

Chapter 6

Access Control and Multilevel Security

The primary purpose of security mechanisms in a system is to control access to information. Until the early 1970s, it was not generally realized that two fundamentally different types of access controls exist. Discretionary access control is the most common: users, at their discretion, can specify to the system who can access their files. Under discretionary access controls, a user (or any of the user's programs or processes) can choose to share files with other users.

Under nondiscretionary or mandatory access control, users and files have fixed security attributes that are used by the system to determine whether a user can access a file. The mandatory security attributes are assigned administratively (such as by a person called the security administrator) or automatically by the operating system, according to strict rules. The attributes cannot be modified by users or their programs. If the system determines that a user's mandatory security attributes are inappropriate for access to a certain file, then nobody—not even the owner of the file—will be able to make the file accessible to that user.

6.1 ACCESS TO THE SYSTEM

Before we worry about access to information within the system, we should pause to consider control of access to the system itself. For some systems, physical controls are entirely adequate, but most systems need to be accessible from locations that are not under the physical control of the site administration.

A system can protect itself in two ways:

1. It can limit who can access the system.
2. It can limit what people can do once they access the system.

The first way requires the system to implement a two-step process of identification (asking you who you are) and authentication (asking you to prove it), as we discussed in section 3.3.2.

Until technology provides something better, the much-maligned password will continue to be the most common authentication technique. Despite their drawbacks, passwords, if properly used, are very effective for user authentication. Following are some time-honored principles for password management:

- Use passwords only for user authentication (see section 6.2.1), not for access control or system identification.
- Encrypt passwords stored in the system database in such a way that someone reading system dumps or the database cannot read the passwords. Using a one-way cipher (National Bureau

of Standards 1985) where, for example, the password is the key to its own encryption makes it impossible to decipher the database.

- Assign a given password to no more than one person.
- Minimize the number of times a password must be entered by the user (to limit its exposure).
- Do not store passwords in programs or files that could be revealed by someone reading the program.
- Minimize the number of different passwords a person has to remember.
- Discourage users from using the same password on different machines.
- Educate users who choose their own passwords about easy-to-guess passwords. Instead of allowing users to choose passwords, some systems (such as Honeywell's Multics and DEC's vms) provide an automated password generator that assigns random pronounceable words (Gasser 1975).
- Have users change passwords occasionally, but not so frequently that they need to write them down.
- Change a user's password the day that person leaves the organization. In a large organization with scores of machines of various sizes, this means keeping good enough records to be able to find all the systems on which the user has an account.

The National Bureau of Standards and the National Computer Security Center have published comprehensive guidelines for the creation and management of passwords (Department of Defense 1985; National Bureau of Standards 1985).

The second way for a system to protect itself is to make available a very limited and controlled set of functions for users whom it cannot identify. A transaction processing system, for example, might limit users to a specific set of menu options, with no opportunity for running arbitrary commands. While such limited service systems have their place, they should never be used in lieu of proper authentication. This cannot be stressed too strongly: the only appropriate use for a limited service system as a substitute for user authentication is where it is impractical to register users in advance, such as on a public terminal in an airport providing flight information and reservation services.

If you think you can avoid a lot of implementation effort and password management headaches by implementing a limited service interface for a given application, you are thinking dangerously. Limiting what a user can do on a general-purpose operating system is extraordinarily difficult. Try as you might to close the loopholes, there always seems to be a way for a clever user to break out of the limited system and obtain access to the operating system's underlying facilities. Even if you do succeed in containing the user, it may be nearly impossible to prevent malicious misuse of the limited system, except in the case of extremely limited systems that provide read-only access to a small amount of data. If a security breach occurs and you have not taken steps to require proper identification, there is no way to track down the perpetrator.

As dangerous as they are when used as a substitute for authentication, limited service systems make sense in cases where you need to limit what certain users (who have been properly identified and authenticated) can do.

6.2 DISCRETIONARY ACCESS CONTROL

Early systems had no internal access controls; any user could access any file simply by knowing its name. Access control consisted of an operator's deciding whether to mount a tape or card deck for reading or writing. This decision was rarely reliable. For example, the operator might look at the user name punched on a special ID card at the head of a batch card deck to ensure that the job requesting a tape to be mounted belonged to the owner of the tape. These ID cards might contain colored stripes to make them more difficult to forge. Such systems worked despite their flaws because the value of the information that could be gained by a penetration was rarely worth the risk or effort.

Access control became a more serious issue with the emergence of disk storage, on which files of many users could be stored online well before the days of networks or interactive computing. Indeed, controlling access to disk files was probably the first widespread computer security concern, because for the first time the system, rather than the operator, was required to enforce access control.

6.2.1 Passwords for File Access

Very simple password-based access control mechanisms were used to protect files at first; and even as technology changed from batch computing to online interactive computing, these password schemes remained the primary protection mechanism.

In a password-based access scheme, each file is given a password. A user can access a file by providing to the system the password for that file. This password has nothing to do with any password the user might need to log into the system. Each new user who needs to access the file must be notified of the file's password. In some systems that use passwords on files, only system managers can assign the passwords; in others, the owner of a file can change the password at will. There usually must be at least two passwords per file: one to control reading, and one to control writing.

While passwords are excellent for user authentication, they are unsuitable for file access control. The following problems (some of which were discussed in section 2.4) render such use highly dangerous:

- There is no way to revoke one user's access to the file (by changing the password) without revoking everyone's access. This problem is only partially corrected by using multiple passwords per file.
- There is no way for the system to keep track of who has access to the file, since passwords are distributed manually without the system's knowledge.
- Passwords for file access tend to be embedded as character strings within programs that need to use the files; so one user's program can be run by another person who does not necessarily know the passwords for all of the files the program needs in order to operate properly. Accidental and undetected exposure of passwords is greatly increased whenever passwords are written down in any form.
- Requiring a user to remember a separate password for each file is an unreasonable burden. Most likely the user will end up writing down a list of the passwords on a sheet of paper and

taping it to the terminal. in a large organization where users come and go daily, a password-based protection scheme for all files becomes impossible to manage.

6.2.2 Capability List

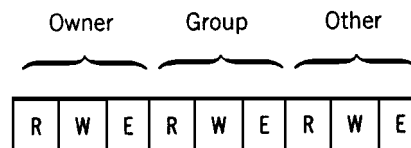
Another type of access control is the capability list or access list. A capability is a key to a specific object, along with a mode of access (read, write, or execute). A subject possessing a capability may access the object in the specified mode. At the highest levels in the system, where we are concerned with users and files, the system maintains a list of capabilities for each user. Users cannot add capabilities to this list except to cover new files they create. Users might, however, be allowed to give access to files by passing copies of their own capabilities to other users, and they might be able to revoke access to their own files by taking away capabilities from others (although revocation can be difficult to implement).

This type of access control, while much better than passwords, suffers from a software management problem. The system must maintain a list for each user that may contain hundreds or thousands of entries. When a file is deleted, the system must purge capabilities for the file from every user's list. Answering a simple question such as “who has access to this file?” requires the system to undergo a long search through every user's capability list.

The most successful use of capabilities is at lower levels in the system, where capabilities provide the underlying protection mechanism and not the user-visible access control scheme. We will discuss this lower-level use of capabilities by hardware in section 8.4.2 and by software in section 11.6.

6.2.3 Owner/Group/Other

A more effective, but simple and very common discretionary access control scheme (implemented in Unix, DEC's rsx and vms, and many other systems) uses only a few bits of access control information attached to each file:



These bits specify the access modes for different classes of users. There usually are no more than four classes: the owner of the file, users belonging to the owner's group or project, special system users, and the rest of the world. In a large system where users are grouped by project or department, most access control needs are satisfied by this technique. The scheme falls apart when access across specific groups is required. A major drawback of the scheme is its inability to specify access rights for an individual user: there is no way for Smith to specify that only Jones, and

nobody else, should have access to a file, unless there is a group defined in the system to which only Smith and Jones belong. This drawback usually results in users giving world access to their files, even though they only want to make the file accessible to specific users.

6.2.4 Access Control Lists

One of the most effective access control schemes, from a user's perspective, is the access control list, or ACL (usually pronounced “ackle”), placed on each file (fig. 6-1). The access control list identifies the individual users or groups of users who may access the file. Because all the access control information for a file is stored in one place and is clearly associated with the file, identifying who has access to a file, and adding or deleting names to the list can be done very efficiently.

FILE ALPHA		FILE BETA	
Jones.CRYPTO	rew	Smith.DRUID	r
*.CRYPTO	re	*.*	n
Green.*	n		
.	r		

Figure 6-1. Access Control List The scheme above, similar to that used in Multics and vms, employs a list of identifiers of the form USER. GROUP, where a * is a wildcard symbol matching any user or group name. When a user opens a file, the list is scanned and the allowed access corresponds to the first match. In this example, user Jones in group CRYPTO has rew access to file ALPHA, while all others in group CRYPTO have re access. Green has no access (n) unless he is in the CRYPTO group. All other users have r access.

One alleged disadvantage of an access control list scheme is performance: the access control list has to be scanned each time any user accesses (or opens) a file. But with suitable defaults and grouping of users, access control lists rarely require more than a handful of entries. The only performance penalty might be due to there being an extra disk I/O required to fetch the ACL each time a file is opened. This could have a noticeable impact on systems where large numbers of files are opened in a relatively short time. Another disadvantage is storage management: maintaining a variable-length list for each file results in either a complex directory structure or wasted space for unused entries. This tends to be a problem only for systems having huge numbers of very small files (typical of the way in which Unix systems are used).

Largely because of the complex management required, only a few systems—such as Honeywell's Multics, DEC's vms, and Data General's AOS—provide the most general form of access control list. If performance is a problem, one approach is to employ a combination of owner/ group/ other and access control lists. The access control list is only used for files where the granularity of owner/group/other is insufficient to specify the desired set of users. (vms uses this dual approach—but for compatibility with older programs, not for performance.) This approach is an example of a

performance and compatibility trade-off that violates the principle of economy of mechanism discussed in section 5.3.

6.2.5 Trojan Horse Threats

Discretionary access controls have one major drawback, regardless of the specific implementation scheme used: they are, by their very nature, subject to Trojan horse attacks. With discretionary controls, programs acting on the user's behalf are free to modify access control information for files that the user owns. The operating system cannot tell the difference between a legitimate request to modify access control information desired by the user and a request made by a Trojan horse that the user did not intend. By eliminating some flexibility, a system can limit the ability to modify access control information to special programs that have privileges. But there is still no general way, under discretionary controls, to prevent a Trojan horse in one process from transmitting information to another process via shared objects: files, messages, shared memory, and so on. See chapter 7 for a more complete discussion of the Trojan horse problem.

6.3 MANDATORY ACCESS CONTROL

Mandatory access controls prevent some types of Trojan horse attacks by imposing access restrictions that cannot be bypassed, even indirectly. Under mandatory controls, the system assigns both subjects and objects special security attributes that cannot be changed on request as can discretionary access control attributes such as access control lists. The system decides whether a subject can access an object by comparing their security attributes. A program operating on behalf of a user cannot change the security attributes of itself or of any object-including objects that the user owns. A program may therefore be unable to give away a file simply by giving other users access to it. Mandatory controls can also prevent one process from creating a shared file and passing information to another process through that file.

Many different mandatory access control schemes can be defined, but nearly all that have been proposed are variants of the U.S. Department of Defense's multilevel security policy (section 6.4). Consequently, it is difficult to discuss mandatory controls apart from multilevel security. A few general concepts, however, apply to all mandatory policies.

Mandatory controls are used in conjunction with discretionary controls and serve as an additional (and stronger) restriction on access. A subject may have access to an object only if the subject passes both discretionary and mandatory checks. Since users cannot directly manipulate mandatory access control attributes, users employ discretionary controls for their own protection from other users. Mandatory controls come into play automatically as a stronger level of protection that cannot be bypassed by users through accidental or intentional misuse of discretionary controls.

As we will see in later examples, mandatory access controls unavoidably impose some severe constraints on users with respect to their own data. Because these constraints are so visible, it is

easy to forget that the underlying purpose of mandatory controls is not to restrict the user. If we simply wanted to prevent users from accessing other users' files, discretionary controls would be sufficient. On the other hand, if we wanted to prevent a user from giving away a file, nothing the computer can do would be sufficient, as it is always possible for a user who can read a file to pass the contents of the file to another user manually. But if our intention is to prevent a program (in the form of a Trojan horse) from giving away a user's file, mandatory controls are needed. Exactly how a Trojan horse is foiled by mandatory controls is discussed in chapter 7.

In practice, mandatory controls do provide a benefit over discretionary controls, even if Trojan horses are not a threat, in cases of accident or irresponsibility. Mandatory controls make it more difficult for a user unintentionally (via an errant program or manual mistake) to give away information in an unauthorized manner. In fact, a mandatory policy can be set up so that the only ways users can pass information to other users is by means of pencil and paper or by giving away their passwords. Using mandatory controls for these purposes is quite reasonable, as long as you remember that mandatory controls can do little to prevent malicious users from revealing their own data.

Mandatory security controls have been implemented in all security kernel-based systems (see chapter 10) and in a handful of conventional (nonkernelized) operating systems. The latter include Honeywell's Multics (Whitmore et al. 1973), DEC's SES/VMS (Blotcky, Lynch, and Lipner 1986), and Sperry (now Unisys Corp.)'s 1100 Operating System (Ashland 1985).

6.4 MULTILEVEL SECURITY

The idea of multilevel security originated in the late 1960s when the U.S. Department of Defense decided it needed to develop some way of protecting classified information stored in computers (Ware 1970). Until that time it was against regulations to process classified information on a system to which uncleared people had access, because no machine was trusted to protect the classified data. Today the situation is not much different, but it should change as systems supporting mandatory controls become more widely available.

6.4.1 Military Security Policy

The Department of Defense has a strict policy for manually handling and storing classified information, which we will call the military security policy. All information (usually in the form of a document) possesses a classification, and every person possesses a clearance. In order to determine whether a person should be allowed to read a document, the person's clearance is compared to the document's classification.

A classification or clearance is made up of two components:

- A security level (also called sensitivity level or just level), consisting of one of a handful of names such as UNCLASSIFIED, CONFIDENTIAL, SECRET, and Top SECRET

- A set of one or more categories (also called compartments), consisting of names such as NATO and NUCLEAR from among a very large number of possible choices used within the Department of Defense

A classification contains a single security level, while its category set may contain an arbitrary number of categories. We will write a classification as a security level name followed by a list of category names: {SECRET; NATO,NUCLEAR,CRYPTO}. In practice the category set is often empty, and it is rarely larger than a handful of names.

The purpose of the multilevel security policy is to prevent compromise, whereby a user is able to read information classified at a level for which he or she is not cleared. In particular, the policy says nothing about the modification or destruction of information.¹

The military classification scheme has many parallels in industry, even though the terms used in industry are different (Clark and Wilson 1987; Lipner 1982). Although industry does not usually employ the concept of hierarchical security levels, most of the theory and practice for handling classified information in a computer are directly applicable to techniques for handling commercially sensitive or “privacy” information. Because a great deal of research has gone into automating the military security policy, and because the concepts are well-defined, we will continue to use terms such as SECRET and TOP SECRET. You can directly map these onto terms used in industry such as PRIVILEGED and COMPANY CONFIDENTIAL. An industry parallel to categories might be the division of a company into departments (ACCOUNTING, PAYROLL, PERSONNEL, and so on), subsidiaries, and various product development groups.

6.4.2 A Note on Terminology

To avoid confusion when reading other literature (or perhaps to confuse you more), you should notice a few things about terminology. In the context of computer security, there is no difference between a classification and a clearance: one term simply applies to an object, and the other applies to a subject. This book uses the term access class for both. Elsewhere you will run into very loose usage of all these terms. Often the terms security level and level are used as synonyms for classification, which is fine as long as the level and category breakdown of the classification is not important (it rarely is). In some documents you may see the meanings of classification and level interchanged from those given here. Again, the distinction rarely matters: access class, security level, clearance, and classification can all be safely taken to mean the same thing. In the remainder of this chapter, we will continue to speak about the components of an access class individually. In the rest of the book, we will note the rare cases where the distinction between the level and the categories matters.

1. Of course, the Department of Defense does care about information destruction, but preventing destruction is not the main reason for classifying information.

6.4.3 Mathematical Relationships

The security levels in an access class are linearly ordered; for example:

UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

One requirement of the military security policy is that, in order to obtain information legally, a person must possess an access class whose level is greater than or equal to the level of the access class of the information.

Categories are independent of each other and not ordered. To obtain access to information, a person must possess an access class whose category set includes all the categories of the access class of the information.

When categories and levels are combined, several relationships are possible between two access classes (mathematically called a *partial ordering*-see section 9.5.3).

1. The first access class *dominates* the second; that is, the level of the first is greater than or equal to the level of the second, and the category set of the first contains all the categories of the second.
2. The second access class dominates the first.
3. The access classes are equal, which is a special case where both 1 and 2 above are true.
4. None of the above is true: the access classes are disjoint and cannot be compared. The first contains a category not in the second, and the second contains a category not in the first.

The word *dominates*, when used to express a partial ordering relationship, has a meaning similar to “greater than or equal to.” While they are not mathematically correct, we will continue to use the words greater than or less than with respect to access classes and will only use the word *dominates* in contexts where a more precise meaning is required.

As an example, consider a document with access class {SECRET;NATO, NUCLEAR}. A user with access class {TOP SECRET; NATO,NUCLEAR,CRYPTO} can read the document, because the user possesses a higher level and all the categories of the document. A user with access class {TOP SECRET; NATO,CRYPTO} cannot read the document, because the user is missing the NUCLEAR category.

6.4.4 Multilevel Security Rules

Multilevel security, also known as MLS, is a mathematical description of the military security policy, defined in a form that can be implemented in a computer. The first mathematical model of a multilevel secure computer system, known as the Bell and La Padula model (Bell and La Padula 1973), defined a number of terms and concepts that have since been adopted by most other models of multilevel security. The Bell and La Padula model is often equated with multilevel security or MLS, but researchers have developed other models of multilevel security. In fact, many of the concepts of the Bell and La Padula model originated in work done at Case Western Reserve University (Walter et al. 1974). Section 9.5.3 discusses the Bell and La Padula model in detail.

Multilevel security has a number of subtleties that make it a not so-obvious transformation of the military security policy. Access classes are easy to represent in the computer, and appropriate checks can readily be made when a user tries to access a file. Enforcing multilevel security in a mandatory way, so that neither users nor their programs can change users' clearances or files' classifications, is also easy to do. This straightforward enforcement of multilevel security is commonly called simple security in the Bell and La Padula model.

Consider a system with two files and two processes (fig. 6-2). One file and one process are UNCLASSIFIED, and the other file and other process are SECRET. The simple security rule prevents the UNCLASSIFIED process from reading the SECRET file. Both processes can read and write the UNCLASSIFIED file. Despite enforcement of the simple security condition, however, a violation of the intent of the military security policy can easily occur if the SECRET process reads information out of the SECRET file and writes it into the UNCLASSIFIED file. This is equivalent to an unauthorized downgrade (lowering of the access class) of information, except that no access class of any file has been changed. Thus, while the letter of the policy has been enforced, the intent of the policy to avoid compromise has been violated. Though the actual compromise does not take place until the downgraded information is read by the unclassified process, the specific act that permits the eventual compromise is the writing of information: When a process writes information into a file whose access class is less than its own, we call the act a write-down. The write-down problem is a continual source of frustration, because even the best technical solutions to the problem adversely affect the usability of systems.

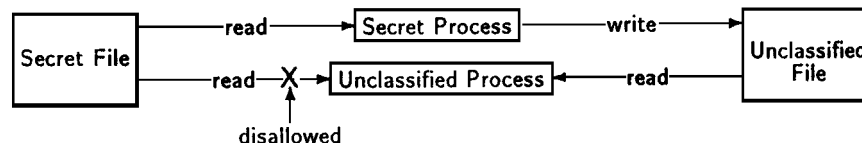


Figure 6-2. Security Violation with Simple Security Controls. In this example, despite the presence of the simple security restriction of multilevel security controls, a Trojan horse in the SECRET process is able to use the UNCLASSIFIED file as a medium for passing SECRET information to the UNCLASSIFIED process.

In general, multilevel security requires the complete prohibition of write-downs by untrusted software. Such a restriction is clearly not present in the world of people and paper: a person with a SECRET clearance is rarely prohibited from writing an UNCLASSIFIED document, despite having a desk cluttered with SECRET documents, because the person is trusted to exercise appropriate judgment in deciding what to disclose.² The restriction on write-downs in a computer is necessary because a bug or Trojan horse in the user's program cannot be trusted to exercise the same judgment. This restriction has been given the rather uninformative name *-property (pronounced "star-property") in the Bell and La Padula model—a term that has become accepted in the

2. As a half-serious proviso, it-might be noted that UNCLASSIFIED reports written by cleared individuals working on a classified project are often subject to a manual review before publication, which is a kind of write-down restriction.

computer security community. We will instead use the more descriptive name *confinement property*.

To summarize, the multilevel security model has two basic properties (fig. 6-3):

- **Simple security:** A subject can only read an object if the access class of the subject dominates the access class of the object. In other words, a subject can read down but cannot read up.
- **Confinement property:** A subject can only write an object if the access class of the subject is dominated by the access class of the object. The subject can write up but cannot write down.

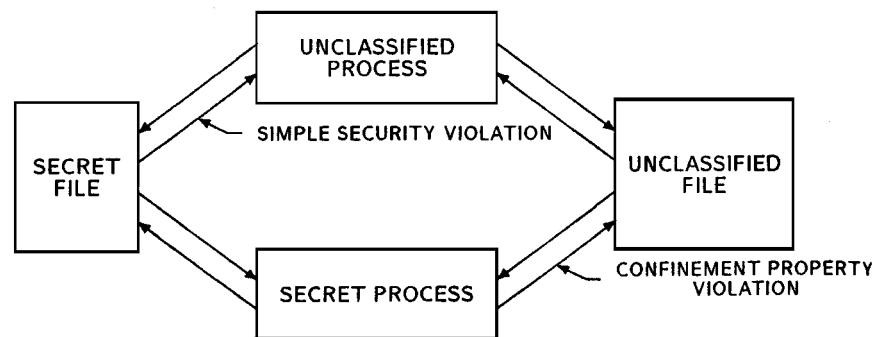


Figure 6-3. Multilevel Security Rules. A process cannot read an object at a higher access class (simple security) nor write an object at a lower access class (*-property or confinement property).

It follows that, in order for a subject to both read and write an object, the access classes of the subject and object must be equal. Although these properties allow a subject to write into an object at a higher access class, the write-up capability is often not too useful, and most systems implementing multilevel security restrict write access to objects that are of an equal access class. But from the standpoint of information compromise, there is no reason why a write-up need be disallowed. The Bell and La Padula model of multilevel security also makes use of an append access mode that allows a subject to attach information to the end of a file it cannot read. Although conceptually this seems a nice idea, implementing practical one-way writes in reality is very difficult.

6.5 INTEGRITY

Even though the confinement property of the multilevel security policy controls the writing of information, its goal is to prevent unauthorized disclosure. The multilevel security policy deals only with secrecy and does nothing to control unauthorized modification of information.

Soon after the Bell and La Padula model of multilevel security was defined, people began to wonder how to model the unauthorized modification of information. One crude way is simply to eliminate the ability to write up. But just as eliminating read-up does not alone prevent the unauthorized disclosure of information (the confinement property is also needed), eliminating write-up

does not fully prevent unauthorized modification. Something akin to the confinement property is needed to prevent a process at a higher access class from reading down and being adversely influenced by information at a lower access class.

The *Biba integrity model* (Biba 1977) addresses the modification problem by mathematically describing read and write restrictions based on integrity access classes of subjects and objects (Biba uses the terms *integrity level* and *integrity compartment*). The integrity model looks exactly the same as the multilevel security model, except that read and write restrictions are reversed:

1. A subject can write an object only if the integrity access class of the subject dominates the integrity class of the object (simple integrity),
2. A subject can read an object only if the integrity access class of the subject is dominated by the integrity class of the object (integrity confinement).

Rule 1 is the logical integrity write-up restriction that prevents contamination of high-integrity data. Figure 6-4 illustrates the reason for rule 2, the equivalent of an integrity confinement property.

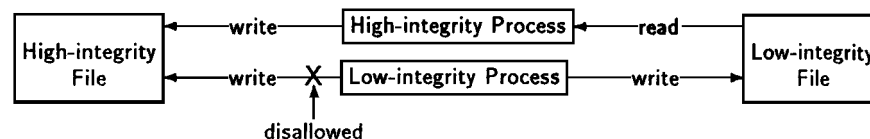


Figure 6-4. Contamination with Simple Integrity Controls. A low-integrity process is not allowed to write into and contaminate a high-integrity file; but through some error, the high-integrity process may receive low-integrity data and may write that data into the high-integrity file.

It is easiest to think about integrity if you completely ignore multilevel security for a moment. A high-integrity file is one whose contents are created by high-integrity processes. The two rules just identified guarantee that the high-integrity file cannot be contaminated by information from low-integrity processes. Furthermore, the high integrity process that writes the file cannot be subverted by low integrity processes or data. The integrity class label on a file therefore guarantees that the contents came only from sources of at least that degree of integrity.

When you consider secrecy (that is, multilevel security) and integrity together, you must be careful not to confuse the secrecy access class with the integrity access class, as they have nothing to do with one another: secrecy and integrity are independent qualities.³ You may, for example, use a spreadsheet program obtained from a public bulletin board to display TOP SECRET data. The process in which the program runs will have low integrity but high secrecy; the output may be erroneous, but the program cannot compromise the top secret data.

3. Biba originated the confusion by using the same names for both secrecy access classes and integrity access classes.

Conversely, an UNCLASSIFIED process may never have access to any classified information, but if the process's job is to perform a system management function that must work correctly to keep the system running, the process should be of high integrity. You would like to feel sure that the process cannot be influenced by low-integrity programs or be tricked by running with low-integrity data.

Although implementing integrity is straightforward, using hierarchical integrity as an adjunct to multilevel security has not fully caught on. Its application is seen as too complicated for many purposes. Whereas there are good reasons for having four or five different secrecy levels and ten or twenty categories, nobody has thought of a reason to use more than a couple of integrity levels; and on top of that, integrity categories are difficult to apply. Some people have warned that, with both secrecy and integrity fully in place, it will be all too easy to set up situations in which processes will be unable to access anything at all.

It has been proposed that eliminating the integrity confinement property restriction (rule 2) might simplify things. After all, a program of high integrity should be trusted to protect itself from low-integrity data. Although it is still possible for an "integrity Trojan horse" in that program to read the low-integrity data and write the data into a high integrity file, one may wonder how a Trojan horse has gotten into high-integrity program. The integrity confinement property is probably more suited to containing errors than Trojan horses.

Probably one of the most important reasons why the idea of integrity as an exact dual of multilevel security has problems in practice is that the notion of integrity is somehow related to the notion of trustedness. Secrecy, on the other hand, says nothing about trust, while requiring trusted software for its enforcement. We can construct any number of simple scenarios that would result in malicious software running at high secrecy access class, but it is hard to think of a reason why malicious software would be running at a very high integrity level. As long as we already have to worry about the distinction between trusted and untrusted software for security purposes, many of the aspects of the integrity model seem superfluous.

Nonetheless, the integrity model is so clean and appealing that aspects of it have been implemented in several systems, leaving its use up to the system managers. In fact, it is possible to combine the integrity and secrecy access classes into a single access class that is rarely separated into the two components; files, processes, and users are then assigned security attributes that combine both secrecy and integrity. In such a system, the rule for reading a file would be as follows: the integrity access class of the file must dominate the integrity access class of the process, and the secrecy access class of the process must dominate the secrecy access class of the file. This rule combined with the appropriate rule for writing result in a rather complex series of checks, especially given that both integrity and secrecy access classes are composed of level and category components.

It may seem at first that integrity addresses the denial-of-service problem by preventing random destruction of data (which other security techniques do not address). But, there are many ways to cause denial of service other than by destroying data: executing illegal instructions or making illegal system calls that halt the system; crashing or slowing down the system by using up

too many resources; and so on. Integrity is strictly a technique to prevent unauthorized modification.

In the systems where integrity has been implemented, the primary application has been to avoid modification of certain system programs and system databases that are important to the operation of the system and yet do not involve information with any secrecy content. For example, the list of users allowed to access the system might not be secret, but it must be protected from modification by untrusted software. This protection must be stronger than the discretionary protection provided for user files, and a mandatory integrity mechanism provides that type of protection. It has been proposed that integrity categories might be quite useful in a commercial environment (Lipner 1982)-perhaps more so than mandatory secrecy controls.

REFERENCES

- Ashland, R. E. 1985. "B1 Security for Sperry 1100 Operating System." In Proceedings of the 8th National Computer Security Conference, pp. 1057. Gaithersburg, Md.: National Bureau of Standards.
A description of mandatory controls proposed for Sperry (now Unisys) operating systems.
- Bell, D. E., and La Padula, L. J. 1973. "Secure Computer Systems: Mathematical Foundations and Model." M74-244. Bedford, Mass.: Mitre Corp. Also available through National Technical Information Service, Springfield, Va., NTIS AD-771543.)
Highly mathematical description of the original Bell and La Padula model.
- Biba, K. J. 1977. "Integrity Considerations for Secure Computer Systems." ESD-TR-76-372. Hanscom AFB, Mass.: Air Force Electronic Systems Division. (Also available through National Technical Information Service, Springfield, Va., NTIS AD-A039324.)
The Biba integrity model.
- Blotcky, S.; Lynch, K.; and Lipner, S. 1986. "SE/VMS: Implementing Mandatory Security in VAX/VMS." In Proceedings of the 9th National Computer Security Conference, pp. 47-54. Gaithersburg, Md.: National Bureau of Standards.
A description of the security enhancements offered by Digital Equipment to upgrade security of its vms operating system.
- Clark, D. D., and Wilson, D. R. 1987. "A Comparison of Commercial and Military Computer Security Policies." In Proceedings of the 1987 Symposium on Security and Privacy, pp. 184-95. Washington, D.C.: IEEE Computer Society.
Argues that a commercial security policy has some characteristics that differ from the lattice nature of the military security policy.
- Department of Defense. 1985. "Password Management Guideline." CSCSTD-002-85. Ft. Meade, Md.: National Computer Security Center.
A thorough discussion of password management guidelines.
- Gasser, M. 1975. "A Random Word Generator for Pronounceable Passwords." ESD-TR-75-97. Hanscom AFB, Mass.: Air Force Electronic Systems Division. (Also available through National Technical Information Service, Springfield, Va., NTIS AD-A017676.)
Versions of this password generator are used in Multics and vax/vms.
- Lipner, S. B. 1982. "Non-Discretionary Controls for Commercial Applications." In Proceedings of the 1982 Symposium on Security and Privacy, pp. 2-10. Silver Spring, Md.: IEEE Computer Society.

Proposes a way to use military-style mandatory security controls in a commercial environment.

National Bureau of Standards. 1985. "Password Usage Standard." FIPS PUB 112. Gaithersburg, Md.: National Bureau of Standards.

Another password management guideline, with recommendations suitable for government and industry.

Walter, K. G.; Ogden, W. F.; Rounds, W. C.; Bradshaw, F. T.; Ames, S. R.; and Shumway, D. G. 1974. "Primitive Models for Computer Security." ESD-TR-4-117. Hanscom AFB, Mass.: Air Force Electronic Systems Division. (Also available through National Technical Information Service, Springfield, Va., NTIS AD-778467.)

An early discussion of a multilevel security model, interesting for historical reasons.

Ware, W. H. 1970. "Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security." R-609-1. Santa Monica, Cal.: Rand Corp. (Reissued October 1979.)

A comprehensive discussion of computer security threats and policy recommendations, providing the foundation for much of the subsequent government work in this field; now largely of historical interest only, although many of the concepts are still valid.

Whitmore, J.; Bensoussan, A.; Green, P.; Hunt, D.; Kobziar, A.; and Stem J. 1973. "Design for Multics Security Enhancements." ESD-TR-74-176. Hanscom AFB, Mass.: Air Force Electronic Systems Division. (Also available through National Technical Information Service, Springfield, Va., NTIS AD-A030801.)

A description of the enhancements incorporated into Multics to support mandatory security controls.