Reformatted from Proceedings 7th DoD/NBS Computer Security Conference, Gaithersburg, Maryland, September 24-26 1984 (pp. 120–136).

The Security Model of Enhanced HDM

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Abstract

The Enhanced HDM Specification and Verification System being developed at SRI International includes an "MLS Checker" that automatically verifies the security of a certain class of system specifications.

This paper gives a brief and informal overview of the security model on which the MLS checker is based and discusses its application and its relationship to other security models and to the requirements of the DoD Trusted Computer System Evaluation Criteria.

1 Introduction

SRI's Enhanced HDM Specification and Verification System will include a subsystem known as "The MLS Checker" that determines whether system specifications are consistent with the DoD Multilevel Security (MLS) policy. In order to do this, the MLS Checker embodies certain assumptions about:

- The "meaning" of specifications written in Revised Special (this is the specification language of Enhanced HDM),
- The sort of systems whose specifications are to be checked, and
- The interpretation of "security" that is appropriate to that class of systems.

The first of these assumptions concerns the semantics of Revised Special and will not be discussed here; the other two assumptions constitute the **security model** of Enhanced HDM and are the subject of this paper. The security model of Enhanced HDM is the same as that of "Old" HDM, which was developed by Feiertag, Levitt, and Robinson in 1977 [4] and which provided the basis for the original MLS Checking Tool developed by Feiertag [3]. The description of the model has been improved over the years (notably by Goguen and Meseguer [5]); the informal presentation given here is based on the current technical description [13]. It should be stressed that it is only the MLS Checking component of Enhanced HDM that has this (or any other) security model built into it; the rest of the HDM system is a general specification and verification environment that may be used to state and verify arbitrary system properties—including those derived from other security models.

Security models are abstract descriptions of computer systems that concentrate on matters relating to the protection and security of information. Such models are helpful in two aspects of the system design process: *synthesis* and *analysis*. By emphasizing just the features relevant to security, a security model can serve to clarify and guide the design (i.e. synthesis) of a secure system; and, by providing a formal basis for the notion of "security", a model can provide the foundation needed to conduct a rigorous informal analysis, or a formal verification, of the security of a system design. Because of its application to the design of the MLS Checker, it is this second aspect that is emphasized in the security model for Enhanced HDM.

As indicated above, a security model has two components: the first embodies assumptions about the sort of systems that are to be considered, while the second defines a notion of security that is appropriate to that class of systems. I will call these, respectively, the **system** and the **security** components of the model. The **utility** of a model is closely related to the realism of the assumptions that constitute its system component; the **correctness** of a model is a function of its security component.

In order to understand what is meant by the "utility" and "correctness" of a model, it is necessary to understand how a system is verified with respect to a security model. Essentially, verification consists of a demonstration that the system is a valid *interpretation* of the model; that is to say, one must establish a correspondence between the elements of the system and those of the model and must show that the elements of the system interact only in ways that are consistent with the model. A model is of limited utility if few, if any, systems of interest can be shown to be consistent with it; a model is incorrect if a system that has been shown to be consistent with the model fails, nonetheless, to meet its security requirements.

Since one of the main purposes of a security model is to capture security requirements formally and unambiguously, there will generally be no independent formal description of a system's security requirements against which to evaluate the correctness of its model. Furthermore, testing is a notoriously unreliable technique for discovering subtle flaws in a system, and this is especially true in the case of security—where flaws may become manifest only under conditions of sophisticated and deliberate abuse. Thus there is no reliable way of determining whether a system meets its security requirements other than by verifying its compliance with a formal security model. It follows that incorrect security models cannot be countenanced: they just *have* to be right.

The best way to ensure that a model is correct is to make it so simple that it can be *totally* comprehended by suitably skilled persons. In this way, the correctness of a model can be established by the social process of peer review—just as the correctness of a mathematical theorem is established. This requirement for simplicity argues for very abstract security models: ones from which all irrelevant issues, and all details peculiar to a given system, have been stripped away so that the single issue of security is isolated and exposed to scrutiny.

Unfortunately, the desire for highly abstract security models conflicts with one of the realities concerning their application. In practice, the demonstration of consistency between a system and its security model is rarely accomplished in a single step—the "gap" between them is just too great to bridged so simply. (The details of the interpretation would be so complex that they would, themselves, be prone to error). Instead, it is generally an abstract **specification** of the system that is verified with respect to a security model. The step of showing that the actual system is a valid interpretation of its verified specification is generally performed informally. (Formal techniques do exist, but they are hugely expensive). The informality, and possible unreliability, of this second step raises the possibility that undetected security flaws may be introduced during the implementation of a secure specification. In order to reduce the likelihood of such flaws, it is desirable that the specification should be "close" to the implementation—so that the complexity of the informal verification step is reduced as much as possible. In particular, the security mechanisms to be employed in the implementation should be described, in all essential details, in the specification. Formal verification of the specification with respect to a security model, followed by informal verification of the system with respect to the verified specification, then gives considerable confidence in the security of the final system.

The desire to verify detailed, concrete, system specifications argues for a detailed, highly concrete, security model—since otherwise the interpretation from model to specification becomes complex and error-prone and we will be back where we started. Unfortunately, however, such concrete security models often raise the very doubts they are meant to allay: if the model is highly detailed and its definition of security correspondingly so, then one naturally wonders whether that definition is correct. The discovery of subtle quirks and outright flaws in certain well established security models shows that these doubts are not idle [9, 15].

Thus we are confronted with a dilemma: in order to be sure that it is itself correct, a security model should be simple and highly abstract; but in order that it can be used to verify usefully detailed system specifications, a model must be detailed and concrete. In my view, the correct escape from this dilemma is through the horns: rather than argue that abstract models are superior to concrete ones, or vice versa, we should recognize the need for *both*. Having said that, however, I also claim that abstract models should be given primacy. My reason is that **abstract models can be used to verify the correctness of concrete ones**—this is accomplished by showing that the axioms of the abstract model are provable as theorems of the more concrete one. Furthermore, although abstract models cannot be used directly to verify the security of a detailed specification of a system's *implementation*, they have a useful role to play in verifying an abstract specification of its *interface*. This is useful in its own right, since a system may be unsecure for (at least) two reasons: either its specification may include inherently unsecure operations (i.e., its **interface specification** may be unsecure), or its mechanisms may fail to correctly implement otherwise secure operations. It is somewhat heavy-handed to detect flaws of the first kind using models intended to detect those of the second kind. For example, it is surely better to discover immediately if a file system contains operations that permit unclassified users to read classified files, rather than wait until the internal mechanisms of the file system are found to be inconsistent with a concrete security model. An abstract security model can be used to perform the useful task of verifying the security of system interface specifications before resources are committed to its implementation.

The security model of Enhanced HDM is a highly abstract one. Its merits are its simplicity and elegance: it is easy to see that it is correct (as far as it goes—and I will discuss the issue of its completeness later). Its applications are the verification of system interface specifications (performed automatically using the MLS Checker of Enhanced HDM), and the verification of more concrete security models (currently performed by hand). An informal description of the HDM model is given in the next section; technical details may be found in [13].

2 The Model

As explained in the introduction, there are two components to a security model: the system and the security components. The system component of the HDM security model is a conventional finite automaton. That is to say, a computer system is regarded as a "black box" that consumes input "tokens" one at a time and, with each token consumed, changes its own internal state in a manner that depends upon its current state and the value of the input token consumed. At the same time, an output token, whose value is determined in the same way, is emitted and returned to the user who sent the input token that initiated the activity. The internal state of the system is not visible outside; all that can be observed is its input/output behavior.

This automaton model captures the essential characteristics of many types of systems, and system components, quite accurately. Consider, for example, a file server that receives requests from users to save and retrieve files and that returns files and status information to users in response to those requests. The requests sent to the file server can be identified with the input tokens of the model, while the results that it returns can be identified with the output tokens. The internal state of the file server consists of the file system that it maintains. The receipt of a request from a user will cause it to update the file system on the basis of information sent with the request, and to return a result determined by the contents of the file system and the nature of the request.

We now need to add a security component to this system model. The first step is simply to interpret the system model a little differently. Instead of tokens being sent by, and returned to, human users directly, we now recognize that those users may be supported by untrusted computer systems or processes. Whereas a human with legitimate access to classified information may be trusted not to reveal that information to unauthorized persons, a computer system cannot. We indicate that the users of the system are now identified with untrusted computer systems or processes by using the term **subject** instead of "user". We associate a **sensitivity label** with each subject to indicate the **clearance** of the (human) user identified with that subject. The sensitivity labels are assumed to be partially ordered by a

dominates relation[1] that defines the **security policy** to be enforced by the system. This policy requires that no information may flow from one subject to another unless the clearance of the recipient dominates that of the sender. The heart of the security component of the HDM security model is the way in which it gives a precise definition to what it means for information to "flow" from one subject to another.

This definition is beautifully simple: an input from one subject causes **information to flow** to another subject if the outputs subsequently seen by the second subject are *different* from those that it would have seen if the input concerned had not been present. The security component of the HDM model then simply requires that information may flow, in the sense just defined, from one subject to another only if the clearance of the recipient dominates that of the sender.

In the case of the file server example, a request to delete a file obviously causes information to flow to all those subjects who may subsequently determine that the file has been deleted. But our definition of information flow is much stronger than this: it says that information flows to all subjects for whom the file server will subsequently behave *differently* than it would have done if the delete-file request had not been issued. Thus, if the delete-file request causes some disk space to become free and another subject can subsequently discover that the amount of free space has changed, then information has flowed to that subject from the one that sent the delete-file request—and this is to be allowed only if the clearance of that subject dominates that of the subject that sent the delete-file request. It can be seen that this *information-flow* characterization of security is very powerful: it embraces covert storage channels (though not timing channels) as well as direct disclosure.

I claim that this formulation of what is meant by security captures clearly, and correctly, the intent behind other formulations of the property. There is, however, a serious charge that can be brought against the model. The charge is that a valid security model should be based on established government regulations regarding the handling of classified information. This would suggest the introduction of *objects* as the repositories of information "within" the system state and, by analogy with the regulations of the "pen and paper" world, we should demand that objects be labelled

[1]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 [2, p110].

with the sensitivity level of their contents (i.e. their **classification**) and that subjects may only read objects whose classifications are dominated by their own clearances. My objection to this approach is that the regulations that would be taken as the starting point in the construction of our model are not a statement of the *intent* of security procedures, but are a particular set of *mechanisms* that are appropriate for safeguarding security in the "pen and paper" world. In effect, they are a security model whose (unstated) system component is a set of assumptions about the way in which the "pen and paper" world operates. Computer systems do not correspond to these assumptions (they are not passive entities like paper and vaults) and this invalidates the security component of the "pen and paper" model. That this is true is manifest by the need to introduce additional axioms (e.g. the "$*$-property" [1]) into those computer security models that follow this approach.

The attempt to base computer security models closely on established regulations is a laudable one; my objection is to taking this approach as a *starting point*—for then we have no "higher-level" notion of security to appeal to in cases where these mechanisms prove inadequate. The "trusted processes" of the Bell and La Padula model [1] are a case in point. These processes are not constrained by the $*$-property because they are trusted not to violate the *intent* of that model property. The problem is then to establish the security of these processes in the absence of a precise description of just what the "intent" of the model is: because the Bell and La Padula model describes a particular set of security mechanisms, it provides no guidance in cases where its mechanisms prove inadequate.

Instead of identifying security with a particular set of mechanisms, I argue that it is preferable to first enunciate a principle of security that attempts to get at the *intent* behind such mechanisms. Once this has been done, we can introduce particular security mechanisms into our model—mechanisms based on government regulations—and attempt to verify them with respect to our more abstract model. If this attempt succeeds, then we have the satisfaction of knowing that two fairly independent efforts to formalize the same set of concerns have converged on the same (and therefore presumably correct) point; if it fails, then investigation of the discrepancy between the models will sharpen our understanding of the issues concerned and may lead to the discovery, and subsequent correction, of errors in one or both of them. I will describe an experiment of this kind later.

Although I claim that the HDM security model, as formulated so far, provides a useful *definition* of security, it is too abstract to give much practical guidance in the construction or analysis of secure systems. The treatment of a computer system as a "black box" with no internal structure is not helpful to those who must design or analyze the internal mechanisms of a computer system. Furthermore, the definition of information flow is quantified over all *sequences* of future state transitions: that is, information is considered to flow from one subject to another if the presence of an input from the first subject can cause *any change whatever* in the subsequent behavior of the system as perceived by the second. What we would really like is a characterization of security that applies to *single* state transitions, rather than to sequences of transitions.

Both these deficiencies in the most abstract formulation of the model can be remedied by adding more structure to its system component. Instead of treating the system state as a "black box" with no internal structure, we will now assume that the system state is a record of the **values** held in **objects**. We will further assume that each object is assigned a *fixed* sensitivity label called its **classification**. We also need a little more terminology: we will say that the individual steps performed by the system are called **operations**. An operation consumes an input token, causes a state change, and produces an output token. The sensitivity level of an operation is taken to be that of the subject that sent the input token that invoked it.

Given these elaborations to the basic model, it is possible to prove the following result.

Theorem: A system is secure if each of its operations satisfy the following three conditions.

1. The output produced by an operation at sensitivity level l depends only on the values of objects whose classifications are dominated by l.

2. An operation at sensitivity level l changes only the values of objects whose classifications dominate l.

3. If an operation changes the value of an object at classification l, then the new value assigned to the object depends only on the prior values of objects whose classifications are dominated by l.

This result is the one on which the MLS Checker is built. I will discuss its interpretation and application in the next section.

3 Applications

The theorem quoted at the end of the previous section provides the basis for the MLS Checker of Enhanced HDM, and for a similar tool developed earlier by Feiertag [3]. These tools process system specifications that have been augmented with information concerning the sensitivity labels associated with the objects and operations defined in the specifications and then check the specifications of the operations to see that they comply with the three conditions stated in the theorem. This checking involves the generation and proof of putative theorems (called **verification conditions**) concerning relationships among the sensitivity labels of the operations and objects defined in the specification. The Checker also ensures that the specifications are sufficiently **complete** that they define the behavior of the system under *all* conditions. Individually, the conditions that need to be checked are conceptually simple, but they are so numerous and detailed that it is unreliable and uneconomic to attempt the process by hand—it is the existence of automatic MLS Checkers that make this a viable method of security analysis. SRI's original MLS Tool [3] has been used in the analysis of several operational systems, including Honeywell's SCOMP [14]. This section considers what is accomplished by such analysis.

The first point to note is that, as stated earlier, it is only the *interface specification* of the system that can be verified in this way. Because the model requires all objects to be assigned *fixed* classifications, it does not address one of the major problems faced in the development of secure systems: the design and verification of mechanisms that permit system resources to be multiplexed securely among entities belonging to different security classifications. In effect, the HDM model assumes that each security classification operates in its own virtual system. It is possible to modify the security model so that the security classifications of objects are not fixed (the Mitre Corporation has built flow analyzers that do this) but this does not completely solve the problem. Instead, the correct solution to the problem is, in my view, to deny that there is one! Checking interface specifications for security is an important process in its own right. Ensuring that those interface specifications are implemented correctly is an equally important, but different, problem that is best approached as a separate issue using techniques (based on more concrete security models) that are specialized to that problem.

The next point to note is that the system component of the HDM security model assumes that input tokens (i.e. requests for operations to be performed) are "tagged" with their correct sensitivity labels. As I have explained earlier, this is a reasonable assumption for certain "application level" systems, or system components, such as file servers. It is not a valid assumption, however, for really basic system components such as an operating system nucleus. This is the component at the heart of a security kernel that establishes and maintains "process isolation through the provision of distinct address spaces" as required by the DoD Trusted Computer System Evaluation Criteria [2] for Evaluation Classes B1 and above. Requests arriving at the interface of the operating system nucleus are not associated with sensitivity labels provided by some outside agency: it is the task of the nucleus to *establish* this association. Also, operations do not arrive at the nucleus in an orderly, one at a time, fashion. Instead, it is the responsibility of the nucleus to respond to asynchronous interrupts and to establish the orderly transmission of operation requests to the processes that it supports. In short, the nucleus creates an environment for its client processes that is consistent with the HDM security model, while it itself operates in a far more complex and demanding environment. Once again, I do not see it as a weakness of the HDM model that it does not address these issues—that is the task of other, specialized security models [6, 10].

Just as it does not address the issue of the secure implementation of verified interface specifications, nor the security problems of an operating system nucleus, so the HDM security model does not confront security issues other than disclosure through information flow. In particular, the *discretionary* aspects of security policy are not addressed, nor those of *inference* and *aggregation*. Neither are the problems of specialized systems and components such as *databases* and *downgraders* considered. Specialized models are needed in all these cases.

In summary, the HDM security model focuses on just one aspect of the general security problem—but within that limited domain, it does a pretty good job. For the rest, let us have lots of specialized security models that each focus on an individual problem with comparable clarity and precision, and let us learn how to combine these different models in ways that give comprehensive guidance and assurance to the developers of trusted computer systems.

4 Comparison with other Models

One of the most influential of security models is the one developed by Bell and La Padula [1]. Recently, some security flaws have been found in the model [9, 15]—some of its *rules* have been found to admit covert storage channels. In this section, I will show how the attempt to verify the Bell and La Padula model with respect to the HDM model provides a systematic technique for detecting such flaws.

In order to verify the Bell and La Padula model, we must demonstrate that it is a valid interpretation of the HDM model. The details of this demonstration are quite complex, since the two models use different mathematical formalisms. The following is a very informal, outline description of the process; those who desire the technical detail are referred to the appropriate reports [11, 12].

The Bell and La Padula security model is a concrete one, in that it describes explicit security mechanisms. Basically, the system state is partitioned into two components: the **value state** and the **protection state**. The first of these is the (usual) record of the values stored in objects, while the second records the current and maximum security level of each subject, the classification of each object, and the type of access each subject is allowed to each object. In the simplest case, only two types of access need be distinguished: **read** and **write**. If a subject has read access to an object, then it may *use* the value of that object when computing the output of an operation or the value to be stored in an object; it may only *change* the value of an object if it has write access to that object. These mechanisms are assumed to be provided by the system's "hardware".

The operations of the system are divided into two classes: the (regular) **operations**, and the **rules**. Operations access only the value state and are constrained by the "hardware" in the manner described above; operations correspond to the ordinary functions like "add", "load" and "store" etc. Rules, on the other hand, access only the protection state; they perform functions such as "give this subject read access to that object", and "change the classification of this object to that level".

The security component of the The Bell and La Padula model identifies security with the following two (slightly simplified) conditions:

simple security property: a subject may have read access only to objects whose classifications are dominated by its own clearance, and

***-property:** an untrusted subject may have write access only to objects whose classifications dominate its own clearance.

It is quite easy to prove that these two conditions imply those of the theorem given earlier—the simple security property (henceforth abbreviated to *ss*-

property) guarantees the first and third conditions in the statement of the theorem, while the $*$-property guarantees the second. Thus we deduce that the Bell and La Padula model is consistent with the HDM model *in the case of the regular operations.* The *rules* are a different matter, however.

Bell and La Padula gave a representative set of rules (based on those found in Multics) and argued that they were secure because they preserved the ss- and the $*$-properties. However, covert storage channels have subsequently been discovered in some of these rules. (A channel in the rule *change-subject-current-security-level* is described in [9], channels in the rule *change-object-security-level* are described in [11, 15]). These channels arise because the ss- and $*$-properties only consider the problem of information flow through the *value* component of the system state—the possibility of information flow through the *protection state* is not considered explicitly.[2] If one attempts to prove that the rules of the Bell and La Padula model are secure with respect to the HDM model, then the system state of the HDM model must be identified with the conjunction of *both* the value and the protection states from the Bell and La Padula model and the proof fails because certain of the rules permit unsecure information flows through the protection state. Although I have described this process as one performed by hand, it is possible (though I haven't tried it, nor thought through all the details) that it could be accomplished mechanically by constructing a specification of the Bell and La Padula model in Revised Special and then submitting it to the MLS Checker.

The lesson to be learned from this exercise is that the construction of concrete security models is a difficult and error-prone task and that informal review may not be an effective technique for uncovering subtle problems or oversights in such models. (The Bell and La Padula model is nearly ten years old, yet Millen and Cerniglia, who attribute the discovery of the covert storage channel in the rule *change-subject-current-security-level* to P.S. Tasker of the Mitre Corporation, observe that this channel was found only "recently" [9].)

As I noted earlier, the rules present in the Bell and La Padula model were based on functions found in Multics. In order to model other systems, it may be necessary to introduce different rules that correspond more closely

[2]In fact, many of the rules perform checks additional to those necessary to preserve the ss- and $*$-properties. The effect of these checks is to prevent covert storage channels that would otherwise have arisen, but the model does not explain why these checks are necessary, nor how to construct them systematically. It is the inadequacy of the checks in the two rules named above that admit the covert storage channels.

to those present in the systems of interest. For some application areas, completely specialized security models have been developed (see [7, 9] for examples), and this trend is likely to continue as novel applications are contemplated. In all these cases, whether they are new variations on established models, or completely new models, it is highly desirable that some objective, formal analysis of their correctness should be undertaken. In many cases, it seems that part of this analysis can be accomplished by verifying these new models with respect to the HDM model.

In comparison with other security models, the HDM model is much more abstract: it has no security mechanisms built in. However, these other security models can often be viewed as more detailed elaborations of the HDM model. Establishing this connection formally is a good way to evaluate some aspects of the correctness of these other models.

5 Relation to the DoD Trusted Computer System Evaluation Criteria

The DoD Trusted Computer System Evaluation Criteria [2] require the use of a formal security model for Evaluation Class B2 and above. For Evaluation Class A1 and beyond, formal methods are required in the analysis of covert channels (Paragraph 4.1.3.1.3) and a combination of informal and formal techniques must be used to demonstrate consistency between the Formal Top-Level Specification (FTLS) and the model (Paragraph 4.1.3.2.2). The Glossary to [2] provides some guidance on what constitutes an acceptable security model:

> "... to be adequately precise, such a model 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.
>
> "... 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 the assumptions required by the model hold, then all future states of the system will be secure."

A theorem satisfying the requirement of the second paragraph in this quotation is often called a *Basic Security Theorem* after a theorem of that name due to Bell and La Padula. A significant criticism of this requirement is that a Basic Security Theorem says essentially nothing about security—as

McLean [8] demonstrated by proving just such a theorem for a model that clearly violates any reasonable notion of "security".[3] In fact, it should be clear that if Φ is *any* effectively decidable property of the system state, then an analog to the Basic Security Theorem can be constructed for that Φ. As McLean observed, a Basic Security Theorem is really a property of the finite-state system model employed (in that states can be indexed to support proof by induction), rather than of the particular definition given for security.

Bell and La Padula actually made very modest claims for their Basic Security Theorem (and made no subsequent use of it after they had proved it). They observed merely that it established:

> "the relative simplicity of maintaining security: the minimum check that the proposed new state is "secure" is both necessary and sufficient for full maintenance of security" [1, p21].

In my view, the *intent* behind the requirements stated in the DoD Criteria is sound, but the particular requirement for a Basic Security Theorem is poorly chosen. If I may be permitted to interpret the intentions of the authors of that document, I would say that their real requirement was for a *concrete* security model. A concrete model is one, such as that of Bell and La Padula, that describes particular security *mechanisms*, as opposed to the HDM model, which describes only security *policy*. A security mechanism must obviously maintain some state information (recording who may access what, and in what way), and not all states will be equally "secure". Thus, it is natural (indeed, necessary) for a concrete model to prescribe a set of "secure states" and a set of rules which are proven (by a Basic Security Theorem) to be sufficient to guarantee that all state transitions are secure-state-preserving.

The identification of a set of secure states and the proof of a Basic Security Theorem do not, however, guarantee that a model enforces a useful form of security—they simply establish the internal consistency of a set of security *mechanisms*. A separate (preferably formal) justification is required in order to establish that those mechanisms enforce a more abstract statement of required security *policy*. As I have already observed, the HDM security model will serve well in this latter capacity.

Given that the verification of compliance between an actual system and its FTLS will be performed only informally, the requirement that a con-

[3]Basically, McLean turned the $*$-property around, so that subjects may transfer information from higher to lower classification levels.

crete security model be used for the verification of the FTLS is entirely reasonable—for we certainly wish to be sure that the security mechanisms of the system are included in the formal stage of its analysis. Nonetheless, and as noted earlier, the security verification of interface specifications provided by the MLS Checker of Enhanced HDM can also make an important contribution to overall security assurance, especially since it is the only formal technique able to detect covert storage channels. It would seem that the DoD Computer Security Center accepts this view since the verification of the Honeywell SCOMP kernel was largely accomplished with the aid of the MLS Checker of "Old" HDM. Also, the HDM security model continues to apply in those cases where the mechanisms of a concrete model prove inadequate, and trusted process are found to be required. Clarification of the Center's requirements and guidelines on all these topics would be welcome.

6 Summary

I have given an informal description of the security model employed by the MLS Checker of Enhanced HDM. This model is a highly abstract one that has no particular security mechanisms built in. The model gives a precise, formal definition of an information-flow interpretation of security that covers covert storage channels as well as direct disclosure. The model is so simple that there can be no doubt about its correctness. The applications of the model are the verification of system interface specifications and the analysis of more concrete security models.

References

[1] D. E. Bell and L. J. La Padula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, MA, March 1976.

[2] *Department of Defense Trusted Computer System Evaluation Criteria.* Department of Defense, December 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[3] R. J. Feiertag. A technique for proving specifications are multilevel secure. Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, CA, January 1980.

[4] R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *Sixth ACM Symposium on Operating System Principles*, pages 57–65, November 1977.

[5] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the Symposium on Security and Privacy*, pages 11–20, Oakland, CA, April 1982. IEEE Computer Society.

[6] B. A. Hartman. A Gypsy-based kernel. In *Proceedings of the Symposium on Security and Privacy*, pages 219–225, Oakland, CA, April 1984. IEEE Computer Society.

[7] C. E. Landwehr. A survey of formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[8] J. McLean. A comment on the "basic security theorem" of Bell and La Padula. Informal note, Naval Research Laboratory, 1983.

[9] J. K. Millen and C. M. Cerniglia. Computer security models. Working Paper WP25068, Mitre Corporation, Bedford, MA, September 1983.

[10] John Rushby. Proof of Separability—A verification technique for a class of security kernels. In *Proc. 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 352–367, Turin, Italy, April 1982. Springer-Verlag.

[11] John Rushby. The Bell and La Padula security model. Draft report, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1984.

[12] John Rushby. Comparison between the Bell and La Padula and the SRI security models. Draft Report, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1984.

[13] John Rushby. The SRI security model. Draft Report, Computer Science Laboratory, SRI International, Menlo Park, CA, July 1994.

[14] J. M. Silverman. Reflections on the verification of the security of an operating system kernel. In *Ninth ACM Symposium on Operating System Principles*, pages 143–154, Bretton Woods, NH, October 1983. (ACM Operating Systems Review, Vol. 17, No. 5).

[15] T. Taylor. Comparison paper between the Bell and La Padula model and the SRI model. In *Proceedings of the Symposium on Security and Privacy*, pages 195–202, Oakland, CA, April 1984. IEEE Computer Society.