CHAPTER 24

OPERATING SYSTEM SECURITY

William Stallings

24.1 INFORMATION PROTECTION AND SECURITY. This chapter reviews the principles of security in operating systems. Some general-purpose tools can be built into computers and operating systems (OSs) that support a variety of protection and security mechanisms. In general, the concern is with the problem of controlling access to computer systems and the information stored in them.

One of the core concepts in all such discussions is the *process*, which is defined as the execution of a specific piece of code by a particular user at a particular time on a particular processor.

Four types of overall protection policies, of increasing order of difficulty, have been identified:

1. **No sharing.** In this case, processes are completely isolated from each other, and each process has exclusive control over the resources statically or dynamically assigned to it. With this policy, processes often "share" a program or data file by making a copy of it and transferring the copy into their own virtual memory.

2. **Sharing originals of program or data files.** With the use of reentrant code, a single physical realization of a program can appear in multiple virtual address spaces, as can read-only data files. Special locking mechanisms are required for the sharing of writable data files, to prevent simultaneous users from interfering with each other.
3. **Confined, or memoryless, subsystems.** In this case, processes are grouped into subsystems to enforce a particular protection policy. For example, a "client" process calls a "server" process to perform some task on data. The server is to be protected against the client discovering the algorithm by which it performs the task, while the client is to be protected against the server's retaining any information about the task being performed.
4. **Controlled information dissemination.** In some systems, security classes are defined to enforce a particular dissemination policy. Users and applications are given security clearances of a certain level, while data and other resources (e.g., input/output [I/O] devices) are given security classifications. The security policy enforces restrictions concerning which users have access to which classifications. This model is useful not only in the military context but in commercial applications as well.[1]

Much of the work in security and protection as it relates to OSs can be roughly grouped into three categories.

1. **Access control.** Concerned with regulating user access to the total system, subsystems, and data, and regulating process access to various resources and objects within the system.
2. **Information flow control.** Regulates the flow of data within the system and its delivery to users.
3. **Certification.** Relates to proving that access and flow control mechanisms perform according to their specifications and that they enforce desired protection and security policies.

This chapter looks at some of the key mechanisms for providing OS security and then examines Windows 2000 as a case study.

24.2 REQUIREMENTS FOR OPERATING SYSTEM SECURITY

24.2.1 Requirements. Understanding the types of threats to OS security requires a definition of security requirements. OS security addresses four requirements:

1. **Confidentiality**. Requires that the information in a computer system be accessible only for reading by authorized parties. This type of access includes printing, displaying, and other forms of disclosure, including simply revealing the existence of an object.
2. **Integrity**. Requires that only authorized parties be able to modify computer system assets. Modification includes writing, changing, changing status, deleting, and creating.

3. **Availability.** Requires that computer system assets are available to authorized parties.
4. **Authentication.** Requires that a computer system be able to verify the identifier of a user, a device or a process.

24.2.2 Computer System Assets. The assets of a computer system can be categorized as hardware, software, and data.

24.2.2.1 Hardware. The main threat to computer system hardware is in the area of availability. Hardware is the most vulnerable to attack and the least amenable to automated controls. Threats include accidental and deliberate damage to equipment as well as theft. The proliferation of personal computers and workstations and the increasing use of local area networks (LANs) increase the potential for losses in this area. Physical and administrative security measures are needed to deal with these threats. Chapters 22 and 23 in this *Handbook* discuss physical security.

24.2.2.2 Software. The OS, utilities, and application programs are what make computer system hardware useful to businesses and individuals. Several distinct threats need to be considered.

A key threat to software is an attack on availability. Software, especially application software, is surprisingly easy to delete. Software also can be altered or damaged to render it useless or dangerous. Careful software configuration management, which includes making backups of the most recent version of software, can maintain high availability. A more difficult problem to deal with is software modification that results in a program that still functions but that behaves differently from before. A final problem is control or possession of software. Although certain countermeasures are available, by and large the problem of unauthorized copying of software has not been solved.

Chapters 38, 39, and 40 of this *Handbook* discuss software security in some detail.

24.2.2.3 Data. Hardware and software security typically are concerns of computing center professionals or individual concerns of personal computer users. A much more widespread problem is data security, which involves files and other forms of data controlled by individuals, groups, and business organizations.

Security concerns with respect to data are broad, encompassing confidentiality, control or possession, integrity, authenticity, availability and utility. For a sound theoretical treatment of the attributes of information that must be protected through security measures, see Chapter 3 in this *Handbook*.

In the case of availability, the concern is with the *destruction* of data files, which can occur either accidentally or maliciously, and with delays in timely access to data.

The obvious concern with confidentiality is the unauthorized reading of data files or databases, and this area has been the subject of perhaps more research and effort than any other area of computer security. A less obvious secrecy threat involves the analysis of data and manifests itself in the use of so-called statistical databases or *data mining*, which provide summary or aggregate information and potentially lead to discovery of unpublicized tendencies, relations, or trends. Aggregate information does not necessarily threaten the privacy of the individuals involved. However, as the use of statistical databases grows, there is an increasing potential for disclosure of personal information through induction or deduction. In essence, characteristics of constituent

individuals may be identified through careful analysis. To take a simple example, if one table records the aggregate of the incomes of respondents A, B, C, and D and another records the aggregate of the incomes of A, B, C, D, and E, the difference between the two aggregates would be the income of E. This problem is exacerbated by the increasing desire to combine data sets. In many cases, matching several sets of data for consistency at levels of aggregation appropriate to the problem requires a retreat to elemental units in the process of constructing the necessary aggregates. Thus, the elemental units, which are the subject of privacy concerns, are available at various stages in the processing of data sets.

Finally, data integrity is a major concern in most installations. Modifications to data files can have consequences ranging from minor to disastrous.

24.2.3 Design Principles. Saltzer and Schroeder identify a number of principles for the design of security measures for the various threats to computer systems. These include:

- **Least privilege**. Every program and every user of the system should operate using the least set of privileges necessary to complete the job. Access rights should be acquired by explicit permission only; the default should be "no access."
- **Economy of mechanisms**. Security mechanisms should be as small and simple as possible, aiding in their verification. This usually means that they must be an integral part of the design rather than add-on mechanisms to existing designs.
- **Acceptability**. Security mechanisms should not interfere unduly with the work of users. At the same time, the mechanisms should meet the needs of those who authorize access. If the mechanisms are not easy to use, they are likely to be unused or incorrectly used.
- **Complete mediation**. Every access must be checked against the access-control information, including those accesses occurring outside normal operation, as in recovery or maintenance.
- **Open design**. The security of the system should not depend on keeping the design of its mechanisms secret. Thus, the mechanisms can be reviewed by many experts, and users can have high confidence in them.[2]

24.3 PROTECTION MECHANISMS. The introduction of multiprogramming brought about the ability to share resources among users. This sharing involves not just the processor but also:

- Memory
- I/O devices, such as disks and printers
- Programs
- Data

The ability to share these resources introduced the need for protection. Pfleeger and Pfleeger point out that an OS may offer protection along this spectrum:

- **No protection**. This is appropriate when sensitive procedures are being run at separate times.

- **Isolation**. This approach implies that each process operates separately from other processes, with no sharing or communication. Each process has its own address space, files, and other objects.
- **Share all or share nothing**. The owner of an object (e.g., a file or memory segment) declares it to be public or private. In the former case, any process may access the object; in the latter, only the owner's processes may access the object.
- **Share via access limitation**. The OS checks the permissibility of each access by a specific user to a specific object. The OS therefore acts as a guard, or gatekeeper, between users and objects, ensuring that only authorized accesses occur.
- **Share via dynamic capabilities**. This extends the concept of access control to allow dynamic creation of sharing rights for objects.
- **Limit use of an object**. This form of protection limits not just access to an object but the use to which that object may be put. For example, a user may be allowed to view a sensitive document but not print it. Another example is that a user may be allowed access to a database to derive statistical summaries but not to determine specific data values.[3]

The preceding items are listed roughly in increasing order of difficulty to implement but also in increasing order of fineness of protection that they provide. A given OS may provide different degrees of protection for different objects, users, or applications.

The OS needs to balance the need to allow sharing, which enhances the utility of the computer system, with the need to protect the resources of individual users. This section considers some of the mechanisms by which OSs have enforced protection for these objects.

24.3.1 Protection of Memory. In a multiprogramming environment, protection of main memory (random-access memory, or *RAM*) is essential. The concern here is not just security but the correct functioning of the various processes that are active. If one process can inadvertently write into the memory space of another process, then the latter process may not execute properly.

The separation of the memory space of various processes is accomplished easily with a virtual memory scheme. Either segmentation or paging, or the two in combination, provides an effective means of managing main memory. If complete isolation is sought, then the OS simply must ensure that each segment or page is accessible only by the process to which it is assigned. This is accomplished easily by requiring that there be no duplicate entries in page and/or segment tables.

If sharing is to be allowed, then the same segment or page may appear in more than one table. This type of sharing is accomplished most easily in a system that supports segmentation or a combination of segmentation and paging. In this case, the segment structure is visible to the application, and the application can declare individual segments to be sharable or nonsharable. In a pure paging environment, it becomes more difficult to discriminate between the two types of memory, because the memory structure is transparent to the application.

Segmentation, especially, lends itself to the implementation of protection and sharing policies. Because each segment table entry includes a length as well as a base address, a program cannot inadvertently access a main memory location beyond the limits of a segment. To achieve sharing, it is possible for a segment to be referenced in the segment tables of more than one process. The same mechanisms are available in a paging system. However, in this case the page structure of programs and data is not visible

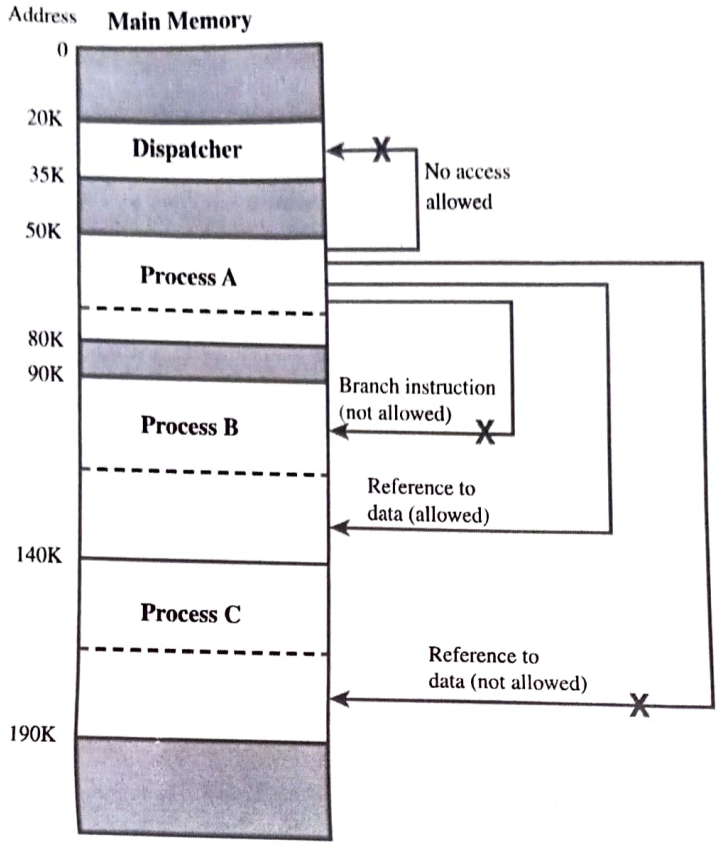


EXHIBIT 24.1 Protection Relationships between Segments

to the programmer, making the specification of protection and sharing requirements more awkward. Exhibit 24.1 illustrates the types of protection relationships that can be enforced in such a system.

An example of the hardware support that can be provided for memory protection is that of the IBM System/370 family of machines, on which OS/390 runs. Associated with each page frame in main memory is a 7-bit storage control key, which may be set by the OS. Two of the bits indicate whether the page occupying this frame has been referenced and changed; these bits are used by the page replacement algorithm. The remaining bits are used by the protection mechanism: a 4-bit access-control key and a fetch-protection bit. Processor references to memory and direct memory access (DMA). DMA I/O memory references must use a matching key to gain permission to access that page. The fetch-protection bit indicates whether the access-control key applies to writes or to both reads and writes. In the processor, there is a program status word (PSW), which contains control information relating to the process that is currently executing. Included in this word is a 4-bit PSW key. When a process attempts to access a page or to initiate a DMA operation on a page, the current PSW key is compared to the access code. A write operation is permitted only if the codes match. If the fetch bit is set, then the PSW key must match the access code for read operations.

24.3.2 User-Oriented Access Control. The measures taken to control access in a data processing system fall into two categories: those associated with the user and those associated with the data.

The most common technique for user access control on a shared system or server is the user logon, which requires both a user identifier (ID) and some form of *authentication*, such as providing a password, a token, or biometric attributes. Authentication refers to the binding of a real-world identity (for example, a named employee or a named role in an organization) and the ID being presented. The system will allow a user to log on only if that user's ID is known to the system and if the user knows the password associated by the system with that ID.

Once a user has established a *session*, the operating system can then *authorize* different forms of access (e.g., read, write, append, lock, or execute) to different types of data (e.g., specific files, databases, devices, or communications).

User access control in a distributed environment can be either centralized or decentralized. In a centralized approach, the network provides a logon service, determining who is allowed to use the network and to whom the user is allowed to connect.

Decentralized user access control treats the network as a transparent communication link, and the destination host carries out the usual logon procedure. The security concerns for transmitting passwords over the network must still be addressed.

In many networks, two levels of access control may be used. Individual hosts may be provided with a logon facility to protect host-specific resources and application. In addition, the network as a whole may provide protection to restrict network access to authorized users. This two-level facility is desirable for the common case, currently, in which the network connects disparate hosts and simply provides a convenient means of terminal-host access. In a more uniform network of hosts, some centralized access policy could be enforced in a network control center.

Chapters 28 and 29 of this *Handbook* present more information about identification and authentication.

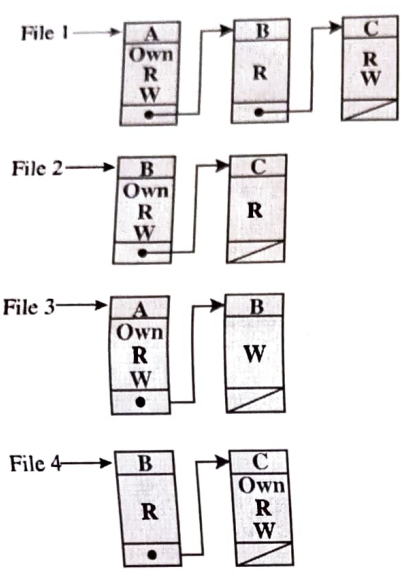
24.3.3 Data-Oriented Access Control. Following successful logon, the user is granted access to one or a set of hosts and applications. This is generally not sufficient for a system that includes sensitive data in its database. Through the user access-control procedure, a user can be identified to the system. Associated with each user, there can be a profile that specifies permissible operations and file accesses. The OS can then enforce rules based on the user profile. The database-management system, however, must control access to specific records or even portions of records. For example, it may be permissible for anyone in administration to obtain a list of company personnel, but only selected individuals may have access to salary information. The issue is more than just one of level of detail. Whereas the OS may grant a user permission to access a file or use an application, following which there are no further security checks, the database management system must make a decision on each individual access attempt. That decision will depend not only on the user's identity but also on the specific parts of the data being accessed and even on the information already divulged to the user.

A general model of access control as exercised by a file or database management system is that of an *access matrix* (see Exhibit 24.2a). The basic elements of the model are:

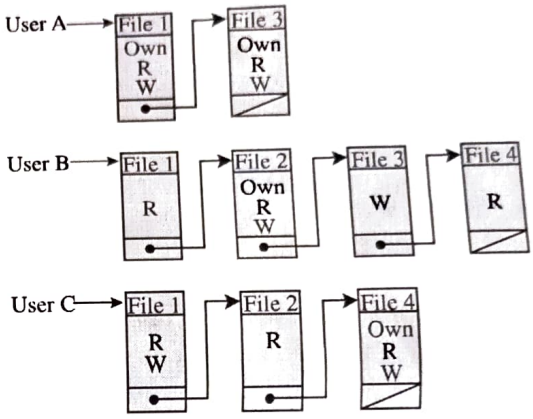
- **Subject**. An entity capable of accessing objects. Generally, the concept of subject equates with that of process. Any user or application actually gains access to an object by means of a process that represents that user or application.
- **Object**. Anything to which access is controlled. Examples include files, portions of files, programs, and segments of memory.
- **Access right**. The way in which an object is accessed by a subject. Examples are read, write, and execute.

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

(a) Access matrix



(b) Access control lists for files of part (a)



(c) Capability lists for files of part (a)

Exhibit 24.2 Example of Access-Control Structures
Source: Based on a figure in Sandhu (1996).

One dimension of the matrix consists of identified subjects that may attempt data access. Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, hosts, or applications, instead of, or in addition to, users. The other dimension lists the objects that may be accessed. At the greatest level of detail, objects may be individual data fields. More aggregate groupings, such as records, files, or even the entire database, also may be objects in the matrix. Each entry in the matrix indicates the access rights of that subject for that object.

In practice, an access matrix usually is sparse and is implemented by decomposition in one of two ways. The matrix may be decomposed by columns, yielding *access-control lists* (see Exhibit 24.2b). Thus, for each object, an access-control list lists users and their permitted access rights. The access-control list may contain a default, or public, entry. This allows users who are not explicitly listed as having special rights to have a default set of rights. Elements of the list may include individual users as well as groups of users.

Decomposition by rows yields *capability tickets* (see Exhibit 24.2c). A capability ticket specifies authorized objects and operations for a user. Each user has a number

of tickets and may be authorized to lend or give them to others. Because tickets may be dispersed around the system, they present a greater security problem than access-control lists. In particular, the ticket must be unforgeable. One way to accomplish this is to have the OS hold all tickets on behalf of users. These tickets would have to be held in a region of memory inaccessible to users.

Network considerations for data-oriented access control parallel those for user-oriented access control. If only certain users are permitted to access certain items of data, then encryption may be needed to protect those items during transmission to authorized users. Typically, data access control is decentralized, that is, controlled by host-based database management systems. If a network database server exists on a network, then data access control becomes a network function.

24.3.4 Protection Based on an Operating System Mode. One technique used in all OSs to provide protection is based on the mode of processor execution. Most processors support at least two modes of execution: the mode normally associated with the OS and that normally associated with user programs. Certain instructions can be executed only in the more privileged mode. These would include reading or altering a control register, such as the program status word; primitive I/O instructions; and instructions that relate to memory management. In addition, certain regions of memory can be accessed only in the more privileged mode.

The less privileged mode often is referred to as the *user* mode, because user programs typically would execute in this mode. The more privileged mode is referred to as the *system mode*, *control mode*, or *kernel mode*. This last term refers to the kernel of the OS, which is that portion of the OS that encompasses the important system functions. Exhibit 24.3 lists the functions typically found in the kernel of an OS.

EXHIBIT 24.3 Typical Kernel Mode Operating System Functions

Process Management

- Process creation and termination
- Process scheduling and dispatching
- Process switching
- Process synchronization and support for interprocess communication
- Management of process control blocks

Memory Management

- Allocation of address space to processes
- Swapping
- Page and segment management

I/O Management

- Buffer management
- Allocation of I/O channels and devices to processes

Support functions

- Interrupt handling
- Accounting
- Monitoring

The reason for using two modes should be clear. It is necessary to protect the OS and key OS tables, such as process control blocks, from interference by user programs. In the kernel mode, the software has complete control of the processor and all its instructions, registers, and memory. This level of control is not necessary, and for safety is not desirable, for user programs.

Two questions arise: How does the processor know in which mode it is to be executing, and how is the mode changed? Regarding the first question, typically there is a bit in the program status word that indicates the mode of execution. This bit is changed in response to certain events. For example, when a user makes a call to an OS service, the mode is set to the kernel mode. Typically, this is done by executing an instruction that changes the mode. When the user makes a system service call, or when an interrupt transfers control to a system routine, the routine executes the change-mode instruction to enter a more privileged mode and executes it again to enter a less privileged mode before returning control to the user process. If a user program attempts to execute a change-mode instruction, it will simply result in a call to the OS, which will return an error unless the mode change is to be allowed.

More sophisticated mechanisms also can be provided. A common scheme is to use a ring-protection structure. In this scheme, lower-numbered, or inner, rings enjoy greater privilege than higher-numbered, or outer, rings. Typically, ring 0 is reserved for kernel functions of the OS, with applications at a higher level. Some utilities or OS services may occupy an intermediate ring. Basic principles of the ring system are:

- A program may access only those data that reside on the same ring or a less privileged ring.
- A program may call services residing on the same or a more privileged ring.

An example of the ring protection approach is found on the VAX VMS OS, which uses four modes:

1. **Kernel**. Executes the kernel of the VMS OS, which includes memory management, interrupt handling, and I/O operations.
2. **Executive**. Executes many of the OS service calls, including file and record (disk and tape) management routines.
3. **Supervisor**. Executes other OS services, such as responses to user commands.
4. **User**. Executes user programs, plus utilities such as compilers, editors, linkers, and debuggers.

A process executing in a less privileged mode often needs to call a procedure that executes in a more privileged mode; for example, a user program requires an OS service. This call is achieved by using a change-mode (CHM) instruction, which causes an interrupt that transfers control to a routine at the new access mode. A return is made by executing the REI (return from exception or interrupt) instruction.

24.3.5 Protection Based on Virtualization. With the growing availability of memory (e.g., tens to hundreds of gigabytes of RAM), disk space (terabytes to petabytes of storage), faster processors (tens of gigahertz), and multicore systems (potentially thousands of processors working in parallel), *virtualization* of computers has progressed to practical and widespread usability. Instantiations of an operating environment can coexist using shared resources without permitting any direct communication among them. Each instantiation is encapsulated and completely protected against

intrusion or interference from processes running on other virtual machines sharing the same physical resources.

24.4 FILE SHARING. Multiuser systems almost always require that files can be shared among a number of users. Two issues arise: access rights and the management of simultaneous access.

24.4.1 Access Rights. The file system should provide a flexible tool for allowing extensive file sharing among users. The file system should provide a number of options so that the way in which a particular file is accessed can be controlled. Typically, users or groups of users are granted certain access rights to a file. A wide range of access rights has been used. The next list indicates access rights that can be assigned to a particular user for a particular file.

- **None**. The user may not even learn of the existence of the file, much less access it. To enforce this restriction, the user would not be allowed to read the user directory that includes this file.
- **Knowledge**. The user can determine that the file exists and who its owner is. The user is then able to petition the owner for additional access rights.
- **Execution**. The user can load and execute a program but cannot copy it. Proprietary programs often are made accessible with this restriction.
- **Locking**. The user can change the status of a logical *flag* that indicates temporary restrictions on access to data. Database management systems provide for locking to control concurrent access to records so that different processes can avoid overwriting each other's modifications.
- **Reading**. The user can read the file for any purpose, including copying and execution. Some systems are able to enforce a distinction between viewing and copying. In the former case, the contents of the file can be displayed to the user, but the user has no means for making a copy.
- **Appending**. The user can add data to the file, often only at the end, but cannot modify or delete any of the file's contents. This right is useful in collecting data from a number of sources.
- **Updating**. The user can modify, delete, and add to the file's data. This normally includes writing the file initially, rewriting it completely or in part, and removing all or a portion of the data. Some systems distinguish among different degrees of updating.
- **Changing protection**. The user can change the access rights granted to other users. Typically, only the owner of the file holds this right. In some systems, the owner can extend this right to others. To prevent abuse of this mechanism, the file owner typically is able to specify which rights can be changed by the holder of this extended right.
- **Deletion**. The user can delete the file from the file system.

These rights can be considered to constitute a hierarchy, with each right implying those that precede it. Thus, if a particular user is granted the updating right for a particular file, then that user also is granted these rights: knowledge, execution, reading, and appending.

One user is designated as owner of a given file, usually this is the person who initially created the file. The owner has all of the access rights listed previously and may grant rights to others. Access can be provided to different classes of users:

- **Specific user**. Individual users who are designated by user ID.
- **User groups**. A set of users who are not individually defined. The system must have some way of keeping track of the membership of user groups.
- **All**. All users who have access to this system. These are public files.

24.4.2 Simultaneous Access. When access is granted to append or update a file to more than one user, the OS or file management system must enforce discipline. A brute-force approach is to allow a user to lock the entire file when it is to be updated. A finer grain of control is to lock individual records during update. Issues of mutual exclusion and deadlock must be addressed in designing the shared access capability. Chapter 52 in this *Handbook* discusses application controls in more detail.

24.5 TRUSTED SYSTEMS. Much of what has been discussed so far has concerned protecting a given message or item from passive or active attack by a given user. A somewhat different but widely applicable requirement is to protect data or resources on the basis of levels of security. This is commonly found in the military, where information is categorized as unclassified (U), confidential (C), secret (S), top secret (TS), or beyond. This concept is equally applicable in other areas, where information can be organized into gross categories and users can be granted clearances to access certain categories of data. For example, the highest level of security might be for strategic corporate planning documents and data, accessible only by corporate officers and their staff; next might come sensitive financial and personnel data, accessible only by administration personnel, corporate officers, and so on.

When multiple categories or levels of data are defined, the requirement is referred to as *multilevel security*. The general statement of the requirement for multilevel security is that a subject at a high level may not convey information to a subject at a lower or incomparable level unless that flow accurately reflects the will of an authorized user. For implementation purposes, this requirement is in two parts and is simply stated. A multilevel secure system must enforce:

1. **No read up**. A subject can only read an object of less or equal security level. This is referred to in the literature as the *simple security property*.
2. **No write down**. A subject can only write into an object of greater or equal security level. This is referred to in the literature as the **-property* (pronounced *star property*).

These two rules, if properly enforced, provide multilevel security. For a data processing system, the approach that has been taken, and has been the object of much research and development, is based on the *reference monitor* concept. This approach is depicted in Exhibit 24.4. The reference monitor is a controlling element in the hardware and OS of a computer that regulates the access of subjects to objects on the basis of security parameters of the subject and object. The reference monitor has access to a file, known as the *security kernel database*, that lists the access privileges (security clearance) of each subject and the protection attributes (classification level) of each

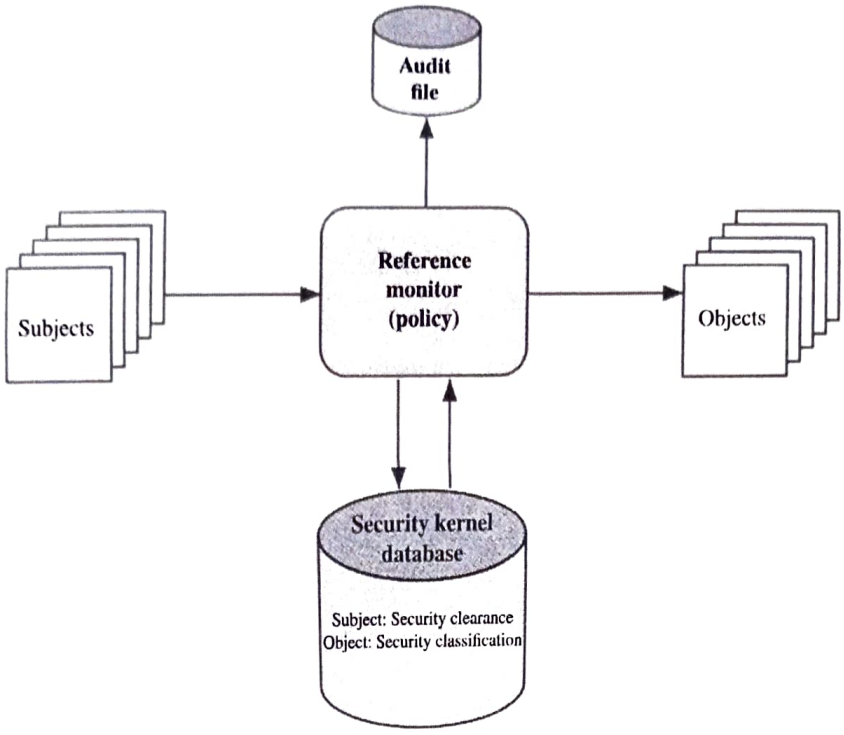


EXHIBIT 24.4 Reference Monitor Concept

object. The reference monitor enforces the security rules (no read up, no write down) and has these properties:

- **Complete mediation**. The security rules are enforced on every access, not just, for example, when a file is opened.
- **Isolation**. The reference monitor and database are protected from unauthorized modification.
- **Verifiability**. The reference monitor's correctness must be provable. That is, it must be possible to demonstrate mathematically that the reference monitor enforces the security rules and provides complete mediation and isolation.

These are stiff requirements. The requirement for complete mediation means that every access to data within main memory and on disk and tape must be mediated. Pure software implementations impose too high a performance penalty to be practical; the solution must be at least partly in hardware. The requirement for isolation means that it must not be possible for an attacker, no matter how clever, to change the logic of the reference monitor or the contents of the security kernel database. Finally, the requirement for mathematical proof is formidable for something as complex as a general-purpose computer. A system that can provide such verification is referred to as a *trusted system.*

Chapter 9 in this *Handbook* discusses mathematical models of computer security in more detail.

A final element illustrated in Exhibit 24.4 is an audit file. Important security events, such as detected security violations and authorized changes to the security kernel database, are stored in the audit file.

In an effort to meet its own needs and as a service to the public, the U.S. Department of Defense in 1981 established the Computer Security Center within the National Security Agency (NSA), with the goal of encouraging the widespread availability of trusted computer systems. This goal is realized through the center's Commercial Product Evaluation Program. In essence, the center attempts to evaluate commercially available products as meeting the security requirements just outlined. The center classifies evaluated products according to the range of security features that they provide. These evaluations are needed for Department of Defense procurements but are published and freely available. Hence, they can serve as guidance to commercial customers for the purchase of commercially available, off-the-shelf equipment.

24.5.1 Trojan Horse Defense. A *Trojan horse* attack involves software that appears to have acceptable functions but which conceals additional, unauthorized functionality. Chapters 2 and 15 in this *Handbook* provide details of many Trojan horse attacks.

One way to secure against Trojan horse attacks is by the use of a secure, trusted OS. Exhibit 24.5 illustrates an example. In this case, a Trojan horse is used to get around the standard security mechanism used by most file management and OSs: the access-control list. In this example, a user named Bob interacts through a program with a data file containing the critically sensitive character string "CPE170KS". User Bob has created the file with read/write permission provided only to programs executing on his own behalf: that is, only processes that are owned by Bob may access the file.

The Trojan horse attack begins when a hostile user, named Alice, gains legitimate access to the system and installs both a Trojan horse program and a private file to be used in the attack as a "back pocket." Alice gives read/write permission to herself for this file and gives Bob write-only permission (see Exhibit 24.5a). Alice now induces Bob to invoke the Trojan horse program, perhaps by advertising it as a useful utility. When the program detects that it is being executed by Bob, it reads the sensitive character string from Bob's file and copies it into Alice's back-pocket file (see Exhibit 24.5b). Both the read and write operations satisfy the constraints imposed by access-control lists. Alice then has only to access Bob's file at a later time to learn the value of the string.

Now consider the use of a secure OS in this scenario (see Exhibit 24.5c). Security levels are assigned to subjects at logon on the basis of criteria such as the terminal from which the computer is being accessed and the user involved, as identified by password/ID. In this example, there are two security levels, sensitive (gray) and public (white), ordered so that sensitive is higher than public. Processes owned by Bob and Bob's data file are assigned the security level sensitive. Alice's file and processes are restricted to public. If Bob invokes the Trojan horse program (see Exhibit 24.5d), that program acquires Bob's security level. It is therefore able, under the simple security property, to observe the sensitive character string. When the program attempts to store the string in a public file (the back-pocket file), however, the *-property is violated and the attempt is disallowed by the reference monitor. Thus, the attempt to write into the back-pocket file is denied even though the access-control list permits it: The security policy takes precedence over the access-control list mechanism.

24.6 WINDOWS 2000 SECURITY. A good example of the access-control concepts discussed in this chapter is the Windows 2000 (W2K) access-control facility, which exploits object-oriented concepts to provide a powerful and flexible access-control capability.

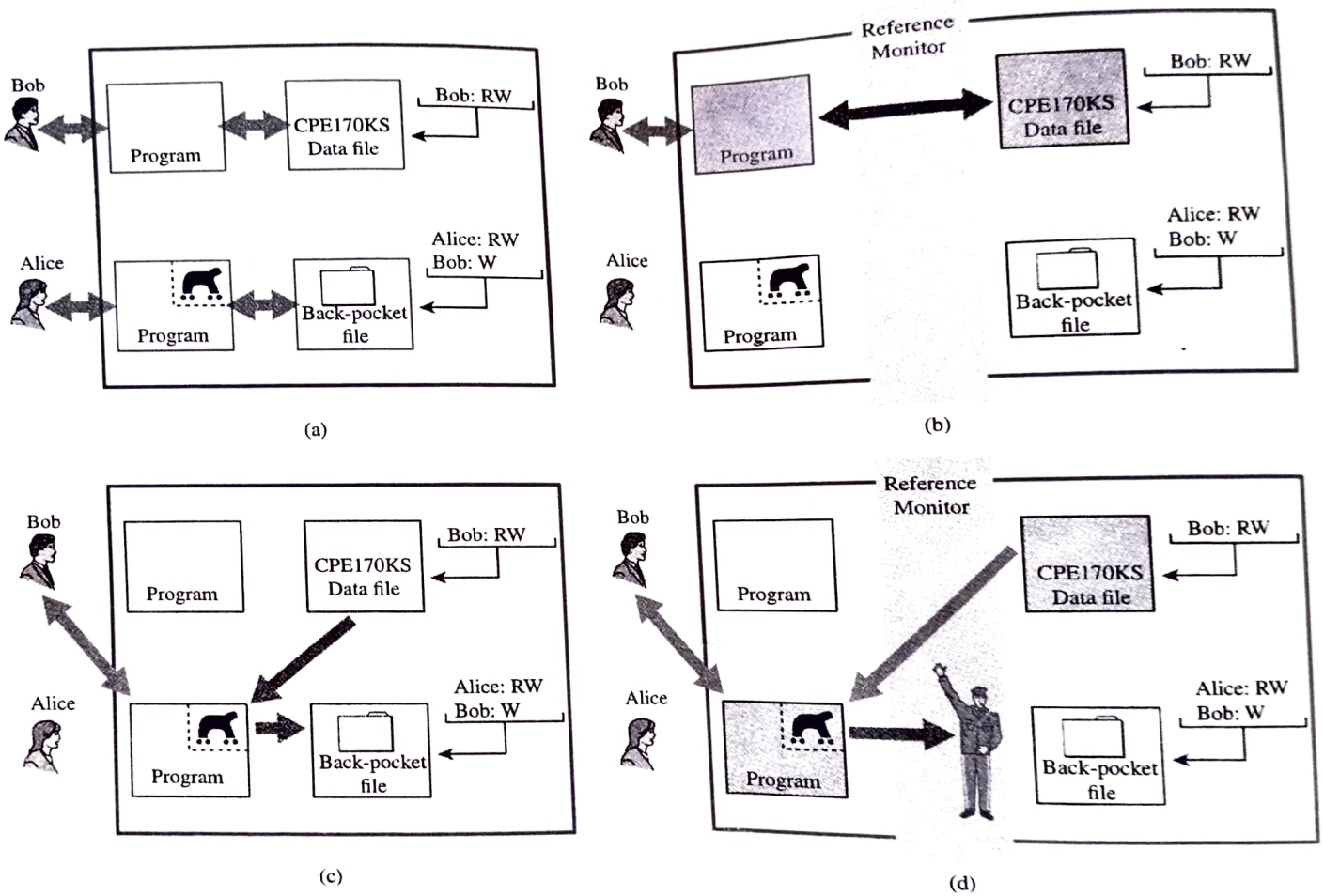


Exhibit 24.5 Trojan Horse and Secure Operating Systems

W2K provides a uniform access-control facility that applies to processes, threads, files, semaphores, windows, and other objects. Access control is governed by two entities: an access token associated with each process and a security descriptor associated with each object for which interprocess access is possible.

24.6.1 Access-Control Scheme. When a user logs on to a W2K system, W2K uses a name/password scheme to authenticate the user. If the logon is accepted, a process is created for the user and an access token is associated with that process object. The access token, whose details are described later, include a security ID (SID), which is the identifier by which this user is known to the system for purposes of security. When the initial user process spawns any additional processes, the new process object inherits the same access token.

The access token serves two purposes:

1. It keeps all necessary security information together to speed access validation. When any process associated with a user attempts access, the security subsystem can make use of the token associated with that process to determine the user's access privileges.
2. It allows each process to modify its security characteristics in limited ways without affecting other processes running on behalf of the user.

The chief significance of the second point has to do with privileges that may be associated with a user. The access token indicates which privileges a user may have. Generally, the token is initialized with each of these privileges in a disabled state. Subsequently, if one of the user's processes needs to perform a privileged operation, the process may enable the appropriate privilege and attempt access. It would be undesirable to keep all of the security information for a user in one systemwide place, because in that case enabling a privilege for one process enables it for all of them.

A security descriptor is associated with each object for which interprocess access is possible. The chief component of the security descriptor is an access-control list that specifies access rights for various users and user groups for this object. When a process attempts to access this object, the SID of the process is matched against the access-control list of the object to determine if access will be allowed.

When an application opens a reference to a securable object, W2K verifies that the object's security descriptor grants the application's user access. If the check succeeds, W2K caches the resulting granted access rights.

An important aspect of W2K security is the concept of impersonation, which simplifies the use of security in a client/server environment. If client and server talk through a remote procedure call (RPC) connection, the server can temporarily assume the identity of the client so that it can evaluate a request for access relative to that client's rights. After the access, the server reverts to its own identity.

24.6.2 Access Token. Exhibit 24.6a shows the general structure of an access token, which includes these parameters:

- **Security ID (SID).** Identifies a user uniquely across all of the machines on the network. This generally corresponds to a user's logon name.
- **Group SIDs.** A list of the groups to which this user belongs. A group is simply a set of user IDs that are identified as a group for purposes of access control. Each

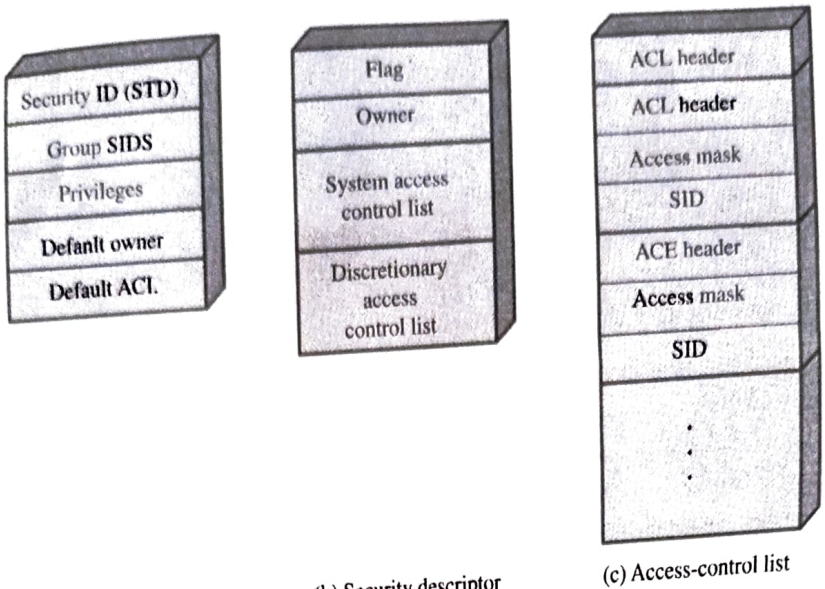


EXHIBIT 24.6 Windows 2000 Security Structures

group has a unique group SID. Access to an object can be defined on the basis of group SIDs, individual SIDs, or a combination.

- **Privileges**. A list of security-sensitive system services that this user may call. An example is create token. Another example is the set backup privilege; users with this privilege are allowed to use a backup tool to back up files that they normally would not be able to read. Most users will have no privileges.
- **Default owner**. If this process creates another object, this field specifies who is the owner of the new object. Generally, the owner of the new process is the same as the owner of the spawning process. However, a user may specify that the default owner of any processes spawned by this process is a group SID to which this user belongs.
- **Default access control list (ACL)**. This is an initial list of protections applied to the objects that the user creates. The user may subsequently alter the ACL for any object that it owns or that one of its groups owns.

24.6.3 Security Descriptors. Exhibit 24.6b shows the general structure of a security descriptor, which includes these parameters:

- **Flags**. Defines the type and contents of a security descriptor. The flags indicate whether the system access-control list (SACL) and discretionary access control list (DACL) are present, whether they were placed on the object by a defaulting mechanism, and whether the pointers in the descriptor use absolute or relative addressing. Relative descriptors are required for objects that are transmitted over a network, such as information transmitted in an RPC.
- **Owner**. The owner of the object generally can perform any action on the security descriptor. The owner can be an individual or a group SID. The owner has the authority to change the contents of the DACL.
- **System access control list (SACL)**. Specifies what kinds of operations on the object should generate audit messages. An application must have the corresponding

privilege in its access token to read or write the SACL of any object. This is to prevent unauthorized applications from reading SACLs (thereby learning what not to do to avoid generating audits) or writing them (to generate many audits to cause an illicit operation to go unnoticed).

- **Discretionary access-control list (DACL).** Determines which users and groups can access this object for which operations. It consists of a list of access-control entries (ACEs).

When an object is created, the creating process can assign as owner its own SID or any group SID in its access token. The creating process cannot assign an owner that is not in the current access token. Subsequently, any process that has been granted the right to change the owner of an object may do so, but again with the same restriction. The reason for the restriction is to prevent a user from covering his or her tracks after attempting some unauthorized action.

Let us look in more detail at the structure of access-control lists, because these are at the heart of the W2K access-control facility (see Exhibit 24.7). Each list consists of an overall header and a variable number of access-control entries. Each entry specifies an individual or group SID and an access mask that defines the rights to be granted to this SID. When a process attempts to access an object, the object manager in the W2K executive reads the SID and group SIDs from the access token and then scans down the object's DACL. If a match is found—that is, if an ACE is found with a SID that matches one of the SIDs from the access token—then the process has the access rights specified by the access mask in that ACE.

Exhibit 24.7 shows the contents of the access mask. The least significant 16 bits specify access rights that apply to a particular type of object. For example, bit 0 for a file object is File_Read_Data access, and bit 0 for an event object is Event_Query_Status access.

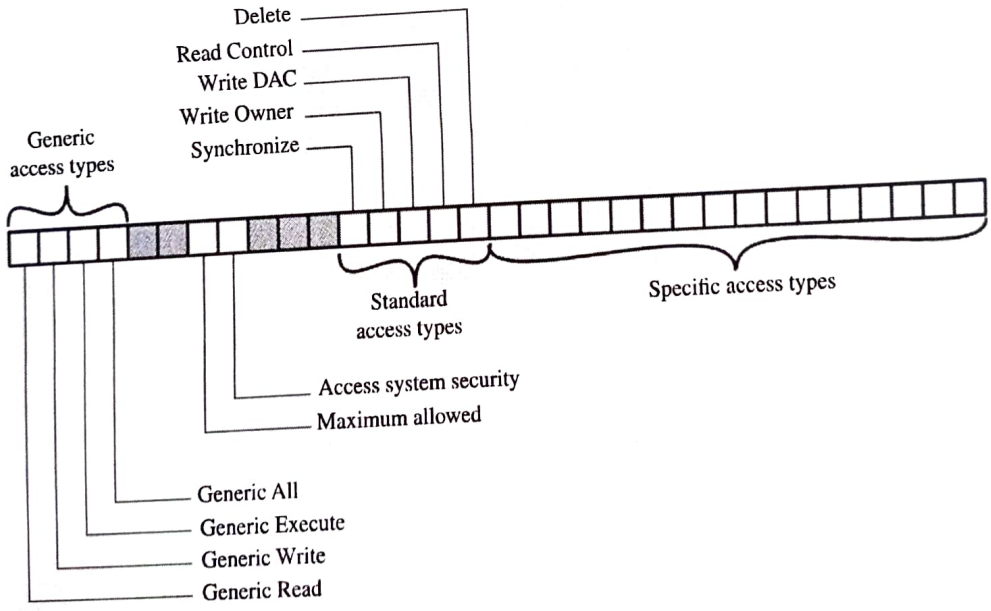


EXHIBIT 24.7 Windows 2000 Access Mask

The most significant 16 bits of the mask contains bits that apply to all types of objects. Five of these are referred to as standard access types:

1. **Synchronize**. Gives permission to synchronize execution with some event associated with this object. In particular, this object can be used in a wait function.
2. **Write_owner**. Allows a program to modify the owner of the object. This is useful because the owner of an object always can change the protection on the object. (The owner may not be denied Write DAC access.)
3. **Write_DAC**. Allows the application to modify the DACL and hence the protection on this object.
4. **Read_control**. Allows the application to query the owner and DACL fields of the security descriptor of this object.
5. **Delete**. Allows the application to delete this object.

The high-order half of the access mask also contains the four generic access types. These bits provide a convenient way to set specific access types in a number of different object types. For example, suppose an application wishes to create several types of objects and ensure that users have read access to the objects, even though read has a somewhat different meaning for each object type. To protect each object of each type without the generic access bits, the application would have to construct a different ACE for each type of object and be careful to pass the correct ACE when creating each object. It is more convenient to create a single ACE that expresses the generic concept *allow read*; simply apply this ACE to each object that is created and have the right thing happen. That is the purpose of the generic access bits, which are:

- **Generic_all**. Allow all access.
- **Generic_execute**. Allow execution if executable.
- **Generic_write**. Allow write access.
- **Generic_read**. Allow read-only access.

The generic bits also affect the standard access types. For example, for a file object, the Generic_Read bit maps to the standard bits Read_Control and Synchronize and to the object-specific bits File_Read_Data, File_Read_Attributes, and File_Read_EA. Placing an ACE on a file object that grants some SID Generic_Read grants those five access rights as if they had been specified individually in the access mask.

The remaining two bits in the access mask have special meanings. The Access_System_Security bit allows modifying audit and alarm control for this object. However, not only must this bit be set in the ACE for a SID, but the access token for the process with that SID must have the corresponding privilege enabled.

Finally, the Maximum_Allowed bit is not really an access bit but a bit that modifies W2K's algorithm for scanning the DACL for this SID. Normally, W2K will scan through the DACL until it reaches an ACE that specifically grants (bit set) or denies (bit not set) the access requested by the requesting process or until it reaches the end of the DACL, in which latter case access is denied. The Maximum_Allowed bit allows the object's owner to define a set of access rights that is the maximum that will be allowed to a given user. With this in mind, suppose that an application does not know all of the

operations that it is going to be asked to perform on an object during a session. There are three options for requesting access:

1. Attempt to open the object for all possible accesses. The disadvantage of this approach is that the access may be denied even though the application may have all of the access rights actually required for this session.
2. Only open the object when a specific access is requested, and open a new handle to the object for each different type of request. This is generally the preferred method because it will not unnecessarily deny access, nor will it allow more access than necessary. However, it imposes additional overhead.
3. Attempt to open the object for as much access as the object will allow this SID. The advantage is that the user will not be artificially denied access, but the application may have more access than it needs. This latter situation may mask bugs in the application.

An important feature of W2K security is that applications can make use of the W2K security framework for user-defined objects. For example, a database server might create its own security descriptors and attach them to portions of a database. In addition to normal read/write access constraints, the server could secure database-specific operations, such as scrolling within a result set or performing a join. It would be the server's responsibility to define the meaning of special rights and perform access checks. But the checks would occur in a standard context, using systemwide user/group accounts and audit logs. The extensible security model should prove useful to implementers of foreign files systems.

24.7 FURTHER READING

Boebert, W., R. Kain, and W. Young. "Secure Computing: The Secure Ada Target Approach." *Scientific Honeyweller* (July 1985). Reprinted in M. Abrams and H. Podell, *Computer and Network Security*. IEEE Computer Society Press, 1987.

Bransted, D. (ed.). *Computer Security and the Data Encryption Standard*. National Bureau of Standards, Special Publication No. 500-27, February 1978.

Denning, P., and R. Brown. "Operating Systems." *Scientific American* 251 (September 1984): 94–106.

Gasser, M. *Building a Secure Computer System*. New York: Van Nostrand Reinhold, 1988.

Gollmann, D. *Computer Security*, 3rd edition. Hoboken, NJ: Wiley & Sons, 2011.

Patterson, D. A., and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*, 4th ed. Morgan Kaufmann, 2011.

Pfleeger, C. P., and S. L. Pfleeger. *Security in Computing*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Saltzer, J., and M. Schroeder. "The Protection of Information in Computer Systems." *Proceedings of the IEEE* (September 1975).

Sandhu, R., and P. Samarati, "Access Control: Principles and Practice." *IEEE Communications* (September 1994).

Singhal, M., and N. Shivaratri. *Advanced Concepts in Operating Systems*. New York: McGraw-Hill, 1994.

Sinha, P. K. *Distributed Operating Systems: Concepts and Design*. Hoboken, NJ: Wiley-IEEE Press, 1996.

Stallings, W. *Cryptography and Network Security: Principles and Practice*, 5th ed. Upper Saddle River, NJ: Prentice-Hall, 2010.

Stallings, W. *Operating Systems: Internals and Design Principles*, 7th ed. Upper Saddle River, NJ: Prentice-Hall, 2011.

Viega, J., and J. Voas. "The Pros and Cons of Unix and Windows Security Policies." *IT Professional* 2, no. 5 (September/October 2000): 40–47.

24.8 NOTES

1. P. Denning and R. Brown, "Operating Systems," *Scientific American* (September 1984).
2. J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE* (September 1975).
3. C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*, 4th ed. (Prentice-Hall PTR, 2006).