A Security Model for Military Message Systems: Retrospective

Carl E. Landwehr
Mitretek Systems, Inc.
Carl.Landwehr@mitretek.org

Constance L. Heitmeyer
Naval Research Laboratory
heitmeyer@itd.nrl.navy.mil

John D. McLean
Naval Research Laboratory
mclean@itd.nrl.navy.mil

Retrospective[1]

In the late 1970's and early 1980's, the military conducted an experiment (the Military Message Experiment, or MME) to investigate replacing existing message systems in use at CINCPAC that were based on the AUTODIN system, with local distribution of copies via a pneumatic tube system, with a new system based on the ARPANET and e-mail that provided a simulated multilevel secure (MLS) interface. At the same time, research was underway to develop multilevel secure operating systems. Experiences with both the MME and with prototype MLS systems led to research conducted at the Naval Research Laboratory to specify and prototype a family of military message systems (MMS) based on software engineering principles and on specifying the desired security behavior at the application level, rather than at the operating system level. The resulting security model was published as an NRL technical report and subsequently in *ACM Transactions on Computer Systems* in August, 1984.

The approach to developing informal security models presented here remains quite relevant. Efforts to develop assurance arguments for today's systems can in many cases be related to the approach taken in this work [25].

This paper was the first in an archival journal to present a security model based on application requirements as opposed to operating system structure. It argues that this is the appropriate orientation for a security model that is to be understood by users, and it presents a framework for developing and expressing security models informally, using natural language, and then formalizing the result. The informal model is accessible to users, while the formal model provides the precision needed for designing a system and determining whether an implementation enforces the model.

The example presented, developed in the context of military message systems, includes a number of concepts that are appropriate for other applications as well. Among these are the concepts of roles—job-related sets of permissions—and of multilevel object—an object (here termed a container) that has a security level of its own and also encloses other objects that retain their own security levels.

Each user had an allowed set of roles, and the access controls on objects in the system could include roles as well as userIDs. A user could occupy one or more roles at a time, and some roles could be occupied only by a single user at a given time. These constraints were based on the observed needs of operational message systems to support one person acting for another as shifts and watches change, and for a single point of control (though not necessarily a single individual) for operations like message release.

The approach to multilevel objects exploits an analogy with the physical world of safes, file folders, and documents to provide a model that application users can understand, and in which they can apply their intuition about familiar objects. Subsequent work has debated whether multilevel objects need to be reflected in the abstractions provided by operating systems, but their appeal to users seems beyond question. In fact, much of the work reported in this paper can fit quite naturally into the framework provided by object-oriented databases.

An informal model has four parts: definitions of terms used in the model, a brief prose description of system operation from the security viewpoint, a set of assumptions, and a set of assertions. Assumptions are statements that must be true if system security is to be preserved, but that cannot be maintained by the computer system itself. For example, users entering message text must be relied on to classify the input properly. Assertions are statements that must be true for system security to be preserved and that the computer system can enforce.

It is the assertions of the model that are re-stated formally. In contrast with the structure of most other security models, security assertions apply without exception to all system users and entities. The formal statement of the model's assertions is notable for being based on both information flow and access control and for being the first state-machine formalization to contain transition restrictions as well as state restrictions.

[1] This retrospective was written by the first author in August, 2001. It draws on an introduction written when the paper was anthologized in about 1990.

Prior to its publication, a draft version of the security model formed the basis for a study of database security problems conducted under the auspices of the National Academy of Sciences in the early 1980's to investigate problems in multilevel secure document handling systems. In this context, the message system example was considered as a restricted version of a database management system.

Many other technical reports and papers were produced by the SMMS research project, covering software design, implementation, and operation of prototype systems based on this model. Some of these are included in the bibliography at the end of this paper [27-38]. None of the prototypes made the transition to an operational system, but the ideas in this paper did influence the design of some classified operational systems. The security modeling approach was subsequently applied to several operational systems as documented in [26].

The paper as presented below is substantially the same as published in *ACM Transactions on Computer Systems,* except for the correction of a few minor errors in the original publication.

Abstract[2]

Military systems that process classified information must operate in a secure manner; that is, they must adequately protect information against unauthorized disclosure, modification, and withholding. A goal of current research in computer security is to facilitate the construction of multilevel secure systems, systems that protect information of different classifications from users with different clearances. Security models are used to define the concept of security embodied by a computer system. A single model, called the Bell and LaPadula model, has dominated recent efforts to build secure systems but has deficiencies. We are developing a new approach to defining security models based on the idea that a security model should be derived from a specific application. To evaluate our approach, we have formulated a security model for a family of military message systems. This paper introduces the message system application, describes the problems of using the Bell-LaPadula model in real applications, and presents our security model both informally and formally. Significant aspects of the security model are its definition of multilevel objects and its inclusion of application-dependent security assertions. Prototypes based on this model are being developed.

Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: General--Security and protection; D.4.6 [Operating Systems]: Security and Protection--access controls; information flow controls; verification; F.3.1 [Logics and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs--assertions; invariants; specification techniques; H.4.3 [Information Systems Applications]: Communications Applications--electronic mail

General Terms: Security, Verification

Additional Key Words and Phrases: Storage channels, message systems, confinement

1 Introduction

A system is *secure* if it adequately protects information that it processes against unauthorized disclosure, unauthorized modification, and unauthorized withholding (also called denial of service). We say "adequately" because no practical system can achieve these goals without qualification; security is inherently relative. A secure system is *multilevel secure* if it protects information of different classifications from users with different clearances; thus some users are not cleared for all of the information that the system processes. Security models have been developed both to describe the protection that a computer actually provides and to define the security rules it is required to enforce [14]. In our view, a security model should enable users to understand how to operate the system effectively, implementors to understand what security controls to build, and certifiers to determine whether the system's security controls are consistent with the relevant policies and directives and whether these controls are implemented correctly [13].

In recent years, the Bell and LaPadula model [4, 8], has dominated efforts to build secure systems. The publication of this model advanced the technology of computer security by providing a mathematical basis for examining the security provided by a given system. Moreover, the model was a major component of one of the first disciplined approaches to building secure systems. The model describes a secure computer system abstractly, without regard to the system's application. Its approach is to define a set of system constraints whose enforcement will prevent any application program executed on the system from compromising system security. The model includes *subjects*, which represent active entities in a system (such as active processes), and *objects*, which represent passive entities (such as files and inactive processes). Both subjects and objects have security levels, and the constraints on the system take the form of axioms that control the kinds of access subjects may have to objects.

One of the axioms, called the **-property* ("star-property"), prohibits a subject from simultaneously having read access to one object at a given security level and write access to another object at a lower security level. Its purpose is to prevent subjects from moving data of a given security level to an object marked with a lower

[2] The work described in this paper was performed while all three authors were associated with the Naval Research Laboratory.

security level. Originally, the model applied this constraint to all subjects, since a subject might execute any arbitrary application program, and arbitrary programs executed without this constraint could indeed cause security violations.

A system that strictly enforces the axioms of the original Bell-LaPadula model is often impractical: in real systems, users may need to invoke operations that, although they do not violate our intuitive concept of security, would require subjects to violate the *-property. For example, a user may need to extract an UNCLASSIFIED paragraph from a CONFIDENTIAL document and use it in an UNCLASSIFIED document. A system that strictly enforces the *-property would prohibit this operation.

Consequently, a class of *trusted subjects* has been included in the model. These subjects are trusted not to violate security even though they may violate the *-property. Systems based on this less restrictive model usually contain mechanisms that permit some operations the *-property prohibits, for example, the *trusted processes* in KS OS [17] and SIGMA [1]. The presence of such mechanisms makes it difficult to determine the actual security policy enforced by the system and complicates the user interface.

To avoid these problems, we propose a different approach. Instead of starting with an application-independent abstraction for a secure computer system and trying to make an application fit on top of it, we start with the application and derive the constraints that the system must enforce from both the functional and security requirements of the application. In this way, it is possible to construct a set of assertions that is enforced uniformly on all the system software. To evaluate our approach, we have formulated a security model for a family of military message systems. Defining an application-based security model is part of a larger effort whose goals are (1) to develop a disciplined approach to the production of secure systems and (2) to produce fully worked-out examples of a requirements document and a software design for such systems. In this paper, we introduce the message system application, discuss the Bell-LaPadula trusted process approach to building secure systems, and present a security model for military message systems both informally and formally.

2 Requirements of Military Message Systems

In recent years, automation has been applied increasingly to the handling of military messages [10]. While the primary purpose of military message systems is to process *formal messages* (i.e., official messages exchanged by military organizations), such systems may also handle informal messages (i.e., unofficial messages exchanged by individuals). Formal messages are transmitted over military networks, such as AUTODIN; their format and use is governed by military standards. Examples of *informal messages* are those currently supported by several message systems (e.g., *HERMES* [19]) available on the ARPA network.

2.1 Functional Requirements

Message system operations may be organized into three categories: operations on incoming messages, operations on outgoing messages, and message storage and retrieval. Operations in the first category permit a user to display and print messages he has received. Second-category operations support the creation, editing, and transmission of outgoing messages. Message storage and retrieval operations allow users to organize messages into message files and to retrieve messages via single keys (e.g., message id) or combinations of keys (e.g., subject and originator). Typically, military systems that process formal messages provide the same operations as systems that process informal messages plus several additional operations, such as distribution determination, action and information assignment, and release [10].

2.2 Security Requirements

Each formal military message is composed of several fields, including *To, From, Info, Date-Time-Group, Subject, Text, Security*, and *Precedence*. A classification, such as UNCLASSIFIED or SECRET, is assigned to each field and to some subfields, for example, the paragraphs of the *Text* field; further, the overall message has a classification that is at least as high as that of any field or subfield. Thus, the *Subject* field of a message may be classified at a lower level than the message as a whole, and two paragraphs of the *Text* field may have different classifications.

In some data processing applications, users process information at a single security level for long periods of time. In contrast, message system users often need to handle data of several classifications during a single computer session. For example, a user may wish to compose an unclassified message based in part on a previous classified message he has received. To accomplish this, he must simultaneously display the classified information and compose the unclassified message. As a further example, the user may wish to scan newly arrived messages and print only those that are unclassifed. To do so, he must display data of several different classifications and then print a hard copy only of the unclassifed data.

Military message systems are required to enforce certain security rules. For example, they must insure that users cannot view messages for which they are not

cleared. Unfortunately, most automated systems cannot be trusted to enforce such rules. The result is that many military message systems operate in "system-high" mode: each user is cleared to the level of the most highly classified information on the system. A consequence of system-high operation is that all data leaving the computer system must be classified at the system-high level until a human reviewer assigns the proper classification.

A goal of our research is to design message systems that are multilevel secure. Unlike systems that operate at system-high, multilevel secure systems do not require all users to be cleared to the level of the highest information processed. Moreover, information leaving such a system can be assigned its actual security level rather than the level of the most highly classified information in the system. Unlike a system that operates at system-high, a multilevel system can preserve the different classifications of information that it processes.

3 Experience with the Bell-Lapadula Model and Trusted Processes

While its complete formal statement is lengthy and complex, the Bell-LaPadula model may be briefly summarized by the following two axioms:

(a) the *simple security rule*, which states that a subject cannot read information for which it is not cleared ("no read up"), and

(b) the **-property*, which states that a subject cannot move information from an object with a higher security classification to an object with a lower classification ("no write down").

These axioms are to be enforced by restricting the access rights that subjects, for example, users and processes, have to objects, for example, files and devices. A less frequently described part of the Bell-LaPadula model is its concept of *trusted subjects*, that is, subjects that are allowed "to operate without the extra encumbrance of the *-property" because they are trusted "never [to] mix information of different security levels" [3]. More precisely, a trusted subject can have simultaneous read access to objects of classification x and write access to objects of classification y, even if the classification of y is less than the classification of x. The formal statement of the Bell-LaPadula model places no constraints on the trusted subject's violations of the *-property.

A number of projects have used the Bell-LaPadula model to describe their security requirements. In these projects, strict enforcement of the Bell-LaPadula axioms without trusted subjects has proved to be overly restrictive. Hence, trusted processes have been introduced as an implementation of the concept of trusted subjects. Below, we summarize experience with the Bell-LaPadula model and trusted processes in four projects: the Military Message Experiment (MME), the Air Force Data Services Center (AFDSC) Multics, the Kernelized Secure Operating System (KSOS), and the Guard message filter.

3.1 MME

The MME's goal was to evaluate the utility of an interactive message system in an operational military environment [23]. During the MME, more than 100 military officers and staff personnel used SIGMA, the message system developed for the experiment, to process their messages [21, 22]. Although SIGMA was built on the nonsecure TENEX operating system, its user interface was designed as though it were running on a *security kernel* (i.e., a minimal, tamperproof mechanism that assures that all accesses subjects have to objects conform to a specified security model). SIGMA's user interface was designed so that it would not change if SIGMA were rebuilt to operate with a security kernel. During the planning phase of the MME, it was decided that SIGMA would enforce the Bell-LaPadula model [1]. This decision led to a number of difficulties, three of which are described below. The first problem arose from the initial decision, later changed, to adopt the model without trusted subjects; the other two problems apply to Bell-LaPadula with or without trusted subjects.

Prohibition of write-downs. The *-property of Bell-LaPadula disallows write-downs; yet, in certain cases, message system users need to lower the classification of information. For example, a user may create a message at TOP SECRET, and, after he has entered the message text, decide that the message classification should be SECRET. A system that strictly enforces the *-property would prohibit a user from reducing the message classification. The user would be required to create a new message at SECRET and re-enter the text.

Absence of multilevel objects. Bell-LaPadula recognizes only single-level objects; some message system data objects (e.g., messages and message files) are inherently multilevel. A computer system that treats a multilevel object as single-level can cause some information to be treated as more highly classified than it really is. For example, when a user of such a system extracts an UNCLASSIFIED paragraph from a SECRET message, the system labels the paragraph SECRET even though the paragraph is actually UNCLASSIFIED.

No structure for application-dependent security rules. Military message systems must enforce some security rules that are absent in other applications. An example is a rule that allows only users with release authority to invoke

the release operation[3]. Such application-dependent rules are not covered by Bell-LaPadula, and, hence, must be defined outside of it.

To address the first problem (and, to some extent, the third), the SIGMA developers designed a trusted process that is not constrained by the *-property and is, therefore, permitted to perform write-downs. For example, a SIGMA user could search a file containing both UNCLASSIFIED and SECRET messages and then display an UNCLASSIFIED message whose citation was returned by the search; such an operation required the intervention of the trusted process since the message citation was transmitted from the SECRET process that did the search to the UNCLASSIFIED process that handled the message display. Unlike the Bell-LaPadula model, which puts no explicit constraints on write-downs performed by the trusted subjects, SIGMA's trusted process narrowly limited the cases in which write-downs were permitted. Ames [1] provides further details on the role of the trusted process in SIGMA.

SiGMA's use of a trusted process was helpful in that it relaxed the rigid constraints of Bell-LaPadula, thus permitting users to perform required operations. However, adding the trusted process also caused a serious problem: it made the security policy that SIGMA enforced difficult to understand. Interviews held during the MME revealed that few SIGMA users clearly understood the security policy that was being enforced. It was an assumption of SIGMA's design that user confirmation of security-relevant operations would prevent security violations. However, because users issued confirmations without comprehending why these confirmations were needed, this assumption was unwarranted.

3.2 AFDSC Multics

In the mid-1970s, Multics was modified to include the Access Isolation Mechanism (AIM). This version of Multics, which has been used at the ADFSC for several years, supports the assignment of security levels to processes and segments and enforces the Bell-LaPadula model. Multics-AIM also contains trusted functions, invoked via a special operating system gate, to enforce access control on objects smaller than a segment, to allow security officers to downgrade files in response to user requests, and to provide other "privileged" operations.

Although Multics-AIM is generally considered a success, experience with it at the AFDSC illustrates some difficulties that arise from strict enforcement of the Bell-LaPadula axioms and from the use of trusted functions. For example, if a user operating at the TOP SECRET level wishes to send an UNCLASSIFIED message to another user operating at the SECRET level, Multics-AIM requires that the message be treated as though it were TOP SECRET. The recipient is not notified of its arrival until he logs in as a TOP SECRET user.

Problems also occur when a user operating at a low security level tries to send mail to a user at a higher level. Mailbox segments in Multics-AIM are special: they have both a minimum and maximum access level. The minimum is defined by the level of the directory that contains the mailbox segment. Thus, a user operating at UNCLASSIFIED is prohibited from sending a message to a mailbox located in a SECRET directory. In this case, the mail could not be sent unless the sender were to log out and log back in at the SECRET level. Because this situation arises frequently, system administrators are allowed to invoke a trusted function that permits them to send mail without logging out and logging back in again.

3.3 KSOS

KSOS [17] was to be a security-kernel based system with a UNIX-compatible program interface on a DEC PDP-11. The KSOS security kernel was designed to strictly enforce the axioms of the Bell-LaPadula model on user-provided programs. To handle those situations where strict enforcement is incompatible with functional requirements, the kernel recognizes certain "privileges" that allow some processes to circumvent parts of this enforcement. These privileges include the ability to violate the *-property to change the security or integrity level [5] of objects, and to invoke certain security kernel functions. KSOS developers defined a special category of software, called *Non-Kernel Security Related* (NKSR), that supports such privileges. For example, the "Secure Server" of the KSOS NKSR allows a user to reduce the security level of files he owns and to print a file classified at a lower security level without raising the security level of the printed output to the level of this process. Both of these operations would be prohibited by strict enforcement of the Bell-LaPadula axioms.

3.4 Guard

The Guard message filter [24] is a computer system that supports the monitoring and sanitization of queries and responses between two database systems operating at different security levels. When a user of the less sensitive system requests data from the more sensitive system, a human operator of the Guard must review the response to ensure that it contains only data that the user is authorized to see. The operator performs this review via a visual display terminal.

One version of the Guard is being built on a security kernel that enforces the axioms of the Bell-LaPadula

[3] Releasing a message is security-relevant because it allows a wider set of users to view the message and because it certifies that a particular military organization originated the message.

model. However, strict enforcement of the *-property is not possible since a major requirement of the Guard system is to allow the operator to violate it, that is, to allow information from the more sensitive system to be sanitized and "downgraded" (or simply downgraded), so that it can be passed to systems that store less sensitive information. An important component of this version's design is the trusted process that performs this downgrading.

3.5 Lessons Learned

Experience has shown that, on one hand, the axioms of the Bell-LaPadula model are overly restrictive: they disallow operations that users require in practical applications. On the other hand, trusted subjects, which are the mechanism provided to overcome some of these restrictions, are not restricted enough. The formal model provides no constraints on how trusted subjects violate the *-property. Consequently, developers have had to develop ad hoc specifications for the desired behavior of trusted processes in each individual system. While such an approach relaxes the rigid enforcement of the *-property, it introduces two additional problems:

(1) Use of the axioms in conjunction with trusted processes makes it difficult to determine the exact nature of the security rules that a system enforces. In the MME and the other three projects described, the security rules enforced by the system as a whole are not the same as the axioms of the model. The actual security rules enforced by each system include both the axioms of the Bell-LaPadula model and the exceptions allowed by the trusted processes.

(2) Because the actual policies in practical systems deviate from the Bell-LaPadula axioms, any inductive proof that such a system maintains a secure state, based on strict enforcement of the axioms of the model, is a proof about only part of the system and cannot apply to the entire system. Moreover, trusted subjects do not address directly[4] the two other problem areas of the Bell-LaPadula model discussed above, that is, its failure to support multilevel objects and its lack of a structure for including application-dependent security rules.

[4] Indirectly, trusted subjects can implement any arbitrary security policy. For example, a trusted subject that acts as a type manager can provide multilevel objects, and application-dependent security rules can be enforced by making controlled operations available only through trusted subjects. Our point here is that the notion of trusted subjects in itself serves only to draw a circle around the aspects of security policy not addressed by the axioms of the Bell-LaPadula model. It does not provide any framework for formulating that policy.

4 Military Message System (MMS) Security Model

Our goal is to define a single, integrated security model that captures the security policy that a military message system must enforce, without mentioning the techniques or mechanisms used to implement the system or to enforce the policy. The security model defined below is intended to allow users to understand security in the context of message systems, to guide the design of military message systems, and to allow certifiers to evaluate such systems. The model presented here is informal; it is the basis for the formal model presented in the following section.

In this section we define some terms, use them to describe how a user views the system's operation, and state assumptions and assertions, based on the terms and the user's view of operation, that are intended to be sufficient to assure the security of the system. The security model comprises the definitions, user's view of operation, the assumptions, and the assertions. It is a revision of earlier work [13, 16].

This model does not address auditing, although secure message systems clearly require auditing mechanisms. The existence of an audit trail may deter potential penetrators, but auditing is primarily a technique for providing accountability and for detecting security violations after the fact. The security model focuses on assertions that, if correctly enforced, will prevent security violations. Consequently, assertions and assumptions about auditing do not appear; in a more detailed system specification, auditing requirements would be explicit.

The model itself places no constraints on the techniques used to implement a military message system or to verify that a particular system correctly enforces the assertions of the model. An implementation based on a complete formal specification and proof of correctness would be as admissible as one based on a security kernel and trusted processes, or even one employing standard software engineering techniques for design, testing, and validation. By separating the statement of the security model from the concerns of implementation and verification, we can allow for advances in these areas without depending on them.

4.1 Definitions

The definitions below correspond in most cases to those in general use and are given here simply to establish an explicit basis for the model. We distinguish between objects, which are single-level, and containers, which are multilevel. We also introduce the concept of user roles, which define job-related sets of privileges.

Classification[5]: a designation attached to information that reflects the damage that could be caused by unauthorized disclosure of that information. A classification includes a sensitivity level (UNCLASSIFIED, CONFIDENTIAL, SECRET, or TOP SECRET) and a set of zero or more compartments (CRYPTO, NUCLEAR, etc.). The set of classifications, together with the relation defining the allowed information flows between levels, form a lattice [7]. Most dissemination controls, such as NATO, NOFORN, and NOCONTRACTOR, can be handled as additional compartment names.

Clearance: the degree of trust associated with a person. This is established on the basis of background investigations and the tasks performed by the person. It is expressed in the same way as classifications are, as a sensitivity level and a {possibly null) compartment set. In a secure MMS, each user will have a clearance, and operations performed by the MMS for that user may check the user's clearance and the classifications of objects to be operated on. Some other characteristics of a user, such as his nationality and employer, may also be treated as part of this clearance so that dissemination controls are handled properly.

UserID: a character string used to denote a user of the system. To use the MMS, a person must present a userID to the system, and the system must authenticate that the user is the person corresponding to that userID. This procedure is called logging in. Since clearances are recorded on the basis of one per userID, each user should have a unique userID.

User: a person who is authorized to use the MMS.

Role: the job a user is performing, such as downgrader, releaser, distributor, and so on. A user is always associated with at least one role at any instant, and the user can change roles during a session. To act in a given role, the user must be authorized for it. Some roles may be assumed by only one user at a time {e.g., distributor). With each role comes the ability to perform certain operations.

Object: a single-level unit of information. An object is the smallest unit of information in the system that has a classification. An object thus contains no other objects; it is not multilevel. There are many kinds of objects; an example is the date-time-group of a message.

Container: A multilevel information structure. A container has a classification and may contain objects (each with its own classification) and/or other containers. In most MMS family members, message files and messages are containers. Some fields of a message (such as the *Text* field) may be containers as well. The distinction between an object and a container is based on type, not current contents: within a family member, if an entity of type message file is a container, then all message files in that family member are containers, even if some of them are empty or contain only objects and/or containers classified at the same level as the message file itself. Devices such as disks, printers, tape drives, network interfaces, and users' terminals will be containers, rather than objects, in most MMS members.

Entity: either a container or an object.

Container Clearance Required (CCR): an attribute of some containers. For some containers, it is important to require a minimum clearance, so that if a user does not have at least this clearance, he cannot view any of the entities within the container. Such containers are marked with the attribute Container Clearance Required (CCR). For example, a user with only a CONFIDENTIAL clearance could be prohibited from viewing just the CONFIDENTIAL paragraphs of a message classified TOP SECRET by marking the message (which is a container) "CCR." On the other hand, given a message file containing both TOP SECRET and CONFIDENTIAL messages, it may be acceptable to allow the user in question to view the CONFIDENTIAL ones, even though the container (message file) as a whole is classified TOP SECRET. In this case, the file would not be marked "CCR."

ID: identifier. An ID names an entity without referring to other entities. For example, the name of a message file is an ID for that file. Some, but not necessarily all, entities can be named by identifiers. Entities may also be named by indirect references (see below).

Direct reference: a reference to an entity is direct if it is the entity's ID.

Indirect reference: a reference to an entity is indirect if it is a sequence of two or more entity names (of which only the first may be an ID). An example is "the current message's Text field's third paragraph."

Operation: a function that can be applied to an entity. It may simply allow that entity to be viewed (e.g., display a message), or it may modify the entity (update a message), or both (create a message). Some operations may involve more than one entity (copy a message from one message file to another).

Access Set: a set of triples (userID or role, operation, operand index) that is associated with an entity. The operations that may be specified for a particular entity depend on the type of that entity. If a given operation requires more than one operand, the operand index specifies the position in which a reference to this entity may appear as an operand. For messages, operations include DISPLAY, UPDATE, DELETE, and so on. The existence of a particular triple in the access set implies that the user corresponding to the specified userID or role

[5] This definition corresponds to that used by other authors for security level. In this paper, security level and classification are synonyms.

is authorized to invoke the specified operation on the entity with which the set is associated.

Message: a particular type implemented by an MMS. In most MMS family members, a message will be a container, though messages may be objects in some receive-only systems. A message will include To, From, Date-Time-Group, Subject, Releaser, and Text fields, and additional fields as well. A draft message also includes a Drafter field.

4.2 User's View of MMS Operation

We present the following as a model of the use of a secure MMS. Terms defined above are printed in uppercase.

People can gain access to the system only by logging in. To log in, a person presents a USERID and the system performs authentication, using passwords, fingerprint recognition, or any appropriate technique. Following a successful authentication, the USER invokes OPERATIONS to perform the functions of the message system. The OPERATIONS a USER may invoke depend on his USERID and the ROLES for which he is authorized; by applying OPERATIONS, the USER may view or modify OBJECTS or CONTAINERS. The system enforces the security assertions listed below (that is, it prevents the user from performing OPERATIONS that would contradict these assertions).

4.3 Security Assumptions

It will always be possible for a valid user to compromise information to which he has legitimate access. To make the dependence of system security on user behavior explicit, we list the following assumptions. These assumptions are really security assertions that can only be enforced by the users of the system.

A1. The System Security Officer (SSO) assigns clearances, device classifications, and role sets properly.

A2. The user enters the correct classification when composing, editing, or re-classifying information.

A3. Within a classification, the user addresses messages and defines access sets for entities he creates so that only users with a valid need-to-know can view the information.

A4. The user properly controls information extracted from containers marked CCR {i.e., exercises discretion in moving that information to entities that may not be marked CCR).

The basis for these assumptions is that when there is no other source of information about the classification of an entity or the clearance of a person, the user is assumed to provide information that is correct.

4.4 Security Assertions

The following statements hold for a multilevel secure MMS:

Authorization

1. A user can invoke an operation on an entity only if the user's userID or current role appears in the entity's access set along with that operation and with an index value corresponding to the operand position in which the entity is referred to in the requested operation.

Classification hierarchy

2. The classification of any container is always at least as high as the maximum of the classifications of the entities it contains.

Changes to objects

3. Information removed from an object inherits the classification of that object. Information inserted into an object must not have a classification higher than the classification of that object.

Viewing

4. A user can view (on some output medium) only an entity with a classification less than or equal to the user's clearance and the classification of the output medium. (This assertion applies to entities referred to either directly or indirectly).

Access to CCR entities

5. A user can have access to an indirectly referenced entity within a container marked "Container Clearance Required" only if the user's clearance is greater than or equal to the classification of that container.

Translating indirect references

6. A user can obtain the ID for an entity that he has referred to indirectly only if he is authorized to view that entity via that reference.

Labeling requirement

7. Any entity viewed by a user must be labeled with its classification

Setting clearances, role sets, device levels

8. Only a user with the role of System Security Officer can set the clearance and role set recorded for a user or the classification assigned to a device. A user's current role set can be altered only by that user or by a user with the role of System Security Officer.

Downgrading

9. No classification marking can be downgraded except by a user with the role of downgrader who has invoked a downgrade operation.

Releasing

10. No draft message can be released except by a user with the role of releaser. The userID of the releaser must be recorded in the "releaser" field of the draft message.

4.5 Discussion

The purpose of this subsection is to clarify the effects of the model in particular cases. The paragraphs below are not part of the model; the previous subsections define the model completely. Here we seek only to show how the model applies in specific circumstances.

(1) What prevents a user from copying a classified entity to an unclassified entity?

The classification of the entity being copied accompanies the data. Moving explicitly classified data to an unclassified container is a violation of assertion 2 (classification hierarchy) and 9 (downgrading), unless the user requesting the operation is the downgrader and is performing a downgrade operation, since the classification of the data in question is effectively changed by the operation. Manipulations that affect only objects are covered by assertion 3 (changes to objects).

(2) What about copying a part of an object into another object?

A part of an object inherits the classification of the whole object (assertion 3, changes to objects). Thus moving part of an object into another object is disallowed by assertions 2 (classification hierarchy) and 3 unless classification of the former object is less than or equal to that of the latter. Note that this constraint does not affect the user's ability to remove an UNCLASSIFIED paragraph (an object) from a CONFIDENTIAL document (a container) and use it in an UNCLASSIFIED document (another container).

(3) Does a user have a "login level"?

Login levels are not explicitly part of the model, but the effect of a login level can be obtained through the classification of the user's terminal. The classification of the terminal is an upper bound on the classification of information that can be displayed on it (assertion 4, viewing). If the user wishes to restrict further the level of the information that appears on the terminal, he may invoke an operation to reduce the classification of the terminal. The right to determine the classification of shared devices (disks, printers, etc.) will generally belong to the SSO. Note that restricting the level of the information that can appear on the user's terminal does not necessarily restrict the level of information that programs he invokes may have access to.

(4) Processes do not appear in the model but surely will be present in the implementation. How will their activities be constrained?

Operations, rather than processes or programs, are in the model because they correspond more closely to the user's view of the system. To the user, the system offers functions that may be invoked by typing strings of characters, pushing function keys, etc. Each function can be understood by the user as an operation. In the implementation, processes are constrained to invoke only operations that preserve the truth of the assertions.

(5) Which entities in a particular message system will be containers and which will be objects?

This decision is part of the next more detailed level of the stated model. Some likely choices are that messages and message files will be containers and that the date-time group will be an object. It is not necessary that all message systems in the family make the same choices. If two message systems that make different choices communicate, some method of mapping between those entities that are objects in one system and containers in the other must be defined.

(6) How are entities created?

For each type of entity that users can create, there will be an operation that, when invoked, creates a new instance of that type. As with all other operations, only users who are authorized for it can invoke it. Thus, it is not necessarily the case that any particular user will be able to create any particular kind of entity; he must be authorized to do so. In particular, only users authorized for certain roles may be allowed to create certain kinds of entities.

(7) How does a user refer to an object or a container?

Some entities have identifers (IDs) that allow them to be named directly. A given entity may have zero, one, or more IDs. An entity may also be referred to indirectly by a qualified name (see the example under the definition of indirect reference). A user (or a program he invokes) can refer to an entity using any valid ID or qualified name. The former is called a direct reference and the latter an indirect reference.

(8) What policy governs access to an entity in a container? Is the classification of the container or of the contents tested and with what is it compared?

The answer to this question depends on the type of access (the operation invoked) and whether the reference is direct or indirect. If the entity is referred to directly for viewing, assertion 4 (viewing) gives the appropriate restriction. If the reference is indirect, there are two cases depending on whether or not the entity is within a container marked CCR. If it is, both assertions 4 and 5 (access to CCR entities) have an effect; otherwise, only assertion 4 is relevant. Note that a user may be permitted to view a particular entity in a CCR container if he refers to it directly, but be denied access if he refers to it indirectly. This provides a means for dealing with the aggregation problem without requiring duplicate copies of protected information: a collection of CONFIDENTIAL aggregation-sensitive objects might be stored in a container marked SECRET-CCR. A user with a CONFIDENTIAL clearance who had been given the ID of an individual object could refer to it directly, but would be unable to view the same item via an indirect reference that identified it as a member of the SECRET-CCR

container. Assertion 1 (authorization) always requires that the user (or his role) be in the access set for the entity-operation pair specified.

(9) Is there anything in the system that is not (or is not part of or a name for) an entity or a user?

From the user's point of view, no. There may be structures in the implementation that the user is unaware of and would be difficult to assign a legitimate classification to (such as internal operating system queues, perhaps). Anything the user can create, display, or modify, however, must be (or be part of or a name for) an entity or a user.

(10) What are the relationships among a user, an operation he invokes, and programs that the operation may invoke on his behalf?. For example, what privileges do the programs inherit, how is it determined whether a given invocation is allowed under the security policy?

A user has a clearance recorded in the system. When a user invokes an operation on an entity, his clearance and role, the appropriate device classifications, and the classification, CCR mark, and access set for that entity determine whether the operation is permitted. The user's ID or current role must be paired with the specified operation in the access set of the entity in question (assertion 1, authorization). If the operation allows information to be viewed via a given device, then the user's clearance and the classification of the output device must equal or exceed the classification of the information (assertion 4, viewing). Similarly, other security assertions must not be violated by the programs invoked as part of the requested operation.

(11) There are no integrity levels or controls defined in the model. What prevents accidental/malicious modification of sensitive data?

The reasons for omitting integrity levels have been discussed separately [15]. Modifications of clearances, classifications, and role sets are covered in the given set of assertions. To alter data, a user must invoke an operation; assertion 1 (authorization) requires that the user be authorized to invoke that operation. In the future, specific cases may be treated in additional assertions similar to assertion 10 (releasing).

5 Formalizing the MMS Security Model

To provide a firm foundation for proofs about the security properties of a system specification or implementation, a formal statement of its security model is needed. This section presents a formal model that corresponds to the informal MMS security model. It serves three purposes: (1) it is an example of how an informal model of a system's security requirements can be made formal; (2) being abstract, it can be interpreted by others for different but related applications; and (3) it is a basis for proofs about particular message system specifications and implementations.

The MMS security model comprises fifteen definitions, a one-paragraph description of MMS operation, four assumptions about user behavior, and ten assertions that hold for the MMS. We focus on formalizing the ten assertions only, although in doing so, some notation is required to define formal entities that correspond to those discussed informally in the fifteen definitions. Below, the assertions are explicated formally in definition (2) concerning system states and definitions (5) through (11) concerning the system transform. Although the correctness of the explication cannot be proven, we discuss the correspondence between the formalism and the informal model briefly following the explication.

Each MMS family member can be modeled as an automaton with inputs, an internal state, and outputs. The inputs correspond to the commands users give to the system. Because this is a security model, we are principally concerned with modeling the categories of inputs that affect system security. The internal state of the automaton corresponds to the information currently stored in the message system--messages, message files, classifications, access sets, and so on. Output from the automaton consists of command responses--the things that users view or obtain in response to particular requests. These may include entities, classification labels, IDs, and so on. We model output as a set of distinguished entities; although output is treated as part of the internal state, it represents that part that is directly visible to users. Some commands cause a state change that affects the output set, others may cause a change of state without changing the output set, and still others {particularly commands that do not satisfy the security assertions) cause no state change at all. A history of the system is a particular sequence of inputs and states.

5.1 Formal Model

We assume the existence of a set of possible users and a set of possible entities. Given these sets we define system state and the notion of a secure state. Next, we define system and history and introduce constraints on the transform that moves a system from one state to another. A system whose transform meets all these constraints is said to be transform secure. Finally, the notions of secure history and secure system are defined.

The structure of the formal model is intended to simplify its application to defining preconditions and postconditions for system operations. To make explicit the entities that a given operation may change, we define the concept of potential modification based, in part, on the work of Popek and Farber [20]. Potential modification is similar to strong dependency, developed by Cohen [6].

5.1.1 System State

In this section we define what it is to be a system state and what it is for a system state to be secure. We assume the existence of the following sets.

OP is a set of operations.

L is a set of security levels. $\geq$ is a partial order on L such that (L, $\geq$) is a lattice.

UI is a set of userID's.

RL is a set of user roles.

US is a set of users. For all $u \in US$, $CU(u) \in L$ is the clearance of u, $R(u) \subseteq RL$ is the set of authorized roles for u, and $RO(u) \subseteq RL$ is the current role set for user u.

RF is a set of references. This set is partitioned into a set, *DR*, of direct references and a set, *IR*, of indirect references. Although the exact nature of these references is unimportant, we assume that the direct references can be ordered by the integers. In this model we treat each direct reference as a unary sequence consisting of a single integer, for example, <17>. Each indirect reference is treated as a finite sequence of two or more integers, for example, $<n_l, ..., n_m>$, where $<n_l>$ is a direct reference.

VS is a set of strings (bit or character). These strings serve primarily as entity values (e.g., file or message contents).

TY is a set of message system data types that includes *"DM"* for draft messages and *"RM"* for released messages.

ES is a set of entities. For all $e \in ES$ $CE(e) \in L$ is the classification of e. $AS(e) \subseteq (UI \cup RL) \times OP \times N$ is a set of triples that compose the access set of e. $(u, op, k) \in AS(e)$ iff u is a userID or user role authorized to perform operation *op* with a reference to e as *op*'s *k*th parameter. $T(e) \in TY$ is the type of entity e. $V(e) \in VS$ is the value of entity e. If $T(e) = DM$ or $T(e) = RM$, then $V(e)$ includes a releaser field $RE(e)$, which if nonempty, contains a userID. *ES* contains as a subset the set of entities that are containers. For any entity e in this set $H(e) = <e_1, ..., e_n>$ where entity e_i is the *i*th entity contained in e. $CCR(e)$ is *true* iff e is marked *CCR*, else *false*. If $T(e_1) = T(e_2)$ then e_1 and e_2 are both containers or both objects. The set *O* of output devices is a subset of the set of containers[6]. Elements $o \in O$ serve as the domain of two further functions. $D(o)$ is a set of ordered pairs $\{(x_1, y_1), (x_2, y_2) .... , (x_n, y_n)\}$ where each y_i is displayed on o. Each x_i is either a user or an entity, and the corresponding y_i is either a reference, a userID, or the result of applying one of the above functions to x_i.[7] We require that $(x, V(x)) \in D(o) \Rightarrow x \in H(o)$[8]. $CD(o)$ gives the maximum classification of information that may be displayed on o. This allows $CE(o)$ to be used as the current upper limit of the classification of information to be displayed by the output device, so that users can restrict the classification of output to be less than the maximum level permitted.

A state maps a subset of userIDs and references (intuitively, those that exist in the state in question) to elements of *US* and *ES* that represent their corresponding properties. A state also maps a subset of userIDs that "exist" into references that correspond to output devices (intuitively, these users are logged on to the specified devices). To this end we define three mappings. An *id function*, *U*, is a one-to-one mapping from a (possibly improper) subset of *UI* into *US*. A reference function, *E*, is a mapping from a (possibly improper) subset of *RF* into *ES* such that for all $n \geq 2$, $E(<i_1, ..., i_n>) = e$ iff $E(<i_1, ..., i_{n-1}>) = e^*$ where e^* is a container such that e is the i_nth element of $H(e^*)$. For any reference r, if $E(r) = e$, we say that r is a reference to e (relative to E). A login function, *LO*, is a one-to-one mapping from a (possibly improper) subset of *UI* into *RF*.[9]

Given a reference function, *E*, each indirect reference of the form $<n_0, ..., n_m>$ to an entity e_m corresponds to a path of entities $<e_0, ..., e_m>$ such that each $e_i \in rng(E)$, e_0 is denoted by the direct reference $<n_0>$, and for all positive integers $i \leq m$, e_i is the n_ith entity in container e_{i-1}. Such an indirect reference is said to be *based* on each entity e_j where $0 \leq j < m$.

---

[6] In implementations, some kinds of output "disappear" from the system state (e.g., information sent to a printer or a telecommunications port) while others persist (e.g., information displayed on the screen of a terminal, which a user may later refer to and modify). In the formalization, we do not distinguish between these types; both are intended to be covered by *O*.

[7] Both the item and what is displayed must be specified so that, for example, cases in which two entities have identical values but different security levels can be distinguished.

[8] We extend the set theoretic notions of membership and intersection to apply to tuples in the obvious sense.

[9] The condition that *LO* is a function reflects an assumption that a user cannot be on two terminals at the same time. We also assume that a user's output is directed to the terminal he is on. These assumptions are merely for ease of exposition and are not an essential part of the model. One way to appropriately restrict output that is not directed to the user's terminal would be to consider a user logged in on a device when he directs output to it.

Definition 1. A system state, s, is an ordered triple[10] *(U, E, LO)* where U is an id function, E is a reference function, and LO is a login function such that $dom(LO) \subseteq dom(U)$ and $rng(LO) \subseteq dom(E \cap (RF \times O))$. We also require that if $o \in rng(E) \cap O$ and $(x, y) \in D(o)$, then $x \in rng(E) \cup rng(U)$ to assure that only information about users and entities that "exist" in the current state can actually be displayed, and that for any reference r, $(x, r) \in D(o) \Rightarrow E(r) = x$. Finally, we require that $E(LO(u_1)) = E(LO(u_2)) \Rightarrow u_1 = u_2$ to prevent two users from being logged in simultaneously on the same terminal.

Given a system state $s = (U, E, LO)$, we abbreviate $E(r)$ by r_s, $U(u)$ by u_s, and $E(LO(u))$ by $\hat{u}_s$.

Definition 2. A state s is *secure* if $\forall x, y \in rng(E)$, $\forall o \in O \cap rng(E)$, $\forall w \in dom(LO)$, and $\forall u \in rng(U)$:

$x \in H(y) \Rightarrow CE(x) \leq CE(y)$,

$x \in H(\hat{w}_s) \Rightarrow CU(w_s) \geq CE(x)$,

$(x, V(x)) \in D(o) \Rightarrow (x, CE(x)) \in D(o)$,

$RO(u) \subseteq R(u)$, and

$CD(o) \geq CE(o)$.

5.1.2 Secure System

In this section we define what a system is and what it is for a system to be secure.

Definition 3. A system Σ is a 4-tuple (I,S, s_0, T) where

I is the set of well-formed system requests, where each request $i \in I$ is of the form $<op, x_1, x_2 \ldots x_n>$ where each $x_j \in RF \cup UI \cup VS$ and $op \in OP$;

S is the set of possible system states;

s_0 designates a special state called the initial state; and

T is the *system transform*, that is, a function from $UI \times I \times S$ into S.

Definition 4. A history, Π, of a system is a function from the set of nonnegative integers N to $UI \times I \times S$ such that (1) the third element of $\Pi(0)$ is s_0, and (2) for all $n \in N$, if $\Pi(n) = (u, i, s)$ and $\Pi(n + 1) = (u^*, i^*, s^*)$, then $T(u, i, s) = s^*$.

Before defining what it means for an operation to potentially modify a reference[11], we notice that a reference function E, and *a fortiori* a state s, induces a set of functions defined on references that are counterparts to the set of functions introduced above that are defined on entities. For example, there is a function, call it V_s, such that $V_s(r) = V(r_s)$. Similarly, there is a counterpart predicate, call it H_s, such that $H_s <r, r^1, \ldots, r^n>$ iff $H(r_s) = <r, r_s^1, \ldots, r_s^n>$. Each counterpart is the user-visible version of the corresponding entity function. We call these *referential counterparts* and use them to define what it means for two states to be equivalent except for a set of references.[12]

State $s = (U, E, LO)$ and $s^* = (U^*, E^*, LO^*)$ are *equivalent except for some set of references* ρ iff (1) $U = U^*$, (2) $LO = LO^*$, (3) $dom(E) = dom(E^*)$, (4) for any entity function F except V, $F_s = F_{s^*}$, and (5) for any reference $r \in dom(E) - \rho$, $V_s(r) = V_{s^*}(r)$.

We now define potential modification as follows:

u, i, s potentially modify r iff $\exists s_1, s_{1^*}$: s_1 is equivalent to s except possibly for some set of references and $T(u, i, s_1) = s^*_1$ and for some entity function F,

$$F(r_{s_1}) \neq F(r_{s_1^*}).$$[13]

Call y a *contributing factor* in such a case iff $y = r$ or $\exists s_1$ as above and s_2, s^*_2: s_1 and s_2 are equivalent except for $\{y\}$ and $T(u, i, s_2) = s^*_2$ and $F(r_{s_2^*}) \neq F(r_{s_1^*})$.

That is, *u, i, s* potentially modifies r if there is some (second) state that may differ from s in the values of some entities, and T maps u, i, and this state into a third state in which some entity function F (value, containment, access set, etc.) on r differs from the second state. The contributing factors are r and those entities whose values affect the final $F(r)$.

For each referential counterpart and each function defined on users, we posit a unique operation that changes an existing entity or user with respect to that function. For example, an operation *set__AS*(*r*, *new_access_set*) is the only operation that affects *r*'s access set, and it has no other user-visible effect. Further, if the transition is, for example, from state s to state s^*, $AS_{s^*}(r)$ is *new_access_set* if *new_access_set* is a character string and V_s(*new_access_set*) if *new_access_set* is an entity reference. Changes to the domain of E or U (creation or deletion of entities or users) are also assumed to occur only by explicit request. The formal release operation defined below is the single exception to this assumption

[10] State is defined as a tuple, rather than as a set of functions, because two states whose elements have the same values are in fact identical, while two entities for which the defined functions return the same values may in fact be different (e.g., two copies of the same citation).

[11] The version of the paper published in ACM TOCS incorrectly had "entity" in place of "reference" at this point. The definition of "potential modification" was given correctly, however.

[12] We could have developed the entire formal model in terms of referential counterparts, but preferred the simplicity of functions to working with the predicate H_s.

[13] This covers cases of creation (and deletion) since $F(r_{s_1})$ will be undefined and $F(r_{s_1^*})$ will be defined (although possibly empty).

(besides *delete*(*r*) and possibly, *create*(*r*)); it changes the type of *r* and, potentially, the releaser field of *r*'s value as well.

The exact nature of these operations is unimportant since these assumptions are included solely for ease of exposition. Their purpose is not to rule out implementation commands that affect different parts of entities, but to eliminate the problem of unspecified side effects in the formal model (e.g., permission to view a message marked *CCR* is not permission to clear the *CCR* mark). Implementation commands that can alter more than a single part of a single entity correspond to a sequence of formal operations. For a given implementation, this correspondence is determined by the semantics of the implementation command language. Once this correspondence has been determined, so that the security-relevant effects of each user command are clear, *I* can be replaced by the set of implementation commands with access sets also changed accordingly. Nevertheless, prudence dictates that modifications (e.g., changing a user's clearance) that can be made only by the *security officer*, be restricted so that there is only a single command that performs them in any implementation.

The following constraints on the system transform lead to the definition of a *secure history* and a *secure system*. Where quantification is not explicit below, universal quantification is assumed.

Definition 5. A transform *T* is *access secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, [($op \in i \cap OP$ and $r_k \in i \cap RF$) $\Rightarrow$ ((u, op, k) $\in AS(E(r_k))$ or $\exists\ l \in RO(u_s)$, and (l, op, k) $\in AS(E(r_k))$)] or $s=s^*$. [14]

Definition 6. A transform *T* is *copy secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, x is potentially modified with y as a contributing factor $\Rightarrow CE(x_s) \geq CE(y_s)$.

Definition 7. A transform *T* is *CCR secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, $r \in i \cap IR$ is based on y and $CCR(y)$ and z is potentially modified with r as a contributing factor $\Rightarrow CU(u_s) \geq CE(y)$.

Definition 8. A transform *T* is *translation secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, $x \in DR$ and $(x_{s^*}, x) \in D(\hat{u}_{s^*})$ $\Rightarrow \exists\, r \in i \cap RF$, $r_s = x_s$ and (r is based on z and $CCR(z)$ $\Rightarrow CU(u_s) \geq CE(z)$). [15]

Definition 9. A transform *T* is *set secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, (a) $\exists\, o \in dom(E \cap (RF \times O))$, $CD(o_s) \neq CD(o_{s^*})$ or $\exists\, x \in dom(U)$, $CU(x_s) \neq CU(x_{s^*})$ or $R(x_s) \neq R(x_{s^*}) \Rightarrow security\_officer \in RO(u_s)$; and (b) $x \in dom(U)$ and $RO(x_s) \neq RO(x_{s^*}) \Rightarrow u_s = x_s$ or $security\_officer \in RO(u_s)$.

Definition 10. A transform *T* is *downgrade secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, $x \in dom(E - (RF \times \{\hat{u}_s\}))$ and $CE(x_s) > CE(x_{s^*}) \Rightarrow downgrader \in RO(u_s)$.

Definition 11. A transform *T* is *release secure* iff $\forall\, u, i, s, s^*$: $T(u, i, s)=s^*$, ($T(x_s)=RM \Rightarrow T(x_{s^*})=RM$ and $RE(x_{s^*})=RE(x_s)$) and ($T(x_s)\neq RM$ and $T(x_{s^*})=RM \Rightarrow RE(x_{s^*})=u$, $\exists\, r$: $r_s = x_s$, i is the operation $<release, r>$, $releaser \in RO(u_s)$ and $T(x_s) = DM$).

Definition 12. A transform is *transform secure* iff it is access secure, copy secure, CCR secure, translation secure, set secure, downgrade secure, and release secure.

Definition 13. A history is *secure* if all its states are state secure and its transform is transform secure.

Definition 14. A system is *secure* if each of its histories is secure.

5.2 Discussion

Perhaps the most basic decision we made in formalizing the MMS model concerns our general conception of a computer system, in particular the relation between a system state and a system. We considered a view where a system state consists of entities and their relations, and where a system adds to this users and user operations on entities. Hence, all restrictions on user properties (in particular, the restriction for all u, $RO(u) \subseteq R(u)$) are included in the definition of a secure system. We chose instead to view the distinction between system states and systems in terms of static as opposed to dynamic properties. *Static properties* are those that hold for all secure states and, hence, can be checked by examining a state in isolation; *dynamic properties* are those that hold for the relation between secure states and, hence, can be checked only by comparing two or more states. In the view we adopted, all static security properties are included in the definition of a secure state.

To a large extent the choice in conceptualizations is a matter of taste. Bell and LaPadula [4] use the latter, while Feiertag et al. [8] lean to the former. By minimizing the notion of a secure state, the former view makes the Basic Security Theorem shorter. The deciding factor in our adopting the latter view is that it makes it impossible for a system to undergo a security-relevant change without undergoing a change in state.

Principal difficulties we encountered in formalizing the MMS security model were in representing "copy" and "view," system output, and the notion of an authorized operation. Assertion 3 (changes to objects) in the informal model requires formal semantics to reflect the movement of information between entities, while assertion 4

[14] For simplicity we disregard error messages in the formalism. In an implementation we assume that if an unauthorized operation is attempted, an appropriate error message will be produced in the next state.

[15] Strictly speaking, references can be written to an entity without violating translation secure only if they are not displayed. A practical implementation that satisfies the spirit of this policy is to permit the writing of references to an entity but only on the condition that the reference could have been displayed without violating translation secure.

(viewing) requires formal semantics to reflect making an entity visible to a user. Assertion 5 (accessing CCR entities) now addresses both copying and viewing. The semantics for "copy," embodied in the definitions of "potential modification" and "contributing factor," are based on a broad interpretation of "copy." Information is considered to be copied, not only if it is directly moved from one entity to another, but also if it contributes to the potential modification of some other entity. For example, if an operation scans message file A and copies messages selected by a filter F to message file B, both A and F contribute to the potential modification of B (and are therefore subject to the constraints imposed by *copy secure* and *CCR secure*), even if both A and F are empty. The semantics for "view" are straightforward: a thing is viewed if an operation makes it a member of an output container. In light of these considerations, we have used "access" instead of "view" in assertion 5.

In the formalization, system output is interpreted as a set of containers; other entities, parts of entities, references, and classifications that are made visible to a user are interpreted as being copied to his output container. We assume that in any implementation the classifications displayed appear close to the entities (or parts) they correspond to, but we have not formalized this assumption. References are explicitly included as a part of output because the same operation applied to the same entities can yield different results, depending on how the entities are referenced. This leads to the constraint (*translation secure*) on operations that produce as output direct references that are translations of indirect ones. To enforce this constraint, the system must recognize references as a particular kind of output.

Formalizing the concept of an authorized operation is difficult because the semantics of authorized operations are unspecified. Our definition of *access secure* requires that, if an operation changes the system state (beyond producing an error message as output), then for each entity in the set of operands the user or role, operation, and operand index must appear in the access set. Unauthorized operations must not alter the system state except to report that they are erroneous.

5.3 Correspondence to the Informal Model

Assertions (2), (4), and (7) of the informal model, concerning classification hierarchy, viewing, and labeling, are incorporated in the formal definition of *secure state*. They correspond respectively to the first three conditions a secure state must satisfy; the last two conditions require that each user's current role set must be a subset of his authorized role set and that the current security level of each output device must be dominated by its maximum allowed level. These last two conditions are implicit in the informal model.

The remaining assertions of the informal model have been translated into constraints on the system transform. Assertion (1) {authorization) corresponds directly to *access secure*, assertion (6) to *translation secure*, and assertions (8)-(10) (setting, downgrading, and releasing) correspond respectively to *set secure*, *downgrade secure*, and *release secure*. *Set secure* restricts the permission to set device classifications and user role sets to security officers and restricts permission to set a user's current role to himself or a security officer. *Downgrade secure* contains an exception for $\hat{u}_s$, so that a user is not prohibited from lowering the current level of his output device. The formal statement of *release secure* makes explicit the requirement that, once released, a message cannot have its type or releaser field altered.

Assertions (3) and (5) correspond to *copy secure* and *CCR secure*. The definition of *copy secure* actually covers parts of both assertion 3 and assertion 4 because output devices are treated as containers. So, if entity x receives information from an object y, $CE(x) \geq CE(y)$ (assertion 3), and if an output container o receives information from entity x, $CE(o) \geq CE(x)$ (assertion 4). *CCR secure* corresponds to assertion 5, under the interpretation that having access to an entity is significant only if that entity is a contributing factor in the potential modification of another entity.

5.4 Storage Channels

Because we have defined *potential modification* and *contributing factor* in terms of changes only to entities, the constraints imposed by *copy secure* and *CCR secure* do not apply to changes made to functions defined on users (clearance, role set, and current role). Thus, there is the potential for information to be transferred from a higher level to a lower one through these functions. However, changes to user clearances and role sets are controlled by *set secure*; the normal user can change only his current role set, and this provides a channel of very limited bandwidth.

Among entities, one class of storage channel remains. Consider two entities with the same classification. The model permits an operation to modify an entity function of one entity based on the value of the other entity. Since entity functions other than value (i.e., containment (H), classification, access set, CCR mark, or type) have no classifications, there is nothing in the model to prohibit a user from viewing those functions, even if he is not cleared to see the entity's value. So information might flow from the value of one entity to the access set of another at the same security level and the change in access set could be observed by a user at a lower security level.

Changes in H offer the greatest opportunity for exploitation, but all of the channels offered by entity

functions could be closed by attributing the classification of the entity value to the other entity functions as well. In practice, the semantics of message system commands should restrict these channels sufficiently so that this will be unnecessary. If designers should find the constraints imposed by the present definitions of potential modification and contributing factor too confining, these could be relaxed by restricting their coverage to a subset of the entity functions. The price of such a change would be the increased potential for storage channels using the excluded functions. The bandwidths of potential storage channels cannot be precisely estimated at the abstract level of the formal model, yet it is clear that the value function should never be excluded from the definitions. Of the other entity functions, *H* is the most problematic both because message system operations are more likely to alter *H* than the other entity functions and because a relatively large amount of information could be encoded in a single change to *H*.

5.5 A Basic Security Theorem for the Formal MMS Security Model

In formalizations where a secure system is a collection of secure states, some feel that a Basic Security Theorem is needed to show the restrictions on system transforms that ensure that a system starting in a secure state will not reach a state that is not secure [4]. Such theorems are of little practical significance, since their proofs do not depend on the particular definition of security provided by the model [18]. Further, in our approach such a theorem is not pressing since the concept of a secure system is defined largely in terms of a secure transform. Nevertheless, we do appeal to the notion of a secure state, and some readers may feel that some form of Basic Security Theorem is needed. Those readers should find it trivial to prove the following analog of the Basic Security Theorem for our definition of a secure state.

THEOREM. *Every state of a system* Σ *is secure if* s_0 *is secure and T meets the following conditions for all u, i, s, s*: T(u, i, s) = s* and for all x, y ∈ RF, w ∈ US:*

(1) $x_s \notin H(y_s)$ and $x_{s^*} \in (y_{s^*}) \Rightarrow CE(x_{s^*}) \leq CE(y_{s^*})$.

(2) $x_s \in H(y_s)$ and $CE(x_{s^*}) \not\leq CE(y_{s^*}) \Rightarrow x_{s^*} \notin H(y_{s^*})$.

(3) $x_s \notin H(\hat{w}_s)$ and $x_{s^*} \in H(\hat{w}_{s^*}) \Rightarrow \mathrm{CU}(w_{s^*}) > CE(x_{s^*})$.

(4) $x_s \in H(\hat{w}_s)$ and $\mathrm{CU}(w_{s^*}) \sim CE(x_{s^*}) \Rightarrow x \notin H(\hat{w}_{s^*})$.

(5) $(x_s, \mathrm{V}(x_s)) \notin \hat{w}_s$ and $(x_{s^*}, \mathrm{V}(x_{s^*})) \in \hat{w}_{s^*} \Rightarrow (x_{s^*}, CE(x_{s^*})) \in \hat{w}_{s^*}$

(6) $(x_s, \mathrm{V}(x_s)) \in \hat{w}_s$ and $(x_{s^*}, CE(x_{s^*})) \notin \hat{w}_{s^*} \Rightarrow (x_{s^*}, \mathrm{V}(x_{s^*})) \notin \hat{w}_{s^*}$

(7) $\mathrm{R}(w_s) \neq \mathrm{R}(w_{s^*})$ or $\mathrm{RO}(w_s) \neq \mathrm{RO}(w_{s^*}) \Rightarrow \mathrm{RO}(w_{s^*}) \subseteq \mathrm{R}(w_{s^*})$.

(8) $CE(\hat{w}_s) \neq CE(\hat{w}_{s^*})$ or $\mathrm{CD}(\hat{w}_s) \neq \mathrm{CD}(\hat{w}_{s^*}) \Rightarrow \mathrm{CD}(\hat{w}_{s^*}) > CE(\hat{w}_{s^*})$.

Together, (1)–(8) are necessary and sufficient conditions for every state of a system to be secure in any system that does not contain states that are unreachable from s_0 .

6 Conclusions

We favor an approach to building secure systems that includes an application-based security model. An instance of such a model and its formalization have been presented. They are intended as examples for others who wish to use this approach. Important aspects of the model are summarized below:

(1) Because it is framed in terms of operations and data objects that the user sees, the model captures the system's security requirements in a way that is understandable to users.

(2) The model defines a hierarchy of entities and references; access to an entity can be controlled based on the path used to refer to it.

(3) Because the model avoids specifying implementation strategies, software developers are free to choose the most effective implementation.

(4) The model and its formalization provide a basis for certifiers to assess the security of the system as a whole.

Simplicity and clarity in the model's statement have been primary goals. The model's statement does not, however, disguise the complexity that is inherent in the application. In this respect, we have striven for a model that is as simple as possible but stops short of distorting the user's view of the system. The work reported here demonstrates the feasibility of defining an application-based security model informally and subsequently formalizing it. The security model described has been used almost without change by another message system project [9], and has been adapted for use in document preparation and bibliographic systems [2].

Judgments about the viability of our approach as a whole must await its application in building full-scale systems. This we are pursuing in the development of message system prototypes [11, 12].

Acknowledgments

Many individuals contributed to the work reported here. Discussions with David Parnas led to an initial version of the security model. Later revisions of the model were based on reviews by Jon Millen, Stan Wilson, Mark Cornwell, Rob Jacob, Jim Miller, Marv Schaefer,

and others too numerous to mention. Participants in the 1982 Air Force Summer Study on Multilevel Data Management Security also provided many helpful comments. For providing the funding that allows us to continue our work, we are grateful to H. O. Lubbes of the Naval Electronic Systems Command and to the Office of Naval Research.

References

1. AMES, S.R., JR., AND OESTRE1CHER, D.R. Design of a message processing system for a multilevel secure environment. In Proceedings of the AFIPS 1978 National Computer Conference (June 5-8), Vol. 47. AFIPS Press, Reston, Va., 765-771.

2. Air Force Studies Board. Multilevel Data Management Security. Commission on Engineering and Technical Systems, National Research Council, National Academy Press, Washington, D.C., 1983.

3. BELL, D.E. Secure computer systems: A refinement of the mathematical model. MTR-2547, Vol. III, MITRE Corp., Bedford, Mass., Apr. 1974, 30-31. Available as NTIS AD 780 528.

4. BELL, D.E., AND LAPADULA, L.J. Secure computer system: Unified exposition and Multics interpretation. MTR-2997, MITRE Corp., Bedford, Mass., Mar 1976. Available as NTIS ADA 023 588.

5. BIBA, K.J. Integrity considerations for secure computer systems. ESD-TR-76-372, ESD/AFSC. Hanscom AFB, Bedford, MA, Apr. 1977 (available as MITRE MTR-3153, NTIS AD A039324).

6. COHEN, E. Information transmission in computational systems. In Proceedings of the 6th ACM Symposium on Operating Systems Principles, West Lafayette, Ind. ACM SIGOPS Oper. Syst. Rev 11, 5, (Nov. 1977), 133-139.

7. DENNING, D.E. A lattice model of secure information flow. Commun ACM 19, 5 (May 1976), 236-243.

8. FEIERTAG, R.J., LEVITT, K.N., AND ROBINSON, L. Proving multilevel security of a system design. In Proceedings of the 6th ACM Symposium on Operating Systems Principles, West Lafayette, Ind. ACM SIGOPS Oper. Syst. Rev. 11, 5 (Nov. 1977), 57-65.

9. FORSDICK, H.C., AND THOMAS, R.H. The design of a Diamond--A distributed multimedia document system. BBN Rep. 5204, Bolt, Beranek, and Newman, Cambridge, Mass., Oct. 1982.

10. HEITMEYER, C.L., AND WILSON, S.H. Military message systems: Current status and future directions. IEEE Trans. Commun., COM-28, 9, (Sept. 1980), 1645-1654.

11. HEITMEYER, C.L., LANDWEHR, C.E., AND CORNWELL, M.R. The use of quick prototypes in the secure military message systems project. ACM SIGSOFT Softw. Eng. Notes 7, 5 (Dec. 1982), 85-87.

12. HEITMEYER, C.L., AND LANDWEHR, C.E. Designing secure message systems: The Military Message Systems (MMS) project. In Proceedings of the IFIP 6.5 Working Conference on Computer-Based Message Services (Nottingham, U.K., May 1984) Elsevier North-Holland, New York, pp. 245-255.

13. LANDWEHR, C.E. Assertions for verification of multilevel secure military message systems. ACM SIGSOFT Softw. Eng. Notes 5, 3 (July 1980), 46-47.

14. LANDWEHR, C.E. Formal models for computer security. ACM Comput. Surv. 13, 3 (Sept. 1981), 247-278.

15. LANDWEHR, C.E. What security levels are for and why integrity levels are unnecessary. NRL Tech. Memo 7590-308:CL:uni, Naval Research Laboratory, Washington, D.C., Feb. 1982.

16. LANDWEHR, C. E., AND HEITMEYER, C.L. Military message systems: Requirements and security model. NRL Memo. Rep. 4925, Naval Research Laboratory, Washington, D.C., Sept. 1982. Available as NTIS ADA 119 960.

17. MCCAULEY, E.J., AND P.J. DRONGOWSKI. KSOS--The design o f a secure operating system. In Proceedings of the AFIPS 1979 National Computer Conference (June 4-7), Vol. 48. AFIPS Press, Reston, Va., 345-353.

18. MCLEAN, J. A comment on the basic security theorem of Bell and LaPadula. Inf. Proc. Lett., Elsevier North-Holland, New York, 1984, to be published.

19. MOOERS, C.D. The HERMES guide. BBN Rep. 4995, Bolt, Beranek, and Newman, Cambridge, Mass., Aug. 1982.

20. POPEK, G.J., AND FARBER, D.A. A model for verification of data security in operating systems. Commun. ACM 21, 9 (Sept. 1978), 737-749.

21.ROTHENBERG, J. SIGMA message service: Reference manual, Version 2.3, Rep. ISI/TM-78-11.2, USC/Inform. Sci. Inst., Marina del Rey, Calif., June 1979. Available as NTIS ADA 072 840.

22. STOTZ, R., TUGENDER, R., AND WILCZYNSKI, D. SIGMA--An interactive message service for the military message experiment. In Proceedings of the AFIPS 1979 National Computer Confer-ence, (June 4-7, 1979), Vol. 48. AFIPS Press, Reston, Va. pp. 855-861.

23. WILSON, S.H., GOODWIN, N.C., BERSOFF, E.H., AND THOMAS, N.M., III. Military message experiment--Vol. I executive summary. NRL Rep. 4454, Naval Research Laboratory, Washington, D.C., Mar. 1982. Available as NTIS ADA 112 789.

24. WOODWARD, J. P.L. Applications for multilevel secure operating systems. In Proceedings of the AFIPS 1979 National Computer Conference (June 4-7), Vol. 48. AFIPS Press, Reston, Va. 1979, pp. 319-328.

Additional references

25. PAYNE, C., FROSCHER, J. AND LANDWEHR, C. Toward a comprehensive INFOSEC certification methodology. Proc. Sixteenth National Computer Security Conference, Baltimore, MD, Sept., 1993. pp. 165-172.

26. FROSCHER, J. AND CARROLL, J. Security requirements of Navy embedded computers. NRL Memorandum Report 5425, Naval Research Laboratory, Washington, D.C., Sept. 1984.

SMMS Chronological Bibliography

27. LANDWEHR, C.E. Assertions for verification of multilevel secure military message systems. ACM SIGSOFT Softw. Eng. Notes 5, 3 (July 1980), 46-47.

28. HEITMEYER, C.L., AND WILSON, S.H. Military message systems: Current status and future directions. IEEE Trans. Commun., COM-28, 9, (Sept. 1980), 1645-1654.

29. HEITMEYER, C. AND CORNWELL, M. Specifications for three members of the military message system (MMS) family. NRL Memorandum Rep. 5654, Naval Research Laboratory, Washington, D.C., Mar., 1982.

30. HEITMEYER, C.L., LANDWEHR, C.E., AND CORNWELL, M.R. The use of quick prototypes in the secure military message systems project. ACM SIGSOFT Softw. Eng. Notes 7, 5 (Dec. 1982), 85-87.

31. HEITMEYER, C.L., AND LANDWEHR, C.E. Designing secure message systems: The Military Message Systems (MMS) project. In Proceedings of the IFIP 6.5 Working Conference on Computer-Based Message Services (Nottingham, U.K., May 1984) Elsevier North-Holland, New York, pp. 245-255.

32. LANDWEHR, C., HEITMEYER, C., AND McLEAN, J. A security model for military message systems. ACM Trans. Computer Syst. Vol. 2, No. 3, Aug., 1984, pp. 198-222.

33. CORNWELL, M. AND JACOB, R. Structure of a rapid prototype secure military message system. Proc. 7th DoD/NBS Computer Security Conf., Gaithersburg, MD, Sept, 1984, p. 48-57.

34. TRETICK, B., CORNWELL, M., LANDWEHR, C., JACOB, R., AND TSCHOHL, J. User's manual for the seucre military message system M2 prototype. NRL Memorandum Rep. 5757, Naval Research Laboratory, Washington, D.C., Mar. 1986.

35. JACOB, R. Survey and examples of specification techniques for user-computer interfaces. NRL Rep. 8948, Naval Research Laboratory, Washington, D.C. April, 1986.

36. CORNWELL, M. R. AND MOORE, A.P. Security architecture for a secure military message system. NRL Rep. 9187, Naval Research Laboratory, Washington, D.C., April, 1989.

37. CORNWELL, M. A software engineering approach to designing trustworthy software. Proc. 1989 IEEE CS Symp. on Security and Privacy, May, 1989, pp. 148-156.

38. QUINN, J. T., BULL, A., EVANS, A. A standard organization for specifying abstract interfaces for the SMMS application. NRL Memorandum Rep. 6552, Naval Research Laboratory, Washington, D.C., Sept., 1989.