NON-DISCRETIONARY CONTROLS FOR COMMERCIAL APPLICATIONS

Steven B. Lipner
Corporate Research Group

Digital Equipment Corporation
77 Reed Road
Hudson, Ma 01749

INTRODUCTION

The lattice model of non-discretionary access control in a secure computer system was developed in the early Seventies [BLaP]. The model was motivated by the controls used by the Defense Department and other "national security" agencies to regulate people's access to sensitive information. Since that time, the lattice model has enjoyed reasonable success in several computer systems used to process national security classified information [MME; Multics; SACDIN]. "Reasonable success", in this context, means that human beings accept the systems and are able to use them to accomplish useful work, without the protection provided by the non-discretionary controls unduly interfering with productivity or perceived convenience.

In the late Seventies and early Eighties, the Defense Department, through its DoD Computer Security Initiative and DoD Computer Security Evaluation Center, has attempted to raise general awareness of computer security issues and to urge manufacturers to improve the security of commercial operating system offerings. The DoD notion of improved security [Nibaldi] seems to constitute (1) better system integrity to assure that users and programs can only access information as authorized; and (2) the introduction of the lattice model as a fundamental operating system mechanism for access authorization.

There is little question as to the need for improved system integrity, though different application environments will require different levels of integrity. (The draft DoD evaluation criteria recognize this variation and specify graded levels of system integrity with suggested technical approaches to achieving them). The lattice model, however, is derived from national security policies and regulations, leading to some question as to its suitability in non-national security data processing environments. This paper examines some ways in which the lattice model non-discretionary controls might be used in commercial data processing.

The first section of this paper introduces the basic notion and derivation of the lattice model. It also introduces, for purposes of comparison, the results of a previous attempt to apply the lattice model in a commercial environment. The second section examines commercial requirements for information security and derives an alternate application of the lattice model more suited to these requirements. The final section introduces a second component of the lattice model (the integrity model) and revisits the commercial application to show an alternate formulation for achieving equivalent security.

The paper's major conclusions are that the lattice model may in fact be applicable to commercial data processing, but that such application will require ways of looking at security requirements different from those prevalent in the national security community. The joint involvement of people experienced in commercial appplications and of people who understand the lattice security model will probably provide the most productive course towards the development of successful commercial applications of the lattice model.

THE LATTICE MODEL

The DoD Security System

The lattice model was developed from an examination of the Defense Department's (DoD) information security policies and the ways in which they might be enforced in the context of a multiuser computer system. The DoD information security policy gives each document a classification level, L, and a (possibly empty) set of categories, C. The security levels are strictly ordered (i.e., form a chain). The categories tend to have no ordering or precedence, but subsets of categories are ordered by set inclusion. The combination of a classification level and a set of categories is referred to as an access class. Figure 1 identifies the DoD classification levels and their order, and a few example categories.

Top Secret > Secret > Confidential > Unclassified

(a) Classification Levels

Nuclear, NATO, Intelligence

(b) Example Categories

Figure 1. DoD Levels and Categories

CH1753-3/82/0000/0002$00.75 © 1982 IEEE

The classification levels and categories in the DoD system are also used to label a set of clearances that can be granted to people. The two fundamental security rules of the DoD system are then: (1) An individual can only have access to (read) a document if the individual's clearance level is greater than or equal to the document's classification level, and if the individual is "cleared" for all categories applied to the document; and (2) Only a specially authorized individual may reduce (downgrade) the level or remove categories from a classified document.

The rules identified above were applied more or less directly in the formulation of the lattice security model. Access rights were associated with processes that execute on behalf of users. Programs are treated as objects that are executed by processes--but have no rights themselves.

In a computer, as in the world of people and documents, a process may not read a file unless it (or the person on whose behalf it operates) is cleared for such access. This rule is called the "simple security condition" (Figure 2a). The second rule of the lattice model derives from the second rule of the DoD system--though its manifestation has often been perceived as overly restrictive: A process may not write a file (or other object) unless the access class of the object is greater than or equal to* that of the process.

This second rule does not reflect a restriction on a cleared individual's ability to write an unclassified document--indeed the purpose of a clearance is to certify that an individual is trusted not to compromise or downgrade sensitive information. Rather, the second rule reflects the reality that a process executing an arbitrary program is not trusted to write files with lower access classes than those it may have read and remembered in its process state. This conservative rule (Figure 2b) is designed to enforce the DoD restriction on downgrading the access class of a document and is called the *-property or confinement property.

A process P may read an object if

$$L_p \geq L_o \quad \& \quad \{C\}_p \supseteq \{C\}_o$$

(a) Simple Security Condition

A process P may write an object O if

$$L_p \leq L_o \quad \& \quad \{C\}_p \subseteq \{C\}_o$$

(b) Confinement Property

Figure 2. Lattice Model Security Rules

---

* - One access Class A is greater than or equal to another B if and only if the classification level for A is greater than or equal to that for B, and A's set of categories contains B's. More formally:

$$L_A \geq L_B \ \& \ \{C\}_A \supseteq \{C\}_B$$

Systems that apply the lattice model have applied it as a "non-discretionary" access control policy in the sense that neither users nor programs may reduce the access class of information (downgrade it) except by appealing to special operating system software that must be invoked directly by a human being (in some cases a system security officer) and not by an untrusted program. The *-property and the restriction on downgrading assure that no chain of program actions can result in an unauthorized flow of information to a file or other object of reduced access class [BlaP]. Discretionary control mechanisms like access control lists, in contrast, allow the construction of sequences of program actions that can result in observation or modification of information by an unauthorized person. For example, a program operating on behalf of an authorized user might, without the person's knowledge, alter an access control list or copy a file to a new version with a different less restrictive access control list. With a non-discretionary lattice mechanism, no program action can allow access by a user who does not have the access class (clearance) corresponding to that of the original file.

Historically, attempts to apply the lattice model in non-DoD environments have begun by identifying levels and categories that reflect the levels of sensitivity and organizational divisions in the subject environment. For example, Figure 3 presents the levels and categories identified in applying a prototype security-enhanced operating system in a hypothetical corporation. The definitions of the categories are more or less self-explanatory. The levels are simply intended to reflect a notion of increasing information sensitivity.

Security Levels

- Public
- Sensitive
- Confidential

Categories

- Manufacturing
- Personnel
- Engineering
- Accounting

Figure 3 - Security levels and categories for a hypothetical corporation

The application of levels and categories like those in Figure 3 has an unfortunate tendency to fall apart on close inspection. While the company may in fact have organizational divisions like those identified in the categories, there will almost never be a notion of individuals having clearance that corresponds to the security levels. Access to information will normally

be granted on a need-to-know basis, depending on individuals' job responsibilities or, worse (from a formal point of view), on a basis of aggregation in which a "little" information is unrestricted, while a "lot" is closely held. The former kind of restriction may be enforced by introducing still more categories; the latter seems to require that an access control system remember large amounts of history, and is thus very difficult to enforce.

There is a more fundamental problem with the lattice model application sketched in Figure 3: It deals only with the reading and compromise of information. In actual practice, commercial instititutions may be more interested in unauthorized modification of information than in its compromise. The remainder of this paper re-examines the commercial information security requirements, then proposes alternate applications of the lattice model to meet those requirements.

CONTROLLING MODIFICATION WITH THE LATTICE MODEL

Revisiting the Requirements

In an attempt to solve the problems identified above, the author initiated an informal search for commercial security policies and requirements. Much of the information gathered centered on who can alter information, rather than who can read it. Furthermore, a great deal of the concern expressed by EDP auditors and EDP security officers had to do with the integrity of financial and finance-related information and with control over the introduction of programs that will operate on that information. Much less concern was expressed about the activities of system users who will write or use programs to process "their own" data of whatever sensitivity.

To make the suggestions above more concrete, the following list of requirements is presented. It is taken from the suggestions of a senior EDP auditor, and is directed toward "production" systems that might operate on financial, material control, or order-entry data:

(1) Users of the application will use production programs and data bases; they will not write their own programs to operate on the production data bases.

(2) Application program developers will do their development and testing in a test environment, and have no access to the production (source or object) programs or data bases. If they need such access, they may be provided with copies of the information they need through a special process.

(3) "Promotion" of programs from development to production status is a controlled event.

(4) The installation system programmers' actions shall be controlled and audited.

(5) Management and audit functions shall have access to the system state and an audit trail of selected activities (both system actions and user interactions with the applications).

The requirements cited above bear little apparent resemblance to the usual statement of DoD security requirements. The following paragraphs will show, however, that the lattice model can be applied to do an effective job of meeting these requirements.

A Lattice Application to Control Modification

In forming a lattice model to meet the requirements identified above, the initial step was to define a set of access classes appropriate to achieving the desired restrictions. Figure 4 shows the levels and categories that will be used in this example. The "system-low" level (SL) corresponds to unclassified in the DoD system, and information at this level is readable by all processes. The application of the remaining level (audit-manager) and categories will be made clear below.

Security Levels

Audit-Manager (AM)

System-Low (SL)

Categories

Production-Data (PD)

Development (D)

System-Development (SD)

Tools (T)

Figure 4. Levels and Categories

The application of the access classes to the system users (and to processes running on their behalf) is outlined in Figure 5. Users (of application programs), application developers, and system programmers each have system low security level and two categories. In each case, the first category is that of the information the population must manipulate (production data in the case of users; application programs and test data in the case of application developers), and the second is that of the programs the population must use. The audit and management population has access to information of any category, and a security level of

audit-manager. Finally, a system control function is defined with system-low level and access to each category, and a "downgrade" privilege to change categories and transform (for example) developing programs into production code -- presumably after suitable testing and quality assurance review.

There is one unusual aspect of the assignment of access classes to user populations in Figure 5. Some populations must be required to operate only through processes having all of the categories or levels for which the users are "cleared". In particular, the "check" on system audit and management personnel derives from the fact that they may not login at system low with one or more categories and operate on data or programs as users or programmers, then use their auditing function to conceal changes that they may have made or actions that they may have taken. Furthermore, if a programmer could somehow introduce a "system low" program for use by production users, there would be nothing in this lattice model to prevent that program from corrupting the production data base to which the users have (lattice model) access. The next section introduces an additional model that helps address this problem. But, in general, users must be restricted as to which subsets of their full "clearance" they exercise.

User Community	Access Class
System Management or Audit	AM; any set of categories
User	SL; PD & PC
Application Developer	SL; D & T
System Programmer	SL; SD & T
System Control	SL; PD, PC, D, SD, T plus "downgrade" privilege

Figure 5. Users' Access Classes

The assignment of access classes to files (objects) is depicted in Figure 6. The program objects (production code,tools*) each have a single category and are intended to be read-only (unmodified). Objects subject to manipulation (production data; system and application programs under development) have two categories each--that of the object itself and that of the program that must operate on it--so that a process executing the program will be allowed by the confinement property to write the object. Audit trail information is developed with the category(ies) of the activity being audited and is "written up" to the higher audit-manager security level.

* Programmer support software--compilers, linkers, library managers,etc.

Files	Access Class
Production Data	SL; PD & PC
Production Code	SL; PC
Developing Code/Test Data	SL; D & T
Software Tools	SL; T
System Programs in Modification	SL; SD & T
System Programs	SL
System and Application Audit Trail	AM; & Appropriate Category(ies)

Figure 6. File Access Class Assignment

The overall effect of the configuration of access classes described above is shown in Figure 7. Their correspondence to the set of requirements stated above is quite close. There are limitations, however. First, the scheme introduces an apparently all-powerful system control function. In fact, the system control population has only the task of moving tested programs and data from category to category and thus could be limited to a specific set of programs at login (e.g., by the discretionary control mechanism). Furthermore, there is a provision for auditing system control actions. Nonetheless, this facet is worrisome.

A second limitation of the scheme deals with its treatment of system programmers. It is not clear that limiting system program development and modification to an application-like environment with someone else (system control) doing installations is either technically or culturally acceptable. In fact, a system programmer who decided to eliminate the *-property altogether for a set of processes of his choice (or to take some equally drastic covert action) could probably do so. Nevertheless, the scheme presented attempts to limit system programmers to their authorized domain, and this is a desired effect.

On the positive side, Figure 7 does reflect a configuration that meets the stated requirements. Programmers, users, and system programmers are each limited to their own sphere of activity. There is an audit trail for management, and management can observe but not change the state of the system. In summary, this application of the lattice model to a commercial environment seems successful if unconventional. The next section introduces an alternative mechanism for achieving the effects presented in Figure 7.

OBJECTS / SUBJECTS	Prod.Data	Prod.Code	Dev. App.Prgm	Dev. Sys.Prgm.	Tools	Sys Prg.	Audit Trail
System Mgt. & Audit	R	R	R	R	R	R	RW
Production Users	RW	R				R	W
Application Programmers			RW		R	R	W
System Programmers				RW	R	R	W
System Control	RW	RW	RW	RW	RW	RW	W

Figure 7. Effects of the Commercial Lattice

THE INTEGRITY MODEL AND THE COMMERCIAL LATTICE

The Integrity Model

In addition to the basic (security) lattice model described above, there is another model directed toward the control of modification (rather than disclosure) of information. This model is the "integrity" model [Biba], which may be considered the mathematical "dual" of the security lattice model. The integrity model seems well-suited to meeting commercial security requirements of the sort outlined above.

The integrity model, like the security lattice model, assigns levels and categories to information and to users. Its objective, however, is to prevent the contamination of high-integrity (highly reliable) information by the infusion of lower-integrity data and by processing with lower integrity programs. The analog of the security policy restriction on a process of high security access class writing information of a lower access class (the *-property) is a restriction on a process of high integrity access class reading (or executing) an object of lower integrity access class. This policy (Figure 8a) assures that computations performed by a process of high integrity access class maintain their high integrity, uncorrupted by low integrity information. Like the security *-property, it is a conservative policy, preventing all reading by a high-integrity process of low-integrity information, even though the information (or the process' use of it) may be harmless.

A process P may read an object O if

$$L_p \leq L_o \;\&\; \{C\}_p \subseteq \{C\}_o$$

a. Integrity *-Property

A process P may write an object O if

$$L_p \geq L_o \;\&\; \{C\}_p \supseteq \{C\}_o$$

b. Simple Integrity Condition

Figure 8. Integrity Lattice Policy.

The integrity analog of the simple security condition and its restriction on a process reading information of a higher access class is the "simple integrity condition". This condition prevents a process (that has read or executed information) of low integrity from writing information of high integrity. Unlike the integrity *-property which restricts a high-integrity process on the basis of conservatism (it may make a mistake or be subtly influenced by flawed data), the simple integrity condition restricts a low-integrity process from corrupting information of higher integrity—which it is simply not allowed to access. A formal statement of the simple integrity condition is given in Figure 8b.

While the integrity model has been in existence for several years, literature on its application is sparse. The discussion below integrates the integrity policy into a commercial processing application based on those described in the previous section.

A Commercial Application of the Integrity Model

The integrity model can be used in an attempt to simplify and render more intuitive a lattice policy application that meets the commercial security requirements described above. The formulation described below will meet an additional requirement:

(6) Special-purpose application software shall be provided to effect "data base repair" on the production data base. This software may be used by members of the application programmer or system control population under special circumstances.

A set of security and integrity levels and categories appropriate to this application is shown in Figure 9. The security levels and categories that remain in this example are substantially the same as those in the previous one. The "tools" category is eliminated and replaced by the functions of the integrity lattice model. The integrity levels are provided to insure against modification of system programs (System-Program) and stabilized production and development support software (Operational). Integrity categories are used to separate the development environment from that for production (Development, Production).

Integrity Levels: System-Program > Operational > System-Low

Integrity Categories: Production, Development

Security Levels: Audit Manager > System-Low

Security Categories: Production, Development, System-Development

Figure 9. Commercial Lattice Security and Integrity Access Classes.

Figures 10 and 11 show the assignment of security and integrity access classes to users and files (subjects and objects). Security access class assignments are somewhat simpler than those for the previous example; integrity access classes are used to isolate development and production environments and prevent undesired modification. The intermediate integrity level (operational) prevents computations that execute production code or software tools from modifying those objects--since no users are authorized to login at the "operational" or "system-program" integrity levels, files at these levels cannot be modified, except by system control installation. The audit and manager population operates at a low integrity access class, and can thus observe any information but modify none. The audit-manager security level prevents processes that create audit trails from observing those trails thereafter. As before, the system control population has the function of moving files among access classes.

Population	Integrity Level	Integrity Categories	Security Level	Security Categories
System Management or Audit	SL	-	AM	(ALL)
Production Users	SL	Production	SL	Production
Application Programmer	SL	Development	SL	Development
System Programmer	SL	Development	SL	System-Development
System Control*	System Program	Production, Development	SL	Production, Development,
Repair	SL	Production	SL	Production

* - Plus "downgrade" privilege

Figure 10. Users' Security and Integrity Access Classes

Object	Integrity Level	Integrity Categories	Security Level	Security Categories
Production Data	SL	Production	SL	Production
Production Code	Operational	Production	SL	Production
Developing Code/ Test Data	SL	Development	SL	Development
Software Tools	Operational	Development	SL	-
System Programs in Modification	SL	Development	SL	System Development
System Programs	Sys.Prog.	Production, Development	SL	-
System & App'n. Audit Trail	SL	-	AM	(Any)
Repair Programs	Production	Production	SL	Production

Figure 11. Files, Security and Integrity Access Classes.

OBJECTS

SUBJECTS \	Production Data	Production Code	Develop. Code & Test Data	Develop. Sys. Prog.	S/W Tools	Sys Prog.	Re-pair Code	Audit Data
System Mgr.	R	R	R	R	R	R	R	RW
Prod.User	RW	R				R		W
App'n.Prog.			RW		R	R		W
Sys.Progrm.				RW	R	R		W
Sys.Control	RW	RW	RW	RW	RW	RW	RW	W
Repair	RW	R				R	R	W

Figure 12. Effects of Commercial Lattice Model with Integrity

The effects of the new lattice model formulation are depicted in Figure 12. They are similar to those shown in Figure 7. The "repair" function introduced in response to an additional requirement appears identical to the other production programs--and the individuals who execute it have rights indistinguishable (at the non-discretionary control level) from those of production users. This result means that repair code must be protected by a discretionary control mechanism rather than the non-discretionary controls. The reason for this is that any attempt to introduce a "repair" security or integrity category either prevents the repair process from reading the data to be repaired (integrity category) or from rewriting it (security category). Perhaps the best approach is to use a repair security category, rewrite the data base with repair and production security categories, and then allow "system control" to remove the repair category from the newly-repaired data base. This form of "two-person" control (repair plus system control) may be appropriate to an exceptional operation like data base repair.

The idea of two-person control for the repair function may also apply in the case of an urgent repair where all rules go "out the window". If a critical application or data base is down, an organization may be willing to abandon its nominal security policy and tell the responsible system or application programmer to "fix it!" If the installation has a lattice security policy enforced by its operating system, the programmer may be given the highest security and integrity levels, every category, and downgrade privilege. In such a case, the remaining security rests on the personal integrity of the individual programmer. If an emergency repair is needed, it may better be made by a team of a system programmer, an application programmer and a system control "installer". Such a team can operate within the context of the lattice model controls, and is probably both a more secure and more effective (by virtue of an element of cross-checking) way of accomplishing emergency system repairs.

Another effect of the integrity formulation is to almost eliminate the restriction on users logging in at access classes less than their "full clearance". The exception is again in the area of "repair" processes. An application programmer with development integrity and security categories (the normal case) and production integrity and security categories issued to support a repair role could observe and modify production data bases in a way that would normally be unauthorized. Such a programmer could not modify the production or repair code because of the absence of the "operational" integrity level. Furthermore, it is not clear what tools such a programmer would have available to support his efforts since the development tools do not have production integrity category and no software they produce will either. Nonetheless, repair personnel should be required to login either in a repair role (production categories) or a development role (development categories). More generally, it appears that some limitation on a user's selection of login levels and categories is a desirable feature of systems that implement the lattice model.

The integrity lattice formulation, unlike that using only the security model, limits the possibility of introducing a "Trojan Horse" to modify a sensitive data base. A development programmer can write a "system-low" program to manipulate production data (though it will actually appear from the compiler with development categories), but only the system control "promotion" process can give it the integrity level to operate on a production data base. Compromise of production data to a development programmer is prevented by the *-property and the "production" category in both security and integrity lattices.

With the exceptions and caveats noted above, the integrity and security lattice formulations both meet the stated requirements. The integrity formulation offers somewhat better control, but it is not clear that users and security personnel will find the set of levels and categories required intuitively understandable.

REVIEW AND COMMENT

The early paragraphs of this article "threw out" the idea of partitioning a commercial system by levels and categories, based on the claim that there is no notion of clearance in most commercial organizations. Revisiting this assumption, it may be observed that some commercial organizations do have organizational security partitions corresponding to the roles of components (manufacturing vs. engineering) or special activities (a critical new product). These partitions may be reflected by security categories of the lattice model, and overlaid on the sort of lattice formulation described above. Thus a data base and production code might have security categories "Production" and "Project_XYZ_Engineering". The application programming function might likewise be divided into (Project_XYZ_Engineering, Development versus Project_ABC_Engineering, Development) and application programmers either isolated or (in a smaller organization) allowed to login with the security category corresponding to the function they were supporting.* This application of the security lattice model might even be extended back to the DoD environment and DoD security levels incorporated as well. It is probable that the reintroduction of security levels and categories is most compatible with a formulation that applies the integrity lattice to control writing, and thus reduces the proliferation of security categories.

Whether the formulation of choice is security lattice only or security lattice plus integrity lattice, it appears that the lattice model can help address real commercial computer security requirements. This result may be of interest both to individuals and organizations that have such requirements and to developers and advocates of systems that incorporate the lattice model.

---

* - In order to partition files, activities and audit trails, though not to isolate information from the programmers themselves.

ACKNOWLEDGEMENT

The ideas presented here were heavily influenced by initial discussions with Bill Hill, Bruce Parker, Ashby Woolf and Tom Bailey. Paul Karger, Ed Balkovich, Maurice Wilkes, and Joe Tardo reviewed the draft, and Pat Bright prepared the paper in final form. The effort reported was supported by the Digital Equipment Corporation's Corporate Research and Government Systems Groups.

REFERENCES

[Biba] Biba, K.J., "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1977, (AD A039324).

[BlaP] Bell, D.E., and LaPadula, L.J., "Secure Computer Systems", ESD-TR-73-278, Volume I-III, The MITRE Corporation, Bedford, MA, November 1972-June 1974.

[MME] Heitmeyer, Constance L.. and Wilson, Stanley H., "Military Message Systems: Current Status and Future Directions", IEEE Transactions on Communications, Vol. Com-28, No. 9, Sept. 1980, pp. 1645-1654.

[Multics] Whitmore,J.C., et al., "Design for Multics Security Enhancements", ESD-TR-74-176, Honeywell Information Systems, 1974, (AD A030801).

[Nibaldi] Nibaldi, G.H. "Proposed Technical Evaluation Criteria for Trusted Computer Systems", M79-225, The MITRE Corporation, Bedford, Ma, Oct. 25, 1979.

[Sacdin] Chandersekaran, C.S. and Shankar, K.S., "Towards Formally Specifying Communications Switches", Trends and Applications 1976: Computer Networks, IEEE, New York, N.Y., Nov. 17, 1976, 1976, pp. 104-112.