Chapter 8
Principles of Security Models, Design, and Capabilities

THE CISSP EXAM TOPICS COVERED IN THIS CHAPTER INCLUDE:

- ✓ **Domain 3: Security Architecture and Engineering**
  - 3.1 Implement and manage engineering processes using secure design principles
  - 3.2 Understand the fundamental concepts of security models
  - 3.3 Select controls based upon systems security requirements
  - 3.4 Understand security capabilities of information systems

Understanding the philosophy behind security solutions helps to limit your search for the best controls for specific security needs. In this chapter, we discuss security models, including state machine, Bell-LaPadula, Biba, Clark-Wilson, Take-Grant, and Brewer and Nash. This chapter also describes Common Criteria and other methods governments and corporations use to evaluate information systems from a security perspective, with particular emphasis on U.S. Department of Defense and international security evaluation criteria. Finally, we discuss commonly encountered design flaws and other issues that can make information systems susceptible to attack.

The process of determining how secure a system is can be difficult and time-consuming. In this chapter, we describe the process of evaluating a computer system's level of security. We begin by introducing and

explaining basic concepts and terminology used to describe information system security concepts and talk about secure computing, secure perimeters, security and access monitors, and kernel code. We turn to security models to explain how access and security controls can be implemented. We also briefly explain how system security may be categorized as either open or closed; describe a set of standard security techniques used to ensure confidentiality, integrity, and availability of data; discuss security controls; and introduce a standard suite of secure networking protocols.

Additional elements of this domain are discussed in various chapters: Chapter 6, “Cryptography and Symmetric Key Algorithms,” Chapter 7, “PKI and Cryptographic Applications,” Chapter 9, “Security Vulnerabilities, Threats, and Countermeasures,” and Chapter 10, “Physical Security Requirements.” Please be sure to review all of these chapters to have a complete perspective on the topics of this domain.

Implement and Manage Engineering Processes Using Secure Design Principles

Security should be a consideration at every stage of a system's development. Programmers should strive to build security into every application they develop, with greater levels of security provided to critical applications and those that process sensitive information. It's extremely important to consider the security implications of a development project from the early stages because it's much easier to build security into a system than it is to add security onto an existing system. The following sections discuss several essential security design principles that should be implemented and managed early in the engineering process of a hardware or software project.

Objects and Subjects

Controlling access to any resource in a secure system involves two entities. The *subject* is the user or process that makes a request to access a resource. Access can mean reading from or writing to a resource. The *object* is the resource a user or process wants to access. Keep in mind that the subject and object refer to some specific access request, so the same resource can serve as a subject and an object in different access requests.

For example, process A may ask for data from process B. To satisfy process A's request, process B must ask for data from process C. In this example, process B is the object of the first request and the subject of the second request:

First request	process A (subject)	process B (object)
Second request	process B (subject)	process C (object)

This also serves as an example of transitive trust. *Transitive trust* is the concept that if A trusts B and B trusts C, then A inherits trust of C through the transitive property—which works like it would in a mathematical equation: if a = b, and b = c, then a = c. In the previous example, when A requests data from B and then B requests data from

C, the data that A receives is essentially from C. Transitive trust is a serious security concern because it may enable bypassing of restrictions or limitations between A and C, especially if A and C both support interaction with B. An example of this would be when an organization blocks access to Facebook or YouTube to increase worker productivity. Thus, workers (A) do not have access to certain internet sites (C). However, if workers are able to access to a web proxy, virtual private network (VPN), or anonymization service, then this can serve as a means to bypass the local network restriction. In other words, if workers (A) are accessing VPN service (B), and the VPN service (B) can access the blocked internet service (C); then A is able to access C through B via a transitive trust exploitation.

Closed and Open Systems

Systems are designed and built according to one of two differing philosophies: A *closed system* is designed to work well with a narrow range of other systems, generally all from the same manufacturer. The standards for closed systems are often proprietary and not normally disclosed. *Open systems*, on the other hand, are designed using agreed-upon industry standards. Open systems are much easier to integrate with systems from different manufacturers that support the same standards.

Closed systems are harder to integrate with unlike systems, but they can be more secure. A closed system often comprises proprietary hardware and software that does not incorporate industry standards. This lack of integration ease means that attacks on many generic system components either will not work or must be customized to be successful. In many cases, attacking a closed system is harder than launching an attack on an open system. Many software and hardware components with known vulnerabilities may not exist on a closed system. In addition to the lack of known vulnerable components on a closed system, it is often necessary to possess more in-depth knowledge of the specific target system to launch a successful attack.

Open systems are generally far easier to integrate with other open systems. It is easy, for example, to create a local area network (LAN) with a Microsoft Windows Server machine, a Linux machine, and a

Macintosh machine. Although all three computers use different operating systems and could represent up to three different hardware architectures, each supports industry standards and makes it easy for networked (or other) communications to occur. This ease comes at a price, however. Because standard communications components are incorporated into each of these three open systems, there are far more predictable entry points and methods for launching attacks. In general, their openness makes them more vulnerable to attack, and their widespread availability makes it possible for attackers to find (and even to practice on) plenty of potential targets. Also, open systems are more popular than closed systems and attract more attention. An attacker who develops basic attacking skills will find more targets on open systems than on closed ones. This larger "market" of potential targets usually means that there is more emphasis on targeting open systems. Inarguably, there's a greater body of shared experience and knowledge on how to attack open systems than there is for closed systems.

Open Source vs. Closed Source

It's also helpful to keep in mind the distinction between open-source and closed-source systems. An *open-source* solution is one where the source code, and other internal logic, is exposed to the public. A closed-source solution is one where the source code and other internal logic is hidden from the public. Open-source solutions often depend on public inspection and review to improve the product over time. *Closed-source* solutions are more dependent on the vendor/programmer to revise the product over time. Both open-source and closed-source solutions can be available for sale or at no charge, but the term *commercial* typically implies closed-source. However, closed-source code is often revealed through either vendor compromise or through decompiling. The former is always a breach of ethics and often the law, whereas the latter is a standard element in ethical reverse engineering or systems analysis.

It is also the case that a closed-source program can be either an

open system or a closed system, and an open-source program can be either an open system or a closed system.

Techniques for Ensuring Confidentiality, Integrity, and Availability

To guarantee the confidentiality, integrity, and availability of data, you must ensure that all components that have access to data are secure and well behaved. Software designers use different techniques to ensure that programs do only what is required and nothing more. Suppose a program writes to and reads from an area of memory that is being used by another program. The first program could potentially violate all three security tenets: confidentiality, integrity, and availability. If an affected program is processing sensitive or secret data, that data's confidentiality is no longer guaranteed. If that data is overwritten or altered in an unpredictable way (a common problem when multiple readers and writers inadvertently access the same shared data), there is no guarantee of integrity. And, if data modification results in corruption or outright loss, it could become unavailable for future use. Although the concepts we discuss in the following sections all relate to software programs, they are also commonly used in all areas of security. For example, physical confinement guarantees that all physical access to hardware is controlled.

Confinement

Software designers use process confinement to restrict the actions of a program. Simply put, process *confinement* allows a process to read from and write to only certain memory locations and resources. This is also known as *sandboxing*. The operating system, or some other security component, disallows illegal read/write requests. If a process attempts to initiate an action beyond its granted authority, that action will be denied. In addition, further actions, such as logging the violation attempt, may be taken. Systems that must comply with higher security ratings usually record all violations and respond in some tangible way. Generally, the offending process is terminated.

Confinement can be implemented in the operating system itself (such as through process isolation and memory protection), through the use of a confinement application or service (for example, Sandboxie at www.sandboxie.com), or through a virtualization or hypervisor solution (such as VMware or Oracle's VirtualBox).

Bounds

Each process that runs on a system is assigned an authority level. The authority level tells the operating system what the process can do. In simple systems, there may be only two authority levels: user and kernel. The authority level tells the operating system how to set the bounds for a process. The *bounds* of a process consist of limits set on the memory addresses and resources it can access. The bounds state the area within which a process is confined or contained. In most systems, these bounds segment logical areas of memory for each process to use. It is the responsibility of the operating system to enforce these logical bounds and to disallow access to other processes. More secure systems may require physically bounded processes. Physical bounds require each bounded process to run in an area of memory that is physically separated from other bounded processes, not just logically bounded in the same memory space. Physically bounded memory can be very expensive, but it's also more secure than logical bounds.

Isolation

When a process is confined through enforcing access bounds, that process runs in isolation. Process isolation ensures that any behavior will affect only the memory and resources associated with the isolated process. *Isolation* is used to protect the operating environment, the kernel of the operating system (OS), and other independent applications. Isolation is an essential component of a stable operating system. Isolation is what prevents an application from accessing the memory or resources of another application, whether for good or ill. The operating system may provide intermediary services, such as cut-and-paste and resource sharing (such as the keyboard, network interface, and storage device access).

These three concepts (confinement, bounds, and isolation) make designing secure programs and operating systems more difficult, but they also make it possible to implement more secure systems.

Controls

To ensure the security of a system, you need to allow subjects to access only authorized objects. A *control* uses access rules to limit the access of a subject to an object. Access rules state which objects are valid for each subject. Further, an object might be valid for one type of access and be invalid for another type of access. One common control is for file access. A file can be protected from modification by making it read-only for most users but read-write for a small set of users who have the authority to modify it.

There are both mandatory and discretionary access controls, often called mandatory access control (MAC) and discretionary access control (DAC), respectively (see Chapter 14, “Controlling and Monitoring Access,” for an in-depth discussion of access controls). With mandatory controls, static attributes of the subject and the object are considered to determine the permissibility of an access. Each subject possesses attributes that define its clearance, or authority, to access resources. Each object possesses attributes that define its classification. Different types of security methods classify resources in different ways. For example, subject A is granted access to object B if the security system can find a rule that allows a subject with subject A’s clearance to access an object with object B’s classification.

Discretionary controls differ from mandatory controls in that the subject has some ability to define the objects to access. Within limits, discretionary access controls allow the subject to define a list of objects to access as needed. This access control list serves as a dynamic access rule set that the subject can modify. The constraints imposed on the modifications often relate to the subject’s identity. Based on the identity, the subject may be allowed to add or modify the rules that define access to objects.

Both mandatory and discretionary access controls limit the access to objects by subjects. The primary goal of controls is to ensure the

confidentiality and integrity of data by disallowing unauthorized access by authorized or unauthorized subjects.

Trust and Assurance

Proper security concepts, controls, and mechanisms must be integrated before and during the design and architectural period in order to produce a reliably secure product. Security issues should not be added on as an afterthought; this causes oversights, increased costs, and less reliability. Once security is integrated into the design, it must be engineered, implemented, tested, audited, evaluated, certified, and finally accredited.

A *trusted system* is one in which all protection mechanisms work together to process sensitive data for many types of users while maintaining a stable and secure computing environment. *Assurance* is simply defined as the degree of confidence in satisfaction of security needs. Assurance must be continually maintained, updated, and reverified. This is true if the trusted system experiences a known change or if a significant amount of time has passed. In either case, change has occurred at some level. Change is often the antithesis of security; it often diminishes security. So, whenever change occurs, the system needs to be reevaluated to verify that the level of security it provided previously is still intact. Assurance varies from one system to another and must be established on individual systems. However, there are grades or levels of assurance that can be placed across numerous systems of the same type, systems that support the same services, or systems that are deployed in the same geographic location. Thus, trust can be built into a system by implementing specific security features, whereas assurance is an assessment of the reliability and usability of those security features in a real-world situation.

Understand the Fundamental Concepts of Security Models

In information security, models provide a way to formalize security policies. Such models can be abstract or intuitive (some are decidedly mathematical), but all are intended to provide an explicit set of rules that a computer can follow to implement the fundamental security concepts, processes, and procedures that make up a security policy. These models offer a way to deepen your understanding of how a computer operating system should be designed and developed to support a specific security policy.

A *security model* provides a way for designers to map abstract statements into a security policy that prescribes the algorithms and data structures necessary to build hardware and software. Thus, a security model gives software designers something against which to measure their design and implementation. That model, of course, must support each part of the security policy. In this way, developers can be sure their security implementation supports the security policy.

Tokens, Capabilities, and Labels

Several different methods are used to describe the necessary security attributes for an object. A security *token* is a separate object that is associated with a resource and describes its security attributes. This token can communicate security information about an object prior to requesting access to the actual object. In other implementations, various lists are used to store security information about multiple objects. A *capabilities list* maintains a row of security attributes for each controlled object. Although not as flexible as the token approach, capabilities lists generally offer quicker lookups when a subject requests access to an object. A third common type of attribute storage is called a *security label*, which is generally a permanent part of the object to which it's attached. Once a security label is set, it usually cannot be altered. This permanence provides another safeguard against tampering

that neither tokens nor capabilities lists provide.

You'll explore several security models in the following sections; all of them can shed light on how security enters into computer architectures and operating system design:

- Trusted computing base
- State machine model
- Information flow model
- Noninterference model
- Take-Grant model
- Access control matrix
- Bell-LaPadula model
- Biba model
- Clark-Wilson model
- Brewer and Nash model (also known as Chinese Wall)
- Goguen-Meseguer model
- Sutherland model
- Graham-Denning model

Although no system can be totally secure, it is possible to design and build reasonably secure systems. In fact, if a secured system complies with a specific set of security criteria, it can be said to exhibit a level of trust. Therefore, trust can be built into a system and then evaluated, certified, and accredited. But before we can discuss each security model, we have to establish a foundation on which most security models are built. This foundation is the trusted computing base.

Trusted Computing Base

An old U.S. Department of Defense standard known colloquially as the Orange Book/Trusted Computer System Evaluation Criteria (TCSEC) (DoD Standard 5200.28, covered in more detail later in this chapter in

the section “Rainbow Series”) describes a *trusted computing base (TCB)* as a combination of hardware, software, and controls that work together to form a trusted base to enforce your security policy. The TCB is a subset of a complete information system. It should be as small as possible so that a detailed analysis can reasonably ensure that the system meets design specifications and requirements. The TCB is the only portion of that system that can be trusted to adhere to and enforce the security policy. It is not necessary that every component of a system be trusted. But any time you consider a system from a security standpoint, your evaluation should include all trusted components that define that system’s TCB.

In general, TCB components in a system are responsible for controlling access to the system. The TCB must provide methods to access resources both inside and outside the TCB itself. TCB components commonly restrict the activities of components outside the TCB. It is the responsibility of TCB components to ensure that a system behaves properly in all cases and that it adheres to the security policy under all circumstances.

Security Perimeter

The *security perimeter* of your system is an imaginary boundary that separates the TCB from the rest of the system (Figure 8.1). This boundary ensures that no insecure communications or interactions occur between the TCB and the remaining elements of the computer system. For the TCB to communicate with the rest of the system, it must create secure channels, also called *trusted paths*. A trusted path is a channel established with strict standards to allow necessary communication to occur without exposing the TCB to security vulnerabilities. A trusted path also protects system users (sometimes known as subjects) from compromise as a result of a TCB interchange. As you learn more about formal security guidelines and evaluation criteria later in this chapter, you’ll also learn that trusted paths are required in systems that seek to deliver high levels of security to their users. According to the TCSEC guidelines, trusted paths are required for high-trust-level systems such as those at level B2 or higher of TCSEC.

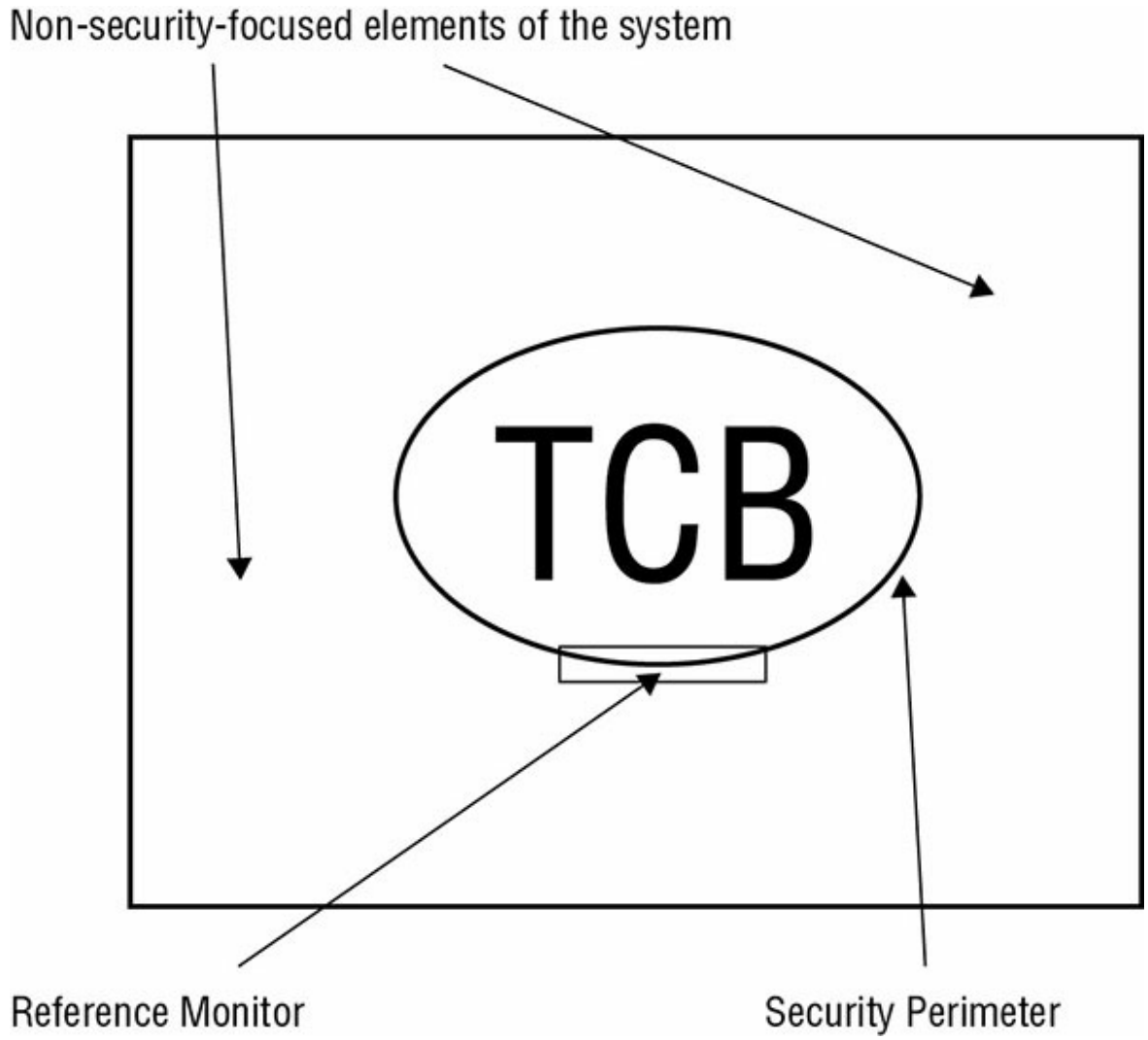

FIGURE 8.1 The TCB, security perimeter, and reference monitor

Reference Monitors and Kernels

When the time comes to implement a secure system, it's essential to develop some part of the TCB to enforce access controls on system assets and resources (sometimes known as objects). The part of the TCB that validates access to every resource prior to granting access requests is called the *reference monitor* (Figure 8.1). The reference monitor stands between every subject and object, verifying that a requesting subject's credentials meet the object's access requirements before any requests are allowed to proceed. If such access requirements aren't met, access requests are turned down. Effectively,

the reference monitor is the access control enforcer for the TCB. Thus, authorized and secured actions and activities are allowed to occur, whereas unauthorized and insecure activities and actions are denied and blocked from occurring. The reference monitor enforces access control or authorization based on the desired security model, whether Discretionary, Mandatory, Role Based, or some other form of access control. The reference monitor may be a conceptual part of the TCB; it doesn't need to be an actual, stand-alone, or independent working system component.

The collection of components in the TCB that work together to implement reference monitor functions is called the *security kernel*. The reference monitor is a concept or theory that is put into practice via the implementation of a security kernel in software and hardware. The purpose of the security kernel is to launch appropriate components to enforce reference monitor functionality and resist all known attacks. The security kernel uses a trusted path to communicate with subjects. It also mediates all resource access requests, granting only those requests that match the appropriate access rules in use for a system.

The reference monitor requires descriptive information about each resource that it protects. Such information normally includes its classification and designation. When a subject requests access to an object, the reference monitor consults the object's descriptive information to discern whether access should be granted or denied (see the sidebar "Tokens, Capabilities, and Labels" for more information on how this works).

State Machine Model

The *state machine model* describes a system that is always secure no matter what state it is in. It's based on the computer science definition of a *finite state machine (FSM)*. An FSM combines an external input with an internal machine state to model all kinds of complex systems, including parsers, decoders, and interpreters. Given an input and a state, an FSM transitions to another state and may create an output. Mathematically, the next state is a function of the current state and the input next state; that is, the next state = F(input, current state).

Likewise, the output is also a function of the input and the current state output; that is, the output = F(input, current state).

Many security models are based on the secure state concept. According to the state machine model, a *state* is a snapshot of a system at a specific moment in time. If all aspects of a state meet the requirements of the security policy, that state is considered secure. A transition occurs when accepting input or producing output. A transition always results in a new state (also called a *state transition*). All state transitions must be evaluated. If each possible state transition results in another secure state, the system can be called a *secure state machine*. A secure state machine model system always boots into a secure state, maintains a secure state across all transitions, and allows subjects to access resources only in a secure manner compliant with the security policy. The secure state machine model is the basis for many other security models.

Information Flow Model

The *information flow model* focuses on the flow of information. Information flow models are based on a state machine model. The Bell-LaPadula and Biba models, which we will discuss in detail later in this chapter, are both information flow models. Bell-LaPadula is concerned with preventing information flow from a high security level to a low security level. Biba is concerned with preventing information flow from a low security level to a high security level. Information flow models don't necessarily deal with only the direction of information flow; they can also address the type of flow.

Information flow models are designed to prevent unauthorized, insecure, or restricted information flow, often between different levels of security (these are often referred to as multilevel models). Information flow can be between subjects and objects at the same classification level as well as between subjects and objects at different classification levels. An information flow model allows all authorized information flows, whether within the same classification level or between classification levels. It prevents all unauthorized information flows, whether within the same classification level or between classification levels.

Another interesting perspective on the information flow model is that it is used to establish a relationship between two versions or states of the same object when those two versions or states exist at different points in time. Thus, information flow dictates the transformation of an object from one state at one point in time to another state at another point in time. The information flow model also addresses covert channels by specifically excluding all nondefined flow pathways.

Noninterference Model

The *noninterference model* is loosely based on the information flow model. However, instead of being concerned about the flow of information, the noninterference model is concerned with how the actions of a subject at a higher security level affect the system state or the actions of a subject at a lower security level. Basically, the actions of subject A (high) should not affect the actions of subject B (low) or even be noticed by subject B. The real concern is to prevent the actions of subject A at a high level of security classification from affecting the system state at a lower level. If this occurs, subject B may be placed into an insecure state or be able to deduce or infer information about a higher level of classification. This is a type of information leakage and implicitly creates a covert channel. Thus, the noninterference model can be imposed to provide a form of protection against damage caused by malicious programs such as Trojan horses.

Real World Scenario

Composition Theories

Some other models that fall into the information flow category build on the notion of how inputs and outputs between multiple systems relate to one another—which follows how information flows between systems rather than within an individual system. These are called *composition theories* because they explain how outputs from one system relate to inputs to another system. There are three recognized types of composition theories:

- *Cascading*: Input for one system comes from the output of another system.
- *Feedback*: One system provides input to another system, which reciprocates by reversing those roles (so that system A first provides input for system B and then system B provides input to system A).
- *Hookup*: One system sends input to another system but also sends input to external entities.

Take-Grant Model

The *Take-Grant model* employs a directed graph (Figure 8.2) to dictate how rights can be passed from one subject to another or from a subject to an object. Simply put, a subject with the grant right can grant another subject or another object any other right they possess. Likewise, a subject with the take right can take a right from another subject. In addition to these two primary rules, the Take-Grant model may adopt a create rule and a remove rule to generate or delete rights. The key to this model is that using these rules allows you to figure out when rights in the system can change and where leakage (that is, unintentional distribution of permissions) can occur.

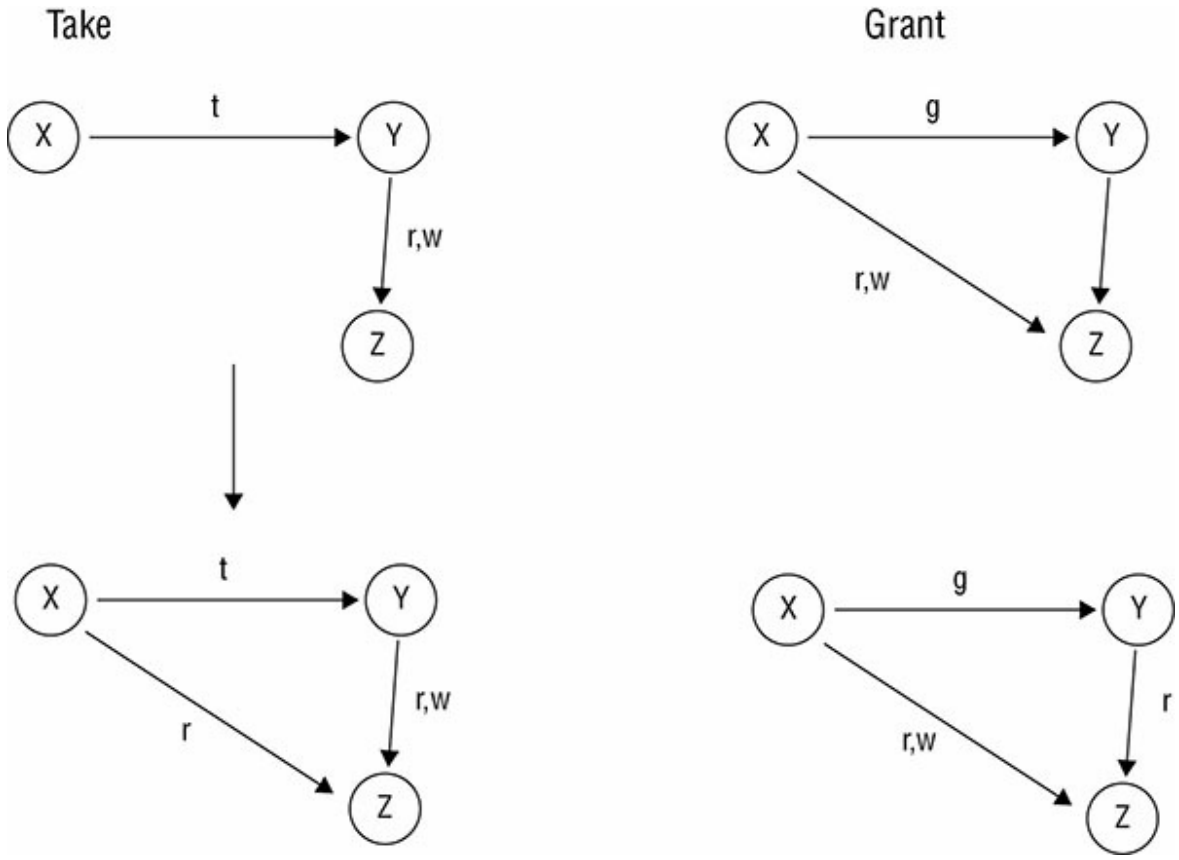

FIGURE 8.2 The Take-Grant model's directed graph

Take rule	Allows a subject to take rights over an object
Grant rule	Allows a subject to grant rights to an object
Create rule	Allows a subject to create new rights
Remove rule	Allows a subject to remove rights it has

Access Control Matrix

An *access control matrix* is a table of subjects and objects that indicates the actions or functions that each subject can perform on each object. Each column of the matrix is an access control list (ACL). Each row of the matrix is a *capabilities list*. An ACL is tied to the object; it lists valid actions each subject can perform. A capability list is tied to the subject; it lists valid actions that can be taken on each object. From an administration perspective, using only capability lists for access control is a management nightmare. A capability list method

of access control can be accomplished by storing on each subject a list of rights the subject has for every object. This effectively gives each user a key ring of accesses and rights to objects within the security domain. To remove access to a particular object, every user (subject) that has access to it must be individually manipulated. Thus, managing access on each user account is much more difficult than managing access on each object (in other words, via ACLs).

Implementing an access control matrix model usually involves the following:

- Constructing an environment that can create and manage lists of subjects and objects
- Crafting a function that can return the type associated with whatever object is supplied to that function as input (this is important because an object's type determines what kind of operations may be applied to it)

The access control matrix shown in Table 8.1 is for a discretionary access control system. A mandatory or rule-based matrix can be constructed simply by replacing the subject names with classifications or roles. Access control matrixes are used by systems to quickly determine whether the requested action by a subject for an object is authorized.

TABLE 8.1 An access control matrix

Subjects	Document file	Printer	Network folder share
Bob	Read	No Access	No Access
Mary	No Access	No Access	Read
Amanda	Read, Write	Print	No Access
Mark	Read, Write	Print	Read, Write
Kathryn	Read, Write	Print, Manage Print Queue	Read, Write, Execute
Colin	Read, Write, Change	Print, Manage Print Queue, Change	Read, Write, Execute, Change

	Permissions	Permissions	Permissions

Bell-LaPadula Model

The U.S. Department of Defense (DoD) developed the *Bell-LaPadula model* in the 1970s to address concerns about protecting classified information. The DoD manages multiple levels of classified resources, and the Bell-LaPadula multilevel model was derived from the DoD's multilevel security policies. The classifications the DoD uses are numerous; however, discussions of classifications within the CISSP Common Body of Knowledge (CBK) are usually limited to unclassified, sensitive but unclassified, confidential, secret, and top secret. The multilevel security policy states that a subject with any level of clearance can access resources at or below its clearance level. However, within the higher clearance levels, access is granted only on a need-to-know basis. In other words, access to a specific object is granted to the classified levels only if a specific work task requires such access. For example, any person with a secret security clearance can access secret, confidential, sensitive but unclassified, and unclassified documents but not top-secret documents. Also, to access a document within the secret level, the person seeking access must also have a need to know for that document.

By design, the Bell-LaPadula model prevents the leaking or transfer of classified information to less secure clearance levels. This is accomplished by blocking lower-classified subjects from accessing higher-classified objects. With these restrictions, the Bell-LaPadula model is focused on maintaining the confidentiality of objects. Thus, the complexities involved in ensuring the confidentiality of documents are addressed in the Bell-LaPadula model. However, Bell-LaPadula does not address the aspects of integrity or availability for objects. Bell-LaPadula is also the first mathematical model of a multilevel security policy.

Real World Scenario

Lattice-Based Access Control

This general category for nondiscretionary access controls is covered in Chapter 13, "Managing Identity and Authentication." Here's a quick preview on that more detailed coverage of this subject (which drives the underpinnings for most access control security models): Subjects under *lattice-based access controls* are assigned positions in a lattice. These positions fall between defined security labels or classifications. Subjects can access only those objects that fall into the range between the least upper bound (the nearest security label or classification higher than their lattice position) and the highest lower bound (the nearest security label or classification lower than their lattice position) of the labels or classifications for their lattice position. Thus, a subject that falls between the private and sensitive labels in a commercial scheme that reads bottom up as public, sensitive, private, proprietary, and confidential can access only public and sensitive data but not private, proprietary, or confidential data. Lattice-based access controls also fit into the general category of information flow models and deal primarily with confidentiality (that's the reason for the connection to Bell-LaPadula).

This model is built on a state machine concept and the information flow model. It also employs mandatory access controls and the lattice concept. The lattice tiers are the *classification levels* used by the security policy of the organization. The state machine supports multiple states with explicit transitions between any two states; this concept is used because the correctness of the machine, and guarantees of document confidentiality, can be proven mathematically. There are three basic properties of this state machine:

- The *Simple Security Property* states that a subject may not read information at a higher sensitivity level (no read up).
- The * *(star) Security Property* states that a subject may not write information to an object at a lower sensitivity level (no write down). This is also known as the *Confinement Property*.
- The *Discretionary Security Property* states that the system uses an access matrix to enforce discretionary access control.

These first two properties define the states into which the system can transition. No other transitions are allowed. All states accessible through these two rules are secure states. Thus, Bell-LaPadula–modeled systems offer state machine model security (see Figure 8.3).

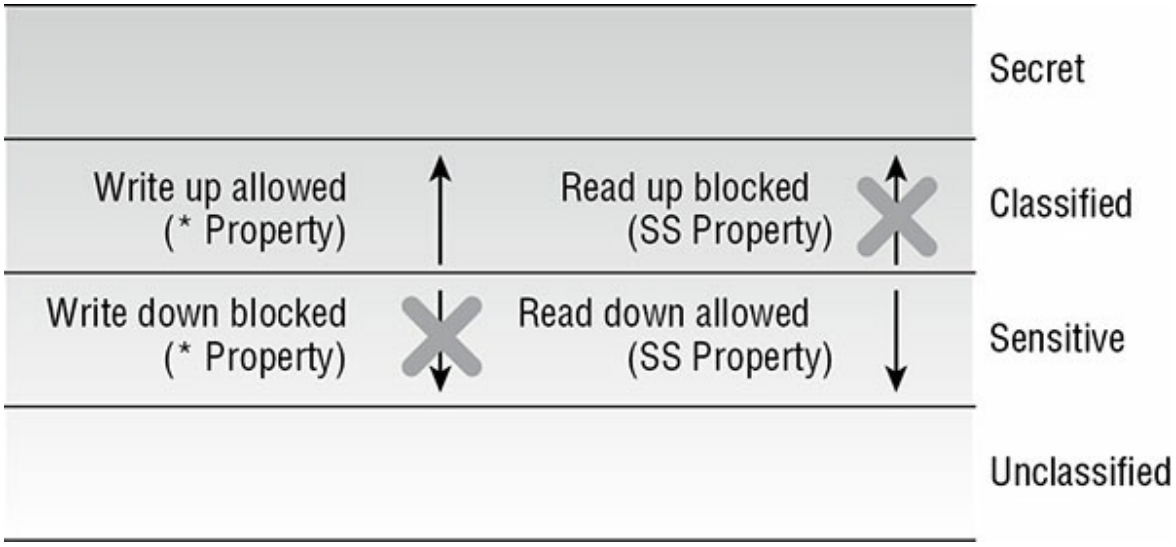

FIGURE 8.3 The Bell-LaPadula model

The Bell-LaPadula properties are in place to protect data confidentiality. A subject cannot read an object that is classified at a higher level than the subject is cleared for. Because objects at one level have data that is more sensitive or secret than data in objects at a lower level, a subject (who is not a trusted subject) cannot write data from one level to an object at a lower level. That action would be similar to pasting a top-secret memo into an unclassified document file. The third property enforces a subject's need to know in order to access an object.

An exception in the Bell-LaPadula model states that a "trusted subject" is not constrained by the * Security Property. A trusted subject is defined as "a subject that is guaranteed not to consummate a security-breaching information transfer even if it is possible." This means that a trusted subject is allowed to violate the * Security Property and perform a write-down, which is necessary when performing valid object declassification or

reclassification.

The Bell-LaPadula model addresses only the confidentiality of data. It does not address its integrity or availability. Because it was designed in the 1970s, it does not support many operations that are common today, such as file sharing and networking. It also assumes secure transitions between security layers and does not address covert channels (covered in Chapter 9, "Security Vulnerabilities, Threats, and Countermeasures"). Bell-LaPadula does handle confidentiality well, so it is often used in combination with other models that provide mechanisms to handle integrity and availability.

Biba Model

For some nonmilitary organizations, integrity is more important than confidentiality. Out of this need, several integrity-focused security models were developed, such as those developed by Biba and by Clark-Wilson. The *Biba model* was designed after the Bell-LaPadula model. Where the Bell-LaPadula model addresses confidentiality, the Biba model addresses integrity. The Biba model is also built on a state machine concept, is based on information flow, and is a multilevel model. In fact, Biba appears to be pretty similar to the Bell-LaPadula model, except inverted. Both use states and transitions. Both have basic properties. The biggest difference is their primary focus: Biba primarily protects data integrity. Here are the basic properties or axioms of the Biba model state machine:

- The *Simple Integrity Property* states that a subject cannot read an object at a lower integrity level (no read-down).
- The * *(star) Integrity Property* states that a subject cannot modify an object at a higher integrity level (no write-up).


In both the Biba and Bell-LaPadula models, there are two properties that are inverses of each other: simple and * (star). However, they may also be labeled as axioms, principles, or rules.

What you should focus on is the *simple* and *star* designations. Take note that *simple* is always about reading, and *star* is always about writing. Also, in both cases, simple and star are rules that define what cannot or should not be done. In most cases, what is not prevented or disallowed is supported or allowed.

Figure 8.4 illustrates these Biba model axioms.

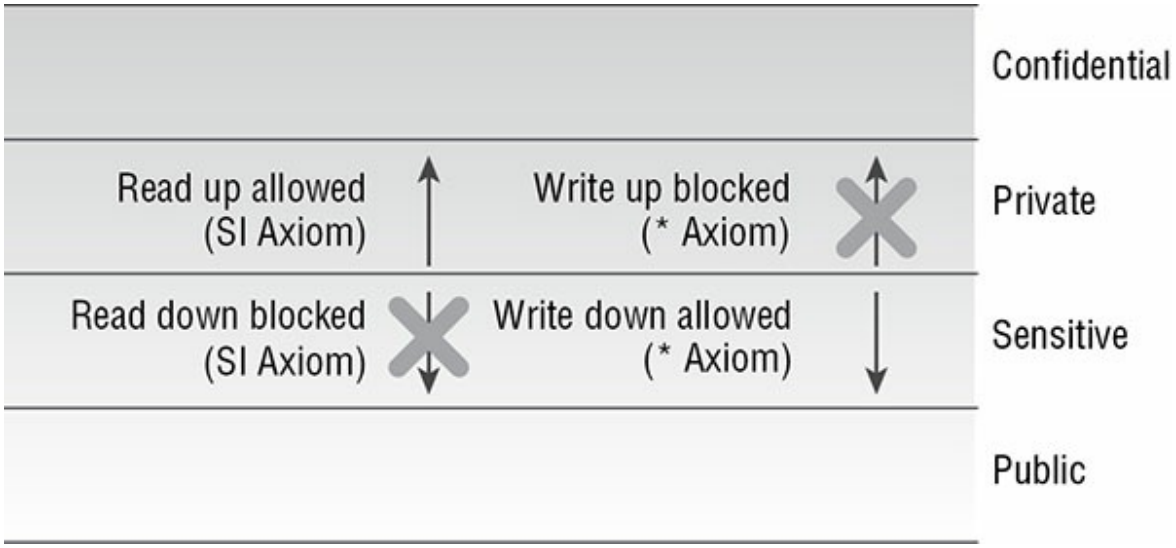

FIGURE 8.4 The Biba model

When you compare Biba to Bell-LaPadula, you will notice that they look like they are opposites. That's because they focus on different areas of security. Where the Bell-LaPadula model ensures data confidentiality, Biba ensures data integrity.

Biba was designed to address three integrity issues:

- Prevent modification of objects by unauthorized subjects.
- Prevent unauthorized modification of objects by authorized subjects.
- Protect internal and external object consistency.

As with Bell-LaPadula, Biba requires that all subjects and objects have a classification label. Thus, data integrity protection is dependent on data classification.

Consider the Biba properties. The second property of the Biba model is pretty straightforward. A subject cannot write to an object at a higher integrity level. That makes sense. What about the first property? Why can't a subject read an object at a lower integrity level? The answer takes a little thought. Think of integrity levels as being like the purity level of air. You would not want to pump air from the smoking section into the clean room environment. The same applies to data. When integrity is important, you do not want unvalidated data read into validated documents. The potential for data contamination is too great to permit such access.

Critiques of the Biba model reveal a few drawbacks:

- It addresses only integrity, not confidentiality or availability.
- It focuses on protecting objects from external threats; it assumes that internal threats are handled programmatically.
- It does not address access control management, and it doesn't provide a way to assign or change an object's or subject's classification level.
- It does not prevent covert channels.

Because the Biba model focuses on data integrity, it is a more common choice for commercial security models than the Bell-LaPadula model. Some commercial organizations are more concerned with the integrity of their data than its confidentiality. Commercial organizations that are more focused on integrity than confidentiality may choose to implement the Biba model, but most organizations require a balance between both confidentiality and integrity, requiring them to implement a more complex solution than either model by itself.

Clark-Wilson Model

Although the Biba model works in commercial applications, another model was designed in 1987 specifically for the commercial environment. The *Clark-Wilson model* uses a multifaceted approach to enforcing data integrity. Instead of defining a formal state machine, the Clark-Wilson model defines each data item and allows modifications through only a small set of programs.

The Clark-Wilson model does not require the use of a lattice structure; rather, it uses a three-part relationship of subject/program/object (or subject/transaction/object) known as a *triple* or an *access control triple*. Subjects do not have direct access to objects. Objects can be accessed only through programs. Through the use of two principles—well-formed transactions and separation of duties—the Clark-Wilson model provides an effective means to protect integrity.

Well-formed transactions take the form of programs. A subject is able to access objects only by using a program, interface, or access portal (Figure 8.5). Each program has specific limitations on what it can and cannot do to an object (such as a database or other resource). This effectively limits the subject's capabilities. This is known as a constrained interface. If the programs are properly designed, then the triple relationship provides a means to protect the integrity of the object.

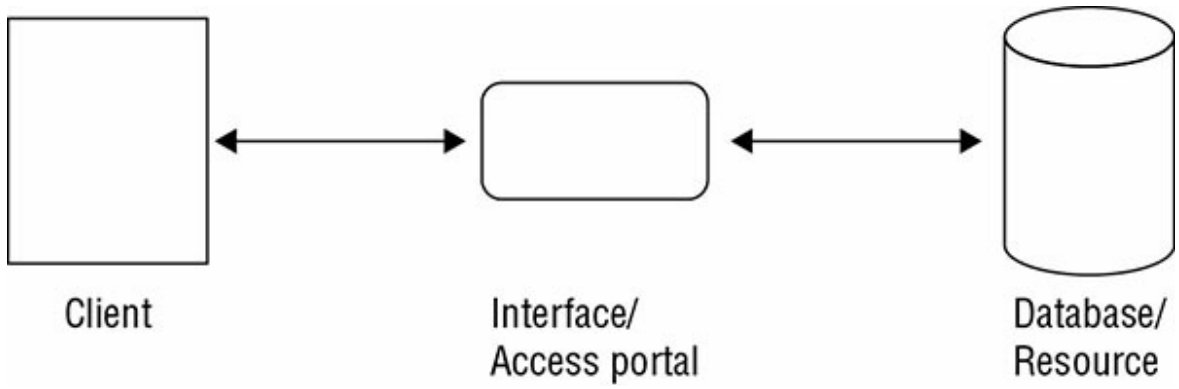

FIGURE 8.5 The Clark-Wilson model

Clark-Wilson defines the following items and procedures:

- A *constrained data item (CDI)* is any data item whose integrity is protected by the security model.
- An *unconstrained data item (UDI)* is any data item that is not controlled by the security model. Any data that is to be input and hasn't been validated, or any output, would be considered an unconstrained data item.
- An *integrity verification procedure (IVP)* is a procedure that scans data items and confirms their integrity.

- *Transformation procedures (TPs)* are the only procedures that are allowed to modify a CDI. The limited access to CDIs through TPs forms the backbone of the Clark-Wilson integrity model.

The Clark-Wilson model uses security labels to grant access to objects, but only through transformation procedures and a *restricted interface model*. A restricted interface model uses classification-based restrictions to offer only subject-specific authorized information and functions. One subject at one classification level will see one set of data and have access to one set of functions, whereas another subject at a different classification level will see a different set of data and have access to a different set of functions. The different functions made available to different levels or classes of users may be implemented by either showing all functions to all users but disabling those that are not authorized for a specific user or by showing only those functions granted to a specific user. Through these mechanisms, the Clark-Wilson model ensures that data is protected from unauthorized changes from any user. In effect, the Clark-Wilson model enforces separation of duties. The Clark-Wilson design makes it a common model for commercial applications.

Brewer and Nash Model (aka Chinese Wall)

The *Brewer and Nash model* was created to permit access controls to change dynamically based on a user's previous activity (making it a kind of state machine model as well). This model applies to a single integrated database; it seeks to create security domains that are sensitive to the notion of conflict of interest (for example, someone who works at Company C who has access to proprietary data for Company A should not also be allowed access to similar data for Company B if those two companies compete with each other). This model is known as the *Chinese Wall model* because it creates a class of data that defines which security domains are potentially in conflict and prevents any subject with access to one domain that belongs to a specific conflict class from accessing any other domain that belongs to the same conflict class. Metaphorically, this puts a wall around all other information in any conflict class. Thus, this model also uses the principle of data isolation within each conflict class to keep users out

of potential conflict-of-interest situations (for example, management of company datasets). Because company relationships change all the time, dynamic updates to members of and definitions for conflict classes are important.

Another way of looking at or thinking of the Brewer and Nash model is of an administrator having full control access to a wide range of data in a system based on their assigned job responsibilities and work tasks. However, at the moment an action is taken against any data item, the administrator's access to any conflicting data items is temporarily blocked. Only data items that relate to the initial data item can be accessed during the operation. Once the task is completed, the administrator's access returns to full control.

Goguen-Meseguer Model

The *Goguen-Meseguer model* is an integrity model, although not as well known as Biba and the others. In fact, this model is said to be the foundation of noninterference conceptual theories. Often when someone refers to a noninterference model, they are actually referring to the Goguen-Meseguer model.

The Goguen-Meseguer model is based on predetermining the set or domain—a list of objects that a subject can access. This model is based on automation theory and domain separation. This means subjects are allowed only to perform predetermined actions against predetermined objects. When similar users are grouped into their own domain (that is, collective), the members of one subject domain cannot interfere with the members of another subject domain. Thus, subjects are unable to interfere with each other's activities.

Sutherland Model

The *Sutherland model* is an integrity model. It focuses on preventing interference in support of integrity. It is formally based on the state machine model and the information flow model. However, it does not directly indicate specific mechanisms for protection of integrity. Instead, the model is based on the idea of defining a set of system states, initial states, and state transitions. Through the use of only

these predetermined secure states, integrity is maintained and interference is prohibited.

A common example of the Sutherland model is its use to prevent a covert channel from being used to influence the outcome of a process or activity. (For a discussion of covert channels, see Chapter 9.)

Graham-Denning Model

The *Graham-Denning model* is focused on the secure creation and deletion of both subjects and objects. Graham-Denning is a collection of eight primary protection rules or actions that define the boundaries of certain secure actions:

- Securely create an object.
- Securely create a subject.
- Securely delete an object.
- Securely delete a subject.
- Securely provide the read access right.
- Securely provide the grant access right.
- Securely provide the delete access right.
- Securely provide the transfer access right.

Usually the specific abilities or permissions of a subject over a set of objects is defined in an access matrix (aka access control matrix).