The Specification and Modeling of Computer Security

John McLean

Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, D.C. 20375

Computer security models are specifications designed, among other things, to limit the damage caused by Trojan Horse programs such as computer viruses. Recent work in such models has revealed limitations of the widely accepted model of Bell and LaPadula. This paper provides an introduction to computer security modeling in general, the Bell and LaPadula model in particular, and the limitations of the model. Many of the issues raised are of interest not simply to the security community, but for the software specification community as a whole. We then construct a framework for security models that address these limitations. The result is a model that not only better addresses government security policies, but nongovernment security policies as well.

1. Introduction

Since proving programs correct by formal means is expensive, where feasible at all, such an approach to assurance is cost effective in only a few areas. One of the most important of these is the area of security where the cost of a mistake can be billions of dollars and, at least in the national security arena, human life. Unfortunately, formally proving that a program is secure is especially hard. It's not that formal proofs about security are intrinsically more difficult than proofs about other properties, but rather that the concept of security, itself, is harder to explicate formally. For this reason, there has been a great deal of focus on formally explicating *security* by constructing formal models of security.

If we assume the existence of a set of *objects*, which can be intuitively viewed as consisting of information receptacles, and a set of *subjects*, which can be intuitively viewed as consisting of agents who can operate on objects in various ways, security is the problem of appropriately governing subjects' access to objects. Although identification, authentication, and auditing requirements all fall within the security arena, our concern in this paper is with requirements that restrict the access a legitimate user can have to files. To render the problem nontrivial, we assume that it is not the case that every subject is permitted to access every object.

Access requirements fall into two categories, Discretionary Access Control (DAC) and Mandatory Access Control (MAC). DAC restricts access rights that are based, e. g., on file ownership. The most important property of such access rights is that they can be passed to other users. For example, any right I have to a file based solely on the fact that I own the file, I can pass to another user. As a result, such DAC protection is limited since it is subject to a Trojan Horse attack where I am tricked into executing a program that, without my knowledge, passes my rights on to another user. The most widely known examples of Trojan Horse programs are computer viruses.

MAC provides access restrictions that are not subject to user discretion. As such, MAC limits the damage a Trojan Horse can cause. For example, if we assume the existence of a (partially) ordered set of security levels, such that each subject and each object is assigned a level, and the MAC restriction that subjects assigned to a level cannot read objects assigned to a higher level, then even if we can trick a high-level subject into executing a Trojan Horse program, the program will be unable to pass a low-level subject access rights to a high-level object. In the national security arena, an example of such a set of levels is the familiar hierarchy of *{top secret, secret, confidential, unclassified}*, where *top secret* > *secret* > *confidential* > *unclassified*. The level assigned a subject is called the subject's *clearance*, and the level assigned an object is called the object's *classification*. However, MAC, perhaps under a different name, appears in other arenas as well. For example, medical employees have the right to view patient records, but cannot pass this right on to nonmedical employees. Similarly, advisers cannot pass the right to look at student grades on to other students and payroll officers cannot pass the right to look at employee salaries on to other employees.

In the national security arena, the most widely-used model is that of David Bell and Leonard LaPadula [2]. This model, known as the Bell and LaPadula Model (BLP), serves as the backbone of the National Computer Security Center's evaluation process for trusted computer systems. However, although widely used, BLP has limitations. For example, it has little to say about systems in which users may change security levels of themselves or their files. Yet, such changes are often necessary in real-world systems. Further, it is inadequate for expressing requirements that certain operations cannot be performed by a single individual operating alone. Such n-person rules, or "dual-custody" policies, are contained in government security policies, e. g., policies governing missile launchings, and are an important part of industrial security [3].

This paper describes computer security models, in general, and BLP, in particular. Although we address DAC briefly at the end of the paper, our main topic is MAC, and although our primary focus is the arena of national security, the issues raised are relevant to any setting in which MAC-like restrictions arise. Further, we shall see that security provides a fruitful research area for those with a general interest in software specification since some of the most difficult issues that arise in specifying security have analogues in other domains. We shall examine the limitations of BLP and see how they can be remedied by a framework of models. This framework will help render BLP more useful to those interested in industrial security.

2. Computer Security Specifications and Models

The task of computer security, as distinct from security in general, is to assure that when we computerize an information system, and hence provide users with greater processing power and less human supervision, we do not introduce *new* security threats to the system. We are not concerned with espionage, e. g., in the sense of a user viewing a file he is entitled to see, memorizing its contents, and reproducing the file from memory, or with sabotage or integrity violations, e. g., in the sense of a person maliciously altering a file he is entitled to change. Nor are we concerned with unauthorized users breaking into the system nor denial of service attacks where, e. g., authorized users disable a system by performing authorized operations that reduce system response to an unacceptable level. Our concern is limited to assuring that valid users and programs are not permitted to view data they are not authorized to view and that programs are not permitted to distribute in an unauthorized manner data they are authorized to access. These two components of *security*, not being able to view data classified above one's authorized level and not being able to copy data from one level to a lower level, are known as *simple security* and the **-property*, respectively. They play an integral part in traditional explications of MAC security.

One problem with *simple security* and the **-property* is that they really constitute a possible implementation of security, called a *reference monitor*, rather than an abstract specification that all secure systems must satisfy. By concerning themselves with particular controls over files inside the computer rather than limiting themselves to the relation between input and output, they make it harder to reason about the requirements, themselves, and prejudice the programmer against alternative implementations which may be better [8]. A more abstract specification would ignore system internals and deal directly with the system's input/output relation, requiring, e. g., that output to a user with a low security level does not enable such a user to infer properties of input from users of a high security level. The best known formalization of this constraint requires that high-level inputs do not interfere with low-level outputs [6, 11]. One problem with this approach is that it is limited to systems where high-level output is not generated from low-level input. Hence, it cannot take into account the fact that some systems raise the security level of their input simply by the processing they perform on the input. A second problem is that in nondeterministic systems high-level input cannot simply be restricted from interfering with the possibility of a low-level output occurring, it must be restricted from interfering with the probability that it occurs. For example, a system in which high-level input can significantly reduce the probability of a low-level output occurring can pass significant information even if the high-level input cannot render the low-level output impossible. However, the most serious problem with this approach is that, in general, it is too strong: in most systems we are willing to tolerate high-level input having some effect on low-level output, *e. g.*, with respect to system response, operating system messages concerning available storage, device status, *etc*. Disallowing any such information flow may lead to performance degradation we are unwilling to accept.

By using the concepts of *simple security* and the **-property* we hope to be more selective about the sorts of effects that we consider impermissible. We then leave as a separate task (called a *covert channel analysis*) the job of showing that a malicious program that transfers information *via* system response, operating system messages, *etc.* cannot transfer information at a rapid enough rate to be of concern.

It should be noted that the issues we are dealing with concerning the advantages and disadvantages of specification abstraction arise not only in specifications of security. Consider a mailing system, for example. Ideally, we may want our specification to require that all messages sent are delivered within some time interval t, but when we see the cost of such a system, we may settle for a specification that makes delivery time a function, f, of system load. No matter how precisely we specify f, however, if our specification contains no implementation suggestions, only the most talented programmer will be able to build a system that convincingly satisfies our specification. As a compromise, we may develop a system model of operation and show that the model system satisfies our abstract requirements within acceptable limits. The programmer is then presented the model as an implementation guideline. Whereas the initial abstract specification is analogous to noninterference, the model is analogous to the role of *simple security* and the **-property*. The move from an abstract specification to an implementation-oriented model is motivated by the fact that an abstract specification of our exact requirements provides too little implementation guidance for the programmer.

A more serious problem with *simple security* and the **-property* is that not all explications of *security* agree exactly on what they mean. With respect to *simple security*, should a user whose clearance is *confidential* be able to copy the contents of a file classified *secret* to another file classified *secret*? That is, should we regard such an operation as a case of the *confidential* user viewing a *secret* file? Explications of *simple security* differ in their answer to this question depending on the how fine-grained we assume is our knowledge of process behavior, that is, whether we can differentiate between reading for the sake of viewing and reading for the sake of copying. If we regard any *read* access as a viewing of a file, then the major explications agree on the correct formulation of *simple security*.[1] [7] Rather than trying to distinguish between different ways of reading a file, we adopt this view.

It may seem that we could side-step the issue by treating the viewing of a file as copying a file to a screen or printer. *Simple security* then becomes a special case of the **–property* if we mandate that the classification of an output device for the purposes of an operation is the minimum of some basic classification of that device and the clearance of the user in control of the device at the time of the operation.[2] A problem with this approach is that if a program can read a file, it can communicate the file's contents to a user without having to write to an output device or another file. For example, suppose *F* contains our opening move as white in an upcoming chess tournament, and our opponent knows that we always play either p-k4 or p-q4. Consider the following program:

```
CHESS:
open F for read
while F="p-k4" do end
close F
```

A user can obtain our opening move by starting *CHESS* and passively waiting for the system prompt. Our move is "p-q4" if and only if *CHESS* successfully terminates. Hence, to prevent invalid information flow, we must prevent all accesses above one's security level. The **–property*, by itself, is insufficient.

Even if we could eliminate the need for *simple security*, there is the problem that explications of the **-property* differ [2, 6]. For example, if we assume that processes have memory, we may not wish to allow a process that has read a secret file to downgrade itself and write to a confidential file without first being "sanitized". Alternatively, we may choose to permit a process to read a secret file and write a confidential file even simultaneously, but prohibit information flow from the former to the latter. As in the case of *simple security*, the difference between the two approaches reflects different

1. This equivalence must be taken with a grain of salt. The important issue is whether a system that conforms to one formulation of *simple security* conforms to another. To answer this question, one needs to consider not just the two explications, but the mappings from the system to the explications as well.

2. Such an approach is taken in the Military Message System Model (MMS) [10].

assumptions about how fine-grained our knowledge of process behavior is.

For example, assume that we have two secret files *S1* and *S2*, two confidential files *C1* and *C2*, and a program *P*. If *P* reads *S1*, we may, for ease of implementation, simply prevent *P* from writing to *C1*. However, if we examine *P* more closely, we may discover that we are being overly restrictive. For example, *P* may be of the following program:

```
DONOTHING:
open S1 for read
open C1 for write
close S1,C1
```

or more realistically, the program:

```
SIMCOPY:
open S1,C1 for read
open S2,C2 for write
copy S1 to S2
copy C1 to C2
close S1,S2,C1,C2
```

The trouble with allowing such programs is that it is hard to draw the line between them and programs that allow harmful information flow from high-level files to low-level files. Even if we can examine code, it is not always clear whether a program passes information from one file to another. The same statement can leak information or not, depending on the environment. For example, the command **if A=0 then B:=0** passes information from *A* and *B* only if neither *A* nor *B* has been previously set to 0 in the program by an unconditional assignment. In general, the problem of detecting all and only nonsecure information flows is undecidable [4].

A solution to this problem is to take a coarse-grained approach to the **–property* and prevent nonsecure information flow by limiting program access to files. A reference monitor is a software module that enforces such limitations. Since access restrictions are necessary in any case, as demonstrated by our earlier program, *CHESS*, it makes sense to use this approach to enforce information flow as well. BLP is the epitome of a model that embodies this approach.

3. The Bell and LaPadula Model

In this section we develop a formal model for *simple security* and the **–property* that is identical to BLP in all essentials as far as MAC is concerned, but simpler. We assume the existence of a finite set of subjects S, consisting of system users and perhaps, programs; a finite set of objects O, consisting of system files (possibly including programs); the standard lattice of security levels L; and a finite set of access modes A = *{read, write}*, the set of modes in which an element of S can have access to an element of O.

Subjects who have *read* access to an object, can read it and copy it; those who have *write* access can modify it. To edit an object requires both types of accesses.

A system is a state machine such that each state $v=(b,f)$ is an element of $V=(B \times F)$, where

B is $P(S \times O \times A)$, the set of all possible current access sets such that a subject s has access a to an object o if and only if $(s,o,a) \in b$, and

F is $L^S \times L^O$, the set of all possible ordered pairs of functions (f_s,f_o) such that f_s gives the security level (clearance) associated with each subject, and f_o gives the security level (classification) associated with each object.

The set of requests (to change an element of a system state) is denoted by R. The state transition function, $T: R \times V \to V$ moves the system from one state to another: a request r is issued in state v that moves the system from v to its successor. A system $\Sigma(V,T,v_{init})$ is the finite state machine consisting of states V, transition function T, and initial state v_{init}. A state v is *reachable* in a system $\Sigma(V,T,v_{init})$ if and only if there is a sequence $<(r_0,v_0), \cdots ,(r_n,v_n)>$ such that $v_0 = v_{init}$, $v_n = v$, and for all $0 \le i < n$, $T(r_i,v_i) = v_{i+1}$. Note that for any system, v_{init} is trivially reachable.

A state is *simple secure* if and only if for every subject that has *read* access to an object in the state, the clearance of the subject dominates (in the lattice) the classification of the object. More formally, (b,f) is *simple secure* if and only if $(s,x,read) \in b \to f_s(s) \ge f_o(x)$. A state is $*-secure$ if and only if no subject has *read* access to an object x and *write* access to an object y in the state unless the classification of y dominates the classification of x. More formally, *(b,f)* is **-secure* if and only if $((s,x,read) \in b \wedge (s,y,write) \in b) \to f_o(y) \ge f_o(x)$. A system is secure if and only if all of its reachable states are *simple secure* and $*-secure$.

To apply BLP to a real system, we must appropriately map the model's primitive access types, *read* and *write*, to the system. Ideally, what counts as an appropriate mapping is one where a file cannot affect a program's behavior unless the program has *read* access to it and a file cannot be affected by a program unless the program has *write* access to it. Determining these mappings is not trivial. Program behavior can be affected in subtle ways, as for example, in the program *CHESS* described above. If we took all such effects into account, BLP would be too stringent for most applications. What is done in practice is to classify a set of operations as *read*'s and *write*'s and then use covert channel analysis to assess the possible damage that may result from the classification. Hence, there is always a give and take in mapping a model to a system and performing the accompanying covert channel analysis. The more information flow we can rule out categorically by our mapping of the model to the system, the less flexible our system is but the less information flow we have to examine for potential damage during covert channel analysis.

Further, we must appropriately map the model's subjects and objects to the system so that information is not improperly stored in program variables or system variables, e. g., the instruction counter. For example, assume that $F1$ contains our chess opening as described above and consider the following two programs:

```
PASS1(F1: file, F2: file)
open F1 for read
read A from F1
close F1
open F2 for write
write A to F2
close F2

PASS2(F1: file, F2: file)
open F1 for read
if F1= "p-k4"
    then close F1
        open F2 for write
        write "p-k4" to F2
    else close F1
        open F2 for write
        write "p-q4" to F2
close F2
```

Both *PASS* 1 and *PASS* 2 obviously pass information from *F* 1 to *F* 2 even though neither violates the $*-property$ as explicated in our version of BLP.[3] The most natural way to prevent such flow violations is to restrict file opening to the beginning of programs, restrict file closing to the end of programs, and disallow global and static variables.

Given these assumptions, it is straightforward to show that if a file can affect a program only if the program has *read* access to it and a program can affect a file only if the program has *write* access to it, then a system with only $*-secure$ states is a special case of a system that does not allow invalid information flow. One way to see this is to move to a neutral framework that does not have machine states. We can do this by considering sequences of operations, called *traces* [1, 12]. We represent the sequence consisting of the operation x followed by the operation y by $x.y$, and we use Q, R, S, and T as variables ranging over such (possibly empty) sequences. All variables are assumed to be universally quantified unless preceded by an occurrence of the existential quantifier, $\exists$. For simplicity, we assume that security levels do not change, using the statement $\mathbf{f1} \geq \mathbf{f2}$ to assert that the classification of file $f1$ dominates that of $f2$. If we ignore the complications that arise from permitting the closing of files and represent the operation of a user s opening a file f for access ϕ as **open(s,f,ϕ)**, the $*-secure$ state formulation of the $*-property$ is captured by two simple axioms that specify when a trace satisfies the predicate $*-s$, which intuitively says that the trace satisfies the $*-property$:

3. Bell and LaPadula's explication of the **-property* is more complicated, but no more satisfactory. They include in their formulation a current security level for each subject above which the subject cannot read and below which it cannot write. However, there is no restriction on changing this level in their model, so the flow violation given above is not blocked. Even in the Multics interpretation that accompanies their model, the only restriction on changing this function is that it does not violate the **-property*, which neither version of *PASS* does. For example, *PASS1* can lower its current security level after reading *F1* and before reading *F2*.

(1) *-s(T.open(s,f1,r).R.open(s,f2,w).S) → f2≥f1

(2) *-s(T.open(s,f2,w).R.open(s,f1,r).S) → f2≥f1

To capture the information flow version of $*\!-\!security$, we assume that any program that passes information from a file $f1$ to another file $f2$ on behalf of user s is represented by the operation **cp(s,f1,f2)**. Further, we assume that we can execute **cp(s,f1,f2)** only if $f1$ has been opened for read and $f2$ for write by s. Again, ignoring the closing of files, this requirement is specified using the predicate L, which denotes the set of legal traces:

(3) L(T.cp(s,f1,f2)) →
(∃R)(∃S)(∃Q)(T=R.open(s,f1,r).S.open(s,f2,w).Q ∨
T=R.open(s,f2,w).S.open(s,f1,r).Q)

The requirement that information not flow down is then captured by the assertion:

(4) *-s(T.cp(s,f1,f2)) → f2≥f1

It is easy to see that (1), (2), and (3) together imply (4) for all legal system operations, *i. e.*, that the $*\!-\!secure$ state version of $*\!-\!security$ implies the information flow version.

4. General Security Models: Paradigm Lost

To examine BLP in more detail, let us call security as defined by BLP $BLP\!-\!security$, to distinguish it from *access security*, the informal security requirement BLP purports formally to specify. The most notable theorem known about $BLP\!-\!security$ is called the "Basic Security Theorem" (BST), which gives necessary and sufficient conditions for a system starting in a secure state to never reach a nonsecure state.

Theorem: A system $\Sigma(V,T,v_{init})$ is $BLP\!-\!secure$ if and only if v_{init} is a $BLP\!-\!secure$ state and T satisfies (1) and (2) below:

(1) If $T(r,v)=v^*$ where $v=(b,f)$ and $v^*=(b^*,f^*)$, then for each $(s,o,read)\in b^*-b$, $f^*_s(s)\geq f^*_o(o)$, and for each $(s,o,read)\in b$ such that $f^*_s(s)\not\geq f^*_o(o)$, $(s,o,x)\notin b^*$.

(2) If $T(r,v)=v^*$ where $v=(b,f)$ and $v^*=(b^*,f^*)$, then for each $\{(s,x,read),(s,y,write)\}\subseteq b^*-b$, $f^*_o(y)\geq f^*_o(x)$; and for each $\{(s,x,read),(s,y,write)\}\subseteq b$ such that $f^*_o(y)\not\geq f^*_o(x)$, $\{(s,x,read),(s,y,write)\}\not\subseteq b^*$.

The theorem states that a system with a secure initial state is secure if and only if whenever the system moves from one state to a new state no new accesses are added that would be nonsecure with respect to the new state's security level functions and no accesses from the original state are retained that would be nonsecure with respect to the new state's security level functions. Bell and LaPadula seem simply to have intended for this theorem to establish an interesting property of $BLP\!-\!security$. We shall disregard the question of whether the results of the BST are interesting and consider a stronger claim: the accepted response of the computer security community to the BST was that it shows that $BLP\!-\!security$ completely explicates *access security*.[4] [13] In other words,

4. In [2] there is also a set of operations, called "rules", that satisfy further properties besides *sim-*

the BST was interpreted as establishing that *access security* $=$ *BLP−security*.

The justification for this opinion was never articulated. However, the position can be made plausible by considering an analogy [14]. In the 1930's logicians were struggling to explicate the informal concept, *computability*. The first explication to be offered was Church's suggestion of *recursiveness*, followed soon thereafter by Turing's suggestion of *Turing computability*. What gives credence to both suggestions is that *recursiveness* and *Turing computability* are coextensive. Hence, we have two different explications of an informal concept that pick out the same class of functions.

Returning to *access security*, what BLP offers us is an explication in terms of the notion of a restricted (*BLP−secure*) state. What would add credence to this explication is a coextensive explication in terms of the notion of a restricted state transition function. On the face of it, this is what the BST provides. It seems to offer a set of transition restrictions adequate for explicating *access security* and shows that the class of state machines that satisfy these restrictions is the same class of machines that satisfy the conditions of being *BLP−secure*. If this view were correct, the BST would certainly succeed in adding credence to the claim that *BLP−security* $=$ *security*, though the view that the BST *proves* that *BLP−security* $=$ *security* would be overstated.

Unfortunately, even this moderate view of the BST is untenable. The confidence it gives us that *BLP−security* $=$ *security* fades when we realize that an analogous theorem holds for obviously incorrect explications of security, e. g., one in which subjects are permitted to read only information classified higher than their security level [13]. The BST is transparent with respect to the concept of *secure state* used in our explication and would hold no matter what restrictions (or lack of restrictions) we placed on such states.

The failure of the BST to justify BLP does not demonstrate that no justification is forthcoming. We may try to construct our own justification by constructing a definition of security that truly is based on the notion of a *secure transition* rather than a *secure state* [14]. To this end, call a transition function T *simple secure* if and only if whenever $T(r,v)=v^*$ then the following three conditions are met:

(1) if $(s,o,read) \in b^* - b$, then $f_s(s) \geq f_o(o)$, and $f = f^*$;

(2) if $f_s(s) \neq f^*_s(s)$ then (i) b does not contain any triples of the form $(s,o,read)$ where $f^*_s(s) \ngeq f_o(o)$, and (ii) $f_o = f^*_o$ and $b = b^*$; and

(3) if $f_o(o) \neq f^*_o(o)$ then (i) b does not contain any triples of the form $(s,o,read)$ where $f_s(s) \ngeq f^*_o(o)$, and (ii) $f_s = f^*_s$ and $b = b^*$.

A transition function T is *$*$−secure* if and only if whenever $T(r,v)=v^*$ then the following two conditions are met:

(1) if $\{(s,x,read),(s,y,write)\} \subseteq b^*$ and $\{(s,x,read),(s,y,write)\} \not\subseteq b$, then $f_o(y) \geq f_o(x)$ and $f = f^*$;

---

ple security and the **-property*. However, these rules are Multics-dependent and are presented merely as one possible implementation of the model. If there are further constraints besides *simple security* and the **-property* that all such rule sets must enforce, these should be made explicit and placed on the same level as *simple security* and the **-property*. In general, sets of rules are hard to analyze for flaws since they are complicated by system-dependent information. We need a simple abstraction with which we can validate various rule sets [14].

(2) if $f_o(y) \neq f^*_o(y)$ then (i) b does not contain any subsets of the form $\{(s,x,read),(s,y,write)\}$ where $f^*_o(y) \not\geq f^*_o(x)$ or of the form $\{(s,y,read),(s,x,write)\}$ where $f^*_o(x) \not\geq f^*_o(y)$, and (ii) $b^*=b$.

A transition function is *secure* if and only if it is *simple secure* and $*-secure$. Our definition states that a transition function is secure if and only if whenever it changes an element of a state, the change cannot violate security with respect to the other elements of the state and these other elements cannot be changed during the transition. This differs from the conditions of the *BST* by not allowing a single transition to change, e. g., both a state's current accesse set and security level functions. Since a secure transition from a state v to v^* allows only one element of v to change and that that element can be changed only in ways that preserve state security with respect to v, it is straightforward to prove the following theorem [14]:

Theorem: A system is $BLP-secure$ if its initial state is $BLP-secure$ and its transition function is secure.

Unfortunately, the converse is false. A system may be $BLP-secure$ yet not have a secure transition function. As an example, consider the system Z whose initial state is $BLP-secure$ and which has only one type of transition [14]:

> When a subject s requests any type of access to an object o, every subject and object in the system is downgraded to the lowest possible level and the access is recorded in the current access set b.

It is easy to see that system Z's transition always leads to a $BLP-secure$ state, and hence, that system Z is $BLP-secure$. However, Z's transition function does not meet our definition of a secure transition. Since, Z permits anyone to access anything, *BLP* obviously fails to capture *access security*. Further, it should be clear that no adequate definition of *access security* can be based entirely on the notion of a secure state. We can define *simple security* and the $*-property$ in terms of a secure state since object contents are not part of our model and so we do not have to restrict how they can change from state to state.[5] However, since security levels are part of our model, we do need to say how they can change from state to state.

5. General Security Models: Paradigm Regained

In this section we develop a set of frameworks for MAC models that allow changes in security levels. A single model is insufficient since policies on changing security levels are intrinsically application-specific: the adequacy of the trade-off between flexibility and security inherent in such a policy can be judged only in light of the policy's intended application. To bring order to the resulting proliferation of models, we need theoretical frameworks that allow us to compare models and construct new models from old ones.

A framework is determined by S, O, L, and A, and a model within a framework is determined by an ordered pair $C=(c_s,c_o)$ such that c_s is a total function from S to $P(S)$ and c_o is a total function from O to $P(S)$. Intuitively, $c_s(x)$ and $c_o(y)$ are the set of subjects who can change the security levels of subject x and object y, respectively [15].

---

5. Though as we have seen, we need to include transition restrictions when determining when a program requires *read* and *write* access to an object.

Our notion of a system is the same as above except that our transition function is now a function $T\colon S \times R \times V \rightarrow V$ where a subject s issues a request r in state v that moves the system from state v to a new state. A transition function is *transition secure* if and only if each transition $T(s,r,v) = v^*$, where $v = (b,f)$ and $v^* = (b^*, f^*)$, is such that (1) for all $x \in S$ if $f^*_s(x) \neq f_s(x)$, then $s \in c_s(x)$, and (2) for all $y \in O$ if $f^*_o(y) \neq f_o(y)$, then $s \in c_o(y)$. In other words, a transition can change the security level of a subject x or object y only if the subject executing the transition is in $c_s(x)$ or $c_o(y)$, respectively.

A system is *secure* only if (1) all its reachable states are *simple secure* and $*-$*secure*, and (2) its transition function is *transition secure*. All models in a framework share this security policy, though the consequences of the policy differ from model to model depending on the particular model's function pair C. A *model* is the set of secure systems that share S, O, L, A, and C. A *framework* is the set of models that share S, O, L, and A.

Note that our security policy gives necessary, but not sufficient, conditions for a system to be *secure*. Hence, different models within a framework do not contradict one another. This would not be the case if our policy followed BLP in giving sufficient conditions for *security* as well. It allows us to construct new models from old ones.[6] In fact it is possible to define model construction operations $\cap$, $\cup$, and $'$ that form a Boolean Algebra [15]. This gives us a natural partial ordering of the set of models within a framework, *viz.*, $M1 \leq M2$ if and only if $M1 \cap M2 = M1$. For example, if we consider a framework where $S = \{x, y\}$ and $O = \{z\}$, we have the following ordering among models that do not allow the clearance of a subject to change:

$$\{x,y\}$$

$$\{x\} \qquad\qquad \{y\}$$

$$\{\}$$

The top policy says that either x or y can change z's classification; the leftmost policy on the second row says that only x can change z's classification; the rightmost policy of that row is the analogous policy for y; and the bottom policy says that z's classification cannot be changed. Although the distinction between the two policies on the second row can be useful when we are considering specific individuals, there are times we may wish to ignore differences between policies that can be eliminated by renaming and treat all policies that appear on a row as equivalent. This can be done formally by slightly modifying the definition of $\leq$ [15].

A given security model corresponds to a model in a framework if and only if the set of systems that is nonsecure by the criteria of the former is the set of systems that is nonsecure by the criteria of the latter. Given this sense of correspondence and sets S, O, L, and A, BLP corresponds to that element of the framework for which $c_s(x) = c_o(y) = S$ for all $x \in S$ and $y \in O$. It is the least stringent of the framework's models. The most

6. The fact that models do not contradict one another is important. Consider a model *M1* that says a system is secure if and only if *C1* and a model *M2* that says a system is secure if and only if *C2*. Unless *C1* and *C2* are equivalent the model that says a system is secure if and only if *C1* $\wedge$ *C2* has no useful logical relation to *M1* and *M2*. All three models contradict each other.

stringent is BLP^{+}, BLP supplemented by tranquility, the restriction that security levels cannot change[7] and corresponds to the framework model for which $c_S(x) = c_O(y) = \emptyset$ for all $x \in S$ and $y \in O$. An intermediate position is one where a subject $sso \in S$ acts as system security officer and is in charge of all changes to security levels.[8] Such a system corresponds to the framework model in which for all $x \in S$ and $y \in O$, $c_S(x) = c_O(y) = \{sso\}$. It is easy to see that $\mathrm{BLP}^{+} \leq \mathrm{MMS} \leq \mathrm{BLP}$. In fact, BLP and BLP^{+} are the top and bottom elements, respectively, of the Boolean Algebra formed by a framework. Hence, as we move toward the top of the algebra, the models become less stringent, and in general, $M1 \leq M2$ if any system that satisfies $M1$ satisfies $M2$ as well.

Since this way of modeling the ability to change security levels places all the theory of Boolean Algebra at our disposal for comparing and constructing models, we should be reluctant to forsake it. Nevertheless, it has the limitation that it does not provide us the ability to formulate n-person rules on our system. Such rules are widespread in government and industry. For example, we may require that the classification of an object can be changed only with the approval of both the owner of the object and the system security officer. This requirement can be implemented in several ways. One possibility is to require that the classification of an object may be changed only if the owner of the object and the system security officer concurrently execute a special program that changes the classification. Another is to require that the classification of an object may be changed only if the owner of the object executes a program that requests a change and the system security officer then executes a second program that grants the request. On a more abstract level, however, it is worthwhile being able to ignore implementation differences and focus solely on the concept of an action being executed by several people jointly, whether concurrently or sequentially. The abstract formulation provides a criterion of correctness for the lower level rules.

To this end we consider frameworks whose subject set has a particular structure [15]. We replace our set of subjects, S, by $P(S)-\{\emptyset\}$, the set of nonempty subsets of S, which we denote by $\mathbf{S}$ and whose elements we denote by $\mathbf{s}$.[9] In our definition of a system, we replace B by $\mathbf{B} = P(\mathbf{S} \times O \times A)$, and f_S by $\mathbf{f_s}$, a function from $\mathbf{S}$ to L such that $\mathbf{f_s}(\mathbf{s})$ is the greatest lower bound (in L) of $\{f_S(\{s\})\text{: } s \in \mathbf{s}\}$. For example, if $x \in S$ and $y \in S$ with security levels of *secret* and *top secret*, respectively, then $\{x\} \in \mathbf{S}$ with security level *secret*, $\{y\} \in \mathbf{S}$ with security level *top secret*, and $\{x,y\} \in \mathbf{S}$ with security level *secret*. If $\{(\{x\}, o_1, write), (\{x,y\}, o_2, write)\} \subseteq \mathbf{b}$, then x has *write* access to o_1, and x and y have joint *write* access to o_2. The latter access signifies that x and y can change o_2, but only if they do so by operating together.

Given these changes to the framework, our definition of *simple security* remains unchanged except for being in terms of our new subject set $\mathbf{S}$, i. e., a state is *simple secure* if and only if $(\mathbf{s}, x, read) \in \mathbf{b} \rightarrow \mathbf{f_s}(\mathbf{s}) \geq f_O(x)$. Our definition of the *$*$–property*, however, requires more substantial modification. We now say that a state is *$*$–secure* if and only if for any subjects $\mathbf{s}_1, \mathbf{s}_2$ and objects x, y if $(\mathbf{s}_1, x, read) \in \mathbf{b}$ and

7. See, for example, the SeaView model [5].
8. See, for example, the MMS model [10].
9. What follows can be applied to the set of objects in an analogous manner if we wish to capture the notion of being able to operate on an object only conjointly with operations on another object, as for example, in double-entry bookkeeping.

$(\mathbf{s}_2, y, write) \in \mathbf{b}$ and $f_o(y) \not\geq f_o(x)$, then $\mathbf{s}_1 \cap \mathbf{s}_2 = \emptyset$. In other words, we must guard against a subject violating the $*$*–property* by virtue not only of its singleton accesses, but also its joint accesses.

To formulate transition restrictions we redefine our notion of a state transition and replace $T: S \times R \times V \rightarrow V$ with the function $\mathbf{T}: \mathbf{S} \times R \times V \rightarrow V$, and we replace c_s and c_o by the functions $\mathbf{c_s}$ and $\mathbf{c_o}$, respectively, where each new function's range is $P(\mathbf{S})$ instead of $P(S)$. Since the security level of the subject $\{x, y\}$ is determined by the security levels of $\{x\}$ and $\{y\}$, we need concern ourselves only with how the security levels of singleton subjects can change. Hence, the domain of $\mathbf{c_s}$ becomes $\{\{x\}: \{x\} \in \mathbf{S}\}$ instead of S. Our definition of *transition security* is as before except modified to reflect these changes, i. e., a transition function is *transition secure* if and only if each transition $\mathbf{T}(\mathbf{s}, r, v) = v^*$, where $v = (b, f)$ and $v^* = (b^*, f^*)$, is such that (1) for all $\{x\} \in \mathbf{S}$ if $\mathbf{f}^*_\mathbf{s}(\{x\}) \neq \mathbf{f}_\mathbf{s}(\{x\})$, then $\mathbf{s} \in \mathbf{c_s}(\{x\})$, and (2) for all $y \in O$ if $f^*_o(y) \neq f_o(y)$, then $\mathbf{s} \in \mathbf{c_o}(y)$.

Each model in the sense of the previous section yields a unique model in the current sense since the security levels of all joint subjects is determined by the levels of the individual member subjects. It is easy to verify that if an instance of our new framework allows only singleton subjects, then the restrictions it places on a subject $\{s\}$ is identical to the restrictions placed on s by our previous framework. Further, the same restrictions that apply to a subject s in the previous framework now apply to any subject that contains s in the current framework. Our current model is more selective, however, in the sense that, for example, a subject may be denied joint *write* access to an object o which it could have single *write* access to because the subject it wishes to share the access with may have *read* access to an object of higher classification.

Though our treatment of joint access is straightforward and has the advantage that the models of a framework form a Boolean Algebra, it has the drawback that it is not clear that all the instances of a framework make sense. For example, consider a system that contains two users x and y. The set of subjects in our new model will be $\{\{x, y\}, \{x\}, \{y\}\}$. Hence, it is possible to have $\mathbf{c_o}(w) = \{\{x\}\}$ for some object w. Some may regard as nonsensical a policy where x can change w's security level, but is unable to change it operating in conjunction with y. If we wish to rule out such systems as bizarre, we can either treat such models as merely theoretical elements necessary to round out our framework or we can exclude them by appropriately restricting the range of $\mathbf{C}$ so that, e. g., $\{x\} \in \mathbf{c_o}(w)$ only if $\{x, y\} \in \mathbf{c_o}(w)$. The advantage of so restricting $\mathbf{C}$ is that we have no policies that are nonsensical. The disadvantage is that the domain of the components of $\mathbf{C}$ no longer form a Boolean Algebra. However, it is straightforward to prove that the possible domains for each component do form a distributive lattice under the set theoretic operations $\cap$, and $\cup$ [15].

6. Discretionary Security

Though we have so far formally considered solely the notion of MAC, DAC can also be handled in our framework and is useful for models that capture the notion of limiting access for reasons of privacy and "need to know". Since DAC is a useful addition to government security and serves as one pillar of industrial security, it is worthwhile examining it briefly here.

In BLP, discretionary security is captured by a discretionary access matrix which, for any subject-object pair, lists the type of accesses that the subject may have to the object. This approach has the disadvantage of being extremely coarse grained, especially in the industrial world, where one does not want to give a subject blanket rights to alter an object in any way, but rather only the right to alter an object in a set of specified ways [3]. In such a setting discretionary access is the right to execute certain programs on a specified object.[10] Ignoring the order of program arguments, it can be viewed as a function $D : S \times O_1 \times O_2 \rightarrow Boolean$ where $D(s, o_1, o_2) = true$ if and only if s has the right to execute program o_1 on o_2.

The advantage of the present framework for such access restrictions is that it allows us to capture the requirement that a user can access an object via a program, but only in conjunction with other users. Such restrictions form the second pillar of security for industrial purposes [3]. Hence, one may wish to say, at an abstract level, that the action of reimbursement for an order can only be performed by a receiving agent and an accountant operating jointly to make sure that nothing is payed for that has not been delivered. The details of modeling such policies are beyond the scope of this paper, but it is plain to see that the resulting framework will support the comparison and construction of DAC policies in a way similar to our comparison and construction of MAC policies.

7. Conclusion

We have seen that BLP has limitations stemming from its state-based-only approach to security. It is interesting to note that the response of the computer security community to these limitations displays differing views of BLP's role in computer security.[11] Those who formulated BLP seem to view it primarily as a research tool developed to explore the properties of one possible explication of *security*. However, those who evaluate systems for adequate MAC enforcement seem to view BLP as correctly capturing our informal concept of security. Since the limitations described in this paper are problems only from the latter point of view, the reaction to the community resembled Thomas Kuhn's descriptions of paradigm shifts in science where two communities fail to understand each other because of differing assumptions [9].

No matter which view is correct, there can be not doubt that our framework for traditional MAC security models and our framework for security models that contain n-person rules form a more useful setting for performing research in computer security and for evaluating the security of systems. Both frameworks support the rigorous comparison of models and the systematic creation of new models from existing ones.

The next step is to extend the framework to cover input/output models of security and models that permit the components of C to change as well. For example, who grants or revokes permission for a user to change an object's security level? In the mean time, both frameworks should support future research in security modeling by suggesting new

10. This is the approach taken, for example, in the MMS [10] and SeaView [5].

11. A representative sample of this response is contained in *Computer Security Forum 5*, 18 (July 5, 1986), ed. Ted Lee for Arpanet distribution. System *Z* was originally presented in issue 14 (June 22, 1986) of the *Forum*, and additional responses appeared in issues 25 (September 23, 1986), 26 (October 5, 1986), 27-29 (all October 16, 1986), and 30-31 (all December 9, 1986).

models to be considered and the relations between these models and existing ones. For example, if we discover a model M is too lax for our purposes, we can conclude that any model $M*$ such that $M* \geq M$ will also be too lax. Similarly if M is too restrictive, no model $M* \leq M$ need be considered. By creating and experimenting with such models, we will create a tool box of models that can serve a variety of purposes.

Acknowledgments

I am grateful to Thor Bestul, Dick Kemmerer, Carl Landwehr, Teresa Lunt, Catherine Meadows, and Jeannette Wing for their comments on various sections of this paper.

References

1. W. Bartussek and D. L. Parnas, ‘‘Using Traces To Write Abstract Specifications For Software Modules,’’ Report TR 77-012, University of North Carolina, Chapel Hill, N. C., December 1977.
2. D. E. Bell and L. J. LaPadula, ‘‘Secure Computer System: Unified Exposition and Multics Interpretation,’’ MTR-2997, MITRE Corp., Bedford, MA, March, 1976. Available as NTIS AD A023 588.
3. D. D. Clark and D. R. Wilson, ‘‘A Comparison of Commercial and Military Security Policies,’’ in *Proc. 1987 IEEE Symposium on Security and Privacy*, pp. 184-194, IEEE Computer Society Press, April, 1987.
4. D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, 1982.
5. D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. Shockley, ‘‘The SeaView Formal Security Policy Model,’’ SRI Interim Report A003, SRI International, 1987.
6. J. A. Goguen and J. Meseguer, ‘‘Security Policies and Security Models,’’ in *Proc. 1982 IEEE Symposium on Security and Privacy*, pp. 11-20, IEEE Computer Society Press, April, 1982.
7. J. T. Haigh, ‘‘A Comparison of Formal Security Models,’’ in *Proc. 7th National Computer Security Conference*, pp. 88-119, Gaithersburg, MD., Sept. 1984.
8. C. Heitmeyer and J. McLean, ‘‘Abstract Requirements: A New Approach and Its Application,’’ *IEEE Transactions on Software Engineering*, vol. SE-9, no. 5, pp. 580-589, September 1983.
9. T. Kuhn, *The Structure of Scientific Revolutions*, University of Chicago Press, Chicago, 1970.
10. C. Landwehr, C. Heitmeyer, and J. McLean, ‘‘A Security Model for Military Message Systems,’’ *ACM Transactions on Computer Systems*, vol. 2, no. 3, pp. 198-222, August 1984.
11. D. McCullough, ‘‘Noninterference and the Composability of Security Properties,’’ in *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, April 1988.
12. J. McLean, ‘‘A Formal Method for the Abstract Specification of Software,’’ *J. ACM*, vol. 31, no. 3, pp. 600-627, July 1984.

13. J. McLean, ‘‘A Comment on the ’Basic Security Theorem’ of Bell and LaPadula,’’ *Information Processing Letters*, vol. 20, no. 2, pp. 67-70, February 1985.
14. J. McLean, ‘‘Reasoning about Security Models,’’ in *Proc. 1987 IEEE Symposium on Security and Privacy*, pp. 123-131, IEEE Computer Society Press, April 1987. Also in *Advances in Computer System Security*, vol. III, ed. R. Turn, Artech House, Dedham, MA, 1988.
15. J. McLean, ‘‘The Algebra of Security,’’ in *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, April 1988.