RC 21717 (97406) 15 March 2000
Computer Science/Mathematics

IBM Research Report

A New Mandatory Security Policy Combining Secrecy and Integrity

Paul A. Karger, Vernon R. Austel, and David C. Toll
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

A New Mandatory Security Policy Combining Secrecy and Integrity

Paul A. Karger, Vernon R. Austel, and David C. Toll
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Submitted to the 6th European Symposium on Research in Computer Security (ESORICS 2000) to be held 4-6 Oct 2000, Toulouse, France

15 March 2000

ABSTRACT

This paper describes a new mandatory security model that better combines secrecy and commercial data integrity requirements than previous models based on Bell and LaPadula and Biba. The new model solves many of the previously perceived drawbacks and limitations of the Biba model, and makes use of a modified-Biba lattice to incorporate into the model the operation of so-called *trusted processes* that in the past have always been viewed as annoying exceptions to the existing mandatory security models. It then applies this new model as a better approach to security of mobile code.

Keywords: Mandatory access control, Lattice security models, Biba Integrity model, Bell and LaPadula model, mobile code security.

1 Introduction

Existing mandatory security models such as the Bell and LaPadula model [11] and the Biba model [13] have a number of drawbacks when put into practical use. Actual implementations of both models have required the use of *trusted processes* to meet a variety of administrative and downgrading requirements, yet trusted processes typically have been allowed to violate the requirements of the models – not an appealing procedure mathematically. The Biba integrity model has received little practical use, due to difficulties in actually assigning levels of integrity and because the model does not reflect actual practice in many cases.

Existing mandatory security models have also not directly addressed the needs for mobile code, such as Java or ActiveX. While there have been informal proposals to apply mandatory access controls to downloading code from the World Wide Web [19], there have not been any real models or implementations.

This new mandatory security model has been developed as part of a larger project to design and implement a high-assurance smart card operating system [9]. However, the model is in no way limited to only smart cards, and smart cards should not be viewed as central to this paper. Some of the examples of use of the model are based on smart cards, but only because it was convenient to use examples developed for our smart card operating system project.

The model has been designed for use in systems ranging from smart cards up to large servers and supercomputers. The model is particularly designed to address the needs of downloaded code, distinguishing between downloaded or mobile code that cannot be trusted from that which can be trusted. However, unlike the existing Java 2.0 and ActiveX trust models that simply rely on the reputation of the software developer, the new model relies on independent third-party evaluation

of downloaded code, based on either the ITSEC [8] or the Common Criteria [7] security evaluation schemes.

2 Models of Security

A formal model of security is essential when reasoning about the security of a system. Without an unambiguous definition of what security means, it is impossible to say whether a system is secure. Security models can be broken down into three major categories, listed in order of complexity:

- Models that protect against unauthorized disclosure of information,
- Models that protect against unauthorized tampering or sabotage, and
- Models that protect against denial of service.

Protection against disclosure of information has been understood the longest and has the simplest models. Protection against tampering or sabotage has been less well understood and appropriate models are only now under development. Protection against denial of service is not well understood today, although the recent rash of distributed denial of service attacks on the Internet is focusing much more attention than before. This paper does not deal with denial of service.

2.1 Preventing Information Disclosure

The first requirement of most security systems is preventing unauthorized disclosure of information. Indeed, the basic point of Lampson's access matrix [21] and the various access-control-list and capability-based systems that have been implemented is to control who may have access to which objects, both to prevent information disclosure and to prevent unauthorized tampering or sabotage. This section examines two classes of mechanisms: discretionary access controls and mandatory access controls.

2.1.1 Discretionary Access Controls

Discretionary access controls are the commonly available security controls based on the fully general Lampson access matrix. They are called discretionary, because the access rights to an object may be determined at the discretion of the owner or controller of the object. Both access-control-list and capability systems are examples of discretionary access controls. The presence of Trojan horses in the system can cause great difficulties with discretionary controls. The Trojan horse could surreptitiously change the access rights on an object or could make a copy of protected information and give that copy to some unauthorized user. All forms of discretionary controls are vulnerable to this type of Trojan-horse attack. Even if the Trojan horse couldn't directly leak the information, it could change the access control rules of an object to effectively cause a leak to occur. For example, a Trojan horse in an access-control-list system could surreptitiously change the ACL of an object. A Trojan horse in a capability system could make a copy of a capability for a protected object and then store that capability in some other object to which a penetrator would have read access. In both cases, the information is disclosed to an unauthorized recipient.

Harrison, Ruzzo, and Ullman [17] have shown that the *safety* property is undecidable for fully general, discretionary access controls, such as either a general access-control-list or capability system. Their argument is based on modeling the state transitions of the access matrix as the state transitions of a Turing machine. They show that solving the safety problem is equivalent to solving the Turing-machine halting problem.

2.1.2 Mandatory Access Controls

Mandatory access controls have been developed to deal with the Trojan horse problems of discretionary access controls. The distinguishing feature of mandatory access controls is that the system manager or security officer may constrain the owner of an object in determining who may have access rights to that object. Mandatory access controls were developed to solve what Lampson has called the *confinement* problem [20] to control the leaking of information by Trojan horses.[1]

Lipner [23] and Denning [16] have shown that for *lattice security models*, unlike for fully general access matrices, it is possible to solve the confinement problem. All mandatory controls, to date, have been based on lattice security models.

2.1.2.1 Elements of the Lattice Model

A lattice security model consists of a set of *access classes* that form a partial ordering. Any two access classes may be less than, greater than, equal to, or not ordered with respect to one another. Two access classes that are not ordered are called *incomparable*. Furthermore, there exists a lowest access class, called *system low*, such that system low is less than or equal to all other access classes. There also exists a highest access class, called *system high*, such that all other access classes are less than or equal to system high.

A very simple lattice might consist of two access classes: LOW and HIGH. LOW is less than HIGH. LOW is system low, and HIGH is system high. A slightly more complex example might be a list of secrecy levels, such as UNCLASSIFIED, CONFIDENTIAL, SECRET, and TOP SECRET. Each level in the list represents data of increasing secrecy.

There is no requirement for strict hierarchical relationships between access classes. The U.S. military services use a set of access classes that have two parts: a secrecy level and a set of categories. Categories represent compartments of information for which an individual must be specially cleared. To gain access to information in a category, an individual must be cleared, not only for the secrecy level of the information, but also for the specific category. For example, if there were a category NUCLEAR, and some information classified SECRET-NUCLEAR, then an individual with a TOP SECRET clearance would not be allowed to see that information, unless the individual were specifically authorized for the NUCLEAR category.

Information can belong to more than one category, and category comparison is done using subsets. Thus, in the military lattice model, for access class A to be less than or equal to access class B, the secrecy level of A must be less than or equal to the secrecy level of B, and the category set of A must be an improper subset[2] of the category set of B. Since two category sets may be incomparable, the complete set of access classes has only a partial ordering. There is a lowest access class, {UNCLASSIFIED-no categories}, and a highest access class, {TOP SECRET-all categories}. The access classes made up of levels and category sets form a lattice.

Many other security policies also form lattices. One could define other lattices that model commercial security requirements. Section 2.2.1 describes one such commercial security lattice.

---

[1] Lampson's confinement problem [20] and Harrison, Ruzzo, and Ullman's safety property [17], although related, are NOT the same property. A full treatment of these issues is beyond the scope of this paper. The interested reader should consult Denning [15, chapters 4 and 5] for a more complete treatment.

[2] Proper subsets must contain fewer elements than the set of which they are a subset. Improper subsets may contain all of the elements as the set of which they are a subset. Thus, every set is its own improper subset, and all proper subsets of a set are also improper subsets of that set, but not all improper subsets are proper subsets of a set.

2.1.2.2 Defeating Trojan Horses

Lattice models were first developed at the MITRE Corporation by Bell and LaPadula [11] and at Case Western Reserve University by Walter [33] to formalize the military security model and to develop techniques for dealing with Trojan horses that attempt to leak information.[3] At the time, no one knew how to deal with Trojan horses at all, and it came as quite a surprise that two quite simple properties could prevent a Trojan horse from compromising sensitive information.

First, the *simple security property* says that if a subject wishes to gain read access to an object, the access class of the object must be less than or equal to the access of the subject. This is just a formalization of military-security-clearance procedures that one may not read a document unless one is properly cleared.

Second, the *confinement property or *-property* [4] requires that if a subject wishes to gain write access to an object, the access class of the subject must be less than or equal to the access class of the object. The net effect of enforcing the confinement property is that any Trojan horse that attempts to steal information from a particular access class cannot store that information anywhere except in objects that are classified at an access class at least as high as the source of the information. Thus, the Trojan horse could tamper with the information, but it could not disclose the information to any unauthorized individual. A more detailed discussion of the confinement property and its interpretation in the context of a practical time-sharing system can be found in [11]. A survey on formal security models in general can be found in [22].

2.2 Preventing Tampering and Sabotage

The military's emphasis on the lattice security models, as typified by the requirements of the National Computer Security Center's evaluation criteria [3], has often been criticized as neglecting the issues of information tampering. However, the history of the development of the lattice security model shows that the military services have always been concerned with both unauthorized release and tampering. When the original work on the lattice security model was done at the MITRE Corporation [11] and at Case Western Reserve University [33] no one knew how to make formal statements about security policy. Indeed, protection against Trojan horses was considered an unsolvable problem at the time, and security researchers were all quite surprised when the *-property made it possible to formalize Trojan horse protection. The later emphasis on protecting against unauthorized release was because no one knew how to protect against tampering but protection against unauthorized release was understood.

2.2.1 Biba Integrity Model

Biba [13] later developed a model of mandatory integrity that is a mathematical dual of the Bell and LaPadula mandatory-security model. Biba defines a set of *integrity access classes* that are analogous to security access classes and defines *simple-integrity* and *integrity-confinement properties* that are analogous to the simple-security and confinement properties. The difference between integrity and security is that the direction of the less-than signs are all reversed, so that a program of high integrity is prevented from reading or executing low integrity objects that could

[3] The lattice models were based on earlier work on the ADEPT-50 operating system [34] and on the security enhancements to the WWMCCS-GCOS operating system [25] (pp. 147—148). Neither of those systems could solve the Trojan-horse problem, although ADEPT-50 contained a partial solution.

[4] The confinement property was called the **-property* in [12]. It was so named as a place holder until a better name could be found. No better name was found prior to publication, so *-property was used, and much of the literature on non-discretionary controls continues to use the name *-property (pronounced *star property*). In 1977, Jerry Saltzer [29] urged that a more meaningful name be found. Thus, some of the literature has since used the term confinement property.

be the source of tampering or sabotage. The principal difficulty with the Biba integrity model is that it does not model any practical system. Unlike the security models that developed from existing military security systems, the Biba integrity model developed from a mathematical analysis of the security models. However, Biba did not suggest how to actually decide which programs were deserving of a high integrity access class and which were not. This has made practical application of the Biba model very difficult.

2.2.2 Lipner Commercial Integrity Model

Lipner developed a commercial integrity model [24] that uses both the mandatory security and mandatory integrity models to represent a software development environment in a bank. It tied the integrity modeling much closer to reality than the Biba model did, but it was still quite complex. To our knowledge, no effort has been made to actually implement the Lipner commercial integrity model.

2.2.3 Clark and Wilson Commercial Integrity Model

A more recent development in preventing tampering and sabotage is the Clark and Wilson commercial integrity model [14]. They have proposed a model of data integrity that they assert more accurately describes the needs of a commercial application than the Bell and LaPadula lattice security model [11]. Clark and Wilson's model focuses on two notions: *well-formed transactions* and *separation of duties*. Separation of duties is commonly used in commercial organizations to protect against fraud.[5] Clark and Wilson contrasted their work with Lipner's commercial security interpretation of the lattice security and integrity models [24] and concluded that Lipner's commercial model does not adequately deal with limiting data manipulation to specific programs to implement the well-formed transactions.

The Clark and Wilson model consists of a set of certification and enforcement rules to be applied to a computer system. These rules apply to the operations of *transformation procedures* (TPs) that actually carry out the data manipulation in their system. Clark and Wilson clearly identified that the TPs needed to be certified to be valid. Unfortunately, they did not suggest any way to decide which TPs were valid and which were not. This is the same as the problem in the Biba model.

3 Using Mandatory Policies on a Smart Card

This section discusses how mandatory security policies could be used in a smart card system. Smart card examples are used, because the model was developed as part of a smart card operating system project. However, the examples could equally well apply to other types of limited-memory systems, such as personal digital assistants, cellular phones, etc. They could also apply in larger systems, such as desktops or servers.

3.1 Mandatory Secrecy Policy

We need a mandatory secrecy policy to ensure that information does not leak from one application to another, even in the presence of downloaded Trojan horse code. A Bell and LaPadula lattice model [11] will handle this nicely. The principal requirement will be categories,

[5] Separation of duties is also a familiar concept to the military. Launch of nuclear weapons is done under a concept of *two-person control* in which no one individual can ever launch a nuclear weapon. Two separate individuals must turn separate keys simultaneously. The keys are placed such that it is physically impossible for one person to perform the necessary actions.

while secrecy levels will take a subsidiary role. Presumably, each major application source will need its own category. Thus, an airline loyalty card might assign one category to the airline and one category to each rental car partner. Information would be constrained from flowing from one rental car partner to another rental car partner. Data to flow from a rental car company to the airline would be marked with both the categories of the rental car company and of the airline. A downgrading process would ultimately remove the rental car marking, before applying the frequent flyer miles to the customer's account. The downgrading process would only be allowed to downgrade from a single partner's marking to the airline's marking. It would not be allowed to apply the marking of some other partner company. Control of downgrading will be discussed in section 3.2 below on integrity policies.

The number of categories required will be relatively large for any given smart card. For example, a typical airline program might have 6-8 partner airlines, 3-4 partner rental-car companies, and 10-15 partner hotel chains. This means that typical applications will need at least 20 categories, with large applications needing 50-100 categories. On the other hand, the number of category combinations that will actually be used is relatively small. Given N categories, we could safely allocate storage for roughly 2*N category combinations, rather than the 2^N possible category combinations, assuming that each category is usually combined with only one other category.

To summarize the standard Bell and LaPadula security rules:

Read permission:	Secrecy access class of the process >= Secrecy access class of the data.
Write permission:	Secrecy access class of the process <= Secrecy access class of the data (no downgrading permitted.)
Execute permission:	Same as read permission.

3.2 Mandatory Integrity Policy

A commercial system, however, cannot be limited to only protecting the secrecy of information. Assuring that information is not tampered with is often much more important in a commercial setting. Whether a smart card is used as a cash card or as a loyalty card, ensuring that the correct amount of money or loyalty points are transferred may be much more important than keeping secret how much money or how many loyalty points were transferred.

A simple Biba [13] hierarchy of integrity levels of the form from high to low:

- Operating system high integrity code,
- Card issuer high integrity code,
- Partner company's high integrity code,
- Normal applications code.

Data files could be marked with an integrity level and a process' integrity level would have to be greater than or equal to the integrity level of the data for write permission, and the process' integrity would have to be less than or equal to the integrity level of the data for read permission. This way, we could assure the card issuer that applications added later could never modify the issuer's data.

However, the simple Biba integrity model suffers from a serious problem – how do you actually decide which programs are worthy of a higher integrity level? Since card issuers will be particularly worried about the security of applications on their cards (since they might be held liable in a court), we need to improve on the Biba model. The Biba model also prevents high integrity applications from reading low-integrity data, in fear that the application might be compromised in some form. This, however, makes it difficult to describe applications that have

been designed with high integrity to specifically process low integrity data input and to rule on its appropriateness. Therefore, we need some changes to the model.

Just as for the ActiveX and Java policies, developers digitally sign their applications. However, we go beyond this. If an application has been independently reviewed and digitally signed by the certifying body, then we can grant it a higher level of integrity. For example, we could define integrity levels for ITSEC-evaluated [8] applications. The Commercially Licensed Evaluation Facility (CLEF) would evaluate the application and the certifying body would digitally sign the application and its ITSEC E-level. A card issuer (such as a bank) might lay a requirement on vendors who want to download applications onto their cards. Your application must have received at ITSEC evaluation of some level to be acceptable. The NSA and NIST are developing a US equivalent of ITSEC evaluation CLEFs. That program is called the National Information Assurance Partnership (NIAP) and is based on the Common Criteria [5-7]. NIAP is designed to replace an existing NSA program for commercial evaluations called the Trust Technology Assessment Program (TTAP). Similarly, European evaluation agencies are also converting gradually to the Common Criteria.

The approach we have defined for assigning ITSEC E-levels as integrity levels does not address integrity categories. Biba defined integrity categories, and Lipner proposed use of them in his commercial data integrity model [24]. Interestingly, the Shirley and Schell program integrity model [32] also does not use integrity categories. However, we have not yet identified a use for integrity categories in this new model. We continue to include them for mathematical completeness and because someone may develop a use for integrity categories in the future, but all examples in this paper will only have integrity levels.

There could be provisions for less formal evaluations than full ITSEC. For example, a commercial security laboratory could check an application for obvious security holes (buffer overflows and the like) and for Trojan horses or trapdoors. While not as formal as an ITSEC evaluation, it might be sufficient for loyalty applications to gain access to communications files that are used to send data between applications (as discussed above).

There is one problem with using more than one kind of evaluation criteria. If an application has been evaluated under one criteria, and another application has been evaluated under a very different criteria, then if a user wishes to download both of those application onto the same card, it is not clear how to compare the integrity classes. If the two criteria have defined mappings (such as the E levels of the ITSEC and the EAL levels of the Common Criteria [5-7]), then there is not a problem. The Canadian criteria [1] and the US Orange book [3] can probably also be mapped. However, if the card issuer chose to use some very different and incompatible criteria, then downloading of other applications that were ITSEC evaluated might be difficult.

Essentially, this is a security policy to allow a card issuer to set some minimum quality standards on downloaded applications and to grant more or less access, based on the independent security reviews of applications.

3.2.1 Integrity Policy Details

Processes could be marked with an integrity level, based on the signatures of the independent reviewer of the code. That is pure Biba integrity. However, we will instead mark a process with TWO separate integrity access classes – one for read permission and one for write and execute permission. The write/execute integrity access class will be the level determined by the independent evaluator. If the read integrity access class equals the write/execute class, then we have the standard Biba model. If the read integrity access class is lower than the write/execute integrity access class, then we have a process permitted to sanitize and upgrade input that is initially marked at a low integrity access class. Not just any program will be trusted for

sanitization, but rather only programs explicitly evaluated for the purpose. Identifying the range of levels across which a program is allowed to sanitize will be specified as part of the digitally signed information from the CLEF. The operating system kernel will read that information when starting a program into execution to know what range of integrity classes to assign the process.

Separating the execute permission from the read permission originated in the program integrity model of Shirley and Schell [32]. The policy was further developed in the GEMSOS security model [30] that specified a range of levels within which integrity downgrading could occur.

However, our model allows a further generalization beyond that of GEMSOS. GEMSOS required that the trusted processes execute within a range of access classes. For integrity, this means that the write/execute integrity access class must be greater than or equal to the read integrity access class. Our model recognizes that allowing the read integrity access class to be greater than the write/execute integrity access class allows modeling firewall policies. In such a case, a particularly untrustworthy process executing at a low integrity level can be protected from possible attacks by only allowing it to read data that is at a higher integrity level. All data that the process writes must be marked at the lower integrity level.

The initial access rules will be as follows. These rules will change later in the paper, when we combine this new integrity model with the new secrecy model. Note that we separate execute permission into two subclasses – normal transfers and a special CHAIN operation. A normal transfer is the execution of a branch instruction or a subroutine call instruction. CHAIN is a way to start a separate process executing at some other integrity and secrecy access class. Due to the limited memory of a smart card, the process executing the CHAIN operation is immediately terminated. On a larger system, the process initiating the CHAIN in principle could continue executing. However, in either case, a CHAIN is a one-way operation. The GEMSOS security model did not include any counterpart of the CHAIN operation. The intended use of CHAIN is to start a guard or sanitization process or for a guard process to start a recipient of sanitized information. Note also that data files have only one integrity access class, just as in the original Biba model.

Read permission:	Integrity read access class (process) <= integrity access class (object)
Write permission:	Integrity write/execute access class (process) >= integrity access class (object)
Execute permission:	
Transfer:	Integrity write/execute access class (process) <= integrity access class (object) The target program runs at the integrity level of the caller. Note that a high integrity program cannot call or transfer to lower integrity code.
CHAIN:	No integrity access check is required. The target program runs in a new process at the integrity access class specified in the digitally signed certificate of the program.

3.3 Formalizing Secrecy Downgrading

This same strategy for allowing integrity downgrading can also be used for secrecy downgrading, as discussed in section 3.1 above. However, we would define a pair of secrecy access classes, rather than integrity levels, and the read and write rules would be reversed. There would be two secrecy access classes – one for read/execute and one for write. Note that for secrecy, we keep execute tied to read permission, because a process at a low secrecy level should not be permitted to execute high secrecy program code. This is not for anti-piracy purposes, but rather to maintain the secrecy of algorithms or constants in the code that must not be revealed, even by mere use of the program. Software piracy protection is outside the scope of this security model.

This policy for secrecy downgrading is identical to that used in GEMSOS [30].

The rules for secrecy downgrading are as follows:

Read/Execute permission: Secrecy read/execute access class (process) >= secrecy access class (object)
Write permission: Secrecy write access class (process) <= secrecy access class (object)

For a normal process, the read/execute secrecy access class will be identical to the write secrecy access class. However, if the process has a high enough integrity level, then the process may have a lower write secrecy access class. What integrity level is required for what range of read/execute versus write secrecy access classes is discussed below in section 3.5.

3.4 Combining the New Access Rules

We have now modified both the Bell and LaPadula model for secrecy and the Biba model for integrity and stated the access rules for each. However, when the two models are combined, the resulting access rules are a bit more complex and must be stated together. It is our expectation that with these modifications, the two models will always be used together, because the integrity access classes will be used to determine the secrecy downgrading rules, as specified in section 3.5.

Read permission

Secrecy read/execute access class (process) >= secrecy access class (object)

Integrity read access class (process) <= integrity access class (object)

Write permission

Secrecy write access class (process) <= secrecy access class (object)

Integrity write/execute access class (process) >= integrity access class (object)

Execute permission

Transfer

Secrecy read/execute access class (process) >= secrecy access class (object)

Integrity write/execute access class (process) <= integrity access class (object)

The target program of a transfer runs at the integrity level of the caller. A high integrity program cannot call or transfer to lower integrity code.

Chain

Secrecy read/execute access class (process) >= secrecy access class (object)

Secrecy write access class (process) <=

runtime read/execute secrecy class (new process)

Integrity write/execute access class (process) >=

integrity read access class (new process)

The first rule ensures that chain is possible only to files to which the caller has secrecy read permission. Integrity read permission is NOT required, because a high integrity process is always allowed to start a low integrity process. Contrast this with the rule that

a high integrity program is not allowed to transfer directly to a low integrity program in the same process.

The second rule ensures that the target process must have secrecy read permission to any passed arguments.

The third rule ensures that the target process is not contaminated by a low integrity argument.

The target program runs in a new process at the integrity access class specified in the digitally signed certificate of the program. This is a very important distinction. The process that issues the CHAIN operation does not determine the access class at which the target process executes. Rather, it is determined by the third party evaluator and certifying body who have digitally signed the code file. Section 3.5.3 shows how the certifying body can determine what access class to specify in the digitally-signed code file.

3.5 Using Integrity to Control Security Downgrading

As mentioned in section 3.2.1, we can use the modified integrity model to control security downgrading. To do this, we need a policy, similar to that specified in the NSA's Yellow Books [2, 10] to determine the level of integrity needed to control a particular range of downgrading. The Yellow Books establish US DoD policy on what level of security evaluation is required as a function of the risk range of the data. The methodology is to look at the maximum secrecy level of the data and the minimum trustworthiness level of users to determine a risk range over which the software must preserve security. The higher the risk range, the higher evaluation level that is required.

3.5.1 Summary of Yellow Book Recommendations

The Yellow Books derive their recommendations from the following sets of tables: Table 1assigns numeric levels to the level of clearance that a user may have. Note that the table distinguishes between totally uncleared people and people who do not have security clearances, but are authorized access to sensitive but unclassified information. It also distinguishes between two different kinds of background investigations at the same clearance level.

MINIMUM USER CLEARANCE	RATING
Uncleared (U)	0
Not Cleared but Authorized Sensitive Unclassified Information	1
Confidential (C)	2
Secret (S)	3
Top Secret (TS) with Background Investigation (BI)	4
Top Secret with Special Background Investigation (SBI)	5
One Category (1C)	6
Multiple Categories (MC)	7

Table 1. Rating Scale for Minimum User Clearance

Table 2 assigns numeric values to the secrecy of data. Note that it distinguishes not only the number of categories that may be processed, but also that the secrecy of data in one category may be different from another category.

MAXIMUM DATA SECRECY WITHOUT CATEGORIES	RATING	MAXIMUM DATA SECRECY WITH CATEGORIES	RATING
Unclassified (U)	0	Not Applicable	
Sensitive Unclassified (N)	1	N with one or more categories	2
Confidential (C)	2	C with one or more categories	3
Secret (S)	3	S with one or more categories with no more than one category containing Secret data	4
		S with two or more categories containing Secret data	5
Top Secret (TS)	5	TS with one or more categories with no more than one category containing Secret or Top Secret Data	6
		TS with two or more categories containing Secret or Top Secret Data	7

Table 2. Rating Scale for Maximum Data Secrecy

Table 3 establishes policy for what level of evaluation is required for what level of risk index. The entries in this table are subjective, in that they are based on an assessment of how strong each level of evaluation is against system compromise. Note that even the highest level of evaluation in the DoD Orange Book [3] is not trusted for the highest risk indexes.

RISK INDEX	Security Operating Mode	Minimum Evaluation Level
0	Dedicated	No Prescribed Minimum
0	System High	C2
1	Limited Access, Controlled	B1
2	Limited Access, Controlled	B2
3	Controlled, Multilevel	B3
4	Multilevel	A1
5	Multilevel	Beyond State of the Art
6	Multilevel	Beyond State of the Art
7	Multilevel	Beyond State of the Art

Table 3. Computer Security Requirements for Open Security Environments

Table 4 combines the information in Table 1, Table 2, and Table 3 to indicate the recommended evaluation level for each combination of minimum user clearance and maximum data secrecy.

		Maximum Data Secrecy						
		U	N	C	S	TS	1C	MC
Minimum Clearance of Users	U	C1	B1	B2	B3	*	*	*
	N	C1	C2	B2	B2	A1	*	*
	C	C1	C2	C2	B1	B3	A1	*
	S	C1	C2	C2	C2	B2	B3	A1
	TS(BI)	C1	C2	C2	C2	C2	B2	B3
	TS(EBI)	C1	C2	C2	C2	C2	B1	B2
	1C	C1	C2	C2	C2	C2	C2	B1
	MC	C1	C2	C2	C2	C2	C2	C2

Table 4. Security Index for Open Security Environments

3.5.2 Possible Commercial Version of Yellow Book Recommendations

To make use of these tables for commercial applications requires a different set of clearances and more attention paid to categories. We have experimented with policies based on whether the companies in question are direct competitors or not, and we have found it possible to construct tables similar to the Yellow Book. However, it is not clear whether to weigh competition more or less heavily than secrecy of data within a given company, so much additional research is needed to develop commercial counterparts to the NSA Yellow Books.

3.5.3 Applying the Yellow Book Recommendations

How would the Yellow Book recommendation be applied in practice? When two companies agreed on a policy for controlled downgrading, they would bring the guard program to an ITSEC or Common Criteria CLEF for evaluation. The CLEF would evaluate the program and suggest a recommended integrity access class. The result of the evaluation, together with the two companies agreed upon policy would be sent to the certifying body. After the certifying body approved the evaluation, it would not only issue a certificate on paper. It would also digitally sign the binary code of the evaluated program, together with a binary representation of the E level or EAL level assigned. In addition, it would also sign the four access classes for secrecy read/execute, secrecy write, integrity read, and integrity write/execute. When the program was then downloaded to a secure operating system, that operating system could determine the level of trustworthiness assigned to the guard program and what levels of both secrecy and integrity downgrading should be permitted.

3.6 More Complex Commercial Integrity Policies

It is clear that our new model that revises and combines both Bell and LaPadula and Biba can be relevant to the Clark and Wilson model. As we discussed in section 2.2.3, Clark and Wilson requires that their TPs be validated correct, yet they provided no mechanism for achieving this goal. Our new integrity model clearly provides this, since we can assign an integrity access class, based on the results of an ITSEC or Common Criteria evaluation of a TP, determining not only whether the TP is trustworthy, but assigning a scale of trustworthiness, based on an established international standard.

However, as our first implementation of the model is intended for a smart card system with extremely limited amounts of memory, we have not investigated this combination with the Clark and Wilson model to any depth, because the resulting system would clearly be too large for current smart card systems. More research is needed in this area.

4 Using the Security and Integrity Policies

In this section, we show an example of how the security and integrity policies might be used in an airline, hotel, and rental car loyalty application. The example is significantly different from Lipner's proposals for using non-discretionary controls for commercial applications [24], and we believe it is more easily understood and implemented. Assume we have an airline **A** with ties to hotel chains **H** and **M** and rental car chains **B** and **D**. Staying at the hotel chains earns airline loyalty points. Hotel H gives hotel loyalty points in addition to airline points, while hotel M gives hotel points or airline points, but not both. Hotel H loyalty points and Hotel M loyalty points are completely separate systems. Furthermore, the hotel chains consider the information about where and when the customer has stayed to be valuable marketing information, since the competing hotel chain could use this information to do target marketing. However, the customer and the airline would like all three loyalty systems to be managed from a single smart card, so that the customer need only carry one card, and that card is branded by the airline. Hotel chains H and M do not trust one another, but are both willing to cooperate with the airline A.

Based on these assumptions, the software that manages hotel H loyalty points must behave differently from the software that manages hotel M loyalty points. This is because the hotels have different policies on *double dipping* in which the customer earns points in both hotel and airline plans. Furthermore, the software for both hotel chains and for the airline may need periodic updating to reflect limited time special offers (e.g.: stay five times in one month and earn 500 bonus points), to reflect newly contracted partners, or other significant changes. Another type of special program that would depend on code on the smart card itself would be bonuses if you fly the airline, stay at hotel H, and rent a car from B, all on the same day. Tracking combined bonuses like that could be more easily done on the smart card itself, rather than requiring the central servers for the airline, the hotel, and the rental car companies to all communicate with one another.

We define the following secrecy and integrity levels and categories for the loyalty application:

Integrity Levels:	E6 > E5 > E4 > E3 > E2 > E1
Secrecy Levels:	System-Low
Secrecy Categories:	A, H, M, B, D

There will be data files storing loyalty information for each company. Each company will have a Bell and LaPadula secrecy category. Initially, the customer goes to the airport to fly on airline A. The airline's application with secrecy clearance A will run and grant some airline loyalty points. These are recorded in a file classified A. The airline must also make today's flight information available to all partners. It wants to indicate that the customer has flown today, but it might not want to give full flight details, due to either company confidentiality concerns and/or customer privacy concerns. Therefore, it writes into a different file that is classified system-low that the customer flew today, but with no further details. Now any partner application can read that information. (This assumes that there is only one airline on the card.)

Now the customer rents a car from company B. B's application code runs with secrecy clearance B and computes how many airline loyalty points to grant. It must communicate this information to the airline application, but it does not want the information to be known by rental car company D. Furthermore, B may not want A to know everything about its customer, either. Therefore, B's application writes the number of points earned into a communications file. It then upgrades the classification of that file to require both secrecy category A and secrecy category B. It can do this, because that is a write-up operation from secrecy level B to secrecy level A and B. Next, the B application code must start a B-downgrader process into operation that runs code at a higher integrity level. The B-downgrader will hold a secrecy clearance for both categories A and B. Its program code will be evaluated to a higher level as described in 3.5. The B-downgrader will inspect the communications file contents and ensure that the data being transferred to the airline does not compromise any of B's confidential information. It then downgrades the information by removing the B secrecy category and passes the information to the airline's software which runs in another process with only the A secrecy-category.

Figure 1 shows the procedure for communication between rental car company B and airline A through a guard or downgrader process. Ovals are processes, and boxes are data files. Arrows show the direction of information flow, and lightning bolts show Chain operations.

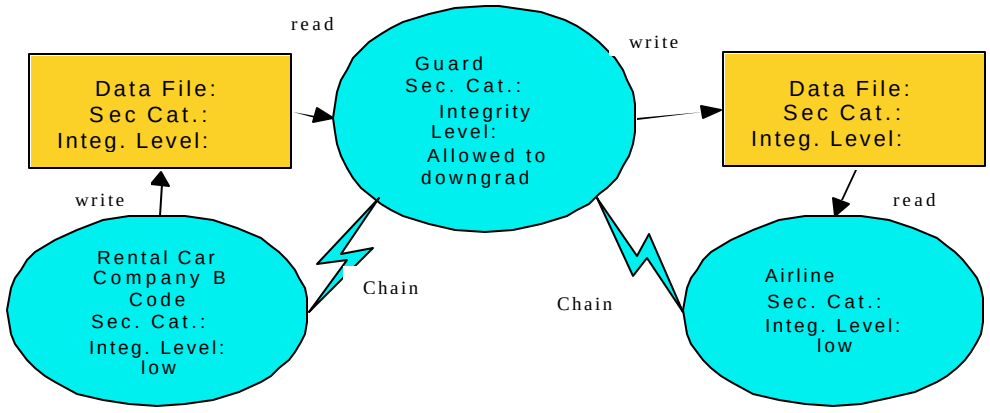

Figure 1. Information flow through a downgrading guard.

The net result is that only the program that actually inspects the information to be passed from B to A has to undergo a higher-level ITSEC evaluation. The bulk of the applications code that runs on behalf of B and on behalf of A can be either unevaluated or only evaluated to a lower level.

Now let us consider a more complex example. Assume that A, B, and H have created a special bonus program in which if your fly A, rent from B, and stay at H, all on the same day, then you get extra bonus points. To implement this, all three partners must share information, since the flight, car rental, and hotel stay could happen in any order.

Each partner, A, B, and H would need to record in an individual file that a rental, flight, or hotel stay had occurred that day. Each of these files would be marked with all three secrecy categories A, B, and H. Writing these files is a write-up operation and does not require special privileges. They would then each start up a high integrity application that was cleared to read A, B, and H category information. That application would check to see if all three transactions had occurred, and only if all three had occurred, then and only then would the high integrity application downgrade some information to category A only to award the bonus points. Note that most of the code for implementing the bonus operation can be written by each of the partners and only has access to that partners data. Only the downgrader needs access to data from all three partners, and it can be formally evaluated to the higher integrity level to ensure that it does not compromise data improperly.

5 Formal Verification of the Model

To be useful in the development of high assurance operating systems, this model needs to be written down in a formal language and verified to be correct and consistent. We have been working together with colleagues from the University of Ulm and the German Artificial Intelligence Center (DFKI) to do this formal verification. They have submitted a companion paper [31] to the 2000 ESORICS conference describing their formal verification of a smart card security model that is similar (although not exactly identical) to ours.

6 Conclusions

We have developed a new model for mandatory access controls that solves a number of problems in the existing Bell and LaPadula model for secrecy and the Biba model for integrity. We solve the Biba problem how to assign integrity levels by using the results of ITSEC or Common Criteria evaluations. We resolve the seeming contradiction that a high integrity program must be protected against contamination by low integrity data yet we need high integrity programs to

validate the quality of suspect data and we provide a model for trusted processes that have always been the bane of implementers of the Bell and LaPadula model by applying access rules from the GEMSOS model to constrain programs to downgrade or upgrade secrecy or integrity access classes. Note that we do not eliminate the need for trusted processes. Rather, we provide a formal mechanism for deciding which processes should be trusted and how much they should be trusted. Furthermore, we show how to write policy in the style of the NSA yellow books to decide what level of integrity is required of programs that wish to downgrade or upgrade specific secrecy or integrity levels. We also show how these models can be used in real commercial applications, such as smart card-based loyalty programs.

Will this new model actually prove successful in the commercial world? It is much too early to tell. None of the previous commercial data integrity models have succeeded in the commercial world, so it would be presumptuous to claim that this new model will succeed where previous ones have not. Even the GEMSOS model, which showed much promise and was applied in the design of the SeaView secure relational database system [27, 28], has not achieved real commercial success. While SeaView has been very influential on subsequent database security research, it also has not been commercially successful. The SeaView prototype was ultimately implemented [18, 26], not on the A1-evaluated GEMSOS [4], but rather on the B1-evaluated Trusted Oracle database management system running on a B1/CMW-evaluated Sun operating system.[6] Neither Trusted Oracle nor the Sun CMW offered the same types of integrity policies that were found in GEMSOS. The SeaView prototype was quite successful demonstrating the SeaView security policies for use within the database management itself, but the prototype could not show practical results for the underlying operating system security and integrity policy combinations that are relevant to this paper.

We will only know whether the new model proposed in this paper will be commercially successful until the model has been exposed both to security experts and to real commercial developers. This paper is a first step in that process.

7 Acknowledgements

We must acknowledge the referees of a previous version of this paper who pointed out the relationship between this model and the Sea View security and integrity models, which are based on the earlier GEMSOS models.

8 References

1. *The Canadian Trusted Computer Product Evaluation Criteria*, Version 3.0e, January 1993, Canadian System Security Centre, Communications Security Establishment, Government of Canada.
2. *Computer Security Requirements -- Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, CSC-STD-003-85, 25 June 1985, DoD Computer Security Center: Ft. George G. Meade, MD.
3. *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985, Department of Defense: Washington, DC.
4. *Final Evaluation Report for the Gemini Trusted Network Processor*, Report No. 34-94, 28 June 1995, National Computer Security Center: Ft. George G. Meade, MD.
5. *Information technology - Security techniques -- Evaluation criteria for IT security -- Part 1: Introduction and general model*, ISO/IEC 15408-1, 1999, International Standards Organization.

[6] This is not intended as a criticism of the SeaView work at all. Their intention was to use an implementation of Trusted Oracle on A1-evaluated GEMSOS, but this implementation was not completed in time.

6. *Information technology - Security techniques -- Evaluation criteria for IT security -- Part 2: Security functional requirements*, ISO/IEC 15408-2, 1999, International Standards Organization.
7. *Information technology - Security techniques -- Evaluation criteria for IT security -- Part 3: Security assurance requirements*, ISO/IEC 15408-3, 1999, International Standards Organization.
8. *Information Technology Security Evaluation Criteria (ITSEC)*, June 1991, Commission of the European Communities: Brussels, Belgium.
9. *Philips Semiconductors and IBM Research to co-develop secure smart cards: Highly secure operating system and processor, suitable for multiple applications*, 4 February 1999: http://www.semiconductors.philips.com/news/content/file_384.html.
10. *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements -- Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, CSC-STD-004-85, 25 June 1985, DoD Computer Security Center: Ft. George G. Meade, MD.
11. Bell, D.E. and L.J. LaPadula, *Computer Security Model: Unified Exposition and Multics Interpretation*, ESD-TR-75-306, June 1975, The MITRE Corporation, Bedford, MA: HQ Electronic Systems Division, Hanscom AFB, MA.
12. Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model*, ESD-TR-73-278, Vol. II, November 1973, The MITRE Corporation, Bedford, MA: HQ Electronic Systems Division, Hanscom AFB, MA.
13. Biba, K.J., *Integrity Considerations for Secure Computer Systems*, ESD-TR-76-732, April 1977, The MITRE Corporation, Bedford, MA: HQ Electronic Systems Division, Hanscom AFB, MA.
14. Clark, D.D. and D.R. Wilson. *A Comparison of Commercial and Military Computer Security Policies*. in **1987 IEEE Symposium on Security and Privacy**. 27-29 April 1987. Oakland, CA: IEEE Computer Society. p. 184-194.
15. Denning, D.E., *Cryptography and Data Security*. 1982, Reading, MA: Addison-Wesley.
16. Denning, D.E., *A lattice model of secure information flow*. **Communications of the ACM**, 1976. **19**(5): p. 236-243.
17. Harrison, M.A., W.L. Ruzzo, and J.D. Ullman, *Protection in Operating Systems*. **Communications of the ACM**, 1976. **19**(8): p. 461-471.
18. Hsieh, D., T.F. Lunt, and P.K. Boucher, *The SeaView Prototype*, A012: Final Report, Contract No. F30602-89-C-0158, 20 August 1993, SRI International, Menlo Park, CA: prepared for Rome Air Development Center, Griffiss AFB, NY.
19. Karger, P.A., *Untrusted Applications Need Trusted Operating Systems*, in *19th National Information Systems Security Conference* 22-25 October 1996, National Institute of Standards and Technology, National Computer Security Center: Baltimore, MD. p. 847-848.
20. Lampson, B.W., *A note on the confinement problem.* **Communications of the ACM**, 1973. **16**(10): p. 613-615.
21. Lampson, B.W., *Protection*. **Operating Systems Review**, 1974. **8**(1): p. 18-24. originally published in Proceedings of the Fifth Princeton Conference on Information Sciences and Systems, March 1971.
22. Landwehr, C.E., *Formal Models for Computer Security*. **Communications of the ACM**, 1981. **13**(3): p. 247-278.
23. Lipner, S.B., *A comment on the confinement problem.* **Operating Systems Review**, 1975. **9**(5): p. 192-196. Proceedings of the Fifth Symposium on Operating Systems Principles.
24. Lipner, S.B. *Non-Discretionary Controls for Commercial Applications.* in **Proceedings of the 1982 Symposium on Security and Privacy**. 26-28 April 1982. Oakland, CA: IEEE Computer Society. p. 2-10.
25. Lobel, J., *Foiling the System Breakers*. 1986, New York: McGraw-Hill Book Company.
26. Lunt, T.F. and P.K. Boucher. *The SeaView Prototype: Project Summary*. in **17th National Computer Security Conference**. 11-14 October 1994. Baltimore, MD: National Institute of Standard and Technology / National Computer Security Center. p. 88-102.
27. Lunt, T.F., D. Denning, R.R. Schell, M. Heckman, and W.R. Shockley, *Secure Distributed Data Views: Volume 2: The SeaView Formal Security Policy Model*, SRI-CSL-88-15, February 1989, SRI International: Menlo Park, CA.
28. Lunt, T.F., P.G. Neumann, D. Denning, R.R. Schell, M. Heckman, and W.R. Shockley, *Secure Distributed Data Views: Volume 1: Security Policy and Policy Interpretation for a Class A1*

Multilevel Secure Relational Database System, SRI-CSL-88-8, August 1988, SRI International: Menlo Park, CA.

29. Saltzer, J., *Personal communication on the name of the *-property*, 1977, Massachusetts Institute of Technology.
30. Schell, R.R., T.F. Tao, and M. Heckman. *Designing the GEMSOS Security Kernel for Security and Performance*. in **Proceedings of the 8th National Computer Security Conference**. 30 September - 3 October 1985. Gaithersburg, MD: DoD Computer Security Center and National Bureau of Standards. p. 108-119.
31. Schellhorn, G., W. Reif, and A. Schairer. *Verification of a formal security model for multiapplicative smartcards.* in **submitted to the 2000 ESORICS conference**. 15-17 May 2000. Oakland, CA: IEEE.
32. Shirley, L.J. and R.R. Schell. *Mechanism Sufficiency Validation by Assignment*. in **Proceedings of the 1981 Symposium on Security and Privacy**. 27-29 April 1981. Oakland, CA: IEEE Computer Society. p. 26-32.
33. Walter, K.G., W.F. Ogden, W.C. Rounds, F.T. Bradshaw, S.R. Ames, and D.G. Shumway, *Primitive Models for Computer Security*, ESD-TR-74-117, 23 January 1974, Case Western Reserve University, Cleveland, OH: HQ Electronic Systems Division, Hanscom AFB, MA.
34. Weissman, C. *Security Controls in the ADEPT-50 time sharing system.* in **1969 Fall Joint Computer Conference**1969: AFIPS Conference Proceedings, AFIPS Press, Montvale, NJ. p. 119-133.