,” *1989 Symposium on Security and Privacy,* pp. 102-109, IEEE, May 1989.

LUNT90 ________, “Multilevel Security for Object-Oriented Database Systems,” in *Database Security III:* Status and Prospects, pp. 199-209, edited by D. L. Spooner and C. Landwehr, North-Holland/IFIP, 1990.

MAIM90 Maimone, W. T., and l. B. Greenberg, “Single-Level Multiversion Schedulers for Multilevel Secure Database Systems,” *Computer Security Applications Conference,* pp. 137-147, IEEE, December 1990.

MANN82 Manna, Z., and A. Pnueli, "Verification of Concurrent Programs: Temporal Proof Principles," *Logic of Programs,* edited by D. Kozen, Lecture Notes in Computer Science 131, Springer-Verlag, 1982.

MARC86 Marcus: L., and T. Redmond, "A Semantics of Read," *9th National Computer Security Conference,* pp. 184-193, NBS/NCSC, September 1986.

MARG85 Margulies, B. l., *Multics Interpretation of the Bell and La Padin La Model,* Honeywell Information Systems, August 1985.

MCCO90 McCollum, C. J., J. R. Messing, and L. Notargiacomo, "Beyond the Pale of MAC and DAC Defining New Forms of Access Control," *1990 Symposium on Research in Security and Privacy* pp. 190-200, IEEE, May 1990.

MCCU88 McCullough, D., "Covert Channels and Degrees of Insecurity," *The Computer Security Foundations Workshop,* pp. 1-33, M88-37, The MITRE Corporation, Bedford MA, June 1988.

MCCU88a , Foundations of Ulysses: *The Theory of Security,* RADC-TR-87-222, NTIS# AD-A200110, Air Force Systems Command, July 1988.

MCLE85 McLean, J., "A Comment on the 'Basic Security Theorem' of Bell and La Padula," *Information Processing Letters,* Vol. 20, pp. 67-70, Elsevier, 1985.

MCLE87 _______, "Reasoning About Security Models," *1987 Symposium on Security and Privacy,* pp. 123-131, IEEE, April 1987.

MCLE88 ________, "The Algebra of Security," *1988 Symposium on Security and Privacy*, pp. 2-7, IEEE, April 1988.

MCLE90 ________, "Security Models and Information Flow," *1990 Symposium on Research in Security and Privacy*, pp. 180-187, lEEE, May 1990.

MEAD90 Meadows, C., "Constructing Containers Using a Multilevel Relational Data Model," *Database Security, III: Status and Prospects,* pp. 127-141, North-Holland/IFIP, 1990.

MEAD90b , "Extending the Brewer-Nash Model to a Multilevel Context," *1990 Symposium on Research in Security and Privacy,* pp. 95-102, IEEE, May 1990.

MEND79 Mendleson, E., *Introduction to Mathematical Logic,* D. Van Nostrand, 1979.

MILL76 Mien, J. K., "Security Kernel Validation in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976.

MILL81 ________ "Information Flow Analysis of Formal Specifications," *1981 Symposium on Security and Privacy,* pp. 3-8, IEEE, April 1981.

MILL84 _____, "Al Policy Modeling," *7th DoD/NBS Computer Security Conference*, pp. 137—145, NBS/NCSC, September 1984.

MILL88 ________, and M. W. Schwartz, "The Cascading Problem for Interconnected Networks," *Fourth Aerospace Computer Security Applications Conference,* pp. 269-274, IEEE, December 1988.

MILL89 ________,"Models of Multilevel Security," Advances in Computers, Vol. 29, pp. 1-45, Academic Press, 1989.

MILL90 ________, and D. J. Bodeau, "A Dual-Label Model for the Compartmented Mode Workstation," M90-51, The MITRE Corporation, Bedford MA, August 1990.

MILN83 Milner, R., "Calculi for Synchrony and Asynchrony," *Theoretical Computer Science,* Vol. 25, No. 3, pp. 267-310, Elsevier, 1983.

MOOR90 Moore, A. P., "The Specification and Verified Decomposition of System Requirements Using CSP," *IEEE Transactions on Software Engineering,* Vol. 16, No. 9, pp. 932-948, September 1990.

NCSC85 National Computer Security Center, *Department of Defense Trusted Computer Security Evaluation Criteria,* DOD 5200.28-STD, December 1985.

NCSC85a _______, *Computer Security Requirements: Guidance for Applying the Department of Defense Trusted Computer System Criteria in Specific Environments,* CSC-STD-03-85, National Computer Security Center, June 1985.

NCSC87 ________, *Trusted Network Interpretation,* NCSC-TG-005, National Computer Security Center, July 1987.

NCSC87a ________, *A Guide to Understanding Discretionary Access Control in Trusted Systems,* NCSC-TG-003, National Computer Security Center, September 1987.

NCSC88 _______, *Glossary of Computer Security Terms,* NCSC-TG-04, National Computer Security Center, October 1988.

NCSC88a ________, *Computer Security Subsystem Interpretation,* NCSC-TG-009, Version-1, National Computer Security Center, September 1988.

NCSC88b ________, "Criteria Interpretations," *Ratings Maintenance Program Notes,* National Computer Security Center, September 1988.

NCSC89 ________, *Guidelines for Formal Verification Systems*, Version-1, NCSC-TG-014, National Computer Security Center, April 1989.

NCSC89a ________ *A Guide to Understanding Trusted Facility Management*, Version-I, NCSC-TG-015, National Computer Security Center, October 1989.

NCSC90a ________, *Guidelines for Writing Trusted Facility Manuals*, National Computer Security Center (in preparation).

NCSC90b _______,*Trusted Unix Working Group (TRUSlX) Formal Security Policy Model for the UNIX System,* NCSC-TG-020-B, National Computer Security Center, (draft) July 1990.

NCSC90c ________, *Trusted Product Evaluations:* A Guide for Vendors, NCSC-TG-002, National Computer Security Center, June 1990.

NCSC91 _______, *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria,* NCSC-TG-021, National Computer Security Center, April 1991.

NCSC9b _______, *Final Evaluation Report, Secure Ware Incorporated, Compartmented Mode Workstation Plus,* National Computer Security Center, January 1991.

PAYN90 Payne, C. N., J. N. Froscher, and J. P. McDermott, "On Models for a Trusted Application System," *Sixth Computer Security Applications Conference*, pp. 58-67, IEEE, December 1990.

PITT87 Pittelli, P. A., "The Bell-La Padula Computer Security Model Represented as a Special Case of the Harrison-Ruzzo-Ullman Model," *10th National Computer Security Conference,* pp. 118-121, NBS/NCSC, September 1987.

PITT88 ________, "Formalizing Integrity Using Non-Interference," *11th National Computer Security Conference,* pp. 38-42, NBS/NCSC, 1988.

PNUE81 Pnueli, A., "Temporal Semantics of Concurrent Programs," *Theoretical Computer Science,* Vol. 13, Elsevier, 1981.

POPE73 Popek, G. J., "Access Control Models," ESD-TR-73-106, NTIS# AD-761807, Electronic Systems Division, Air Force Systems Command, February 1973.

POPE78 ________, and D. A. Farber, "A Model for Verification of Data Security in Operating Systems," *Communications of the ACM,* pp. 737-749, Vol. 21, September 1978.

PORT85 Porter, S., and T. S. Arnold, "On the Integrity Problem," *8th National Computer Security Conference,* pp. 15-16, NBS/NCSC, October 1985.

POZZ86 Pozzo, M. P., and T. E. Gray, "Managing Exposure to Potentially Malicious Programs, *9th National Computer Security Conference*, pp. 75-80, NBS/NCSC, September 1986.

REAG82 Reagan, R. L., *Executive Order 12356,* U. S. Printing Office, April 1982.

REIT79 Reitman, R. P., "A Mechanism for Information Control in Parallel Systems," *Proceedings 7th Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review* Vol.13, No. 4, pp. 55-63, December 1979.

ROSK90 Roskos, J. E., et al., "A Taxonomy of Integrity Models, Implementations and Mechanisms," *13th National Computer Security Conference,* pp. 541-551, NlST/NCSC, October 1990.

ROWE89 Rowe, N. C., "Inference-Security Analysis Using Resolution Theorem- Proving, *Filth International Conference on Data Engineering,* pp. 410-416, IEEE, February 1989.

RUSH85 Rushby, J. M., "The SRI Security Model," Computer Science Laboratory, SRI International, Menlo Park, CA 94025,1985.

RUSH89 ______, and F. von Henke, *Formal Verification of a Fault Tolerant Clock Synchronization Algorithm,* NASA Contractor Report 4239, Langley Research Center, Hampton, VA 23665, June 1989.

RUTH89 Ruthberg, Z. G., and W. T., Polk, editors, *Report of the Invitational Workshop on Data Integrity* Special Publication 500-168, NIST, September 1989.

SALT75 Saltzer, J., and M. Schroeder, "The Protection of Information in Computer Systems," Proceedings of the IEEE, Vol. 63, No. 9, September 1975.

SAND88 Sandhu, R. S., "The N Tree: A Two Dimension Partial Order for Protection Groups," *ACM Transactions on Computer Systems,* Vol. 6, No. 2, pp. 197-222, May 1988.

SAND90 ________, and S. Jajodia, "Integrity Mechanisms in Database Management Systems," *13th National Computer Security Conference*, pp. 526-540, NIST/NCSC, October 1990.

SAYD87 Saydjari, O. S. and J. M. Beckman, "Locking Computers Securely," *10th National Computer Security Conference,* pp. 129-141, NCSC/ICST, September 1987.

SCHA85 Schaefer, M., and D. E. Bell, "Network Security Assurance," *8th National Computer Security Conference, DoD Computer Security Center*, pp. 64-69, October 1985.

SCHE73 Schell, R. R., P. J. Downey, and G. J. Popek, "Preliminary Notes on the Design of Secure Military Computer Systems," ESD-TR-80-127, NTIS# AD-A089433, Electronic Systems Division, U.S. Air Force, January 1973.

SCHE85 Schell, R. R., T. F. Tao, and M. Heckman, "Designing the GEMSOS Security Kernel for Security and Performance, *8th National Computer Security Conference,* pp. 108-119, NBS/NCSC, October 1985.

SCHE89 Scheid, J. and S. Holtsberg, *Ina Jo Specification Language Reference Manual, Unisys* Corporation, Culver City, CA 90230, May 1989.

SCHN85 Schnackenberg, D. D., “Development of a Multilevel Secure Local Area Network,” *8th National Computer Security Conference,* pp. 97-104, NBS/NCSC, October 1985.

SCHO88 Schorre, D. V., et. al., I*nteractive Theorem Prover (ITP) Reference Manual,* Unisys Corporation, Culver City, CA 90230, November 1988.

SHOC88 Shockley, W. R., “Implementing the Clark/Wilson Integrity Policy Using Current Technology,” *11th National Computer Security Conference,* pp. 29-39, NBS/NCSC, October 1988.

SMIT88 Smith, G. W., “Identifying and Representing the Security Semantics of an Application,” *Fourth Aerospace Computer Security Applications Conference,* pp. 1 25-130, IEEE, December 1988.

SMIT90 ________, “Solving Multilevel Database Security Problems: Technology is not Enough,” in *Database Security, III: Status and Prospects,* pp. 115-126, edited by D. L. Spooner and C. Landwehr, North-Holland/IFIP, 1990.

STAC90 Stachour, P. D. and B. M. Thuraisingham, “Design of LDV: A Multilevel Secure Relational Database Management System,” *IEEE Transactions on Knowledge and Data Engineering,* VoI. 2, No. 2, pp. 190-209, lEEE, June 1990.

STER91 Sterne, D. F., “On the Buzzword ‘Security Policy,’ ” *1991 Symposium on Research in Security and Privacy,* pp. 219-230, IEEE, 1991.

TANE87 Tanenbaum, A. S., Operating Systems: *Design and Implementation,* Prentice-Hall, 1987.

TANE88 ________, *Computer Networks*, Second Edition, Prentice Hall, 1988.

TAYL84 Taylor, T., “Comparison Paper Between the Bell and La Padula Model and the SRI Model,” 1984 Symposium on Security and Privacy, pp. 195-202, IEEE, May 1984.

THOM90 Thompson, D. J., “Role-Based Application Design and Enforcement,” *Fourth IFIP WG 11.3 Workshop on Database Security,* IFIP Working Group 1.3 and U.S. Office of Navel Research, Halifax, England, September 1990.

THUR89 Thuraisingham, B. M., “A Functional View of Multilevel Databases,” Computers and Security, Vol. 8, No. 8, pp. 721-730, Elsevier Science, December 1989.

WALT74 Walter, K. G. et a., “Primitive Models of Computer Security,” Case Western Reserve University, and ESD-TR-74-1 17, NTlS #AD778467, Electronic Systems Division, Air Force Systems Command, January 1974.

WALT74a ________, “Modeling The Security Interface,” Case Western Reserve University, August 1974.

WEIS69 Weissman, C., “Security Controls in the ADEPT-SO Time Sharing System,” *1969 AFlPS Fall Joint Computer Conference,* Vol. 35, pp. 119-133, AFIPS Press, 1969.

WHIT84 The White House, *National Policy on Telecommunications and Automated Information Systems Security,* NSDD-145, September 1984.

WILL91 Williams, J. G., “Modeling Nondisclosure in Terms of the Subject-Instruction Stream,” *1991 Symposium on Research in Security and Privacy,* lEEE, May 1991.

WISE90 Wiseman, S. R., “On the Problem of Security in Databases,” *in Database Security, III: Status and Prospects,* pp. 301-310, edited by D. L. Spooner and C. Landwehr, North-Holland/IFIP, 1990.

WITT9O Wittbold, J. T., and D. M. Johnson, “Information Flow in Nondeterministic Systems,” *1990 Symposium on Research in Security and Privacy*, pp. 144-161, lEEE, May 1990.

WOOD87 Woodward, J. P. L., “Exploiting the Dual Nature of Sensitivity Labels,” *1987 Symposium on Security and Privacy,* pp. 23-30, lEEE, April 1987.US.

GOVERNMENT PRINTING OFFICE
1996-717-902/82621

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1992	Final

4. TITLE AND SUBTITLE
A Guide to Understanding Security Modeling in Trusted Systems

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Security Agency
Attn: C81 (Standards, Criteria, and Guidelines Division)
Ft. George G. Meade, MD 20755-6000

8. PERFORMING ORGANIZATION REPORT NUMBER
NCSC-TG-010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Security Agency
Attn: C81 (Standards, Criteria, and Guidelines Division)
Ft. George G. Meade, MD 20755-6000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Library No. S-239,669

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This document provides a guidance as to the formulation of security models for trusted systems at all levels of trust. It helps the developer understand the essential concepts of security modeling, as well as expected content for the model.

14. SUBJECT TERMS
Computer security; Trusted Computer System Evaluation Criteria (TCSEC); automated data processing (ADP); security modeling; modeling; operating systems.

15. NUMBER OF PAGES
159

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	