———. 1990. The Trojan horse virus and other crimoids. In Peter J. Denning, ed., *Computers Under Attack: Intruders, Worms, and Viruses*, 544–554, New York: ACM Press.
Polk, W. Timothy, and Lawrence E. Bassham, III. 1992. *A Guide to the Selection of Anti-Virus Tools and Techniques*. NIST Special Publication 800-5. Gaithersburg, Md.: National Institute of Standards and Technology.
Reynolds, Joyce K. 1991. The helminthiasis of the Internet. *Computer Networks and ISDN Systems* 22: 347–361.
Rochlis, Jon A., and Mark W. Eichin. 1989. With microscope and tweezers: The worm from MIT's perspective. *Communications of the ACM* 32(6): 689–698.
Seeley, Donn. 1989. Password cracking: A game of wits. *Communications of the ACM* 32(6): 700–703.
Shoch, John F., and Jon A. Hupp. 1990. The "worm" programs—Early experience with a distributed computation. In Peter J. Denning, ed., *Computers Under Attack: Intruders, Worms, and Viruses*, 264–281. New York: ACM Press. Reprinted from *Communications of the ACM* 25(3): 172–180, 1982.
Smith, Kirk. 1988. Tales of the damned. *UNIX Review*, February: 45–50.
Spafford, Eugene H. 1989. Crisis and aftermath. *Communications of the ACM* 32(6): 678–687.
Spafford, Eugene H., Kathleen A. Heaphy, and David J. Ferbrache. 1990. A computer virus primer. In Peter J. Denning, ed., *Computers Under Attack: Intruders, Worms, and Viruses*, 316–355. New York: ACM Press.
Stoll, Clifford. 1988. Stalking the Wily Hacker. *Communications of the ACM* 31(5): 484–497.
———. 1989. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York: Doubleday.
Thompson, Ken. 1984. Reflections on trusting trust. *Communications of the ACM* 27(8): 761–763.
van Eck, Wim. 1985. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security* 4: 269–286.
VIRUS-L. 1992. Frequently asked questions on VIRUS-L/comp.virus. *VIRUS-L Forum*, November 1992.
White, Steve R., David M. Chess, and Chengi Jimmy Kuo. 1990. An overview of computer viruses and how to cope with them. *Computer Security Journal* V(2): 37–55.

Chapter

4

Policies and Models

Policy defines security for a computing system; a system is secure if it lives up to its security policy. The policy specifies what security properties the system must provide. Similarly, policy defines computer security for an organization, specifying both system properties and the security responsibilities of people.

The needs of the real world dictate security policy. A *real-world security policy* is the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes resources to achieve its security objectives. Those objectives come from the goals and environment of the organization. Consider, for example, a hospital whose policy is to release records to a patient who signs the right forms. In this example, the resource is information, on paper. The resource could also be a physical object or an action. For example, a radiation dose can be given only if two operators verify the radiation machine's settings. Even if an organization has no explicit security policy, policy assumptions guide its actions.

A *computer security policy* must faithfully represent real-world policy and must interpret that policy for resources that have a computing base, resources like databases and transactions. It must also consider threats to computers, specifying which threats the organization chooses to guard against, and how. Some threats (viruses, for example) bear little relation to real-world policy elements. So computer security policy has two parts: (1) an interpretation and partial automation of the real-world policy—sometimes called the *automated security policy*—and (2) policy about computer security in general (see Fig. 4.1). For example (1) security software enforces rules about who can initiate or approve a banking transaction, and (2) a bank requires

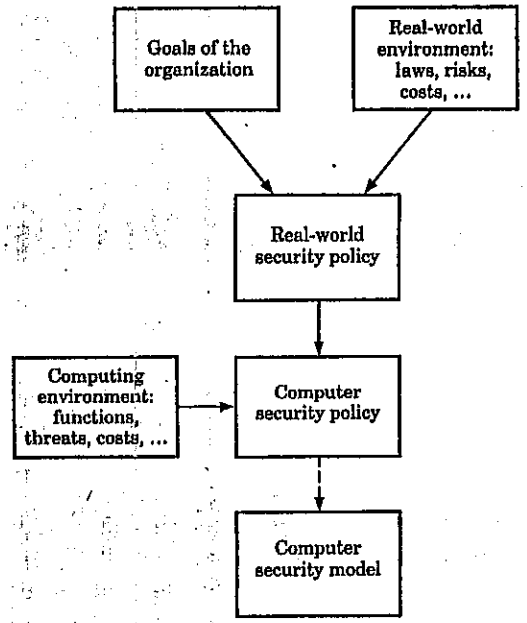

Figure 4.1 Sources of computer security policy.

employees to change their passwords every 30 days. An organization may have different policies for its different applications and systems.

A computer security policy is expressed in a language such as Spanish or English or Japanese. Natural language can be vague or ambiguous, especially if the policy is complex, or if different rules of the policy interact with each other. Most of the time we live with ambiguity, designating who is to interpret the policy when interpretation is needed. Sometimes, however, policy is expressed in a model. A *computer security model* restates the policy in more formal or mathematical terms, allowing its implications to be more fully understood. One purpose of a computer security policy is to guide the design of the security aspects of computing systems. The designers of a system need a clear statement of what policy the system is to carry out. A security model fills that need, defining what services will be provided and, often, what mechanisms will implement them. The model allows designers to get a handle on a complex enterprise.

Of the three computer security properties—confidentiality, integrity, and availability—confidentiality has received by far the most attention. The policies and models for integrity are less developed, and models for availability are in their infancy. This situation resulted in part from the Trusted Computer System Evaluation Criteria (TCSEC) emphasis on confidentiality. All along, however, integrity

has been at least as important for both commercial and military needs. For many applications it is clearly more important—financial systems or medical equipment, for example. As more and more critical devices and activities incorporate computing (communications systems, air traffic control, missile launching, radiation treatment) we depend more and more on the integrity of that computing and its data.

This chapter begins by considering security policy apart from computing, then shows how real-world policy needs carry over into the computing world, where additional needs arise. The idea of security models is introduced. Three major categories of models are considered—access control, information flow, and integrity—along with work on denial of service. The chapter closes with a few words on how computer security policy is applied in building and managing systems.

Real-World Security Policy

Security needs have a long history, with computers entering the picture only late in that history. Everyone is familiar with the need to protect secrets—personal, business, or military. For military secrets, this need led to the encoding of messages so that the enemy could not read them. Almost everyone is familiar with policies designed to control fraud and error in the business world. For example, a large cashier's check must be signed by two bank employees, including a manager. This rule is designed to prevent one person alone from committing fraud and to ensure supervisory control.

Principles underlying real-world security policies

Real-world security policy has been defined as the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes resources to achieve its security objectives. Although the laws, rules, and practices reflect that organization's goals and situation, they also reflect principles that apply broadly. These principles underlie the management controls that are applied in nearly all business practice to reduce the risk from fraud and error. (Many of these real-world security principles were introduced in Chap. 1.)

Individual accountability. Individual persons are held answerable for their actions. The principle implies that people are uniquely and accurately identified and that records of their actions are kept and reviewed.

Authorization. Explicit rules are needed about who can use what resources, and in what ways. For example, only authorized technicians can operate SafeCare's imaging equipment. Rules also are needed about who can authorize the use of resources and how the authorization can be delegated. At SafeCare, the head of the radiology laboratory authorizes access to the equipment and can delegate that authority temporarily.

Least privilege. People should be authorized only for the resources they need to do their jobs. Privileges should be "least" in time as well as scope; they should persist only as long as they are needed. This aspect of least privilege is sometimes called *timely revocation of trust.*

Separation of duty. Functions should be divided between people, so that no one person can commit fraud undetected. For example, two bank officials must sign large cashier's checks. Applying this principle may involve "split knowledge"—no one person knows everything. Separation of duty works best when the different people have different roles and viewpoints.

Auditing. Work and its results must be monitored both while the work is being done and after it is done. Information must be checked for internal consistency and for consistency with other criteria. For example, the linen store's inventory record must be consistent with its records of inventory received and sold, and also with the sheets and towels on the shelves. An *audit trail* must be kept—a record of all actions—that supports a later reconstruction of who did what. The audit trail supports the principle of individual accountability.

Redundancy. The principle of redundancy affects both work and information. Important steps are done twice or more (sometimes by different people—separation of duty), and the results are checked for consistency. Multiple copies are kept of important records, and the copies are often stored in different places. There are backups for critical resources like electrical power.

Risk reduction. It is rarely practical or even possible to enforce security policy absolutely. The strategy must be to reduce risk to an acceptable level, keeping the cost of enforcement proportional to the risks.

Roles in real-world security policy

Security policy involves roles that recur in many situations, and also application-specific roles. For paper documents the following generic roles apply:

- *Originator.* The person who issues a document, often the author, or the manager of the responsible unit.
- *Authorizer.* The person with control over access to the document. The authorizer may or may not be the originator.
- *Custodian.* The person who physically keeps the document and carries out the authorizer's intentions about access to it.
- *User.* The person who reads or alters the document.

If the resource is not a document but a commercial transaction, other roles come into play, including:

- *Creator.* The person who designs the transaction and writes the rules about its steps.
- *Customer.* The person on whose behalf the transaction is carried out.
- *Executor.* The person who actually carries out the transaction—prepares a check or debits a bank account.
- *Supervisor.* The person who authorizes the transaction.
- *Auditor.* The person who checks the actions, results, and controls.

Some roles make sense only for a specific application or transaction. For example, the process of obtaining a prescription drug involves the roles of physician, pharmacist, and patient. Each has certain rights in the process. The physician has the right to prescribe. The pharmacist can dispense, but not prescribe. The patient has the right to receive a copy of the prescription. Security policies refer to both general roles and application-specific roles.

Real-world policies for confidentiality

Of the three computer security properties—confidentiality, integrity, and availability—the first two clearly reflect real-world security properties. We first describe traditional policies for confidentiality, especially of documents.

Some highly sensitive documents are kept secret from everyone, except as explicitly approved by the originator. More typically, documents are grouped or *classified,* according to the type of confidentiality that is needed. Similarly, people are given *clearances.* The policy then can be stated as a relation between the classification of the document and the clearance of the person. In the confidentiality policy of the U.S. military, for example, both documents and people have levels of classification or clearance: Top Secret, Secret,

Confidential, or Unclassified. Access is allowed only if the person's clearance level is at least as high as the document's classification level. A person cleared for Top Secret can see documents at any level. A person cleared for Secret can see Secret, Confidential, and Unclassified documents. Many commercial organizations also classify information according to its level of sensitivity, usually with three or more levels.

Some confidentiality policies cannot be handled with levels alone, however. According to the least-privilege principle, Lieutenant Ray, who works in Army missile procurement, has no need to know about enemy troop number estimates, no matter what her clearance. To meet the least-privilege objective, organizations describe people in terms of *categories* that are relevant to their work, and documents in terms of the categories they cover. The policy for access then states that the categories of the person have to include all the categories of the document. (The military policy is described more precisely later in this chapter.) The categories of a commercial organization may describe the functional areas of its business, such as Engineering and Manufacturing. Also used are categories such as Business Confidential (for sensitive business results), Personal and Confidential (for personnel information), and Proprietary (for trade secrets).

With all this emphasis on confidentiality, it is important to remember that some policy limits confidentiality. The Freedom of Information Act mandates the availability of information, with constraints for confidentiality. A policy of *maximized sharing* would be appropriate for a research database or for a library.

Regardless of its other policies, an organization usually explicitly gives people access to resources, or explicitly denies people such access. Lieutenant Ray has been authorized to see a report summarizing the year's procurement activities for ground-to-air missiles. Many of her colleagues cannot see that report. Another rule denies Lt. Ray access to a report showing vendors' production costs.

Organizations that are concerned about confidentiality devote much effort to classifying documents, labeling them with their classifications, and giving them proper custodial care. For example, BioWidget numbers the copies of its highly sensitive documents. It keeps records about who receives each copy, and it prohibits making new copies. A sensitive document must be kept in a reinforced, fireproof safe, and the location of the document must be audited every three months. Classification is especially important for trade secrets. Not only does it help keep the information confidential, but it is important for obtaining legal recourse if the information does get out. By enforcing its classification policies, BioWidget demonstrates that certain information is proprietary and that it tried to protect it.

Policies and controls for integrity

Management control policies are directed mainly at integrity rather than confidentiality. Many integrity policies are so familiar that they are scarcely noticed. A few of the most important and most common are described here, with a caution to the reader that the definitions and terminology for integrity are very much in flux.

The policy of *authorized actions* states simply that people can take only those actions they are authorized for. (Of course there are realms in any organization where people have implied authorization to act. The policy of authorized actions applies for important resources, such as money in a restaurant cash register. Anyone may retrieve a napkin from the floor.) Another common policy is *supervisory control*—certain actions must be approved by a supervisor. The manager in charge must approve the cashier's going off duty early.

The principle of separation of duty is reflected in many common policies. Dividing a task into parts that are performed by different people is a time-honored way of preventing fraud by one person acting alone. According to the related policy *rotation of duty*, a task should not always be carried out by the same person. If it were, the person would have a better shot at committing fraud without being discovered. Many organizations rotate tasks among employees and insist that employees take their vacations. The policy of *N-person control* requires people to cooperate to carry out an action. To launch a missile, for example, two physically separated controls must be operated simultaneously. This is an example of a *dual-custody* rule.

The policy of *operation sequencing* requires the steps of some task to be carried out in a specific order. Often this policy is combined with separation of duty and *N*-person control, so that a different person or group carries out each step in the sequence. For driver's license renewal, for example, the old license, new forms, and fee go to one clerk, who issues a written exam. A second person corrects the exam. If a driving test is needed, a third person gives it. A fourth person checks vision and takes a picture. Finally, the license is mailed from a different place to the address of record.

The policy of *constrained change* requires data to be changed only in prescribed and structured ways. Forms, such as purchase orders, are a common way of constraining change. In double-entry bookkeeping, every transaction involves at least two changes, and debits must equal credits for every transaction. If a business buys a computer, the asset account is credited and the cash account is debited by the same

amount. A transaction that is constrained and structured has been called a *well-formed transaction.*

Closely related to constrained change is *consistency.* Data must be internally consistent (as in double-entry bookkeeping) and also consistent with external reality. An example of inconsistency is given by the U.S. employment estimates for September 1992. The national figure was 400,000 larger than the sum of the 50 state estimates.

Not all data is as highly structured as accounting data. A written document, for example, cannot be constrained in analogous ways; the best assurance of integrity may be knowing who wrote it. The policy of *attribution of change* aims at attesting to the authorship or "pedigree" of some data. Since various people may change a document, this policy might be enforced by keeping a log showing the description, author, and date of each change. For logs and other history-type documents, a *no-change* policy applies. Once an entry is made, it is never changed or erased.

Real-world policy examples

A few specific policies are worth keeping in mind as test cases for the models described later in this chapter. Since real requirements led to these policies, a good model should handle them.

Chinese Wall policy. Consider an analyst who works for an investment banking firm in the United Kingdom. To do her work for a client—a financial software company—she must obtain insider knowledge about the company. By law she may not advise other competitive software companies but she may advise, for example, a copier manufacturer. According to a model of this policy, information is grouped into "conflict of interest classes," and a person is allowed at most one set of information in each class. Once the analyst has insider information about financial software, a wall is built around all information of that class. Anything else inside the wall is forbidden to her; anything outside is allowed. This is called the *Chinese Wall* policy.

Originator controlled. Another test case is the *originator controlled* (ORCON) policy of the DoD, which is outside the DoD classification and category scheme. A document marked *ORCON* may be released only to organizations on a list specified by the originator of the document. Any other release must be explicitly granted by the originator.

Core policy elements

Although security policy often is tailored to applications, some elements are common to many application-specific policies. A key element of many policies is the concept of *role.* Real-world security policy is often role-based; along with a role come rights to use resources in designated ways. People have these rights because they have to do a job. Once they leave the job, they lose the rights.

One study of tactical military applications (Sterne et al. 1991) identified the same core elements of policy in different applications. These elements included: protection of information from unauthorized disclosure and modification, role-based access control, role exclusion rules (a person who plays one role cannot play another—as in separation of duty), delegation of authority, orders that may not be repudiated, two-person and *N*-person controls, operation sequencing, identification and authentication, and auditing.

Security Policy in the Computing World

The policy principles of the real world continue to hold for the computing world, but the scope of security policy becomes much broader. First, the real-world security policy must be automated faithfully. This means that it must be specified unambiguously. An ambiguous or vague policy may work when it is interpreted by human beings, but it does not work for automated policy. Second, policy choices must be made about the computing situation itself, such as physical security of the computing equipment, or how users identify themselves to the computing system. These choices concern what threats are guarded against and what safeguards are used. Security policy choices must be made when hardware, operating systems, and applications are designed and built, and when they are selected for use. In the computing world, as in the real world, many of the same policies apply to both confidentiality and integrity. More than in the real world, however, confidentiality and integrity may conflict.

Although a few policy needs have been studied intensively, security policy in general is not highly developed. The National Research Council report concluded that "the lack of a clear articulation of security policy for general computing is a major impediment to improved security in computer systems" (National Research Council 1991: 51). The report recommended developing a set of generally accepted system security principles, analogous to those developed over the years by the accounting profession. This means codifying and promulgating principles (about access control, user identification, and auditing, for example) that are applicable to nearly all systems.

A computer security policy is specified in a natural-language document, as clearly and unambiguously as possible. The document specifies what security properties are to be provided, and (to a lesser extent) how. Some policies are abstract and apply to many different

amount. A transaction that is constrained and structured has been called a *well-formed transaction.*

Closely related to constrained change is *consistency.* Data must be internally consistent (as in double-entry bookkeeping) and also consistent with external reality. An example of inconsistency is given by the U.S. employment estimates for September 1992. The national figure was 400,000 larger than the sum of the 50 state estimates.

Not all data is as highly structured as accounting data. A written document, for example, cannot be constrained in analogous ways; the best assurance of integrity may be knowing who wrote it. The policy of *attribution of change* aims at attesting to the authorship or "pedigree" of some data. Since various people may change a document, this policy might be enforced by keeping a log showing the description, author, and date of each change. For logs and other history-type documents, a *no-change* policy applies. Once an entry is made, it is never changed or erased.

Real-world policy examples

A few specific policies are worth keeping in mind as test cases for the models described later in this chapter. Since real requirements led to these policies, a good model should handle them.

Chinese Wall policy. Consider an analyst who works for an investment banking firm in the United Kingdom. To do her work for a client—a financial software company—she must obtain insider knowledge about the company. By law she may not advise other competitive software companies but she may advise, for example, a copier manufacturer. According to a model of this policy, information is grouped into "conflict of interest classes," and a person is allowed at most one set of information in each class. Once the analyst has insider information about financial software, a wall is built around all information of that class. Anything else inside the wall is forbidden to her; anything outside is allowed. This is called the *Chinese Wall* policy.

Originator controlled. Another test case is the *originator controlled* (ORCON) policy of the DoD, which is outside the DoD classification and category scheme. A document marked *ORCON* may be released only to organizations on a list specified by the originator of the document. Any other release must be explicitly granted by the originator.

Core policy elements

Although security policy often is tailored to applications, some elements are common to many application-specific policies. A key element of many policies is the concept of *role.* Real-world security policy is often role-based; along with a role come rights to use resources in designated ways. People have these rights because they have to do a job. Once they leave the job, they lose the rights.

One study of tactical military applications (Sterne et al. 1991) identified the same core elements of policy in different applications. These elements included: protection of information from unauthorized disclosure and modification, role-based access control, role exclusion rules (a person who plays one role cannot play another—as in separation of duty), delegation of authority, orders that may not be repudiated, two-person and *N*-person controls, operation sequencing, identification and authentication, and auditing.

Security Policy in the Computing World

The policy principles of the real world continue to hold for the computing world, but the scope of security policy becomes much broader. First, the real-world security policy must be automated faithfully. This means that it must be specified unambiguously. An ambiguous or vague policy may work when it is interpreted by human beings, but it does not work for automated policy. Second, policy choices must be made about the computing situation itself, such as physical security of the computing equipment, or how users identify themselves to the computing system. These choices concern what threats are guarded against and what safeguards are used. Security policy choices must be made when hardware, operating systems, and applications are designed and built, and when they are selected for use. In the computing world, as in the real world, many of the same policies apply to both confidentiality and integrity. More than in the real world, however, confidentiality and integrity may conflict.

Although a few policy needs have been studied intensively, security policy in general is not highly developed. The National Research Council report concluded that "the lack of a clear articulation of security policy for general computing is a major impediment to improved security in computer systems" (National Research Council 1991: 51). The report recommended developing a set of generally accepted system security principles, analogous to those developed over the years by the accounting profession. This means codifying and promulgating principles (about access control, user identification, and auditing, for example) that are applicable to nearly all systems.

A computer security policy is specified in a natural-language document, as clearly and unambiguously as possible. The document specifies what security properties are to be provided, and (to a lesser extent) how. Some policies are abstract and apply to many different

computing environments—for example, the DoD policy of classification and categories. Others pertain to a specific organization or site. Still others pertain to a computing system or an application.

Jonathan Moffett and Morris Sloman (1988) have characterized security policy for an organization as general or specific. Examples of general policy include

- Access rules should refer to positions or roles, not people.
- An authorizer should be able to grant rights only to positions in his or her organizational domain.

Examples of specific policies would be

- Security administration for the marketing department is delegated to the administrative manager of that department.
- A wire transfer of funds may be executed by any teller assigned to the window authorized for wire transfers. It must be approved by the assistant manager or (if the amount is over $100,000) by the assistant manager and the branch manager.

Security Models

Models serve three purposes in computer security. The first purpose is to provide a framework that aids understanding of concepts. Models designed for this purpose often use diagrams, charts, or analogies. One example is the "threat-vulnerability-safeguard" model. Another example is the original access matrix model of Butler Lampson (1971), represented in Fig. 4.2.

The second purpose is to provide an unambiguous, often formal, representation of a general security policy. One example is the Bell-LaPadula model, which formalizes the military security policy of classification and clearance. Formal methods of reasoning can be used with that model to reveal the detailed implications of the policy. A model like Bell-LaPadula is an *abstract model,* in that it deals with abstract entities like subjects and objects. A *concrete model* translates these abstract entities into the entities of a real system, such as processes and files. The abstract and concrete models also can be seen as one model at different levels of abstraction; there can be intermediate levels between the most abstract and most concrete.

The third purpose for a model is to express the policy enforced by a specific computing system. A precise expression of policy is needed to guide the design of a system; a formal expression opens the way for proving mathematically that a design does enforce the policy. The

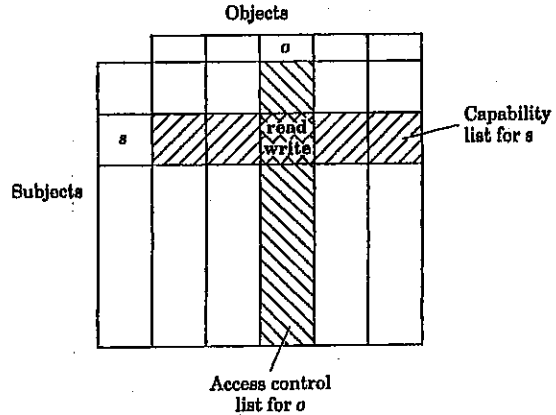

Figure 4.2 Access matrix.

National Computer Security Center (NCSC) definition of a "security policy model" is "a formal presentation of the security policy enforced by the system. It must identify the set of rules and practices that regulate how a system manages, protects, and distributes sensitive information" (NCSC 1988: 42).

The three purposes for models overlap. Conceptual frameworks may evolve with time into abstract formal models. This did happen for the access matrix model. Abstract formal models such as Bell-LaPadula provide a base for concrete models of systems.

History of computer security models

The earliest modeling work, beginning in the 1960s, was stimulated by the development of time-sharing systems, which had to face the problem of sharing information securely. The early systems were developed and used at universities, so the models reflected the university environment. The 1970s and 1980s saw a great expansion of research and a shift to work reflecting military needs. In the late 1980s another shift occurred—toward incorporating the needs of the commercial world and the expertise represented by security controls in business. The 1990s have seen a realization that many different security policies must be modeled (not just one), and that often a single system must be able to support multiple policies. There has been soul-searching about the role that models and formal methods have played—whether the right policies have been modeled, and whether the models have helped or hindered.

Disciplines used in models

Formal models draw on methods of mathematical logic and several fields of applied mathematics. These include information theory, automata theory, complexity theory, and statistics. (See the bibliographic notes for sources on these disciplines.) A brief summary of information theory appears later in the chapter. The descriptions in this chapter include some formal notation, sometimes to convey the essence or flavor of the model, sometimes because the original expression is simpler and clearer than a paraphrase. Figure 4.3 summarizes the notation used. The reader who prefers to skip the notation will always find a paraphrase in words.

Criteria for good models

Assuming that the security policy is appropriate, the first demand on a model is to faithfully represent that policy. There is no way to prove mathematically this faithfulness, since the policy statement is informal. Rather, the model builders must clearly explain in words just how the model corresponds to the policy and must justify the validity of the correspondences.

Another criterion is expressive power. Can the model express all the wrinkles and twists of the policy? For example, the policy may be that a person who has **write** access to a file may grant **write** or **read** access to another person. How can the model express that policy?

A good model helps understanding by clarifying concepts and expressing them forcefully and precisely, focusing attention on the

$X = \{a, b, c, d\}$	X is the set whose elements are a, b, c, and d
$x \in X$	x belongs to the set X
$X \subset Y$	X is a subset of Y; all elements of X belong to Y
$X \cup Y$	All elements of X with all elements of Y
(x, y)	An ordered pair
$X \times Y$	The set of all possible ordered pairs whose first element is a member of set X and whose second element is a member of set Y
$F: X \rightarrow Y$	Function f with domain X and range Y
$y = f(x)$	Function f applied to $x \in X$
$A \Rightarrow B$	A implies B
A iff B	A is true if and only if B is true
$\sum_{i=1}^{n} x$	$x_1 + x_2 + \cdots + x_n$

Figure 4.3 Notation used in describing models.

essentials. A formal model can lead to better understanding of the general problem by way of new truths derived from the axioms of the model. For example, a formal description of the access matrix model led to new understanding of its limitations.

A security model should support decisions about *safety*. The question is whether there is some state of the modeled system in which a specific security property is not preserved. The model should support answering this question, and the decision must not be too complex computationally. Unfortunately, the models that are strong on expressive power tend to be weak in regard to safety. There is a tension between safety and precision. If the model is restricted in order to make safety decidable, it may not represent the security policy precisely enough.

A security model should provide a good foundation for building systems. A system based on the model must be reasonable to build and must perform adequately. A convincing argument must be made that the system is a true implementation of the model. If the model is a formal one, and if the system design is also expressed formally, some steps of the argument can be proved mathematically. This formal *verification* enhances confidence in the security of a system; the TCSEC criteria require formal verification for the highest level of certification. So it is an advantage for the model to be expressed in precise mathematical terms and in a form that can map clearly to design specifications and eventually to code. Since verification is exceedingly difficult, it is a further advantage to express the model in a form that allows proofs to be carried out by computer programs. (This process of going from formal model to implemented system is discussed more fully in Chap. 6.)

It should be possible to model complex systems in parts and then put the parts together. Each part is simpler and more understandable than the whole. It is more likely that the formal specifications and verification will be correct—since they are simpler. Also, parts of the model can be reused when the specification changes as a system evolves. For modeling in parts, formal models of security properties must be *composable*. If two systems are shown individually to have some composable property, and the systems are then hooked together, the resulting system also has that property.

In summary, then, a security model should:

- Validly and precisely represent security policy
- Aid understanding through focused and exact expression and through proofs of properties
- Support safety analysis

- Support system building and system verification
- Allow systems to be modeled in parts that are then put together.

Access Control Models

Access control is the process of ensuring that all access to resources is authorized access. Access control enforces the fundamental security principle of authorization. It supports both confidentiality and integrity.

The access matrix model

The *access matrix model* of access control is simple and intuitive, and it can express many protection policies. The access matrix relates subjects, objects, and rights. *Objects* in the model represent the resources to be controlled. Although objects are abstract entities, they can be thought of as representing files or areas of memory. (Access matrix models were developed in the early 1970s for operating systems, motivated by the protection problems that arose in multiuser systems.) *Subjects* are the active entities of the model. A subject can be thought of as a user or as a process executing on behalf of a user.

The *access matrix* has a row for each subject and a column for each object. In the access matrix *AM*, cell *AM[s,o]* specifies the *rights* that subject *s* has to object *o*. A right represents a type of access to the object, such as **read** or **execute**. In Fig. 4.2, for example, subject *s* has **read** and **write** rights for object *o*. Types of rights are sometimes called *access modes* or *access types*. Subjects may also appear as objects, as when a program subject has **call** rights and also is the object for other programs' **call** rights. A row of the access matrix corresponds to a *capability list*—the list of all the rights of a subject. A column corresponds to an *access control list*—the list of all the rights held by subjects to some object.

The access matrix model can represent many access control policies that support both confidentiality (through controlling reading of objects) and integrity (through controlling modifications to objects and invocation of programs). It supports what is known as *discretionary access control* (DAC), since the access matrix can be changed at the discretion of authorizers. Other models—discussed later—support *mandatory access control* (MAC), which constrains what authorizers can do.

For a useful system, the access matrix is not static. Subjects and objects come and go, and so do rights. This means that the model must include operations to change the access matrix—to create or destroy subjects and objects, and to enter or delete rights. It is crucial to understand what a change to the access matrix implies and what rights subjects potentially can gain.

HRU model and safety. Harrison, Ruzzo, and Ullman (1976) formalized the access matrix model and proved that it cannot provide this crucial understanding, in the general case. In other words, the model is weak in relation to safety.

The *HRU* model defines a *protection system* as consisting of a set of *generic rights R* and a set of *commands C*. A command is like a very simple procedure (with parameters) that has a condition portion and a main portion. The condition portion tests for the presence of certain rights in the access matrix. If the tests succeed, the main portion is executed, carrying out a series of *primitive operations* that change the *protection configuration* (often called the *protection state*). The configuration is a triple (*S,O,AM*), where *S* is the current set of subjects, *O* the current set of objects, and *AM* the access matrix. The primitive operations create and destroy subjects and objects, enter rights in the access matrix, and delete rights. For example, **create object** is a primitive operation. Each operation is defined as a specific change to the access matrix.

Consider, for example, a system where a process may create a file and where the process then becomes the owner of that file and may confer rights to other subjects. This protection system has commands CREATE and CONFER-READ, as follows:

```
command CREATE(process,file)
      create object (file)
      enter own into (process,file)
end
command CONFER-READ(owner,subject,file)
      if own in (owner,file)
      then enter read into (subject,file)
end
```

The HRU model handles many realistic protection systems, such as the original UNIX protection scheme.

Executing a command transforms the protection configuration into a new configuration. HRU notation is

- $Q \vdash_{\alpha(x_1,\ldots,x_k)} Q'$. That is, configuration Q yields Q' when command α is run with the parameters given.
- $Q \vdash_{\alpha} Q'$ if there exist parameters $x_1,\ldots,x_k$ such that $Q \vdash_{\alpha(x_1,\ldots,x_k)} Q'$. That is, Q' can be reached from Q by application of command α.
- $Q \vdash Q'$ if there exists any command α such that $Q \vdash_{\alpha} Q'$. Informally, Q' is reachable from Q.

- $Q \vdash^* Q'$ is also used; $\vdash^*$ represents zero or more applications of $\vdash$.

Returning now to safety, the question is whether a contemplated change to the access matrix can result (indirectly) in *leakage* of rights to unauthorized subjects. HRU defines leakage as follows: A command α leaks a right r from protection configuration Q if, when α is run on Q, it can execute a primitive operation that enters r into some cell of the access matrix that did not previously contain r. (Some subject now has right r on some object; previously the subject did not have that right.) "...the initial configuration Q_0 is *unsafe* for r (or *leaks* r) if there is a configuration Q and a command α such that (1) $Q_0 \vdash^* Q$ and (2) α leaks r from Q" (Harrison, Ruzzo, and Ullman 1976: 467). That is, Q_0 is unsafe for a right (**write**, for example) if we can reach from Q_0 another configuration Q that leaks the right. If a configuration is not unsafe, it is safe.

The important result of HRU is that it is undecidable whether a given configuration is safe for a given right. (The proof involves making a correspondence between (1) leakage of a right and (2) an arbitrary Turing machine entering a final state, which is known to be undecidable.) Although this result about the access matrix model is fundamental, it applies only to a general and unrestricted protection system. For a more restricted *monooperational* system (where each command is a single primitive operation), safety is decidable, but the decision procedure is so computationally complex that it is probably impractical.

The HRU results have led researchers to devise more restricted protection systems, in the hopes of finding useful ones with better safety properties. One such attempt is the *monotonic* system, where rights can only be given and not taken away. This may not sound very practical, but many practical systems can be reduced to monotonic systems for the purpose of safety analysis. Even monotonic systems have quite poor safety properties, however. *Take-Grant* models (described later in this chapter) have better safety properties. Entirely new frameworks for protection have been developed, as well.

In spite of the safety results, the unrestricted access matrix is probably the most widely used security model. On the whole it is successful. Leakage of rights is constrained by controls within the protection system (how the commands are defined) and outside the protection system (the good judgment of users).

Other features of the access matrix model. HRU is a bare-bones model designed to answer fundamental questions. Other features appear in other versions of the access matrix model.

Transfer of rights. In some systems subjects can receive rights that are transferable. That is, the subject may transfer the right to another subject. This transferability of a right is sometimes described as a *copy flag*, following the usage of Lampson. A copy flag attached to a right in $AM[s,o]$ means that subject s can copy that right to another cell in column o, choosing whether or not to copy the flag as well.

The reference monitor. Implied by the access matrix model is some mechanism that monitors all access attempts. The mechanism is sometimes viewed as associated with the object, more often as part of the general protection system. G. Scott Graham and Peter Denning (1972) used the term *monitor* for the mechanism that validates access to each object *type*. James Anderson (1972) introduced the term *reference monitor* for the general mechanism that ensures that each access is authorized by the access matrix (see Fig. 4.4).

Access request and decision. An *access request*, specified by the triple (s,o,r) is the event that the reference monitor mediates. Subject s requests access of type r to object o. The *access decision* may allow or deny the request, or modify it into a different request.

Access validation rules specify how the reference monitor decides the fate of a request—how it makes the access decision.

Authorization rules specify how the access matrix itself can be modified. (Authorization rules would be represented in HRU by the condition portions of the commands.)

Capabilities and the Take-Grant model

A row of the access matrix can be viewed as a capability list specifying all the rights of the subject associated with that row. The two main ways of implementing access control are access control lists

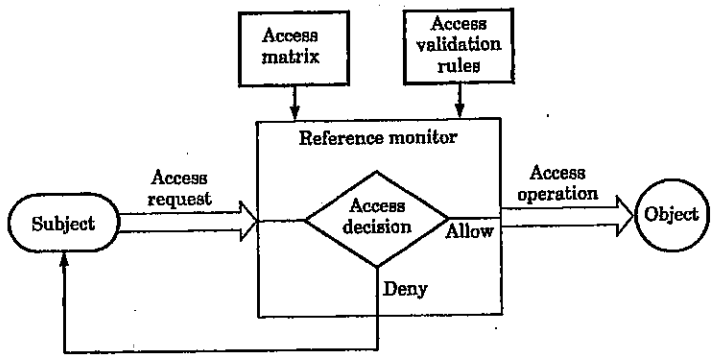

Figure 4.4 Access request mediated by the reference monitor.

(attached to objects and corresponding to columns of the access matrix) and capabilities. A *capability* is held by a subject and is defined as

(*object,rights,random*)

where the third component is a random number generated to prevent forgery. A capability is something like a ticket that the holder can present to obtain access to some object. Since capabilities cannot be forged, they can be passed around without intervention by any monitor. This property of capabilities makes for a great deal of flexibility in system design; operating systems and hardware architectures have been designed around capabilities. This same flexibility also makes it impossible to guarantee certain properties in a pure capability system. For example, a classic capability system cannot enforce the *-property.

Closely identified with capability systems are *Take-Grant* models, which represent the protection state by a directed graph. In Fig. 4.5, vertex b (solid) represents a subject and vertex c (open) represents an object. The directed edge from b to c (labeled α) represents a set of rights that subject b has for object c. The edge then represents a capability for c owned by b. The rights α must be a subset of a fixed set of rights R. For example, R = {**read, write, take, grant**}, or {**r, w, t, g**}. A vertex that represents either a subject or an object is shown as ⊗.

A Take-Grant model specifies a set of rules for transforming protection graphs. These rules control how rights can be passed. By varying the rules, one can define different Take-Grant models. One example model includes rules *Create, Remove, Grant,* and *Take,* where the first two govern how vertices are added and removed, and Grant and Take govern how one subject can grant rights to another or take rights from another. The Grant rule defines a new graph G' formed from graph G, as shown in Fig. 4.6. In graph G, there is an edge from x to y labeled γ, such that **g** is an element of γ (only **g** appears in the figure), and an edge from x to z labeled α, and β is a subset of α. The Grant rule adds a new edge (dashed) labeled β from y to z. That is, x grants y the ability to do β to z. This is allowed because x has **grant** right to y and because β is included in the rights that x has to z.

The Take rule is shown in Fig. 4.7, where in graph G again x is a subject, there is an edge from x to y labeled γ, such that **t** is an element of γ (only **t** appears in the figure), and an edge from y to z labeled α, and β is a subset of α. The Take rule adds an edge from x to z labeled β. That is, x takes from y the ability to do β to z.

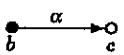

Figure 4.5 Simple protection graph.

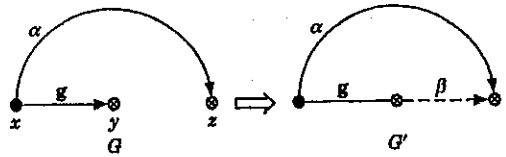

Figure 4.6 Grant rule of Take-Grant model.

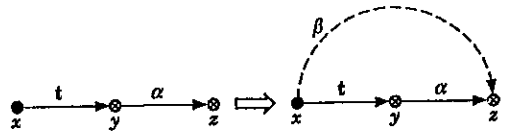

Figure 4.7 Take rule of Take-Grant model.

Compared to the HRU model (which can also be represented graphically) the Take-Grant model is more restrictive, because it has particular rules for transforming the protection graph. Since the model is more restrictive, safety decisions are possible.

Limitations of discretionary access control

One limitation of traditional DAC approaches is the safety problem. Even more serious is the vulnerability of DAC to Trojan horse attacks. A *Trojan horse* is an apparently useful program containing hidden functions that can exploit the rights of the subject, and so violate security policy. For example, the Trojan horse could leak confidential information to an accomplice or could destroy the user's files. Since a Trojan horse executes with the rights of the user who has unwittingly invoked it, DAC is powerless to protect against it. The next section describes a mandatory access control model intended to solve this problem.

The Bell-LaPadula model

The *Bell-LaPadula (BLP) model* formalizes the multilevel security policy. The model, developed by D. E. Bell and L. J. LaPadula (1976), is treated here as an access control model, since it is expressed in terms of access control. The goal of the policy, though, is information flow control. The BLP model has played a key role in the efforts to build systems whose security can be verified.

The multilevel security policy. The multilevel policy classifies information at one of four *sensitivity levels:* Unclassified, Confidential, Secret, or Top Secret—from low to high sensitivity. Information is also described in terms of *compartments,* which represent subject matter. A document might be designated Secret sensitivity level, with compartments Artillery and Nuclear. The *security level* or *access class* of the document is the combination of its sensitivity level and its set of compartments. A person receives *clearance* to some sensitivity level and to a set of compartments, so that both people and information have security levels (or access classes).

The policy (mandated by law) is that persons may access information only up to their clearance level. For example, a person cleared for Secret can access Secret, Confidential, and Unclassified, but not Top Secret. Further, access is allowed only if a person has clearance for all the compartments of the information. A person who has a classified document must exercise *discretion* about giving it to other properly cleared persons.

When a new document is created, it typically contains portions at various levels; the document is classified at the highest level of any of its parts.

Early computing systems separated the security levels, using a different computer or different period of time for processing each level. This scheme was inefficient and limited in function. An important goal, then, was to build *multilevel secure* systems that could work on different levels concurrently.

State machine models. BLP is a *state machine model,* as are many computer security models. A state machine model views a system as a triple (S, I, F), where S is a set of states, I is a set of possible inputs, and F is a *transition function* that takes the system from one state to another. There may also be other sets of entities, such as outputs, and other functions. A model of security also has axioms giving conditions for security. The axioms may restrict the system state, the state transitions, or the outputs. Proofs of security use an inductive method. They show that the initial state is secure and that all transitions from a secure state lead to a secure state.

Sets and properties. The BLP model includes the following sets:

S	*subjects*
O	*objects*
A	*access modes* such as **read (r)** and **write (w)**
L	*security levels*

A *secure state* is defined by three properties that are intended to express the multilevel policy: the *simple security (ss-) property,* the **-property* (pronounced star-property), and the *discretionary security property.* The ss-property expresses clearance-classification policy. The *-property represents the policy of no unauthorized flow of information from a higher level to a lower one. These two properties represent mandatory access control. The discretionary security property reflects the principle of authorization and is expressed in an access matrix. Only the mandatory properties are described here.

System, states, and state transitions. A system is a state machine with states V. Each state v is an element of the set of possible states $V = (B \times F)$, where:

B	is the set of all possible *current access sets.* The current access set b represents the accesses that are currently held—that have been requested and approved, and not released. An element of b is a triple (s, o, a)—subject, object, access mode.
F	is a subset of $L^S \times L^O$. Each element f is a pair of functions (f_S, f_O), where f_S gives the security level associated with each subject, and f_O gives the security level associated with each object. One security level f_1 *dominates* another f_2 if the classification of f_1 is greater than or equal to that of f_2 and the category set of f_1 includes the category set of f_2.

The inputs to the system are a set of *requests* R, and the outputs are *decisions* D. A *rule* is a function that associates a pair (request, state) with a pair (decision, state). An *action* of the system is described as (r, d, v^*, v). That is, a request r made in state v yields a decision d and moves the system from state v to state v^*. The term *request-response* is sometimes applied to this paradigm. When BLP is interpreted for the Multics operating system, the rules correspond to system operations, such as **get-read** (which alters the current access set), **give-access** (which alters the access matrix) and **change-object-security-level.** Some versions of BLP assume that objects do not change security level; this is called the *tranquility* principle. Rules of operation are important for guiding system design, but are often omitted from abstract models.

The security properties. A state satisfies the ss-property if and only if, for each element of the current access set b, the security level of the subject dominates the security level of the object. This property is also called *no read up.* (A formal statement appears below.)

The *-property is satisfied if and only if (1) for each **write** access in the current access set b, the level of the object equals the current

level of the subject, and (2) for each **read** access in b, the level of the subject dominates the level of the object. The *-property ensures that, if a subject has read access to one object and write access to another, then the level of the first is dominated by the level of the second. The *-property is sometimes called *no write down,* or the *confinement* property.

The *-property represents the policy that a subject cannot copy higher level information into a lower level object. The actual specification of the property is stronger than that, since the model deals with access to whole objects and does not delve into what information is being transferred.

Trusted subjects. A system that enforced the model as described so far would be unworkable. For example, the *-property would prohibit moving a paragraph about unclassified topics from a Confidential document to an Unclassified document. It would prevent the downgrading of information by a security officer. To address this problem, the model includes the concept of *trusted subjects,* who are allowed to violate the *-property but are trusted not to violate its intent (as in the examples). Set S', a subset of S, represents the untrusted subjects—those to whom the *-property applies.

Formal statements of properties. Formally, a state $v = (b, f)$ satisfies the ss-property if and only if:

$$s \in S \Rightarrow [(o \in b(s{:}\mathbf{r},\mathbf{w})) \Rightarrow (f_S(s) \text{ dominates } f_O(o))]$$

where $b(s{:}\ \mathbf{r},\mathbf{w})$ stands for $\{o{:}\ (s,o,\mathbf{r}) \in b \text{ or } (s,o,\mathbf{w}) \in b\}$. A state satisfies the *-property relative to S' if and only if

$$s \in S' \Rightarrow [(o \in b(s{:}\mathbf{w}) \Rightarrow (f_O(o) = f_S(s))]$$

and

$$[(o \in b(s{:}\mathbf{r})) \Rightarrow (f_S(s) \text{ dominates } f_O(o))].$$

Secure system. The simple security and *-properties define a secure *state.* The model must also define a secure *system.* A system is defined as all sequences of actions with some initial state that satisfy a relation W on the successive states. The relation W defines how the security properties are preserved. Some additional notation:

W	$W \subset R \times D \times V \times V$; subset of actions
X	Request sequences
Y	Decision sequences

T	$1,2,\ldots,t,\ldots$; t is an index to a state sequence
Z	State sequences; z_t is the tth state in the sequence z
z_0	Initial state of the system

The system $\Sigma(R,D,W,z_0) \subset X \times Y \times Z$ is defined as follows:

$(x,y,z) \in \Sigma(R,D,W,z_0)$ if and only if, for each t in T,

$(x_t, y_t, z_t, z_{t-1}) \in W$

That is, a system is defined by its initial state and its possible actions under the conditions.

A state sequence z is secure if and only if each state in the sequence is secure. An element (x,y,z) of the system is called an *appearance* of the system; it is a *secure appearance* if and only if z is a secure sequence. Finally, a system is secure if and only if each appearance of the system is a secure appearance. The notion of appearance is used in proving the BLP theorems.

Theorems. BLP proves a theorem for each security property, stating that the system satisfies the property, for any initial state z_0 that satisfies the property, if and only if W satisfies certain conditions for each action.

For simple security, the conditions are (1) each element of the new current access set b^* satisfies the ss-property relative to the new security levels f^* and (2) any element of the old current access set b that does not satisfy the ss-property relative to f^* is not in b^*. Similarly, for the *-property, (1) for each new element of b^* specifying **write,** the level f^* of the object equals the level f^* of the subject, and for **read,** f^* of the subject dominates f^* of the object, and (2) each element of b not satisfying the *-property in relation to f^* is not in b^*.

The main result of the model is the *basic security theorem,* a corollary of the theorems about the properties. It states that a system is secure if its initial state is secure and W satisfies the conditions of the separate theorems for each action. This result is important because it shows that preserving security from one state to the next guarantees system security.

Bell-LaPadula as a base for concrete models. Bell-LaPadula has been used as the base for many concrete system models. The concrete model must be a *valid interpretation* of the abstract BLP model. It must satisfy the axioms of BLP. If it does, the same theorems are true and the security properties of BLP carry over. The concrete model will

be stated in terms of entities such as users, processes, and files, rather than subjects and objects. The entities and relations of the concrete model must be mapped to the corresponding entities of BLP.

Bell-LaPadula issues. Although BLP has long been the leading abstract model, issues remain about its concepts and about what role it should play.

Limitations of access control models. One problem is that an access control model can only roughly express the multilevel policy. *Information flow* models (described in the next section) do that more precisely. The *-property of access control has to be stronger, more restrictive, than the policy. Another problem is that important real-world policies (such as *n*-person control or originator controlled) cannot be expressed. Access control models have the advantage, however, of being closer to the way systems are implemented than information flow models and thus providing a better base for concrete models and more guidance for building systems.

Restrictiveness of the *-property. BLP prohibits information flow from high to low. The assumption is that such flow is equivalent to disclosing secrets. However, systems that enforce BLP cannot support activities that are routine in the paper-based world. "In all practical systems information flow from high to low is unavoidable, and, in some systems, it is the main required activity" (Nelson 1994: 76).

Trusted subjects. Because the *-property is so restrictive, the model introduces trusted subjects. Since the model places no constraints on how trusted processes may violate the *-property, each system that is developed makes its own rules about what trusted subjects can do. It is therefore unclear just what security rules a system does enforce, and any proof of security based on the BLP axioms being enforced does not apply to the whole system including trusted processes.

Incompleteness of the model. The BLP model deals with the current access set; it does not explicitly model actual reads and writes. The complexity of current systems (with buffering, file servers, and file sharing as a few examples) means that supplementary models must be used for assurance that reads and writes are consistent with the current access set. (These supplementary policies have been called *entelechy policies.*)

Covert channels. Access control models such as BLP deal with attempts to gain and pass information through normal "overt" access requests, modeled as inputs to the state machine. The model does not deal with information that is passed in more indirect ways known as *covert channels.* One subject can pass information to another through any resource that they share. For example, a Secret process could leak information to an Unclassified process by placing a lock on an Unclassified file. The information being leaked could be coded in the length of time the lock is held.

Secure state transition. The BLP notion of a secure system has been criticized because it has no independent definition of a secure state transition; it has only the Basic Security Theorem statement that a secure transition leads to a secure state. A system can be secure according to BLP, but still exhibit nonsecure transitions. This becomes clear for the imaginary System Z described by John McLean. In System Z, on any request whatsoever, all subjects and objects are downgraded to the lowest possible level, and the requested access is added to the current access set. Although the new state does not violate the security properties, anyone can access anything—clearly not the intent of the policy. This problem could be corrected by adding to the model necessary conditions for secure state transitions.

Policy issues. Finally, the multilevel policy embodied in BLP is far from meeting all policy needs, even for military systems. An example given earlier is ORCON, which is only one form of dissemination control policy. Also mentioned earlier was the Chinese Wall policy, whose support requires extending or reinterpreting BLP. And BLP does not incorporate mandatory integrity for the content of data—only for its classification. (A companion model of mandatory integrity is described later in this chapter.)

Application-oriented models

One appeal of Bell-LaPadula is its abstractness, but that same quality makes it difficult to apply. It has no place for application-dependent security rules. A different approach is taken by Carl Landwehr, Constance Heitmeyer, and John McLean (1984), who begin with the functional and security requirements of a specific application—a military message system. From these requirements they develop a set of security assertions that the system must enforce. Their model distinguishes between single-level objects and *containers,* which are multilevel information structures containing objects and other containers. Examples of assertions are

- The classification of any container is always at least as high as the maximum of the classifications of the entities it contains.

- Any entity viewed by a user must be labeled with its classification.

With this approach there is no discrepancy between an abstract model (such as BLP) and the model of what is actually built. The assertions of the model are enforced by all the software. Further, since the entities of the model are meaningful to users, the model is more understandable.

Authorization

Discretionary access control has two parts: (1) people specify the security policy to the system and (2) the system enforces that specification. The first part is *authorization*. It might seem easy—just make entries in the access matrix—but it is not. A security policy or model for a system should take into account the meaning of:

- *Groups* of subjects and objects
- *Roles* with respect to applications
- *Roles* with respect to authorization
- *Denial* of access
- *Transformation* of rights
- *Access modes* and their ordering or other structure
- *Context-dependent* and *content-dependent* control
- *Object hierarchies* or other structures

Groups and application-specific roles. For access control to be practical, both subjects and objects must be treated as groups. Otherwise the authorizer's task would be immense. A subject (think of a process) operates on behalf of a user, who may belong to more than one group. One possible policy is for the subject to have the union of the rights of these groups. An alternative is for the user to choose one group identity for a session. The user then takes on a specific application-defined role.

A more elegant approach is to group rights rather than people. A role then becomes simply a named group of rights, and a person is given the right to act in that role. This approach corresponds more closely to how the business world works, with a role corresponding to a job responsibility. The system must know both the role and the individual identity of the user, to support the principle of individual accountability.

Authorization roles. Real-world policies involve roles such as originator, authorizer, classifier, and user. Access control policies typically

distinguish rights that affect the protection state and restrict who may gain those rights. For example, only certain users may add new users, or create certain types of objects. In one common policy the creator of an object (a file, for example) automatically becomes its *owner*, with the sole right to grant access to the object. This may be appropriate for systems supporting programmers or researchers, but it is wrong for most operational systems. There, ownership reflects the structure of the organization and the way authority is exercised.

Authorization must be decentralized in a way that corresponds to the organizational structure. The authorizer has two domains of authority. The *organizational domain* defines the people to whom the authorizer can grant rights, and the *resource domain* defines what operations on what resources can be granted. Distributed systems entail decentralization of the authorizer's role on a geographic basis (which sometimes corresponds to an organizational basis).

Transformation of rights. Access rights can be transformed in many ways. They can be granted to additional subjects, as allowed by the copy flag of the access matrix model, or as in the Take-Grant model. Rights can be automatically *amplified*, as when a program's rights amplify the user's rights, or *restricted*, as when the program's rights place a ceiling on what the user may do. For example, the bank teller can write to the account record when using the Deposit transaction, but not when using the Inquiry transaction. (Another way to look at amplification and restriction is to associate rights with a user-program pair. For business transactions this is highly appropriate.) Access rights may be viewed as ordered, so that one right implies another. For example, it is reasonable for **write** to imply **append.** The right to grant access to an object may imply the right to access the object. Such a right may be *attenuated* when it is passed to another subject, the new subject receiving only the right to access, not the right to grant. Alternatively (and usually better) the right to grant is completely separate from the right to access.

Ravi Sandhu and his colleagues have tried to unify these various cases in the concept of *transformation*. With *internal transformation* a subject who has certain rights for an object obtains additional rights, possibly losing existing rights. With *grant transformation* a subject who has a right to an object can grant some rights for the object to another subject. The *Transformation Model* (TRM) focuses on the *authorization scheme* that governs how the protection state gets changed. A scheme for a system is defined by:

1. A set of access rights
2. Sets of subject types and object types

3. State-changing commands, where each command specifies the authorization for its execution and its effect on the protection state

All subjects and objects have types that never change. This means that TRM can enforce nondiscretionary policies, including (it seems) BLP. The propagation of rights in the transformation model is based entirely on existing rights for the object (and not on other rights of the subject or other subjects' rights). This makes it appropriate for distributed systems, since a location that controls some objects can make access decisions based solely on its own authorization scheme and protection state.

Denial of access. Only positive access rights have been described so far. Explicit denial of rights also is used in some systems, and the TCSEC requires (for discretionary access control) the ability to specify individuals and groups who cannot be given access to some object. Denial may be interpreted in alternative ways. According to the *most-specific* rule, grants or denials to an individual take precedence over those for groups. Assume User 1 belongs to Group A, and Group A is denied **read** on File 1. Then, under the most-specific rule, if User 1 has been given **read** rights to File 1, User 1 may read File 1. An alternative rule is that denial takes precedence, whether group or individual. Under this rule User 1 would not be able to read File 1.

Hierarchies of objects. Hierarchical structures of objects are ubiquitous. Many file systems have hierarchical directory structures, and access to a file may require the right to all directories above it in the hierarchy. In a structural hierarchy, one object is part of another. In a relational database system, for example, a relation is part of a database. A reasonable policy then is that access to the relation requires the corresponding right for the database.

Context-dependent and content-dependent access control. An access decision may use information about the context in which the decision is made. This may be environment information (time of day, for example), subject attributes (such as location, job responsibilities, or history of other accesses), and object attributes (such as file size or creation date). Access control using such information is called *context-dependent. Content-dependent* control uses information in the object being accessed. Content-dependent control is especially relevant for database systems, where the data is structured enough to be used in access decisions. For example, a personnel clerk might have read access to the Salary field of some relation if the Department field shows a department that he handles.

A model of authorization

The many policy issues show that it is important to model the abstract semantics of access control, separate from the mechanisms. This separation is achieved in a model of authorization proposed by Thomas Woo and Simon Lam (1992). The term *authorization* here covers both how access rights are *specified* or *represented* and how they are *evaluated.* The specification is given a precise semantics, so that it is computable; thus evaluation is reduced to computation of the semantics of the specification. Authorization is represented in a declarative language as rules, which are like first-order formulas in logic, but with restrictions to ensure computable semantics. A set of these rules can be translated into an extended form of logic program. (A logic program, written in a logic programming language such as Prolog, can execute mathematic logic.)

A request for access goes to an *authorization module,* which is an interpreter that takes as input the authorization requirements—the set of rules—and the system state (because rules may refer to state information such as the history of access or the time of day). The authorization module tries to verify, for *request*(*s, o, r*), that either *grant*(*s, o, r*) or *deny*(*s, o, r*) logically follows from the inputs. If neither *grant* nor *deny* follows, the decision is *fail*(*s, o, r*).

A rule has the form $(f : f')/g$, representing three formulas called the *prerequisite, assumption,* and *consequent* of the rule. The rules can express many useful access control policies. For example, a simple rule (g) can say that users must be able to read and write their home directories:

$$\mathbf{read}^{+}(\text{User,User.home}) \wedge \mathbf{write}^{+}(\text{User,User.home})$$

A rule with a prerequisite can say that a user x who may execute a program P.exe is also authorized to read the associated documentation P.doc:

$$\mathbf{execute}^{+}(x,\text{P.exe}) \Rightarrow \mathbf{read}^{+}(x,\text{P.doc})$$

Example. Woo and Lam express a simplified Bell-LaPadula model (only the ss-property and *-property, with levels High and Low) as follows:

$$BLP = R^{-} \cup W^{-} \cup R^{+} \cup W^{+}$$

where
$$R^{-} = \{s \in \text{Low} \wedge o \in \text{High} \Rightarrow \mathbf{read}^{-}(s,o)\}$$
$$W^{-} = \{s \in \text{High} \wedge o \in \text{Low} \Rightarrow \mathbf{write}^{-}(s,o)\}$$
$$R^{+} = \{o \in \text{Low} \Rightarrow \mathbf{read}^{+}(s,o)\}$$
$$W^{+} = \{s \in \text{Low} \Rightarrow \mathbf{write}^{+}(s,o)\}$$

Information Flow Models

A goal of most security policy is to protect information. Access control models approach this goal indirectly, dealing not with information but

with objects (such as files) that contain information. Another type of model deals directly with *information flow*. But what *is* information? What is information flow?

Information theory

Information theory, which was developed by Claude Shannon for dealing with communication, provides a systematic view of information. Information theory has been used in information flow models and is relevant to other computer security problems and methods, notably cryptography. This is not surprising, since both communication and computer security are concerned with the transmission of information.

Information theory defines information in terms of uncertainty. Giving information is the same as removing uncertainty. Some uncertain situations are clearly more uncertain than others. A race for mayor among three equally ranked candidates is more uncertain than a race between two. The concept of *entropy* captures this idea.

Entropy. A random variable, such as the result of throwing a die, has a set of possible values, such as 1, 2, 3, 4, 5, and 6. The entropy of a random variable depends on the probabilities of its values. Let X be a random variable that takes a finite set of values with probabilities $p_1, \ldots, p_k, \ldots, p_n$. The entropy H of X is defined as

$$H(X) = -\sum_k p_k \log_2 p_k$$

or

$$H(X) = \sum_k p_k \log_2\left(\frac{1}{p_k}\right)$$

(It is customary in information theory to use logarithms to the base 2; from now on the base is omitted.)

Example. A mayoral race has two candidates, A and B, equally likely to win, so $p_A = .5$ and $p_B = .5$. The log of 1/.5 (the log of 2) is 1.

$$H = .5 \times 1 + .5 \times 1 = 1$$

Entropy is always a positive quantity. An upper bound for H is given by $\log n$. H is equal to $\log n$ (is greatest) when all the probabilities are equal, as in the example. The unit of entropy is a *bit*. When we learn who wins the mayoral race we gain one bit of information, which could be represented in a computer by a one-bit field.

Example. A third candidate C enters the race, and the new polls show probabilities of .5 for A, .25 for B, and .25 for C.

$$\log(1/.25) = \log(4) = 2$$

$$H = .5 \times 1 + .25 \times 2 + .25 \times 2 = 1.5$$

We gain 1.5 bits of information by learning the winner of this race.

Conditional entropy. An important concept for information flow models (and also for cryptography) is *conditional entropy*. The conditional entropy of X given Y is a measure of the uncertainty of X given side knowledge about Y. For each value y_j of Y, there is a conditional entropy of X given y_j.

$$H(X|y_j) = -\sum_k p(x_k|y_j) \log p(x_k|y_j)$$

$$= \sum_k p(x_k|y_j) \log \frac{1}{p(x_k|y_j)}$$

where $p(x_k|y_j)$ is the conditional probability that $X = x_k$ given that $Y = y_j$. The conditional entropy of X given Y is defined as:

$$H(X|Y) = \sum_j H(X|y_j)p(y_j)$$

That is, $H(X|Y)$ is the entropy of X given a particular value of Y, averaged over the possible values of Y.

Conditional entropy measures how much information about X can be gained through knowledge about Y. If X and Y are independent, knowing Y has no effect on the entropy of X. That is, the conditional entropy of X given Y is equal to the entropy of X. The conditional entropy can never be greater than the entropy.

Example. Assume that Hawaii women are 75% Democrat and 25% Republican, and the men are 70% Democrat. Also, 80% of Republican women are married to Democratic men. How much information about a husband's party is conveyed by the wife's party? The conditional probabilities are as follows:

$$p(\mathrm{R}|\mathrm{D}) = .333$$
$$p(\mathrm{D}|\mathrm{D}) = .667$$
$$p(\mathrm{R}|\mathrm{R}) = .2$$
$$p(\mathrm{D}|\mathrm{R}) = .8$$

The conditional entropy of a husband's party given a Democratic wife:

$$H(PTY_{\mathrm{H}}|PTY_{\mathrm{W}} = \mathrm{D}) = .333 \log(1/.333) + .667 \log(1/.667)$$
$$= .333 \times 1.585 + .667 \times .585 = .918$$

Given a Republican wife:

$$H(PTY_{\mathrm{H}}|PTY_{\mathrm{W}} = \mathrm{R}) = .2 \log(1/.2) + .8 \log(1/.8)$$
$$= .2 \times 2.322 + .8 \times .322 = .722$$

The conditional entropy of a husband's party given his wife's party is

$$H(PTY_H \mid PTY_W = H(PTY_H \mid PTY_W = D)\, p\,(PTY_W = D)$$
$$+ H(PTY_H \mid PTY_W = R)\, p\,(PTY_W = R)$$
$$= .918 \times .75 + .722 \times .25 = .869$$

The entropy of a husband's party (not knowing his wife's party) is

$$H(PTY_H) = .7 \log 1/.7 + .3 \log 1/.3$$
$$= .360 + .521 = .881$$

So knowing the wife's party reduces the entropy of the husband's party from .881 bits to .869 bits; it provides .012 bits of information.

Channels. A *channel* is a black box that accepts strings of *symbols* from some *input alphabet* and emits strings of symbols from some *output alphabet.* Information theory defines different kinds of channels. A *discrete channel* can transmit only symbols from a finite input alphabet. In a *memoryless* channel the output is independent of any previous input or output. A *discrete memoryless channel* emits a string the same length as the input string. This channel is defined by its input and output alphabets and by a matrix containing the conditional probability of each output symbol given each input symbol. One kind of discrete memoryless channel is the *binary symmetric channel.* It has input and output alphabets {0,1} and a fixed probability q of an output symbol being the same as the input symbol; $q = 1-p$, where p is the probability of error. The channel matrix is shown in Fig. 4.8.

Channel capacity is a measure of the channel's ability to transmit information. It is expressed (depending on the context) as bits per second or bits per symbol and can be computed from the channel matrix. For example, the capacity of the binary symmetric channel is

$$C = 1 + p \log p + q \log q$$

A lattice model of information flow

An information flow policy defines the classes of information that a system can have and how information may flow between these classes. An information flow model developed by Dorothy Denning (1976, 1982) can express the multilevel policy in terms of information flow rather than access control (as in Bell-LaPadula). It can also express other useful policies. The flow policy is defined by a *lattice.*

$$P = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

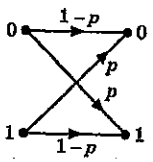

Figure 4.8 Channel matrix for a discrete memoryless channel.

Lattice structure. A lattice is a mathematical structure that neatly represents the meaning of security levels. A lattice consists of a partially ordered set plus a *least upper bound* operator $\oplus$ and a *greatest lower bound operator* $\otimes$. In an information flow model, the lattice $(SC, \leq, \oplus, \otimes)$ represents a set of security classes SC and an ordering relation $\leq$ on the classes. The least upper bound operator $\oplus$ yields the highest security class of the classes it is applied to, and the greatest lower bound operator $\otimes$ yields the lowest security class. The ordering relation $\leq$ is transitive. That is, for classes A, B, and C

$$A \leq B \text{ and } B \leq C \text{ implies } A \leq C$$

Many useful policies can be expressed in lattice form. The U.S. government multilevel policy is often called the *lattice policy,* even when no information flow model is used. For the multilevel policy, if L and L' are sensitivity-level classes, and C and C' are sets of compartments, *dominates* is expressed as

1. $(L, C) \leq (L', C')$ if and only if $L \leq L'$ and $C \subseteq C'$

and (interpreting the least upper bound operator as a class-combining operator)

2. $(L, C) \oplus (L', C) = (\max(L, L'), C \cup C')$

Example. A company's policy forms a lattice with levels *Public* and *Company Confidential* (*CC*) and categories *Engineering* (*E*) and *Marketing* (*M*). The lattice is shown in Fig. 4.9, where an arrow indicates that information may flow along that path.

$$SC = \{Public, CC, E, M\}$$
$$Public \leq CC$$
$$(Public, E) \oplus CC = (CC, E)$$

and

$$(Public, E) \otimes (CC, E, M) = (Public, E)$$

An information flow system. A information flow system is defined as a set of objects, a flow policy (lattice), states, and state transitions. Note that objects here represent all potential holders of information,

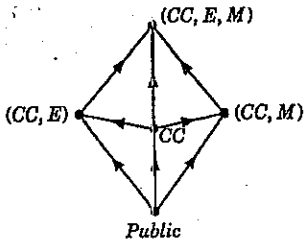

Figure 4.9 Lattice for a company's security policy.

including users. The information state of the system is described by the value and security class of each object. The security class of an object x, written as $\underline{x}$, may be either fixed or variable. A flow from object x to object y (written as $x \rightarrow y$) is permitted if and only if $\underline{x} \leq \underline{y}$, where $\underline{y}$ is the class of y after the flow. If the class $\underline{y}$ is variable, it can change to make the flow permissible. For the multilevel policy without the tranquility principle, the new class $\underline{y}$ would be determined according to 2 above.

State transitions are modeled by operations to create or delete objects, change their values, or change their security class. Only operations that change values can be associated with information flow.

Measuring information flow. An operation causes information to flow from x to y if it reduces the conditional entropy of x given y. That is, new information about x can be obtained from y. The amount of information that flows is measured by the reduction in the conditional entropy $H(x \mid y)$. A potential flow is a channel whose capacity is the maximum information that can be transferred by that flow.

Example. Variable x represents which of four possible locations has been chosen for a military attack. If the locations are equally likely, and the value of x is not known, the flow $x \rightarrow y$ can transfer two bits to y.

Examining a program statically (for example, when it is compiled) reveals its potential information flows, even though the path at execution time is not known. An assignment statement, such as $x := y$, indicates an explicit flow. Conditional statements may indicate implicit flow.

Example. A program contains the statement

```
if x=1 then y := 1
```

where x is 0 or 1 with probability .5 each, and y is initially 0. The entropy of x is one. Before the statement is executed, nothing can be learned about x by examining y. That is, $H(x \mid y) = H(x) = 1$. After the statement is executed, $H(x \mid y) = 0$, since the value of x can be learned with certainty by examining y. One bit of information has been transferred.

In practice, information flow analysis has been applied to formal specifications of programs much more than to programs themselves. Most often it is used to find covert channels.

High water marks and confinement. In the lattice model, the security class of an object can change. When information flows from a higher class into the object, its class rises. The current class of an object is thus a kind of *high water mark* showing the highest class of information that it has "sunk." Simon Foley extends the Denning model by associating a pair of security classes with each entity (subject or object). Each entity x is confined by two classes—a low class $\underline{x}_L$ and a high class $\underline{x}_H$—where

Information in x can flow to class a only if $\underline{x}_L \leq a$

Information at class a can flow into x only if $a \leq \underline{x}_H$

For information to flow from x to y it must hold that $\underline{x}_L \leq \underline{y}_H$.

This confinement flow model can represent flow policies with nontransitive orderings. For example, objects x, y, and z might be confined as follows:

$$confine(x) = [\text{unclassified,unclassified}]$$

$$confine(y) = [\text{unclassified,top-secret}]$$

$$confine(z) = [\text{secret,top-secret}]$$

Information may flow from z to y and from y to x, but not from z to x.

Noninterference

A precise definition of information flow restriction is found in the concept of *noninterference*. One group of users is noninterfering with another group if the actions of the first group using certain commands have no effect on what the second group can see. Noninterference was introduced by Joseph Goguen and José Meseguer (1982, 1984).

The system model. A system is modeled as the following sets:

U users

C state-changing commands

R read commands

O outputs

S internal states, with initial state s_0

along with a transition function *do* and an output function *out*.

do: $S \times U \times C \rightarrow S$, where $do(s,u,c)$ gives the next state after user u executes command c in state s

out: $S \times U \times R \rightarrow O$ where $out(s,u,r)$ gives the result of user u executing a read command r in state s

It is assumed that output commands do not affect the state and that state-changing commands produce no output.

A key concept of the model is the *history* of a system, which is the sequence w of all (*user,command*) pairs since the initial start-up.

$$w = (u_1,c_1)\ldots(u_n, c_n)$$

The state after executing a sequence of state commands, starting from initial state s, is given by function do^*:

$$do^* : S \times (U \times C)^* \rightarrow S$$

The state $[[w]]$ is the state after execution of the sequence w. That is,

$$[[w]] = do^*(s_0,w)$$

Noninterference assertions. Security policies are specified by using noninterference assertions. An assertion is expressed as

$$G, A{:}\,|\,G', B$$

meaning that user group G executing set A of state commands does not interfere with user group G' executing set B of output commands. That is, any A commands by G users are completely invisible to G' users. The notation :| can be thought of as a one-way mirror, where : indicates the "open" side and | the closed side.

The assertion holds if any sequence of commands, given by any users, produces the same effect for users in G' as the corresponding sequence purged of all commands in A given by users in G. That is, for each sequence w, each user v in G', and each output command r in B,

$$out([[w]], v, r) = out([[p_{G,A}(w)]], v, r)$$

where $p_{G,A}(w)$ is the sequence obtained by deleting from w all pairs (u,c) with u in G and c in A. The view of anyone in group G' excludes effects from anything done by anyone in G using A commands. This is expressed as a system history purged of any such actions. The commands that are removed are known as *purgeable*.

Example. Assume that $G = \{u_1,u_2\}$ and $A = \{c_1,c_2\}$. If the sequence

$$w = (u_3,c_1)\,(u_1,c_3)\,(u_1,c_2)\,(u_4,c_1)$$

then the sequence

$$p_{G,A}(w) = (u_3,c_1)\,(u_1,c_3)\,(u_4,c_1)$$

Noninterference assertions must often be conditional. For example, to express the policy that user u can communicate with user v only through commands from set A,

$$u :|\ v \text{ unless } Q_A$$

where Q_A is a predicate expressing that condition. Conditional assertions can also express policies based on past history or the current protection state. They are needed to express the multilevel policy.

Unwinding. Noninterference assertions express security policy as abstract requirements. Useful application requires moving from the assertions to conditions that can be placed on a system's transition function. If those conditions hold, the policy holds. Reaching the conditions from the assertions is known as *unwinding*. Goguen and Meseguer (1984) state assumptions for a multilevel secure system and define multilevel security as noninterference $u{:}|\,v$ for all pairs of users (u,v) except those with $level(u) \leq level(v)$. The definition is not very useful in this form, however, since it must hold for all sequences of (*user,command*) pairs. What is needed is a condition to be applied to state transitions—something like the Bell-LaPadula Basic Security Theorem. Goguen and Meseguer prove such an *unwinding theorem* for multilevel security.

Issues in information flow security

Noninterference is a clear and appealing approach to policies that restrict information flow. Researchers have found problems with it, however, and have suggested new models to correct these problems. One problem is that Goguen and Meseguer modeled systems as deterministic state machines, although systems often are designed with nondeterminism. Scheduling and encryption algorithms, for example, may use randomness. Second, some practical problems cannot be handled, such as the policy that information can flow to a lower level by passing through a trusted downgrader. A third problem is that nonin-

terference does not allow for the generation of high-level data from low-level inputs. This would happen whenever the computer processing itself, not the inputs, determined the security level of the output. For example, a system might take public stock market data as inputs, apply massive computing using public algorithms, and produce private trading decisions. A final and fundamental problem is that noninterference is too strong a requirement. Real systems must settle for some interference, accept some risk. Models must be able to express a quantified amount of interference.

Some progress has been made on these problems. Several theories address nondeterministic systems. Using a model based on logical deduction, Sutherland (1986) defined a property called nondeducibility. Entities at low level should not be able to deduce anything about the activities of high-level entities. Daryl McCullough (1988) defined restrictiveness, which is a composable property. These models still did not handle probabilistic interference, where a higher level event can affect the probability of a lower level event. James Gray introduced P-restrictiveness, which does take into account probabilistic channels. There is also a version of noninterference that accommodates known potential interference.

We have seen that different models use different information-flow confidentiality properties. The models also differ in their models of computation, and that makes them hard to compare. Jeremy Jacob (1991) unifies these different interpretations in a single framework that assumes no specific model of computation. A shared system is treated as a relation. A *window* is a place on the boundary of the system that information can flow from and to. Interactions with the system are made through windows. A system with n windows is modeled as an n-place relation, where the ith place represents interactions through the ith window. Using this framework, Jacob represents, among other properties, noninterference and nondeducibility.

It is important to recognize that researchers do not fully understand confidentiality, let alone integrity and availability.

Composition

Theories of *composition* tell us what security properties result when components or systems are combined in various ways. Since networks and large systems are built by composition, such theories could have great practical value. The main topic studied has been whether a security property is preserved under composition of two systems with that same property.

John McLean (1994) presents a general theory of composition for several security properties called "possibilistic." The most general

form of composition of two systems is *hookup*, where each component can communicate with the other component and also with the outside world. With *cascading*, the output of one system is passed as input to the other. With *feedback* composition, one system serves as front end to a second, which provides feedback to the first. These are *external* composition constructs. *Internal* composition constructs include set union, intersection, and difference. The union of two component systems, for example, accepts any input acceptable to either system and behaves as the relevant system would behave. (The relevant system is the one for which the input is acceptable). If the input is acceptable to both, the composite's output could be the output of either one.

Under cascading, the security properties are preserved when composed with themselves or with stronger properties. Their survival under feedback or internal composition depends on the functions of the systems.

Integrity Models

Integrity means that information is not modified in unauthorized ways, that it is internally consistent and consistent with the real-world objects that it represents, and that the system performs correctly.

Definitions and goals

The definition just given is a good working definition, but it may not include enough. Perhaps a single definition cannot encompass all the properties that have been called *integrity*. It seems that, in order to maintain the constraint

$$\text{Security} = \text{confidentiality} + \text{integrity} + \text{availability}$$

integrity is being defined as all of security except confidentiality and availability. In general, integrity has to do with meeting expectations. *Systems integrity* has to do with a system behaving according to expectations even in the face of attack. *Data integrity* includes two kinds of *consistency*. Data must be internally consistent. For example, the total budget amount must equal the sum of the department budgets. Data must also be consistent with the real-world entities that they represent. For example, the inventory records for the linen department must be consistent with the count of real sheets and towels. A broader concept of data integrity is data *quality*; quality includes such attributes as timeliness, completeness, and pedigree. Robert Courtney and Willis Ware (1994) urge using integrity "to mean that an object has integrity if its quality meets expectations which were determined before the fact."

The following are goals for data integrity:

- Prevent unauthorized modifications
- Maintain *internal* and *external consistency*
- Maintain other data quality attributes
- Prevent authorized but improper modifications

This last goal has to do with those aspects of "proper" that cannot be expressed as consistency requirements. Fraud threatens integrity even if it leaves the data consistent.

For systems integrity the goals are similar, but they apply to the integrity of the computing system itself, rather than its data. Systems integrity is discussed in Chaps. 6 and 7.

All of the security principles discussed in this chapter apply to integrity. The models of discretionary access control also apply to integrity, specifically to the goal of preventing unauthorized modifications. Noninterference can also be interpreted as an integrity model. The Bell-LaPadula model of mandatory access control does not deal with integrity; a companion model was developed to do that.

Biba model

The integrity model of K. J. Biba (1977) assumes a lattice of integrity levels (analogous to security levels) with ordering relation *leq* (less than or equal). Objects are assigned integrity classes according to how damaging it would be if they were improperly modified. Users are assigned integrity classes based on their trustworthiness. Integrity compartments are interpreted like confidentiality compartments. The integrity level of a subject is based on the integrity level of the user it represents and on its needs according to the least privilege principle.

Biba presents several models, all based on the same entities, but representing different integrity policies. The model described here represents the *strict integrity* policy, which is intended as a dual of the Bell-LaPadula confidentiality policy. The entities of the model are

S, O, I	the sets of subjects, objects, and integrity levels
il	a function defining the integrity level of each subject and object
leq	partial ordering relation on integrity levels, *less than or equal*
min	a function returning the greatest lower bound of the subset of *I* specified
o, m	relations defining the ability of a subject *s* to observe (***o***) or modify (***m***) an object *o*
i	a relation defining the ability of subject s_1 to invoke another subject s_2.

The strict integrity policy is characterized by three axioms:

1. $s\ \boldsymbol{o}\ o$ implies $il(s)\ leq\ il(o)$
2. $s\ \boldsymbol{m}\ o$ implies $il(o)\ leq\ il(s)$
3. $s_1\ \boldsymbol{i}\ s_2$ implies $il(s_2)\ leq\ il(s_1)$

According to the first two axioms, a subject may not observe an object of lower integrity and may not modify an object of higher integrity. Axiom 3 states that a subject may not invoke another subject of higher integrity. (The intent here is to prevent the invoking subject from indirectly modifying objects of higher integrity.)

An *information transfer path* is defined as a sequence of objects $o_1,\ldots,o_n$ and a corresponding sequence of subjects $s_1,\ldots,s_{n+1}$ such that, for any $i \in 1,\ldots,n$,

$$s_i\ \boldsymbol{o}\ o_i \text{ and } s_i\ \boldsymbol{m}\ o_{i+1}$$

That is, somewhere in the sequence, some subject reads one object and subsequently modifies another. (Like BLP and unlike information flow models, this model defines information transfer indirectly, in terms of access to objects.) Biba proves that, under the strict integrity axioms, if there is a transfer path from object o_1 to object o_{n+1}, then,

$$il(o_{n+1})\ leq\ il(o_1)$$

That is, information is not transferred to a higher integrity level.

The Biba model has not been used much, perhaps because it does not correspond to an established real-world policy.

Conflict with confidentiality

Integrity sometimes conflicts with confidentiality. For example, protecting against viruses means preserving the integrity of executable modules. This is awkward at best in a system that enforces the Bell-LaPadula model. Using the *- property (no write down) for protection would require classifying a highly sensitive module at the lowest level. If it were classified at the highest level, any subject could write to it. Further, a module classified at the lowest level could be copied and run by a high-level user, who would then risk infection. In these examples the confidentiality policy works against providing integrity. Integrity measures can also work against confidentiality. For example, synchronization methods used to preserve integrity (such as lock-

ing) introduce information flow channels. Confidentiality is also threatened when activity is monitored or data examined to detect or diagnose system problems.

Clark-Wilson model

The integrity model of David Clark and David Wilson (1987, 1989) started a revolution in computer security research. It is not a highly formal model, but rather a framework for describing integrity requirements.

Clark and Wilson point out that, for most computing concerned with business operations and the control of assets, integrity is far more important than confidentiality. Yet most of the effort to provide secure systems, prompted by military needs, addresses only confidentiality. They argue that integrity policies demand different models from confidentiality models, and different mechanisms as well. They focus on two controls that are central in the commercial world: the well-formed transaction and separation of duty (both defined earlier in this chapter.)

Entities of the model. The data items whose integrity is to be preserved are called *constrained data items* (CDIs). The CDIs are manipulated by *transformation procedures* (TPs). The TP of the model represents the well-formed transaction; the purpose of a TP is to transform the set of CDIs from one valid state to another. An *integrity verification procedure* (IVP) has the purpose of confirming that (when it is executed) all of the CDIs are in a valid state; that is, they meet the integrity requirements. The IVP checks for both internal consistency and for consistency with external reality, according to some view of that reality. The particular view of reality is called an *integrity domain.* The role played by the IVP is analogous to the audit function in accounting.

The system must ensure that only TPs can manipulate the CDIs. The TPs and IVPs must be certified with respect to a specific integrity policy. A TP must meet its specifications, and its specifications must be correct. *Unconstrained data items* (UDIs), such as input data, also are relevant because they could be transformed into CDIs.

Rules of the model. The rules of the model define an integrity-enforcing system. *Enforcement rules* (E), which are application-independent, are amenable to being implemented by the system. *Certification rules* (C) involve primarily human analysis and decision making, although some automation is possible. The following rules relate to internal and external consistency:

C1 All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

C2 All TPs must be certified to be valid. That is, they transform a CDI to a valid final state, if the CDI is in a valid state to start. Each TP must be certified for a specific set of CDIs, as expressed in the relation

$$(TP_i, CDI\text{-}list)$$

E1 The system must maintain a list of the relations of rule C2 and must ensure that any manipulation of a CDI is by a TP and is "authorized" by some relation.

Additional rules are needed for separation of duty:

E2 The system must maintain a list of relations of the form

$$(UserId, TP_i, CDI\text{-}list)$$

which relate a user, a TP, and the CDIs that the TP may manipulate on behalf of that user.

C3 The list of relations of E2 must be certified to meet the separation of duty requirement.

The relation of E2, known as the *access triple,* is at the heart of this model. It sums up how the model differs from traditional access control, where subjects gain access to objects directly, rather than through certified programs.

Four other rules complete the model. They specify that:

E3 Users who invoke TPs must be authenticated.

C4 All TPs must be certified to do logging.

C5 TPs that transform UDIs to CDIs must be certified.

E4 Only certain designated users may specify the relations.

These rules enforce a mandatory integrity policy, at least as seen by users. A great deal of human discretion is involved, however, as for example in the certification of TPs.

Implications for computer security. This framework for integrity leads to a set of requirements for computer security services (Clark and Wilson 1989):

Change logs and integrity labels. Authorship must be securely recorded with the data itself (to support the policy of attribution of change). The integrity label records that the data was certified by an IVP, and what integrity domain was used.

Support of the access triple. To enforce the policy of constrained change, the access control triple binds together user, program, and data. Support for it is crucial for this model.

Enhanced user authentication. Although authentication is needed for confidentiality, it takes on special importance for integrity, particularly in relation to the policy of separation of duty. Passwords are inadequate. They can be given away or observed, allowing someone to act as two different persons, thus violating separation of duty rules.

Control of privileged users. Separation of duty must be enforced for the people who maintain the access triples, or who certify TPs. For example, people who can add new users to a system should not be able to change the access triples for those users.

Application program control. A system needs automated tools for managing application software and ensuring its integrity.

Dynamic separation of duty related to TPs. Separation of duty often requires that different steps in a sequence be carried out by different people. Although a static assignment can meet this requirement, a more flexible approach is for the system to keep track of who has executed each step and to enforce separation of duty at each step. This is closer to what happens in the real world. Dynamic separation of duty could be enforced by applications, but mandatory enforcement by the operating system would be better.

Modeling external consistency

One study of integrity elaborated the Clark-Wilson integrity objective into four requirements: external consistency, separation of duty, internal consistency, and error recovery. External consistency is perhaps the most elusive of these, because of its complex relations with the world outside the computing system. The external consistency objective is stated as: "certain users have specified responsibilities for maintaining accurate correspondence between external situations and corresponding information stored in, and available from, the computer. These assigned responsibilities define user roles" (Abrams et al. 1993: 681). An organization must authorize some persons to define roles and others to assign roles to users. The system must ensure that users' authorizations are consistent with their assigned roles. An explicit policy on I/O devices is needed, so that inputs are not forged and outputs not misused.

Taking off from this work, James Williams and Leonard LaPadula (1993) propose two models of external consistency, a simpler one dealing only with correct user inputs, and a more realistic one that allows for user errors.

A system's external interface is modeled as propositions called *sentences.* The "author" of a sentence can be either the system (output) or a user (input). The three kinds of sentences are

- *Assertion:* the author claims that a proposition is true.
- *Question:* the author wants to know whether or not a proposition is true.
- *Request:* the author wants a proposition to become true.

Each user input is associated with a *direct observation* that documents it; the direct observation (but not the input sentence) is assumed to be correct.

The *external-consistency objective* is that each output assertion or request is correct. That is, assertions are true and requests are legitimate according to criteria of the organization or application. However, for efficiency, not all outputs are required to meet this objective; those that do are marked *warranted.* For example, output to a check-writing device would be warranted. A warranted output has an associated *I/O basis* consisting of previous inputs and outputs used in the computation of that output.

The *output-warranty requirement* is for the system's vendor to be able to guarantee that every output marked as warranted is *warrantable,* in that

1. Its I/O basis is warrantable.
2. It is correct provided each sentence in its I/O basis is correct.

An I/O basis is recursively defined to be *warrantable* if and only if all of its outputs are warrantable.

That is, the vendor guarantees that, if the user inputs on which the output is based are correct, then the output is correct. This means the system must be able to perform deductions to show that the output is a consequence of the inputs. The other requirement on systems is that each direct observation is a correct assertion. The responsibility of users (in the first model) is that all their assertions and requests must be correct. These include the software that they install.

Finally, a proof is given of the following result: "If the above system requirements and user responsibility are met, then the external-consistency objectives are satisfied for warranted outputs; in fact, every warrantable sentence is correct."

Williams and LaPadula caution that real systems could only approximate their model, which assumes certified correct software.

Denial of Service

Little has been done to model the third security property—availability—and its inverse, denial of service. One reason is that the requirements are not clearly defined. Jonathan Millen (1992) views a denial of service policy as a guarantee that the system will behave as specified in the face of certain kinds of attack. The policy is enforced by a *Denial-of-Service Protection Base* (DPB), analogous to the reference monitor. The DPB is tamperproof, it cannot be prevented from operating, and it guarantees the availability of the resources under its control. Those resources are a limited subset whose unavailability would be a serious problem.

The model focuses on allocation of shared resources, such as processor time, memory, and files. Since a long delay in service amounts to denial, *waiting time* is used to measure denial. The DPB is characterized by:

- A resource allocation system, which allocates resources to processes. It is modeled as a state machine, with states described by resource allocations, processes' requirements for time and space resources, and the progress of processes.
- A resource monitor algorithm, governing state transitions.
- A waiting time policy, such as fixed maximum waiting time, or finite waiting time.
- *User agreements*—constraints respected by *benign* (as opposed to malicious) processes. For example, benign processes acquire multiple resources in a specified order, to prevent deadlock.

The DPB has to satisfy the conditions that

> (1) Each benign process will make progress in accordance with the waiting time policy.
> (2) No non-CPU resource is revoked from a benign process until its time requirement is zero. (Millen 1992: 146)

These conditions are shown to be satisfied for a simple example DPB with a maximum-waiting-time policy. This model makes a good start, but only a start, toward a realistic model for denial of service.

Security Policy in Practice

This section first describes the role of security policy and models in building trusted systems. Then it introduces the topic of how organizations can set and implement policy. (Chapter 12 returns to this topic.)

Policies, models, and the criteria

The TCSEC specify a methodology for assuring that systems support their security policy (usually BLP). The methodology relies heavily on formal models. For the highest assurance level, there must be a formal model of the policy. The criteria require a formal top-level specification of the *Trusted Computing Base* (TCB), the portion of the system that is trusted to enforce the policy. This specification must be shown to be consistent with the model—by formal methods if possible.

The TCSEC specify one policy. A system certified to meet these criteria may not be able to support other security policies. The European Information Technology Security Evaluation Criteria (ITSEC) also demand that a specific policy be chosen, although they do not restrict what that policy is. It has been argued that the criteria take the wrong approach. For one thing, the approach is inflexible; any change in policy means a complete recertification. Second, cooperation is needed between systems belonging to different organizations, even different nations, and supporting different policies. Third, even one organization needs different policies for its different applications. It would be better (if it could be done) to build systems that can support multiple policies and still be certifiable.

Establishing and carrying out security policy

The computer security policy of an organization must advance the organization's goals and must faithfully represent its real-world security policy. The policy must be practical and usable. It must be cost-effective—addressing the most important threats and accepting some risk where that is appropriate. It should set concrete standards for enforcing security and spell out the response to misuse. The policy must be understandable for users. It must be enforceable technically and must have management behind it. It should aim for the right balance between security and convenient use—a balance that will certainly differ among organizations.

Top management is responsible for setting overall policy, and further responsibility corresponds to the organization's way of distributing authority. Those with authority over people make policy about what rights those people may have. Those responsible for resources make policy about how those resources are used and how they are protected. This pattern applies both to the resources that are available through computers and to the computing resources themselves.

Different roles entail different responsibilities. One generic set of roles in relation to resources is: *owner, user,* and *supplier of services.* The owner of resources is responsible for determining their value,

assigning them the right classification, authorizing access to them, and ensuring that they are properly protected. The owner must also make sure that the policy and controls are clearly understood. People must know what is expected of them.

Users are responsible for understanding and respecting the security policy. They must comply with the controls, using resources only for the approved purposes of the organization, and only in the authorized ways. Users have these responsibilities independent of any security mechanisms that may or may not be available. They must use the mechanisms that are provided and must carry out specific obligations, such as selecting a good password and changing it when the policy specifies. The supplier of computing services is responsible for carrying out the controls specified by owners, as well as for controls that apply to computing resources in general. Everyone, regardless of role, is responsible for alerting management to any security exposures or misuse. Management is responsible for making sure that everyone understands his or her role and its responsibilities.

These generic roles are not the only ones that policy must consider. Some systems have built-in roles, such as the *system administrator* of UNIX. The policy must consider the responsibilities of this role, who may take it on, and who they are accountable to. Is the role centralized, so that, for example, all new accounts are opened at a central place, or is the administration distributed to geographic sites and organizational units? A system administrator is responsible for applying security mechanisms, advising management about security weaknesses and possible improvements, auditing security, and responding appropriately to misuse.

Summary

Long before computers there was a need to protect secrets and control fraud and error. Real-world security policies reflect organizations' goals and situations, as well as principles that apply broadly: individual accountability, authorization, least privilege, separation of duty, auditing, redundancy, and risk reduction.

For confidentiality, documents are often classified and people are given clearances. Access is allowed only if the clearance is at least as high as the classification. Organizations also describe categories; the categories of the person have to include all the categories of the document. For integrity, people can take only those actions they are authorized for. Dividing a task into parts that are performed by different people helps prevent fraud, as does rotating duties. Usually, data should be changed only in prescribed and structured ways. Data must be internally consistent and also consistent with external reality.

Some policy elements—such as role—are common to many application-specific policies.

The scope of security policy broadens for computer security. The real-world security policy must be automated, and policy choices must be made about the computing situation itself.

Models of computer security provide a framework that aids understanding, and an unambiguous representation of policy. A security model should: validly and precisely represent security policy; aid understanding through focused and exact expression and proofs of properties; support safety analysis; support system building and system verification; allow systems to be modeled in parts that are then put together.

Access control is the process of ensuring that all access to resources is authorized access. The access matrix model is simple and intuitive, and it can express many protection policies. The access matrix has a row for each subject and a column for each object. In the access matrix *AM*, cell *AM*[*s*,*o*] specifies the rights that subject *s* has to object *o*. A row of the access matrix corresponds to a capability list—the list of all the rights of a subject. A column corresponds to an access control list—the list of all the rights held by subjects to some object. The access matrix supports discretionary access control (DAC), and other models support mandatory access control (MAC). A security model should support decisions about safety—whether there is some state of the modeled system in which some security property is not preserved. Harrison, Ruzzo, and Ullman showed that it is undecidable whether a given access matrix configuration is safe for a given right.

DAC is vulnerable to Trojan horse attacks. The Bell-LaPadula (BLP) model formalizes a MAC security policy. It has played a key role in relation to the TCSEC. BLP is a state machine model. A secure state is defined by three properties that are intended to express the multilevel policy: the simple security (ss-) property, the *-property, and the discretionary security property. The ss-property expresses clearance-classification policy. The *-property represents the policy of no unauthorized flow of information from a higher level to a lower one. The model includes trusted subjects, who may violate the *-property but are trusted not to violate its intent. The basic security theorem says that preserving security from one state to the next guarantees system security.

BLP has limitations. An access control model cannot express the multilevel policy as precisely as information flow models. The *-property is too restrictive. BLP does not model covert channels or constrain the behavior of trusted processes. The BLP notion of a secure system has been criticized. The multilevel policy does not meet all policy needs.

An important aspect of DAC is *authorization*—specifying the security policy to the system. Models of authorization should take into account: groups of subjects and objects, application roles, authorization roles, denial of access, transformation of rights, access modes and their structure, context-dependent and content-dependent control, and hierarchies of objects.

A goal of most security policy is to protect information. Information flow models approach this goal more directly than access control models. Information theory defines information in terms of uncertainty. Giving information is the same as removing uncertainty. The conditional entropy of x given y is a measure of the uncertainty of x given side knowledge about y. The lattice model of information flow can express the multilevel policy and others. An operation causes information to flow from x to y if it reduces the conditional entropy of x given y.

Noninterference provides a precise definition of information flow restriction. One group of users is noninterfering with another group if the actions of the first group using certain commands have no effect on what the second group can see. Security policies are specified by using noninterference assertions, but useful application requires placing conditions on a system's transition function. Reaching the conditions from the assertions is known as *unwinding*. Noninterference and related concepts are an active research area, but researchers do not fully understand confidentiality, let alone integrity and availability. Theories of composition tell us what security properties result when components or systems are combined.

Integrity means that information is not modified in unauthorized ways, that it is internally consistent and consistent with the real-world objects that it represents, and that the system performs correctly. More generally, integrity has to do with meeting expectations. Goals for data integrity include: prevent unauthorized modifications, maintain internal and external consistency and other data quality attributes, prevent authorized but improper modifications. Integrity sometimes conflicts with confidentiality.

The Clark-Wilson integrity model takes off from two important business controls: the well-formed transaction and separation of duty. Data items whose integrity is to be preserved [constrained data items (CDIs)] are manipulated by transformation procedures (TPs). An integrity verification procedure (IVP) has the purpose of confirming that the CDIs meet the integrity requirements. The system must ensure that only TPs can manipulate the CDIs. The TPs and IVPs must be certified with respect to a specific integrity policy. The model includes enforcement rules, which are application-independent and amenable to automation, and certification rules, which involve primarily human analysis and decision. The access triple controls access to CDIs on the basis of both user ID and TP. This framework leads to requirements for computer security services

Security policy and models are an essential part of building trusted systems. The TCSEC require a formal top-level TCB specification that must be shown to be consistent with the model. An organization's security policy should set concrete standards for enforcing security and spell out the response to misuse. It should aim for the right balance between security and convenient use.

Bibliographic Notes

The definition of security policy is still evolving. The current criteria are inconsistent with one another and even with themselves. Good discussions of the issues are provided by Sterne (1991) and Chizmadia (1992). The ITSEC criteria (European Community—Commission 1991) use the terms *corporate security policy, system security policy* (for a specific installation), and *technical security policy* (for the hardware and software of a system or product). The principles underlying computer security policy are discussed in Chap. 1. Landwehr (1981) describes the multilevel policy. Fernandez, Summers, and Wood (1981) describe maximized sharing. Our discussion of integrity policies draws on Roskos et al. (1990), Clark and Wilson (1987, 1989), Sterne et al. (1991), and NCSC (1991 a and b). The Chinese Wall policy is described by Brewer and Nash (1989). Graubart (1989) discusses the ORCON policy.

Sterne (1991) uses the term *automated security policy.* Delaney et al. (1991) describe generally accepted accounting principles.

Williams and Abrams (1995) give an overview of the use of formal models in developing trusted systems, and Gasser (1988) treats the subject in depth. The process of going from model to implemented system is discussed in Chap. 6.

Sandhu and Samarati (1994) provide an overview of access control. Early versions of the access matrix model were developed by Lampson (1971), Graham and Denning (1972), and Conway, Maxwell, and Morgan (1972). A Typed Access Matrix Model was developed by Sandhu (1992a) and an extension is proposed by Dacier and Deswarte (1994). James Anderson (1972) introduced the term *reference monitor.*

Millen and Cerniglia (1984) discuss concrete models as interpretations of the abstract BLP model. ORCON and dissemination control policy are discussed by McCollum, Messing, and Notargiacomo (1990). Supporting the Chinese Wall policy with BLP is described by Sandhu (1992b, 1993). McLean (1987, 1990b) presents a critique of BLP. Lindgreen and Herschberg (1994) warn of the false (but widely held)

impression that providing a secure computer system is equivalent to providing BLP-like access control. Entelechy policies are discussed by Fellows (1992). Covert channels are discussed in Chap. 6.

Roles are considered by Sterne et al. (1991), Baldwin (1990), Ferraiolo and Kuhn (1992), Sandhu et al. (1996) and in Chaps. 8 and 9. Transformation and transformation models are described by Sandhu (1989, 1992a) and Sandhu and Ganta (1994a, 1994b). Sandhu (1988) describes an earlier Schematic Protection Model. Capabilities are described further in Chap. 7. The Take-Grant model is described by Snyder (1981). Lunt (1989) discusses denial of access. Moffett and Sloman (1988) discuss the right to grant as separate from the right to access. Hierarchies of objects and implied authorization more generally are discussed in Chap. 9. This book simplifies the model of Woo and Lam (1992) by omitting their consideration of multiple sets of independently specified rules.

Information theory is described by Shannon and Weaver (1949). A clear mathematical treatment can be found in Welsh (1988). Another good source is Blahut (1987). Entropy is defined more precisely as a number associated with a probability distribution. The entropy of a random variable X is the entropy of the probability distribution of X. The example about Hawaii women is based on Welsh (1988: 12). Denning (1982) applies information theory to information flow and cryptography and gives many examples and exercises. The lattice model is described by Denning (1976, 1982). Compile-time analysis of programs to reveal information flows is treated by Denning and Denning (1977). Information flow analysis for detecting covert channels is discussed in Chap. 6. The term *high water mark* was used for the ADEPT system (Weissman 1969), where the security level of a process reflects the highest level of data that it has read.

Our discussion of issues in noninterference models draws from Gray (1990) and McLean (1990a). The term *traces* is often used rather than *history*. It comes from *Communicating Sequential Processes* (CSP), a mathematical framework for proving properties of processes (Hoare 1985). A trace represents a sequence of events in the life of a process. *Event-based* traces, while useful for abstract models like noninterference, are not appropriate for specifying and verifying systems. Approaches based on allowable sequences of procedure calls are proposed for that purpose (Meadows 1992). Murphy, Crocker, and Redmond (1992) deal with known potential interference. Shi, McDermid, and Moffett (1993) apply noninterference to composition. Other work on composition is reported by Dinolt, Benzinger, and Yatabe (1994) and Millen (1994). Chapter 9, "Database Security," discusses the problem of inference.

According to U.S. government classification policy (Clinton 1995) integrity means "the state that exists when information is unchanged from its source and has not been accidentally or intentionally modified, altered, or destroyed." Principles of integrity are discussed in Sandhu and Jajodia (1990) and NCSC (1991a,b). This chapter draws heavily from NCSC (1991a), but sums things up differently. The dual custody policy is described by McLean (1990b). Appearance of the Clark and Wilson model led to two integrity workshops (Katzke and Ruthberg 1989, Ruthberg and Polk 1989). Data quality is defined in Ruthberg and Polk. Dynamic separation of duty is discussed by Nash and Poland (1990). Abrams et al. (1993) elaborate on the Clark-Wilson objectives. Limoges et al. (1994) describe a model that supports the access triple. Lipner (1982) shows how the Biba model could be applied in a commercial environment. A system that enforces the Biba model is described in Chap. 9. Motro (1989) considers integrity definitions and theory. Applying the BLP model to protecting against viruses is discussed by Cohen (1987, 1990).

Millen's work on denial of service (1992) builds on work of Yu and Gligor (1990).

The process of developing systems to meet the TCSEC criteria is described in Chap. 6. Multipolicy approaches are described by Abrams et al. (1990), LaPadula (1995), Hosmer (1992), and Bell (1994).

Our discussion of establishing and carrying out security policy draws from Fites, Kratz, and Brebner (1989), the report of the National Research Council (1991), guidelines for Internet site security (Holbrook and Reynolds 1991), and a study of security policy for the National Research and Education Network (Oldehoeft 1992). More about this topic can be found in Chap. 12.

Exercises

4.1 What are the parts to computer security policy? Give three additional examples of each part.

4.2 Discuss the principles of least privilege and separation of duty. How do they support the principle of authorization?

4.3 Why is composability an important quality for models?

4.4 What is the significance of the HRU access matrix model?

4.5 Describe precisely the main result obtained by Harrison, Ruzzo, and Ullman.

4.6 The Typed Access Matrix Model of Sandhu can be used for the ORCON policy. The problem is to prevent a person p_1 granted read access to an ORCON document from copying the information into another document and granting p_2 access to that copy. The solution involves the special access

rights confined read and parent, and the security types confined subject and confined object. The originator grants p_1 confined read access to the confined object. How would you define the command use-confined-read? Hint: create a temporary subject of type "confined subject"; define the type confined subject with appropriate limitations.

4.7 Discuss the advantages and limitations of discretionary access control. Which limitation is the most serious?

4.8 Describe the purpose of the *-property. Describe precisely in words its specification in the Bell-LaPadula model. What is the relationship between the purpose and the specification?

4.9 Define and draw a lattice representing a security policy for a company with three separate areas of product development and two levels of information sensitivity.

4.10 Can a virus spread between the levels of a multilevel system? Explain.

4.11 Describe the concept of noninterference.

4.12 Assume a noninterference assertion

$$G, A: G', B$$

Group $G = \{u_1, u_2\}$ and command set $A = \{c_1, c_2\}$. The sequence $w = (u_3, c_1)$ (u_1, c_3) (u_2, c_1) (u_1, c_2) (u_4, c_1). List the purgeable commands of the sequence.

4.13 List three types of constraints that are placed on the Constrained Data Items of the Clark-Wilson model.

4.14 What is the significance of the access triple of the Clark-Wilson model? Give an example of a business situation where this type of control is needed.

4.15 The TCSEC and ITSEC both assume that a system will enforce a specific security policy. Discuss the pros and cons of this approach.

4.16 Write a computer security policy for a large public library. Assume the computing system is used for acquisitions and cataloging, records about library card holders, and circulation. The on-line catalog is available to all patrons. Access is through PCs, many of them in public areas.

References

Abrams, Marshall D., Edward G. Amoroso, Leonard J. LaPadula, Teresa F. Lunt, and James G. Williams. 1993. Report of an integrity research study group. *Computers & Security* 12(7): 679–689.

Abrams, Marshall D., Leonard J. LaPadula, Kenneth W. Eggers, and Ingrid M. Olson. 1990. A generalized framework for access control: An informal description. *Proceedings of the 13th National Computer Security Conference*, 135–143. NIST/NCSC.

Anderson, James P. 1972. *Computer Security Technology Planning Study.* Report ESD-TR-73-51, Vol. I. AD 758206. Bedford, Mass.: U.S. Air Force Electronic Systems Division.

Baldwin, Robert W. 1990. Naming and grouping privileges to simplify security management in large databases. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 116–132. Los Alamitos, Calif.: IEEE Computer Society Press.

Bell, D. Elliott. 1994. Modeling the "Multipolicy Machine." *Proceedings of the 1994 ACM SIGSAC New Security Paradigms Workshop*, 2–9. Los Alamitos, Calif.: IEEE Computer Society.

Bell, D. E. and L. J. LaPadula. 1976. *Secure Computer System: Unified Exposition and Multics Interpretation.* Report MTR-2997 Rev. 1. AD A023 588. Bedford, Mass.: The Mitre Corporation.

Biba, K. J. 1977. Integrity Considerations for Secure Computer Systems. Report ESD-TR-76-372, AD A039324. Bedford, Mass.: U.S. Air Force Electronic Systems Division.

Blahut, Richard E. 1987. *Principles and Practice of Information Theory.* Reading, Mass.: Addison-Wesley.

Brewer, David F. C. and Michael J. Nash. 1989. The Chinese Wall security policy. *Proceedings of the 1989 IEEE Computer Society Symposium on Research in Security and Privacy*, 206–214. Los Alamitos, Calif.: IEEE Computer Society Press.

Chizmadia, David M. 1992. Some more thoughts on the buzzword "security policy." *Proceedings of the 15th National Computer Security Conference*, Vol. II, 651–660. NIST/NCSC.

Clark, David D., and David R. Wilson. 1987. A comparison of commercial and military computer security policies. *Proceedings of the 1987 IEEE Computer Society Symposium on Research in Security and Privacy*, 184–194. Los Alamitos, Calif.: IEEE Computer Society Press.

Clark, David D. and David R. Wilson. 1989. Evolution of a model for computer integrity. *Report of the Invitational Workshop on Data Integrity*, A.2-1–A.2-13. NIST Special Publication 500-168.

Clinton, William J. 1995. *Classified National Security Information.* Executive Order, April 17. Washington, D.C.: The White House.

Cohen, Fred. 1987. Computer viruses: Theory and experiments. *Computers & Security* 6(1): 22-35.

Cohen, Fred. 1990. Implications of computer viruses and current methods of defense. In *Computers under Attack: Intruders, Worms, and Viruses*, Peter J. Denning, ed., 381–406. New York: ACM Press.

Conway, R. W., W. L. Maxwell, and H. L. Morgan. 1972. On the implementation of security measures in information systems. *Communications of the ACM* 15(4): 211–220.

Courtney, Robert H., and Willis H. Ware. 1994. What do we mean by integrity? *Computers & Security* 13(3): 206–208.

Dacier, Marc, and Yves Deswarte. 1994. Privilege graph: An extension to the Typed Access Matrix model. *Computer Security—ESORICS 94. Proceedings of the Third European Symposium on Research in Computer Security*, 319–334. Berlin: Springer-Verlag.

Delaney, Patrick R., James R. Adler, Barry J. Epstein, and Michael F. Foran. 1991. *GAAP: Interpretation and Application of Generally Accepted Accounting Principles.* New York: John Wiley & Sons.

Denning, Dorothy E. 1976. A lattice model of secure information flow. *Communications of the ACM* 19(5): 236–243.

———. 1982. *Cryptography and Data Security.* Reading, Mass.: Addison-Wesley.

Denning, D. E., and P. J. Denning. 1977. Certification of programs for secure information flow. *Communications of the ACM* 20(7): 504–513.

Dinolt, G. W., L. A. Benzinger, and M. G. Yatabe. 1994. Combining components and

policies. *Proceedings of the Computer Security Foundations Workshop VII*, 22–33. Los Alamitos, Calif.: IEEE Computer Society.
European Communities—Commission. 1991. ITSEC: Information Technology Security Evaluation Criteria: Provisional Harmonised Criteria. Luxembourg: Office for Official Publications of the EC.
Fellows, Jonathan. 1992. Mandatory policy issues of high assurance composite systems. *Proceedings of the 15th National Computer Security Conference*, Vol. I, 350–358. NIST/NCSC.
Fernandez, Eduardo B., Rita C. Summers, and Christopher Wood. 1981. *Database Security and Integrity*. Reading, Mass.: Addison-Wesley.
Ferraiolo, David and Richard Kuhn. 1992. Role-based access controls. *Proceedings of the 15th National Computer Security Conference*, 554–563. NIST/NCSC.
Fites, Philip E., Martin P. J. Kratz, and Alan F. Brebner. 1989. *Control and Security of Computer Information Systems*. Rockville, Md.: Computer Science Press.
Foley, Simon N. 1989. A model for secure information flow. *Proceedings of the 1989 IEEE Computer Society Symposium on Research in Security and Privacy*, 248–258. Los Alamitos, Calif.: IEEE Computer Society Press.
Gasser, Morrie. 1988. *Building a Secure Computer System*. New York: Van Nostrand Reinhold.
Goguen, J. A., and J. Meseguer. 1982. Security policies and security models. *Proceedings of the 1982 Symposium on Security and Privacy*, 11–20. Los Alamitos, Calif.: IEEE Computer Society Press.
Goguen, Joseph A., and José Meseguer. 1984. Unwinding and inference control. *Proceedings of the 1984 Symposium on Security and Privacy*, 75–86. Los Alamitos, Calif.: IEEE Computer Society Press.
Graham, G. Scott, and Peter J. Denning. 1972. Protection—principles and practice. *Proceedings of the 1972 Spring Joint Computer Conference*, 417–429. Montvale, N.J.: AFIPS Press.
Graubart, Richard. 1989. On the need for a third form of access control. *Proceedings of the 12th National Computer Security Conference*, 296–304. NIST/NCSC.
Gray, James W. III. 1990. Probabilistic interference. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 170–179. Los Alamitos, Calif.: IEEE Computer Society Press.
Harrison, M. A., W. L. Ruzzo, and J. D. Ullman. 1976. Protection in operating systems. *Communications of the ACM* 19(8): 461–471.
Hoare, C. A. R. 1985. *Communicating Sequential Processes*. Englewood Cliffs, N.J.: Prentice-Hall.
Holbrook, J. Paul, and Joyce K. Reynolds. 1991. *Site Security Handbook*. Internet Site Security Policy Handbook Working Group. RFC 1244.
Hosmer, Hilary H. 1992. The multipolicy paradigm. *Proceedings of the 15th National Computer Security Conference*, vol. II, 409–422. NIST/NCSC.
Jacob, Jeremy. 1991. A uniform presentation of confidentiality properties. *IEEE Transactions on Software Engineering* 17(11): 1186–1194.
Katzke, Stuart W., and Zella G. Ruthberg, eds. 1989. *Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)*. NIST Special Publication 500-160.
Lampson, Butler W. 1971. Protection. *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, 437–443. Reprinted in *Operating Systems Review* 8(1): 18–24, January 1974.
Landwehr, Carl E. 1981. Formal models for computer security. *Computing Surveys* 13(3): 247–278.
Landwehr, Carl E., Constance L. Heitmeyer, and John McLean. 1984. A security model for military message systems. *ACM Transactions on Computer Systems* 2(3): 198–222.
LaPadula, Leonard J. 1995. Rule-set modeling of a trusted computer system. In *Information Security: An Integrated Collection of Essays*, 187–241. Los Alamitos, Calif.: IEEE Computer Society.
Limoges, Charles G., Ruth R. Nelson, John H. Heimann, and David S. Becker. 1994. Versatile integrity and security environment (VISE) for computer systems. *Proceedings of the 1994 ACM SIGSAC New Security Paradigms Workshop*, 109–118. Los Alamitos, Calif.: IEEE Computer Society.
Lindgreen, E. Roos, and I. S. Herschberg. 1994. On the validity of the Bell-LaPadula model. *Computers & Security* 13(4): 317–333.
Lipner, Steven B. 1982. Non-discretionary controls for commercial applications. *Proceedings of the 1982 Symposium on Security and Privacy*, 2–10. Los Alamitos, Calif.: IEEE Computer Society Press.
Lunt, Teresa F. 1989. Access control policies: Some unanswered questions. *Computers & Security* 8(1): 43–54.
McCollum, Catherine J., Judith R. Messing, and LouAnna Notargiacomo. 1990. Beyond the pale of MAC and DAC—defining new forms of access control. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 190–200. Los Alamitos, Calif.: IEEE Computer Society Press.
McCullough, Daryl. 1988. Noninterference and the composability of security properties. *Proceedings of the 1988 IEEE Computer Society Symposium on Research in Security and Privacy*, 177–186. Washington, D.C.: IEEE Computer Society Press.
McLean, John. 1987. Reasoning about security models. *Proceedings of the 1987 IEEE Computer Society Symposium on Research in Security and Privacy*, 123–131. Los Alamitos, Calif.: IEEE Computer Society Press.
———. 1990a. Security models and information flow. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 180–187. Los Alamitos, Calif.: IEEE Computer Society Press.
———. 1990b. The specification and modeling of computer security. *Computer* 23(1): 9–16.
———. 1994. A general theory of composition for trace sets closed under selective interleaving functions. *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, 79–93. Los Alamitos, Calif.: IEEE Computer Society.
Meadows, Catherine. 1992. Using traces based on procedure calls to reason about composability. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 177–188. Los Alamitos, Calif.: IEEE Computer Society Press.
Millen, Jonathan K. 1992. A resource allocation model for denial of service. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 137–147. Los Alamitos, Calif.: IEEE Computer Society Press.
———. 1994. Unwinding forward correctability. *Proceedings of the Computer Security Foundations Workshop VII*, 2–10. Los Alamitos, Calif.: IEEE Computer Society.
Millen, J. K., and C. M. Cerniglia. 1984. *Computer Security Models*. Report MTR9531. AD A166 920. Bedford, Mass.: The MITRE Corporation.
Moffett, Jonathan D., and Sloman, Morris S. 1988. The source of authority for commercial access control. *Computer* 21(2): 59–69.
Motro, Amihai. 1989. Integrity = validity + completeness. *ACM Transactions on Database Systems* 14(4): 480–502.
Murphy, Sandra R., Stephen Crocker, and Timothy Redmond. 1992. Unwinding and the LOCK proof referees study. *Proceedings of the Computer Security Foundations Workshop V*, 9–21. Los Alamitos, Calif.: IEEE Computer Society Press.
Nash, Michael J., and Keith R. Poland. 1990. Some conundrums concerning separation of duty. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 201–207. Los Alamitos, Calif.: IEEE Computer Society Press.
National Research Council. 1991. *Computers at Risk: Safe Computing in the Information Age*. Washington, D.C.: National Academy Press.
NCSC. 1988. *Glossary of Computer Security Terms*. Report NCSC-TG-004. Fort Meade, Md.: NCSC.
———. 1991a. *Integrity in Automated Information Systems*. Prepared by Terry Mayfield et al. C Technical Report 79-91. Fort Meade, Md.: NCSC.
———. 1991b. *Integrity-Oriented Control Objectives: Proposed Revisions to the Trusted Computer System Evaluation Criteria (TCSEC)*. Prepared by Terry Mayfield et al. C Technical Report 111-91. Fort Meade, Md.: NCSC.
Nelson, Ruth. 1994. What is a Secret—and—What does that have to do with Computer

Security? *Proceedings of the 1994 ACM SIGSAC New Security Paradigms Workshop*, 74–79. Los Alamitos, Calif.: IEEE Computer Society.
Oldehoeft, Arthur E. 1992. *Foundations of a Security Policy for Use of the National Research and Educational Network*. Report NISTIR 4734. Gaithersburg, Md.: NIST.
Roskos, J. Eric, Stephen R. Welke, John M. Boone, and Terry Mayfield. 1990. A taxonomy of integrity models, implementations and mechanisms. *Proceedings of the 13th National Computer Security Conference*, 541–551. NIST/NCSC.
Ruthberg, Zella G., and William T. Polk, eds. 1989. *Report of the Invitational Workshop on Data Integrity*. NIST Special Publication 500-168. Gaithersburg, Md.: NIST.
Sandhu, Ravinderpal Singh. 1988. The Schematic Protection Model: Its definition and analysis for acyclic attenuating schemes. *Journal of the Association for Computing Machinery* 35(2): 404–432.
Sandhu, Ravi. 1989. Transformation of access rights. *Proceedings of the 1989 IEEE Computer Society Symposium on Research in Security and Privacy*, 259–268. Los Alamitos, Calif.: IEEE Computer Society Press.
———. 1992a. The typed access matrix model. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 122–136. Los Alamitos, Calif.: IEEE Computer Society Press.
———. 1992b. A lattice interpretation of the Chinese wall policy. *Proceedings of the 15th National Computer Security Conference*, Vol. I, 329–339. NIST/NCSC.
———. 1993. Lattice-based access control models. *Computer* 26(11): 9–19.
Sandhu, Ravi S., Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-based access control models. *Computer* 29(2): 38–47.
Sandhu, Ravi S., and Srinivas Ganta. 1994a. On the expressive power of the unary transformation model. *Computer Security—ESORICS 94. Proceedings of the Third European Symposium on Research in Computer Security*, 301–318. Berlin: Springer-Verlag.
———. 1994b. On the minimality of testing for rights in transformation models. *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, 230–241. Los Alamitos, Calif.: IEEE Computer Society.
Sandhu, Ravi, and Sushil Jajodia. 1990. Integrity mechanisms in database management systems. *Proceedings of the 13th National Computer Security Conference*, Vol. II, 526–540. Washington, D.C.: NIST/NCSC.
Sandhu, Ravi S., and Pierangela Samarati. 1994. Access control: Principles and practice. *IEEE Communications Magazine* 32(9): 40–48.
Shannon, Claude E., and Warren Weaver. 1949. *The Mathematical Theory of Communication*. Urbana, Ill.: The University of Illinois Press.
Shi, Qi, J. A. McDermid, and J. D. Moffett. 1993. Applying noninterference to composition of systems: A more practical approach. *Proceedings of the Ninth Annual Computer Security Applications Conference*, 210–220. Los Alamitos, Calif.: IEEE Computer Society.
Snyder, Lawrence. 1981. Formal models of capability-based protection systems. *IEEE Transactions on Computers* C-30(3): 172–181.
Sterne, Daniel F. 1991. On the buzzword "security policy." *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 219–230. Los Alamitos, Calif.: IEEE Computer Society Press.
Sterne, Daniel F., Martha A. Branstad, Brian S. Hubbard, Barbara A. Mayer, and Dawn M. Wolcott. 1991. An analysis of application-specific security policies. In *Proceedings of the 14th National Computer Security Conference*, Vol. I, 25–36. NIST/NCSC.
Sutherland, D. 1986. A model of information. *Proceedings of the 9th National Computer Security Conference*, 175–183. NIST/NCSC.
Weissman, C. 1969. Security controls in the ADEPT-50 time-sharing system. *AFIPS Conference Proceedings, 1969 FJCC*, 119–133. Montvale, N.J.: AFIPS Press.
Welsh, Dominic. 1988. *Codes and Cryptography*. Oxford: Clarendon Press.
Williams, James G., and Marshall D. Abrams. 1995. Formal methods and models. In *Information Security: An Integrated Collection of Essays*, 170–186. Los Alamitos, Calif.: IEEE Computer Society.
Williams, James G., and Leonard J. LaPadula. 1993. Automated support for external consistency. *Proceedings of the Computer Security Foundations Workshop VI*, 71–81. Los Alamitos, Calif.: IEEE Computer Society.
Woo, Thomas Y. C., and Simon S. Lam. 1992. Authorization in distributed systems: A formal approach. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 33–50. Los Alamitos, Calif.: IEEE Computer Society Press.
Yu, Che-Fn, and Virgil D. Gligor. 1990. A specification and verification method for preventing denial of service. *IEEE Transactions on Software Engineering*, 16(6): 581–592.