CHAPTER 5

Security Models and Architecture

In this chapter, you will learn about the following topics:

- Computer architecture and the items that fall within it
- Trusted computing base and security mechanisms
- Components within an operating system
- Various security models
- Security criteria and ratings
- Certification and accreditation processes

Computer and information security covers many areas within an enterprise. Each area has security vulnerabilities and, hopefully, some corresponding countermeasures that raise the security level and provide better protection. Not understanding the different areas and security levels of network devices, operating systems, hardware, protocols, and applications can cause security vulnerabilities that can affect the environment as a whole.

Two fundamental concepts in computer and information security are the security model, which outlines how security is to be implemented—in other words, providing a "blueprint"—and the architecture of a computer system, which fulfills this blueprint.

A security policy outlines how data is accessed, what level of security is required, and what actions should be taken when these requirements are not met. The policy outlines the expectations of a computer system or device. A ***security model*** is a statement that outlines the requirements necessary to properly support and implement a certain security policy. If a security policy dictates that all users must be identified, authenticated, and authorized before accessing network resources, the security model might lay out an access control matrix that should be constructed so that it fulfills the requirements of the security policy. If a security policy states that no one from a lower security level should be able to view or modify information at a higher security level, the supporting security model will outline the necessary logic and rules that need to be implemented to ensure that under no circumstances can a lower-level subject access a higher-level object in an unauthorized manner. A security model provides a deeper explanation of how a computer operating system should be developed to properly support a specific security policy.

NOTE Individual systems and devices can have their own security policies. We are not talking about organizational security policies that contain management's directives. The systems' security policies and models they use should enforce the higher-level organizational security policy that is in place.

Security Models and Architecture

Computer security can be a slippery term because it means different things to different people. There are many aspects of a system that can be secured, and security can happen at various levels and to varying degrees. We have stated in previous chapters that information security is made up of the following main attributes:

- **Availability** Prevention of loss of access to resources and data
- **Integrity** Prevention of unauthorized modification of data
- **Confidentiality** Prevention of unauthorized disclosure of data

From here these main attributes branch off into more granular security attributes such as authenticity, accountability, non-repudiation, and dependability. How does a company know which of these it needs, to what degree they are needed, and if the operating systems and applications they use actually provide these features and protection? These questions get much more complex as one looks deeper into the questions and systems themselves. Companies are not just concerned about e-mail messages being encrypted as they pass through the Internet. They are also concerned about the confidential data stored in their databases, the security of their Web farms that are connected directly to the Internet, the integrity of data entry values going into applications that process business-oriented information, the internal users sharing trade secrets, the external attackers bringing down servers and affecting productivity, viruses spreading, the internal consistency of data warehouses, and much more. These issues not only affect productivity and profitability, but also raise legal and liability issues with securing data. Companies, and the management that runs them, can be held accountable if many of the previously mentioned issues go wrong. So it is, or at least it should be, very important for companies to know what security they need and how to be properly assured that the protection is actually being provided by the products they purchase.

Many of these security issues must be thought through before and during the design and architectural phase for a product. Security is best if it is designed and built into the foundation of operating systems and applications and not added on as an afterthought. Once security is integrated as an important part of the design, it has to be engineered, implemented, tested, audited, evaluated, certified, and accredited. The security that a product provides has to be rated on the availability, integrity, and confidentiality it claims. Consumers then use these ratings to determine if specific products provide the level of security they require. This is a long road, with many entities involved with different responsibilities. This chapter takes you from the steps necessary before actually developing an operating system to how these systems are evaluated and rated by governments and other agencies, and what these ratings actually mean.

However, before we dive into these concepts, it is important to understand how the basic elements of a computer system work. These elements are the pieces that make up any computer's architecture.

Computer Architecture

Put the processor over there by the plant, the memory by the window, and the secondary storage upstairs.

Computer architecture encompasses all the parts of a computer system necessary for it to function, including the operating system, memory chips, circuits, hard drive, security components, buses, and networking components. The interrelationships and internal working of all of these parts can be quite complex, and making them work together in a secure fashion is comprised of complicated methods and mechanisms. Thank goodness for the smart people who figured this stuff out! Now it is up to us to learn how they did it and why.

The more you understand how these different pieces work and process data, the more you will understand how vulnerabilities actually occur and how countermeasures work to impede and hinder vulnerabilities from being introduced, found, and exploited.

Central Processing Unit

Hey, when is it my turn to use the CPU? Answer: When the control unit says it's your turn.

The ***central processing unit (CPU)*** is a microprocessor that contains a control unit, an ***arithmetic logic unit (ALU)***, and registers, which are holding places for data and instructions. The ***control unit*** manages and synchronizes the system while different applications' code and operating system instructions are being executed. It determines what application instructions get processed and in what priority and time slice. It controls when instructions are executed and this execution enables applications to process data. The control unit does not actually process the data; it is like the traffic cop telling traffic when to stop and start again, as shown in Figure 5-1.

The chips within the CPU cover only a couple of square inches, but contain over a million transistors. All operations within the CPU are performed by electrical signals at different voltages in different combinations, and each transistor holds this voltage, which represents 0s and 1s to the computer. The CPU contains registers that point to memory locations that contain the next instructions to be executed and enable the CPU to keep status information of the data that needs to be processed. The ALU performs mathematical functions and logical operations on data. The ALU can be thought of as the brain of the CPU and the CPU as the brain of the computer.

Software holds its instructions and data in memory. When action needs to take place on the data, the instructions and data are passed to the CPU portion of the system, as shown in Figure 5-2. The CPU components handle the flow of instructions from the operating system and applications. The data that needs to be processed is passed into the instruction registers. When the control unit indicates that the CPU can process them, they are passed to the CPU for actual processing, number crunching, and data manipulation. The results are sent back to the computer's memory so the application can use this processed data to continue its tasks.

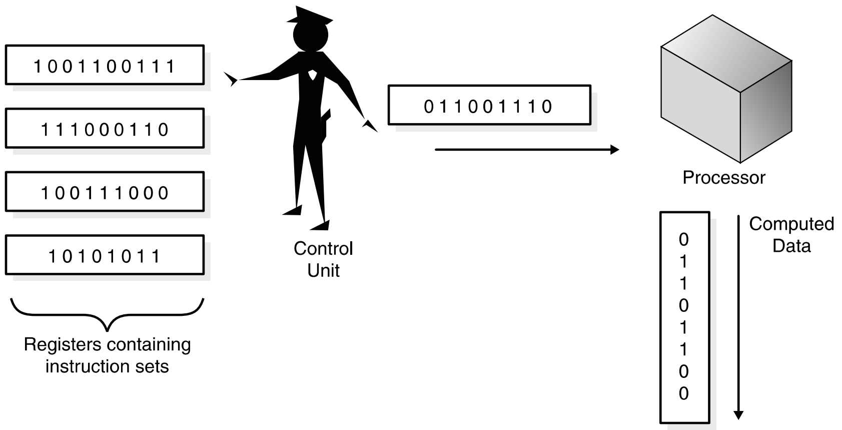


Figure 5-1 The control unit works as a traffic cop, indicating when instructions are sent to the processor.

Instructions and data are held in registers until needed by the CPU. The software instructions are first ported into the CPU because these instructions indicate what actually needs to happen to the data. The registers are not permanent storage areas, but a temporary memory area to hold instructions that are to be interpreted by the CPU and used for data processing.

The data being processed is entered into the CPU in blocks at a time. If the software instructions do not properly set the boundaries for how much data can come in as a block (for example, 64 bits at a time), extra data can slip in and be executed. This is how ***buffer overflows*** work. If a buffer overflow takes place, it is due to the operating system or

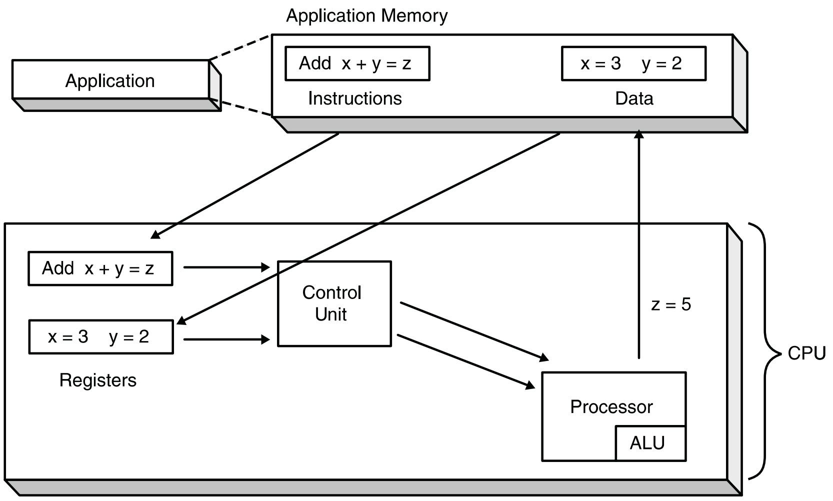


Figure 5-2 Instructions and data are passed to the CPU for processing.

application software instructions that are processing the data, not the computer system itself. If extra data slips in, it can be executed in a privileged mode and cause disruption and lead to unauthorized access or different degrees of damage. This can result in the computer system freezing, rebooting, or allowing data corruption. Buffer overflows can be corrected by well-written programs that verify how much data is being accepted and sent to the CPU at any given point in time.

A CPU's time and processing power has to be shared between many tasks. Software and system interrupts are used to make sure that all data is processed in a timely manner and priorities are used to ensure that critical tasks are performed before less important tasks.

Memory

The operating system instructions, applications, and data are held in memory, but so are the basic input/output system (BIOS), device controller instructions, and firmware. They do not all reside in the same memory location or even the same type of memory. The different types of memory, what they are used for, and how each is accessed can get a bit confusing because the CPU deals with several different types for different reasons.

The following paragraphs quickly outline the different types of memory within a computer system.

Random access memory (RAM) is a type of temporary storage facility where data can be held and altered. It is used for read/write activities by the operating system and applications. It is described as *volatile* because if the computer's power supply is terminated, then all information within this type of memory is lost. There are different types of RAM, but the most well-known types are dynamic and static RAM. *Static RAM* lives up to its name, because when it stores data, it stays there without the need of being continually refreshed. *Dynamic RAM*, on the other hand, requires that the data held within it be periodically refreshed because the data can dissipate and decay.

Read-only memory (ROM) is a *nonvolatile* storage facility, meaning that when a computer's power is turned off, the data is still held within the memory chips. For the most part, when data is inserted into ROM memory chips, it cannot be altered. The software that is stored within ROM is called *firmware*.

Erasable and programmable read-only memory (EPROM) can be modified, deleted, or upgraded. EPROM holds data that can be electrically erased or written to.

References

How RAM Works: **www.howstuffworks.com/ram.htm**

Unix/Linux Internals Course and Links: **www.softpanorama.org/Internals**

Cache Memory

I am going to need this later, so I will just stick it into cache for now.

Cache memory is a type of memory that is used for high-speed writing and reading activities. It holds instructions and data from primary storage and is accessed when application instructions and data are being executed. When the system assumes that it will need to access specific information many times throughout its processing activities, it will store it in cache memory so that it is easily and quickly accessible. Data being retrieved from cache can be accessed much more quickly than if it was stored in real memory; thus, it affects the overall speed of the computer system. Therefore, any information needed by the CPU very quickly, and very often, is often stored in cache memory.

An analogy is how the brain stores information that is used often. If one of Marge's primary functions at her job is ordering parts and telling vendors the company's address, this information is held within a portion of her brain that is easily and quickly accessible for Marge when she needs it. This information is held in a type of cache. If Marge was asked to recall her third grade teacher's name, this information would not necessarily be held in cache memory, but in a more long-term storage facility within her noggin. The long-term storage within her brain is comparable to a system's hard drive. It takes more time to track down and return information from a hard drive than specialized cache memory.

Memory Mapping

Okay, here is your memory, here is my memory, and here is Bob's memory. No one use each other's memory!

Because there are different types of memory holding different types of data, a computer system does not want to let every user, process, and application access all types of memory anytime they want to. Access to memory needs to be controlled to ensure that data does not get corrupted. This type of control takes place through memory mapping and addressing.

The CPU is one of the most trusted components within a system, and therefore it can access memory directly. It uses physical addresses instead of pointers to memory segments. The CPU has physical wires connecting it to the memory chips within the computer. Because there are physical wires connecting the two types of components, physical addresses are used to represent the intersection between the wires and the transistors on a memory chip. Software does not use physical addresses; instead, it uses virtual or logical memory. Accessing memory indirectly provides an access control layer between the software and the memory, which is done for protection and efficiency. Figure 5-3 illustrates how the CPU can access memory directly using physical addresses and how software must use memory indirectly through a memory mapper.

Let's look at an analogy. You would like to talk to Mr. Marshall about possibly buying some acreage in Iowa. You don't know Mr. Marshall personally, and you do not want to give out your physical address and have him show up at your doorstep. Instead, you would like to use a more abstract and controlled way of communicating, so you give Mr. Marshall your phone number so you can talk about the land and you can make a determination if you want to meet Mr. Marshall in person. The same type of thing happens in computers. When a computer runs software, it does not want to expose itself unnecessarily to software written by good and bad programmers. Computers enable software to use memory indirectly using index tables and pointers, instead of giving them the right to access the memory directly. Only the system itself can access memory directly and programs can access the memory indirectly, but it is the same memory storage. This is one way the computer system protects itself.

When a program attempts to access memory, its access rights are verified and then instructions and commands are carried out in a way to ensure that badly written code does not affect other programs or the system itself. Applications, and their processes, can only

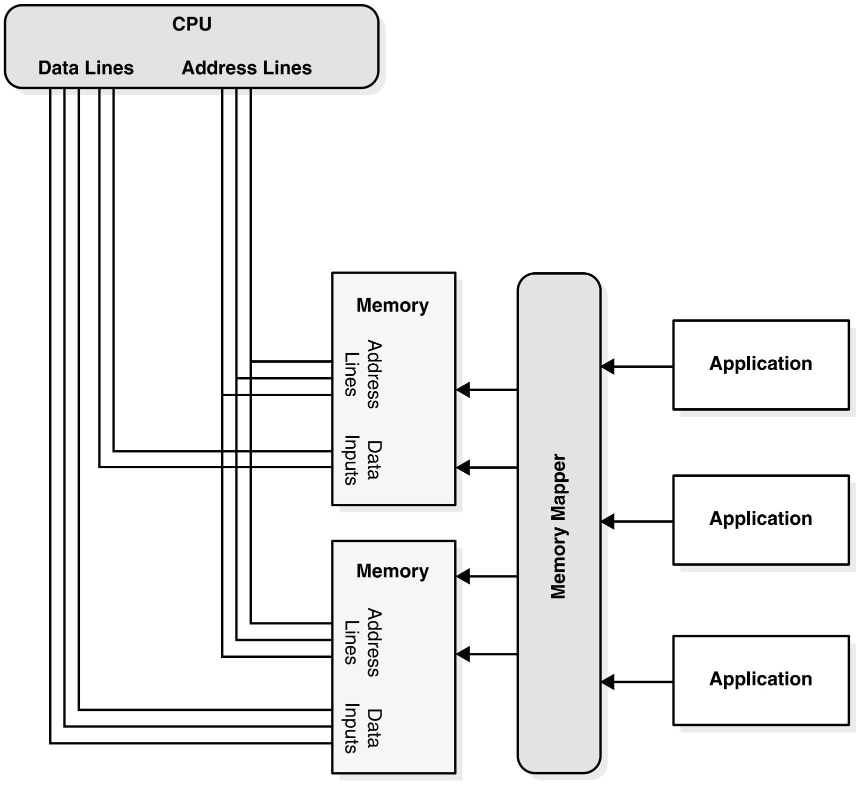


Figure 5-3 The CPU and applications access memory differently.

access the memory allocated to them, as shown in Figure 5-4. This type of memory architecture provides protection and efficiency.

If programs accessed data held in memory directly, each program would have to wait until the prior program is done before it could access and process data. Mapped memory enables different programs to access the data and perform their own separate functions on it in a more economical and resourceful manner.

Secondary storage is considered nonvolatile storage media, which can be the computer's hard drive, floppy disks, or CD-ROM.

When RAM and secondary storage are combined, the result is ***virtual storage***. The system uses hard drive space to extend RAM memory space capability. The hard drive space that is used to extend the RAM memory capabilities is incremented in pages. When a system fills up its volatile memory space, it will write data from memory onto the hard drive. When a program or user requests access to this data, it is brought from the hard drive back into memory. This process is called *paging*. Accessing data that is kept in pages on the hard drive takes more time than accessing data kept in memory because actual disk access has to take place; however, the payoff is that it seems as

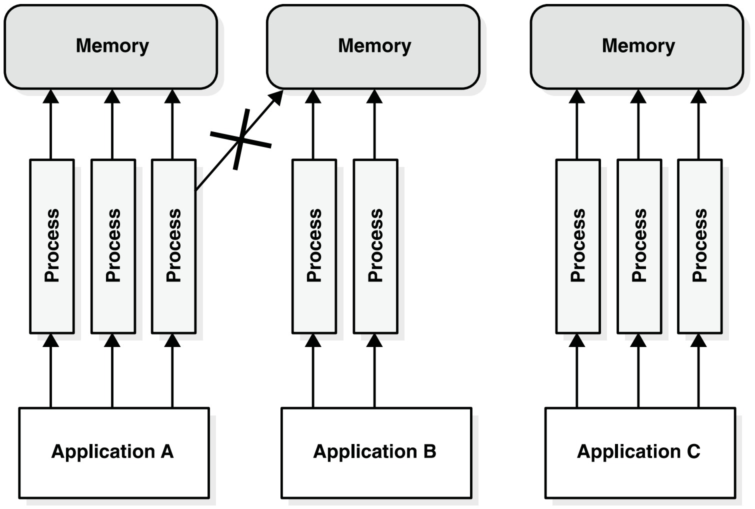


Figure 5-4 Applications, and the processes they use, only access their own memory segments.

though the system can hold an incredible amount of information in memory, as shown in Figure 5-5.

The different types of memory we looked at are summed up here:

- **Primary storage** Main memory directly accessed by the CPU and indirectly accessed by applications, considered volatile memory
- **Secondary storage** Nonvolatile storage (floppy disk, CD-ROM disk, hard drive, and so on)
- **Virtual storage** RAM and secondary storage used together
- **RAM** Random access memory, where instructions and data are placed when being executed

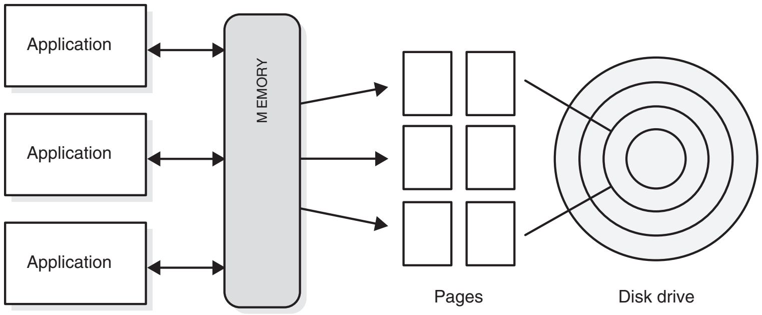


Figure 5-5 Systems send data from memory to the hard drive in units of pages to enable memory to reach gigabyte sizes.

CPU Modes and Protection Rings

If I am corrupted, very bad things can happen. Response: Then you need to go into ring 0.

If an operating system is going to be stable, it must be able to protect itself from its users and their applications. This requires the capability to distinguish between operations performed on behalf of the operating system itself and operations performed on behalf of the users or applications. This can be complex because the operating system software can be accessing memory segments, sending instructions to the CPU for processing, and accessing secondary storage devices. Each user application (e-mail client, antivirus, Web browser, word processor, personal firewall, and so on) can be attempting the same types of activities at the same time. The operating system must keep track of all of these events and ensure that none of them violates the security policy.

The operating system has several protection mechanisms to ensure that processes do not negatively affect each other or the critical components of the system itself. One has already been mentioned: memory segments. Another security mechanism that the system uses is ***protection rings***. These rings provide strict boundaries and definitions for what the processes that work within each ring can access and what operations they can successfully execute. The processes that operate within the inner rings have more privileges than the processes operating in the outer rings. This is because the inner rings only permit the most trusted components and processes to operate within them. Although operating systems can vary in the number of protection rings, processes that execute within the inner rings are usually referred to as existing in a ***privileged***, or supervisor, mode. The processes working in the outer rings are said to execute in a ***user mode***.

NOTE The actual ring architecture that is being used by a system is dictated by the processor and the operating system. The hardware chip (processor) is constructed to work with a certain number of rings, and the operating system must be developed to also work in this ring structure. This is one reason why one operating system platform may work with an Intel chip, but not an Alpha chip. They have different architectures and ways to interpret instruction sets.

Operating system components operate in a ring that gives them the most access to memory locations, peripheral devices, system drivers, and sensitive configuration parameters. Because this ring provides much more dangerous access to critical resources, it is the most protected. Applications usually operate in ring 3, which limits the type of memory, peripheral device, and driver access activity, and is controlled through the operating system functions or system calls. The different rings are illustrated in Figure 5-6. The type of commands and instructions that are sent to the CPU from applications in the outer rings are more restrictive in nature. If an application tries to send instructions to the CPU that fall outside of its permission level, the CPU treats this violation as an exception and may show a general protection fault, panic, or exception error and attempt to shut down the offending application.

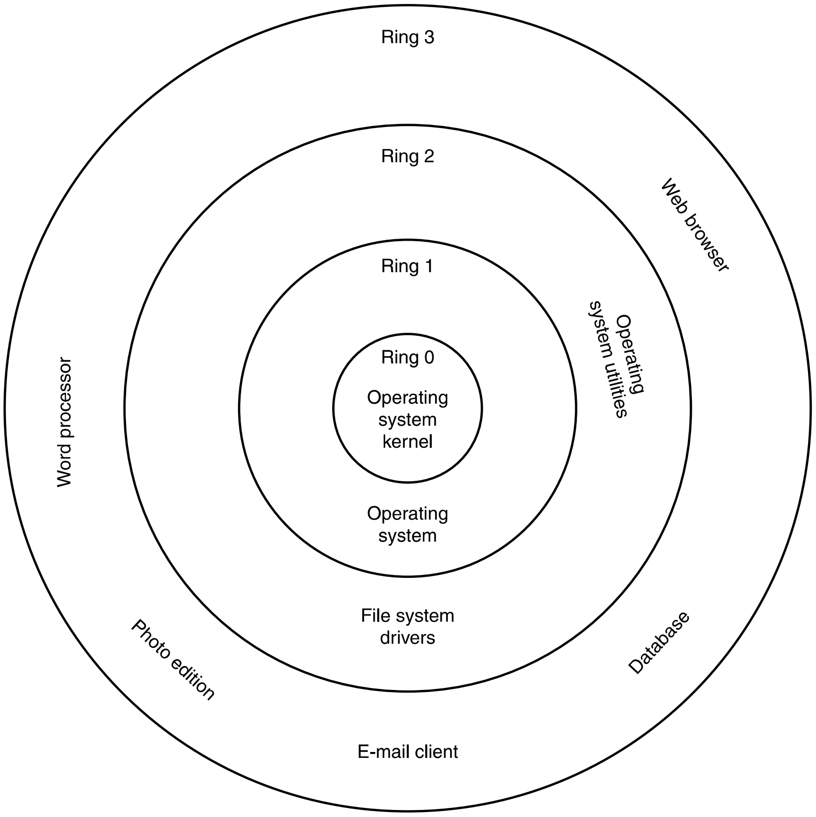


Figure 5-6 More trusted processes operate within lower-numbered rings.

Protection rings support the availability, integrity, and confidentiality requirements of multitasking operating systems. Many systems use four protection rings:

- **Ring 0** Operating system kernel
- **Ring 1** Remaining parts of the operating system
- **Ring 2** I/O drivers and utilities
- **Ring 3** Applications and programs

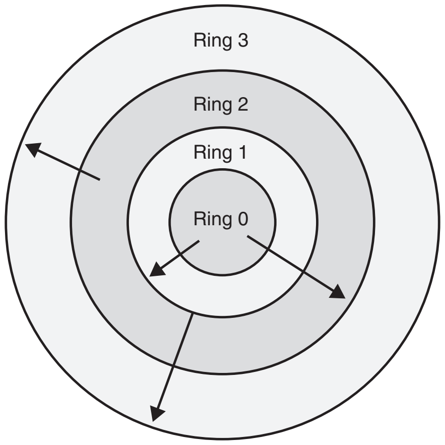
These protection rings provide an intermediate layer between subjects and objects, and are used for access control when a subject tries to access an object. The ring determines the access level to sensitive system resources. The lower the number, the greater the amount of privilege that is given to the process that runs within that ring. Each subject and object is logically assigned a number (0 through 3) depending upon the level of trust the operating system assigns it. A subject in ring 3 cannot directly access an object in ring 1, but subjects in ring 1 can directly access an object in ring 3. Entities can only access objects within their own ring and outer rings, as shown in Figure 5-7. When an application needs access to components in rings that it is not allowed to directly access, it makes a request of the operating system to perform the necessary tasks. This is handled through system calls, where the operating system executes instructions not allowed in user mode. The request is passed off to an operating system service, which works at a higher privilege level and can carry out the necessary task.

When the operating system executes instructions for processes in rings 0 and 1, it operates in system mode or privileged mode. When the operating system executes instructions for applications and processes in ring 3, it operates in user mode. User mode provides a much more restrictive environment for the application to work in, which in turn protects the system from misbehaving programs.

Process Activity

Computers can run different applications and programs at the same time. They have to share resources and play nice with each other to ensure a stable and safe computing environment that maintains its integrity. Some memory, data files, and variables are actually shared between different applications. It is critical that more than one process does not attempt to read and write to these items at the same time. The operating system is the master program that prevents this type of action from taking place and ensures that programs do not corrupt each other's data held in memory. The operating system works with the CPU to provide time slicing and interrupts to ensure that processes are provided with adequate access to the CPU. This also ensures that critical system functions are not negatively affected by rogue applications.

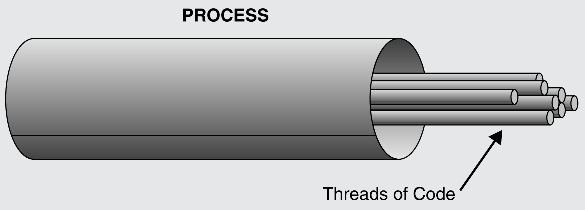
Figure 5-7 Processes in inner rings can directly access processes in outer modes but not vice versa.



Ring 0—Operating system kernel
Ring 1—Remaining parts of the operating system
Ring 2—I/O drivers and utilities
Ring 3—Applications and programs

Process versus Thread

A *process* is a program in execution that works in its own address space and can only communicate with other processes in a controlled manner. A *thread* represents a piece of code that is executed within a process, as shown here. A process can have one or more threads running at one time, depending upon the tasks it needs to carry out.

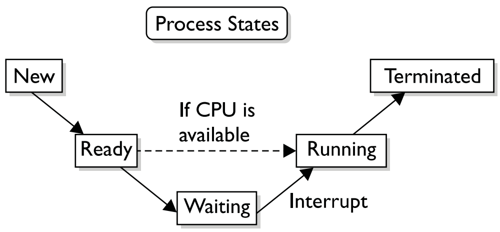


Operating States

I am in a problem state. Is that bad? Answer: Nope, the CPU is just carrying out your request.

The operating system itself can be in one of two modes at any given point in time. The operating system can operate in privileged (supervisor state) or user mode (problem state) depending on what process made a request to execute instructions, but there are also different *operating states* in which the different processes can work within. The following outlines the four possible states a process can operate in:

- **Stopped** The process is not running.
- **Waiting** The process is waiting for an interrupt to be able to be interact with the CPU.
- **Running** The process's instructions are being executed by the CPU.
- **Ready** The process is available to be used and waiting for an instruction.



These protection rings, operational modes, and operating states work together to ensure a secure and predictable operating environment. They enable the system to protect itself and work to protect the availability, integrity, and confidentiality of the system, the information that is being processed, and the applications installed.

When an application runs on a computer, it thinks it is the only program running. It does not necessarily know that it is sharing resources with several other types of programs, the operating system, and other processes. This simplifies the issues that programmers need to be concerned about when writing an application. Because the application thinks it is the only one executing, it needs to be presented with an environment that reflects this type of reality. This can be done through virtual machines and virtual memory. The operating system creates a virtual environment (virtual machine) for the application to work in and allots it a segment of virtual memory, as shown in Figure 5-8. Another application could have its own virtual machine and virtual address segment, and the two would not know that each other even existed. This way the two applications do not interact with each other's data in memory or step on each other's toes while being executed by the CPU. It is a very orchestrated, controlled, and timed event-oriented atmosphere. An analogy is two people with two different offices. George has his office, computer, file cabinet, closet, and window. Marge has an office with the same materials. If George and Marge had to use the same computer, file cabinet, and closet, they could easily move each other's items around, accidentally delete each other's files, or wear each other's coat home. However, because they each have their own office and components, these events will not take place.

The granularity of this type of separation and protection becomes very detailed within a computer system. A process is used to request resource access and carry out data manipulation activities, and is the mechanism used to enable applications and the operating system to communicate. A process is like a phone call. If an application wants to print a form, it calls (or communicates via a process) the operating system, which calls the printer driver, and the printer driver calls the printer. These processes can only communicate in a way that has been previously approved by the operating system and outlined in the security model. The application cannot make a direct call to the printer driver because it does not operate in a privileged ring; thus, it does not have the necessary type of authority. The security architecture of the operating system dictates this type of communication authority.

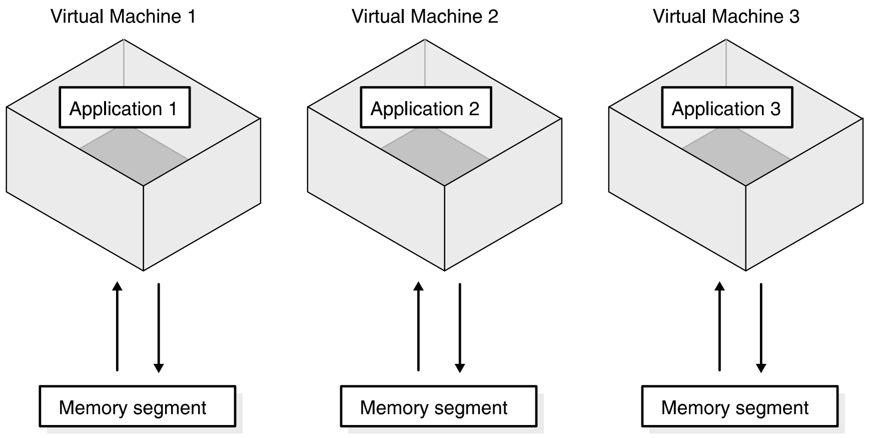


Figure 5-8 Virtual machines only access their allotted memory segments.

One application can make several calls at one time. It can ask to print a form, accept a fax, display data to the monitor, and ask for an address from the address book. These different calls use different threads and are an example of ***multithreading***. A system that can process more than one request at a time is capable of multithreading. If the CPU can process more than one process, or task, at one time, it is considered a ***multitasking*** system. If a computer has more than one CPU (also called processor), it can use them in parallel to execute instructions; this activity is called ***multiprocessing***. ***Multiprogramming*** is the interleaved execution of two or more programs by a CPU. The system can run more than one application, but not necessarily perform multithreading or multitasking. Older operating systems used multiprogramming instead of multitasking, but this was not an effective practice because a process could commit a resource and no other processes could use that resource until it was properly released by the first process. This approach to operating system operations gives the processes too much control. Newer operating systems gave this control to the operating system instead of the individual processes.

An operating system has to protect its integrity and ensure that users do not accidentally or intentionally access someone else's data. Multitasking systems interleave the execution of processes belonging to different users, which adds complexity to protecting itself and users' data. The operating system uses logical separation between users' data, which takes place in file and memory management.

Input/Output Device Management

When a user chooses to print a document, or a word processor displays a file previously stored, or if a user saves files to a zip drive, these requests are going from the application the user is working in, through the operating system, and to the device requested. The operating system uses a device driver to communicate to a device controller, which is an electrical component with its own software used to provide a communication path that enables the device and operating system to exchange data. The operating system sends commands to the device controller's registers and the controller then writes data to the peripheral device or extracts data to be processed by the CPU, depending on the given commands. This communication happens through specific communication channels dedicated just for these types of functions. Once the device controller retrieves the requested information, it is stored in main memory for the operating system, application, and CPU to perform their necessary functions.

It is important that the devices and computer resources are properly accessed and released. Different operating systems handle accessing devices and resources differently. For example, Windows NT is considered a more stable and safer data processing environment than Windows 9*x* because applications cannot make direct requests to hardware devices. Windows NT and Windows 2000 have a much more controlled method of accessing devices than Windows 9*x*. This helps to protect the system from badly written code that does not properly request and release resources. This level of protection helps to ensure the resources' integrity and availability.

Proper input/output management is a core component of operating systems. When a request for a resource (memory allocation, printer, secondary storage devices, disk space) is made, certain data structures are built and processes are dedicated for this action to take place. Once the action takes place (a document is printed, a file is saved, or data is retrieved from the drive), the program, or operating system, needs to tear down these built structures and release the resources back into a pool to be available for other programs and processes. If this does not happen properly, a ***deadlock*** situation can occur or a computer may not have enough resources to process other requests (result of a denial-of-service attack). A deadlock situation can happen with peripheral devices, memory, database, or file access. If Sean accesses a database and his process never properly releases the necessary resources back to the system, when Debbie attempts to perform the same action, a deadlock situation can occur and Sean, Debbie, and the system that contains the database can be negatively affected. This has a direct correlation to a main security principle: availability. The database software should apply a software lock when Sean accesses information in the database, which will ensure that no one else, including Debbie, can access and modify the same data until Sean is done.

Another common deadlock situation is when process A commits resource 1 and needs to use resource 2 to properly complete its task. But process B has committed resource 2 and needs resource 1 to finish its job. So both processes are "hung" because they do not have the resources they need to finish the function they are to carry out. This activity does not take place as much as it used to because better programming is usually performed. Also the environment (operating system, database software, application) now has the intelligence to detect this activity and will either release committed resources or control how resources are going to be properly shared between processes.

Operating systems have different methods of dealing with resource requests and releases and solving deadlock situations. In some systems, if a requested process is unavailable for a certain period of time, some operating systems will kill that process. This action releases the process from the application that had committed it, releases the supporting system resources necessary to commit this resource, and restarts the process so it is "clean" and available to be used by other applications. Other operating systems might require a program to request all of the resources it needs *before* it actually starts executing instructions or requires a program to release its currently committed resources before being able to acquire more. Input/output management is a core responsibility of the operating system, along with memory management, CPU tasks, and file system management.

Tying It Together

Each of the topics discussed in earlier sections have complete books written on them because of their complexity and importance to the computing world. Each topic also relates in some way to a part of security. An operating system is a complex beast that performs many complicated tasks in order to provide a useful and secure work environment for a user. It must recognize each process and program that is running, make access decisions, support system calls, protect memory segments, validate requested commands, properly process instructions, and ensure that none of these activities introduce vulnerabilities that can violate the security policy of the system. It does this by enforcing protection rings, mapping memory, implementing virtual machines, working in different states, and assigning trust levels to each and every process. These are integral parts of the security model for the operating system.

System Architecture

Designing a system from the ground up is a complicated task and has many intricate and abstract goals that have to be achieved through mathematics, logic, design, programming code, and implementation. There are fundamental design decisions that need to be made when constructing a system. Security is only one of the goals of a system, but it is the goal security professionals are most concerned about.

Availability, integrity, and confidentiality can be enforced at different places within an enterprise. For example, a company may store customer credit card information in a database that many users can access. This information, obviously, requires protection to ensure that it is not accessed or modified in an unauthorized manner. We start with general questions and gradually drill down into the details. Where should this protection be placed? Should there be access controls that screen users when they log in and assign them their rights at that point dictating what data they can and cannot access? Should the data files holding the credit card information be protected at the file system level? Should protection be provided by restricting users' operations and activities? Or should there be a combination of all of these? The first and most general question is "Where should the protection take place: at the user's end, where the data is stored, or by restricting user activities within the environment?" This is illustrated in Figure 5-9.

Once these general questions have been answered, the placement of the mechanisms needs to be addressed. Security mechanisms can be placed at the hardware, kernel, operating system, services, or program layers. At which layer(s) should security mechanisms be implemented? If protection is implemented at the hardware layer, the protection mechanisms will be more simplistic and provide broad and general protection. As we ascend up the layers, more complexity is added, and functionality becomes more specific and granular. The top layer holds the most complexity because it is directed toward providing the user with a vast amount of functionality and options. Functionality and complexity of security increases as it approaches the layers that are closer to the user. The increased complexity lowers the assurance levels of the security mechanisms. This is shown in Figure 5-10.

The more complex a security mechanism becomes, the less assurance it provides. This is because the complexity of the mechanism demands more technical understanding from the individuals who install, test, maintain, and use it. The more complex the tools, the more chances there are for errors, and therefore increased chances for security compromises. The more complex the security mechanism, the harder it is to fully test it under

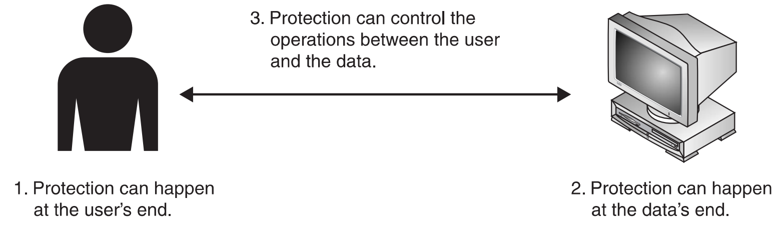

3. Protection can control the operations between the user and the data.
1. Protection can happen at the user's end.
2. Protection can happen at the data's end.

Figure 5-9 Security can take place at three main areas.

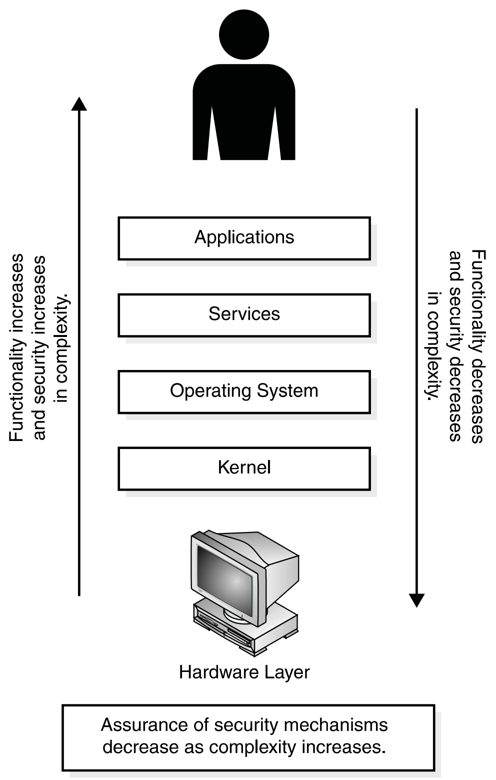

Applications
Services
Operating System
Kernel
Hardware Layer
Functionality increases and security increases in complexity.
Functionality decreases and security decreases in complexity.
Assurance of security mechanisms decrease as complexity increases.

Figure 5-10 As functionality increases, complexity increases and the assurance of security decreases.

all possible conditions. On the other hand, simplistic mechanisms cannot provide the desired richness of functionality and options, although they are easier to install, maintain, use, and test. So the tradeoffs between functionality and assurance need to be fully understood to make the right security mechanism choices when designing a system.

Once the designers have an idea of what the security mechanisms should focus on (users, operations, or data), what layer(s) the mechanisms should be placed at (hardware, kernel, operating system, services, or program), and the complexity of each mechanism, the mechanisms need to be built and integrated in a way to have a proper relationship with other parts of the system.

The first step is to decide what system mechanisms need to be trusted and specify how these entities can interact in a secure manner. Although it might seem that you would want to trust all the components within the system, this would cause too much overhead, complexity, and performance bottlenecks. For a mechanism to be trusted, it means that it protects itself and the data it is processing, it performs in predictable and secure manners, and it does not adversely affect other trusted or untrusted mechanisms. In return, these trusted components have access to more privileged services, have direct access to memory, usually have higher priority when requesting CPU processing time, and have more control over system resources. So the trusted subjects and objects need to be identified and distinguished from the untrusted ones and placed into defined subsets.

Defined Subset of Subjects and Objects

I totally trust you. You can do whatever you want. Response: I want to leave.

As stated previously, not all components need to be trusted and therefore do not fall within the ***trusted computing base (TCB)***. The TCB is defined as the total combination of protection mechanisms within a computer system. The TCB includes hardware, software, and firmware. These are part of the TCB because the system is sure that these components will enforce the security policy and not violate it.

The components that do fall under the TCB need to be identified and their accepted capabilities defined. For example, a system that has a lower trust level rating may permit all authenticated users to access and modify all files on the computer. This subset of subjects and objects is large and the relationship between them is loose and relaxed. A system with a much higher trust level rating may permit only two subjects to access all files on a computer system, and only one of those subjects can actually modify all the files. This subset is much smaller and the rules being enforced are more stringent and detailed.

Again, it depends upon what type of system the developers are aiming at building. If developers want to develop a system that achieves an Orange Book security rating of D (very low), then what makes up the TCB or not is not much of an issue because the system will not be expected to provide a very high level of security. However, if the developers want to develop a system with an Orange Book rating of B2 or B1 (much higher than rating D), they will need to specify what makes up the TCB. They need to enforce strict rules that dictate how subjects and objects interact. The developers also need to ensure that these items are identified, authenticated, and audited because these are the components that will be scrutinized, tested, and evaluated before a rating of B2 or B1 can be given. (The Orange Book and its ratings are addressed in the section "The Orange Book" later in the chapter.)

Trusted Computing Base

The term "trusted computing base" originated from the Orange Book and does not address the level of security a system provides, but the level of trust, albeit from a security sense. This is done because no computer system can be totally secure. The types of attacks and vulnerabilities change and evolve over time, and with enough time and resources, most attacks can become successful. However, if a system meets a certain criteria, it is looked upon as providing a certain level of trust, again, from a security perspective.

The TCB does not just address operating systems, because a computer system is not made up of only an operating system. The TCB addresses hardware, software components, and firmware. Each can affect the computer's environment in a negative and positive manner, and each has a responsibility to support and enforce the security policy of that particular system. Some components and mechanisms have direct responsibilities in supporting the security policy, such as firmware that will not let a user boot a computer from a floppy disk or the memory manager that will not let users overwrite other users' data. Then there are components that do not enforce the security policy, but must behave properly and not violate the trust of a system. The types of violations a component could cause against the system's security policy could be an application that attempts to make a direct call to a piece of hardware instead of using the proper calls through the operating system, a program that attempts to read data outside of its approved memory space, or a piece of software that does not properly release resources after use.

Not every part of a system needs to be trusted. Part of evaluating the trust level of a system is to identify the architecture, security services, and assurance mechanisms that make up the TCB. It must be shown how the TCB is protected from accidental or intentional tampering and compromising activity. For systems to achieve a higher trust level, they must meet well-defined TCB requirements, and the details of their operational states, developing stages, testing procedures, and documentation will be reviewed with more granularity than systems that are attempting to achieve a lower trust rating.

By using specific security criteria, trust can be built into a system, evaluated, and certified. This approach can provide a measurement system for customers to use when comparing one system to another. It also gives vendors guidelines on what expectations are put upon their systems and provides a common security rating so when one group talks about a C2 rating, everyone else is on the same page and understands what these terms mean.

The Orange Book defines a trusted system as hardware and software that utilize measures to protect the integrity of unclassified or classified data for a range of users without violating access rights and the security policy. It looks at all protection mechanisms within a system to enforce the security policy and provide an environment that will behave in a manner expected of it. This means that each layer of the system must trust the underlying layer to perform the expected functions, provide the expected functionality, and operate in an expected manner under many different situations. When the operating system makes calls to hardware, it is anticipating data to be returned in a specific data format and to behave in a consistent and predictable manner. Applications that run on top of the operating system expect to be able to make certain system calls and receive the required data in return and to be able to operate in a reliable and dependable environment. Users expect the hardware, operating system, and applications to perform in particular fashions and provide a certain level of functionality. For all of these actions to behave in such predicable manners, the requirements of a system must be addressed in the planning stages of development, not afterwards.

Security Perimeter

Now, whom do we trust? Answer: Anyone inside the security perimeter.

As stated previously, not every component and resource falls within the TCB, so some resources fall outside of this imaginary boundary referred to as the ***security perimeter***. A perimeter is a boundary that divides the trusted from the untrusted. For the system to stay in a secure and trusted state when a component within the TCB needs to communicate with a component outside of the TCB, precise communication standards must be developed to ensure that this type of communication cannot bring on unexpected security compromises. This type of communication is handled and controlled through interfaces.

For example, a resource that is within the boundary of the TCB, or security perimeter, must not pass confidential information to resources outside the TCB. The resource within the TCB must also be careful about the commands and information it accepts from less-trusted resources. These limitations and restrictions are built into the interfaces that permit this type of communication to take place and are the mechanisms that enforce the security perimeter, as shown in Figure 5-11. Communication between trusted components and untrusted components needs to be controlled to ensure that confidential information does not flow in an unintended way.

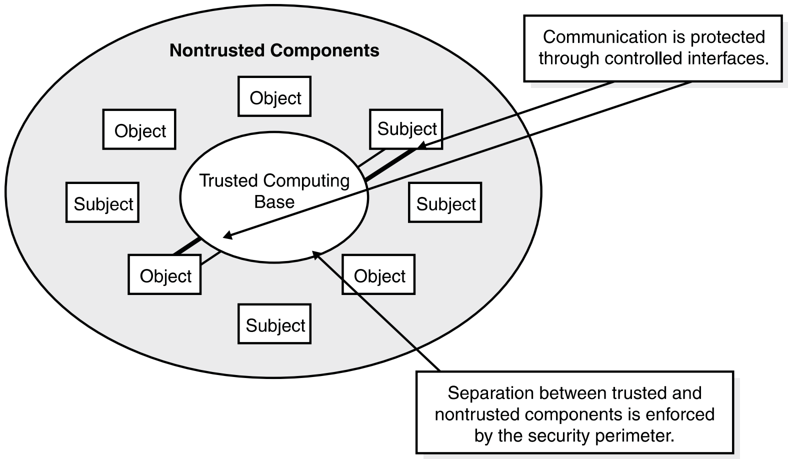

Nontrusted Components
Object
Object
Subject
Subject
Trusted Computing Base
Subject
Object
Object
Subject
Communication is protected through controlled interfaces.
Separation between trusted and nontrusted components is enforced by the security perimeter.

Figure 5-11 Interfaces control the communication between trusted and untrusted processes.

NOTE The TCB and security perimeter are not physical entities, but conceptual constructs used by the operating system to delineate between trusted and untrusted components. They are illustrated here to show the relationship of the TCB, security perimeter, and different components.

Reference Monitor and Security Kernel

Up to now, our computer system architecture developers have accomplished many things in developing their system. They have defined where the security mechanisms will be located (hardware, kernel, operating system, services, or program), they have defined the objects that are within the TCB and how those components will interact with each other. The security perimeter that separates the trusted components and the untrusted components has been constructed. They have developed proper interfaces for these entities to communicate securely. Now they need to develop and implement a mechanism that ensures that the subjects that access objects have been given the necessary permissions to do so. This means the developers need to develop and implement a reference monitor and security kernel.

The ***reference monitor*** is an abstract machine, which mediates all access subjects have to objects to ensure that the subjects have the necessary access rights and to protect the objects from unauthorized access and destructive modification. For a system to achieve a higher level of trust, it must enforce subjects (programs, users, or processes) to be fully authorized prior to accessing an object (file, program, or resource). It must be made certain that the subject has been granted access privileges prior to letting the subject use the requested object. The reference monitor is an access control concept, not an actual physical component, and that is why it is normally referred to as the "reference monitor concept."

The ***security kernel*** is made up of mechanisms that fall within the TCB and implements and enforces the reference monitor concept. The security kernel is made up of hardware, firmware, and software components that mediate all access and functions between subjects and objects. The security kernel is the core of the TCB and is the most commonly used approach to building trusted computing systems. There are four main requirements of the security kernel:

- It must provide isolation for the processes carrying out the reference monitor concept and they must be tamperproof.
- The reference monitor must be invoked for every access attempt and must be impossible to circumvent. Thus, the reference monitor must be implemented in a complete and foolproof way.
- The reference monitor must be verifiable as being correct. This means that all decisions made by the reference monitor should be written to an audit log, and verified as being correct.
- It must be small enough to be able to be tested and verified in a complete and comprehensive manner.

These are the requirements of the reference monitor; therefore, they are the requirements of the components that provide and enforce the reference monitor concept—the security kernel.

These issues work in the abstract, but are implemented in the physical world of hardware devices and software code. The assurance that the components are enforcing the abstract idea of the reference monitor is proved through testing and functionality.

NOTE The reference monitor is a concept where an abstract machine mediates all accesses to objects by subjects. The security kernel is the hardware, firmware, and software of a TCB that implements this concept. The TCB is the totality of protection mechanisms within a computer system that work together to enforce a security policy. The TCB contains the security kernel and all other security protection mechanisms.

Figure 5-12 provides a quick analogy to show you the relationship between the components that make up the kernel, the kernel itself, and the reference monitor concept. Individuals make up a society. The individuals can represent the components and the society can represent the kernel. For a society to have a certain standard of living, it needs to react in specific ways, which is why we have laws. The laws represent the reference monitor, which enforces proper activity. Each individual is expected to stay within the bounds of the laws and act in specific ways so society as a whole is not adversely affected and the standard of living is not threatened. The components within a system must stay within the bounds of the reference monitor's laws so that they will not adversely affect other components and threaten the security of the system.

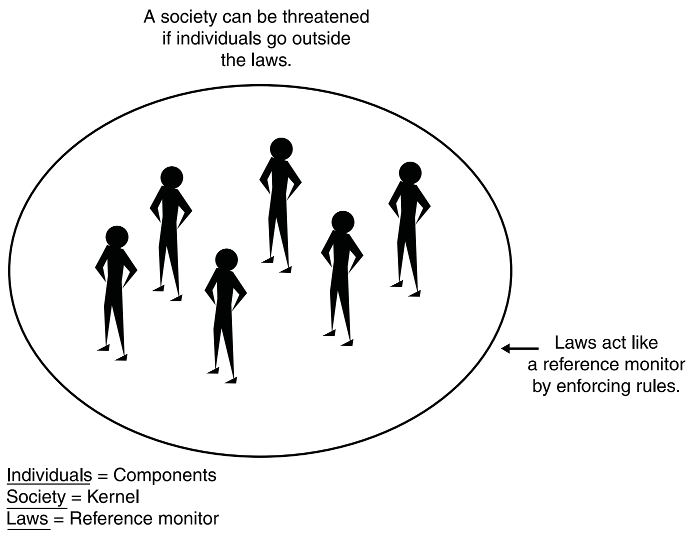

A society can be threatened if individuals go outside the laws.
Laws act like a reference monitor by enforcing rules.
Individuals = Components
Society = Kernel
Laws = Reference monitor

Figure 5-12 Components, like people, need to say inside the bounds of the rules.

References

Rationale Behind the Evaluation Classes: **www.kernel.org/pub/linux/libs/security/Orange-Linux/refs/Orange/OrangeI-II-6.html**

The Reference Monitor Concept: **citeseer.nj.nec.com/299300.html**

Implementing Complete Mediation: **www.cs.cornell.edu/html/cs513-sp99/NL05.html**

Domains

Okay, here are all the marbles you can play with. We will call that your domain of resources.

A ***domain*** is defined as a set of objects that a subject is able to access. This domain can be all the resources a user can access, all the files available to a program, the memory segments available to a process, or the services and processes available to an application. A subject needs to be able to access and use objects (resources) to perform tasks, and the domain defines which objects are available to the subject and which objects are untouchable and therefore unusable by the subject.

These domains have to be identified, separated, and strictly enforced. An operating system may work in a privileged mode and a user mode. The reason to even use these two different modes is to define two different domains. The privileged mode has a much larger domain to work with (or more resources to access); thus, it can provide much more functionality. When an operating system works in privileged mode, it can physically access memory modules, transfer data from an unprotected domain to a protected domain, and directly access and communicate with hardware devices. An application that functions in user mode cannot access memory directly and has a more limited amount of resources available to it. Only a certain segment of memory is available to this application and it must be accessed in an indirect and controlled fashion. The application can copy files only within its own domain and cannot access hardware directly.

A program that resides in a privileged domain needs to be able to execute its instructions and process its data with the assurance that programs in a different domain cannot negatively affect its environment. This is referred to as an ***execution domain***. Because programs in a privileged domain have access to sensitive resources, the environment needs to be protected from rogue program code or unexpected activities resulting from programs in other domains. Some systems may only have distinct user and privilege areas, whereas other systems may have complex architectures that contain up to ten security domains.

A security domain has a direct correlation to the protection ring that a subject or object is assigned to. The lower the protection ring number, the higher the privilege and the larger the security domain. This concept is depicted in Figure 5-13.

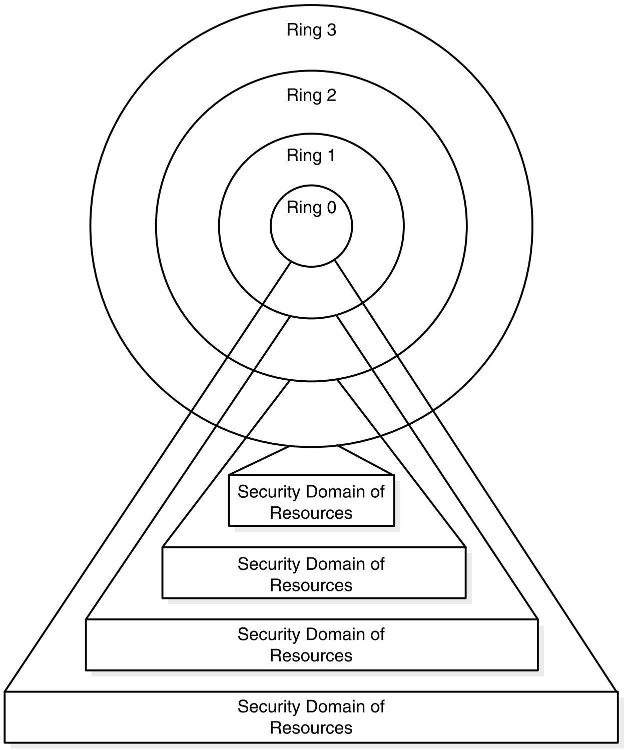

Ring 3
Ring 2
Ring 1
Ring 0
Security Domain of Resources
Security Domain of Resources
Security Domain of Resources
Security Domain of Resources

Figure 5-13 The higher the level of trust, the larger the number of available resources.

Resource Isolation

You, go into this room and play all by yourself.

To properly enforce access control, auditing, and determining what subjects and objects reside in specific domains, each resource has to be clearly separated from one another. This modularity requirement enables each subject and object to be identified uniquely, permissions and rights to be assigned independently, accountability to be enforceable, and intricate activities to be tracked precisely. The subjects, objects, and protection controls need to be clearly isolated from each other, and the isolation methods and enforcement are a requirement of the architecture of a system and its security model.

Processes are also resources that need to be isolated and this is usually done through distinct address allocation. Virtual memory (explained in the section "Process Activity" earlier in the chapter) techniques are used to make sure different processes have their own memory addresses to use and do not access each other's memory. If a process has a particular memory range allocated to it, it does not know there is other memory in the system. The process works happily in the memory allocated and does not stomp on another process's data in another segment of memory.

Systems of a higher trust level may need to implement ***hardware segmentation*** of the memory used by different processes. This means that memory is separated physically instead of just logically. This adds another layer of protection to ensure that a lower-privileged process does not access and modify a higher-level process's memory space.

Security Policy

We have stated that the TCB contains components that directly enforce the security policy, but what is a security policy?

A ***security policy*** is a set of rules and practices dictating how sensitive information is managed, protected, and distributed. A security policy expresses exactly what the security level should be by setting the goals of what the security mechanisms are to accomplish. This is an important element that has a major role in defining the design of the system. The security policy is a foundation for the specifications of a system and provides the baseline for evaluating a system. In Chapter 3, security policies were examined in depth, but those policies were directed toward the company itself. The security policies being addressed here are for operating systems and applications. The different policies are similar but have different targets: an organization as opposed to an individual computer system.

A system provides trust by fulfilling and enforcing the security policy and typically deals with the relationships between subjects and objects. The policy must indicate what subjects can access individual objects, and what actions are acceptable and unacceptable. The definition of what trust means is derived from a framework and the security policy works as this framework for computing systems.

For a system to provide an acceptable level of trust, it must be based on an architecture that provides the capabilities to protect itself from untrusted processes, intentional or accidental compromises, and attacks at different layers of the system. A majority of the trust ratings require a defined subset of subjects and objects, explicit domains, and the isolation of resources so their access can be controlled and activities performed on them can be audited.

Let's regroup. We know that a system's trust is defined by a set of criteria. When a system is tested against this set of criteria, a rating is assigned to the system and this rating is used by customers, vendors, and the computing society as a whole. The criteria will determine if the security policy is being properly supported and enforced. The security policy lays out the rules and practices pertaining to how a system will manage, protect, and distribute sensitive information. The reference monitor is a concept that says all subjects must have proper authority to access objects and is implemented by the security kernel. The security kernel is made of all the resources that supervise system activity in accordance with the system's security policy and is part of the operating system that controls access to system resources. For this to work correctly the individual resources need to be isolated from each other and domains need to be defined to dictate what objects are available to what subjects.

Security policies that prevent information from flowing from a high security level to a lower security level are called ***multilevel security policies***. These types of policies permit a subject to access an object only if the subject's security level is higher than or equal to the object's classification.

As we said, these are abstract ideas that will be manifested into physical hardware components, firmware, software code, and activities through designing, building, and implementing a system. These ideas are like abstract goals and dreams we would like to accomplish, which are accomplished by our physical hard work and items that we produce.

Least Privilege

Once resources and processes are isolated properly, ***least privilege*** needs to be enforced. This means that a process has no more privileges than necessary to be able to fulfill its functions. Only processes that *need* to carry out critical system functions should be allowed to, and other less-privileged processes should call upon the more privileged processes to carry out these types of activities when necessary. This type of indirect activity protects the system from poorly written or misbehaving code. Processes should possess a level of privilege only as long as they really need it. If a process needs to have its status elevated so it can interact directly with sensitive information, as soon as its tasks are complete, the process's status should be dropped to a lower privilege to ensure that another mechanism cannot use it to adversely affect the system. Only processes needing complete system privileges are located in the kernel; other less-privileged processes call upon them to process sensitive or delicate operations.

References

Access Control Policies and Mechanisms: **www.cs.cornell.edu/html/cs513-sp99/NL03.html**

Information Warfare Site: **www.iwar.org.uk/comsec/resources/standards/rainbow/NCSC-TG-028.htm**

Discretionary Protection: **www.kernel.org/pub/linux/libs/security/Orange-Linux/refs/Orange/OrangeI-II-2.html**

Layering, Data Hiding, and Abstraction

You can't see me, you don't know that I exist, so you can't talk to me. Response: Fine by me.

Systems that meet certain trust levels must supply mechanisms that force processes to work in layers, which means certain functionality takes place in different layers of a system. This requires a structured and hierarchical architecture that has the basic functionality taking place at lower layers and more complex, sensitive functions at the higher layers. ***Layering*** further separates processes and resources and adds modularity to the system. The layers can communicate, but only through detailed interfaces that uphold the security integrity of the system. In some instances, it is required that processes in different layers do not communicate; therefore, they are not supplied with interfaces to interact with each other. This process is called ***data hiding*** in that the data in one layer is hidden because the subjects in another layer do not even know that the data exists. If a subject in one layer has no interface to be able to communicate with data at another layer, this data is hidden from that subject.

Objects can be grouped into sets called classes. When a class of objects is assigned specific permissions and acceptable activities are defined, it is called ***abstraction***. This makes management of different objects easier because classes can be dealt with instead of each and every individual object. When a class is defined, all the objects within that class are assigned an abstract data type, which is a precise definition of the format the object will accept data and the format it will present its processed data to other objects and subjects. This provides a predictable way of communicating and helps prevent authorized entities from modifying the data within an object in an inappropriate way. For example, when one object passes a registry value to another object, it is done in a predefined manner and the receiving object will not accept values outside of these predefined boundaries. If the receiving object is expecting a binary value, it will not accept a hexadecimal value in its place. This type of restriction is defined in the object's abstract data type.

Layering, data hiding, and abstraction are all methods to protect subjects, objects, and the data within the objects. These concepts are foundational pieces to a security model.

Security Models

An important concept in the design and analysis of secure systems is the security model, because it incorporates the security policy that should be enforced in the system. A model is a symbolic representation of a policy. It maps the desires of the policy makers into a set of rules that are to be followed by a computer system.

We have continually mentioned the security policy and its importance, but it is an abstract term that represents the objectives and goals a system must meet and accomplish to be deemed secure and acceptable. How do we get from an abstract security policy to an administrator being able to uncheck a box on the graphical user interface (GUI) to disallow David from accessing configuration files on his system? There are many complex steps in between that take place during the system's design and development.

A security model maps the abstract goals of the policy to information system terms by specifying explicit data structures and techniques necessary to enforce the security policy. A security model is usually represented in mathematics and analytical ideas, which are then mapped to system specifications, and then developed by programmers through

Analogy of the Relationship Between a Security Policy and a Security Model

If someone tells you to live a healthy and responsible life, this is a very broad, vague, and abstract notion. So when you ask this person how this is accomplished, they outline the things you should and should not do (do not harm others, do not lie, eat your vegetables, and brush your teeth). The security policy provides the abstract goals, and the security model provides the dos and don'ts necessary to fulfill these goals.

programming code. So we have a policy that encompasses security goals like "each subject must be authorized to access each object." The security model takes this requirement and provides the necessary mathematical formulas, relationships, and structure to be followed to accomplish this goal. From here, specifications are developed per operating system type (Unix, Windows, or Macintosh), and individual vendors can decide how they are going to implement mechanisms that meet these necessary specifications.

So in a very general and simplistic example, if a security policy states that subjects need to be authorized to access objects, the security model would provide the mathematical relationships and formulas explaining how *x* can access *y* only through outlined specific methods. Specifications are then developed to provide a bridge to what this means in a computing environment and how it maps to components and mechanisms that need to be coded and developed. The developers then write the program code to produce the mechanisms that provide a way for a system to use access control lists and give administrators some degree of control. This mechanism presents the network administrator with a GUI representation, like check boxes, to choose which subjects can access what objects, to be able to set this configuration within the operating system. This is a rudimentary example because security models can be very complex, but it is used to demonstrate the relationship between the security policy and the security model.

Some security models enforce rules to protect confidentiality, such as the Bell- LaPadula model. Other models enforce rules to protect integrity, such as the Biba model. Formal security models, such as Bell-LaPadula and Biba, are used to provide high assurance in security. Informal models, such as Clark-Wilson, are used more as a framework to describe how security policies should be expressed and executed.

A security policy outlines goals with no idea of how they would be accomplished. A model is a framework that gives the policy form and solves security problems for particular situations. Several security models have been developed to enforce security policies, and the following sections provide overviews of each model.

State Machine Models

No matter what state I am in, I am always trustworthy.

In ***state machine models***, to verify the security of a system, the state is used, which means all current permissions and all current instances of subjects accessing objects must be captured. Maintaining the state of a system deals with each subject's association with objects. If the subjects can only access objects by means that are concurrent with the security policy, the system is secure. State machines have provided a basis for important security models. A state of a system is a snapshot of a system in one moment of time. There are many activities that can alter this state, which is referred to as a ***state transition***. The developers of an operating system that will implement the state machine model need to look at all of the different state transitions that are possible and assess if a system starts up in a secure state, can any of these events put the system into an insecure state? If all of the activities that are allowed to happen in the system do not compromise the system and put it into an insecure state, then the system executes a secure state machine model.

So a system that has employed a state machine model will be in a secure state in each and every instance of its existence. It will boot up into a secure state, execute commands and transactions securely, allow subjects to access resources only in secure states, and shut down and fail in a secure state. Failing in a secure state is extremely important. It is imperative that if anything unsafe takes place that the system can "save itself" and not make itself vulnerable. When a user sees error messages, or a system reboots or freezes, these are all safety measures. The operating system has experienced something that is deemed illegal and it cannot take care of the situation itself, so to make sure it does not enter an insecure state it reacts in one of these fashions. So if an application or system freezes on you, know that it is the system just trying to protect itself and your data.

Bell-LaPadula Model

I don't want anyone to know my secrets. Response: We need Mr. Bell and Mr. LaPadula in here then.

In the 1970s, the U.S. military used time-sharing mainframe systems and was concerned about the security of these systems and leakage of classified information. The ***Bell-LaPadula model*** was developed to address these concerns. It was the first mathematical model of a multilevel security policy used to define the concept of a secure state machine and modes of access and outline rules of access. Its development was funded by the U.S. government to provide a framework for computer systems that would be used to store and process sensitive information. The model's main goal is to prevent secret information from being accessed in an unauthorized manner.

A system that employs the Bell-LaPadula model is called a multilevel security system because users with different clearances use the systems, and the systems process data with different classifications. The level at which information is classified determines the handling procedures that should be used. These access rights together form a ***lattice***, which is an upper bound and lower bound of authorized access. A subject that has top secret clearance can access top-secret, secret, and unclassified data. Top secret is the upper bound and unclassified is the lower bound. Mandatory access control is based on a lattice of security labels.

The Bell-LaPadula model is a state machine model enforcing the confidentiality aspects of access control. A matrix and security levels are used to determine if subjects can access different objects. The subject's clearance is compared to the object's classification;

if the clearance is higher or equal to the object's classification, the subject can access the object without violating the security policy.

The model uses subjects, objects, access operations (read, write, and read/write), and security levels. Subjects and objects can reside at different security levels and have relationships and rules dictating the acceptable activities between them. If properly implemented and enforced, this model has been mathematically proven to prevent data from a higher security level from flowing to a lower security level. It is an ***information flow security model*** also, which means that information does not flow in an insecure manner.

The Bell-LaPadula model is a subject-to-object model. An example would be how you could take an element from a specific database and write it into another database. The Bell-LaPadula has a focus of ensuring that subjects are properly authenticated, by having the necessary security clearance and need-to-know, before accessing an object.

There are three main rules used and enforced in the Bell-LaPadula model: the simple security rule, the *-property rule, and the strong star property rule. The ***simple security rule*** states that a subject at a given security level cannot *read* data that resides at a higher security level. The ****-property* rule** states that a subject in a given security level cannot *write* information to a lower security level. The simple security rule is referred to as the "no read up" rule, and the *-property rule is referred to the as the "no write down" rule, as shown in Figure 5-14. The third rule, the ***strong star property rule***, states that a subject that has read and write capabilities can only perform those functions at the same security level, nothing higher and nothing lower. The simple security and *-property rules, and the strong star property rule, indicate what states the system can go into.

The state of a system changes as different operations take place. The Bell-LaPadula model defines a secure state, meaning a secure computing environment and the allowed actions, which are security-preserving operations. This means that the model provides a secure state and only permits operations that will keep the system within a secure state and not let it enter into an insecure state. So if 100 people access 2,000 objects in a day using this one system, this system is put through a lot of work and several complex activities have to take place. However, at the end of the day, the system is just as secure as it

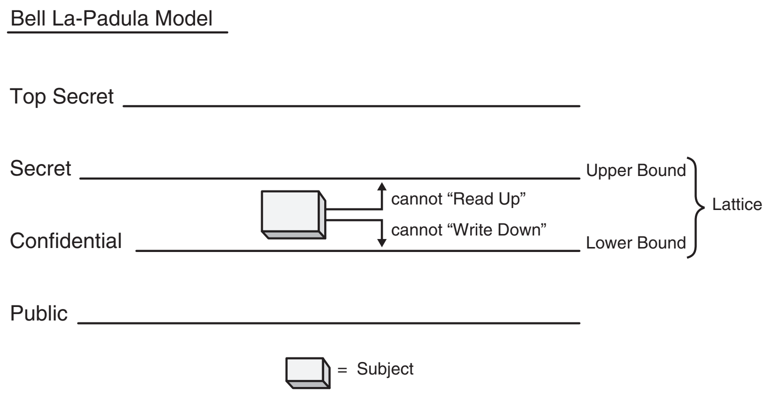


Figure 5-14 In the Bell-LaPadula model, each subject has a lattice of rights.

was in the beginning of the day. This is the definition of the Basic Security Theorem used in computer science.

NOTE The definition of the Basic Security Theorem is that if a system initializes in a secure state and all state transitions are secure, then every subsequent state will be secure no matter what inputs occur.

An important thing to note is that the Bell-LaPadula model was developed to make sure secrets stay secret; thus, it provides and addresses confidentiality only. This model does not address integrity of the data the system maintains—only who can and cannot access the data.

So What Does This Mean and Why?

Subjects and objects are assigned labels. The subject's label is a clearance label (top secret, secret, confidential, and so on) and the object's label is a classification label (top secret, secret, confidential, and so on). When a subject attempts to access an object, the system compares the subject's clearance label and the object's classification label and looks at a matrix to see if this is a legal and secure activity. So let's say it is a perfectly fine activity, and the subject is given access to the object. Now if the subject's clearance label is top secret and the object's classification label is secret, the subject cannot write to this object, because of the *-property rule. This makes sure that subjects cannot accidentally or intentionally share confidential information by writing to an object at a lower security level. Let's say a busy and clumsy general in the army opens up a briefing letter that will go to all clerks at all bases all around the world. He attempts to write that the United States is attacking Cuba. The Bell-LaPadula model will come into action and not permit this general to write this information to this type of file because his clearance is higher than that of the memo.

Likewise, if a nosey clerk tried to read a memo that was available only to generals and above, the Bell-LaPadula model would stop this activity also. The clerk's clearance is lower than that of the object, the memo, and this violates the simple security rule of the model. It is all about keeping secrets secret.

The following lists some criticisms of the Bell-LaPadula model:

- It deals only with confidentiality and does not address integrity.
- It does not address management of access control, because there is no mechanism to modify access rights.
- The model does not prevent or address covert channels.
- The model does not address file sharing used in more modern systems.

Biba Model

The ***Biba model*** was developed after the Bell-LaPadula model. It uses a state machine model and is very similar to the Bell-LaPadula model. Biba addresses the integrity of data being threatened when subjects at lower integrity levels are able to write to objects at higher integrity levels and when subjects can read data at lower levels. If implemented and enforced properly, the Biba model prevents data from any *integrity* level from flowing to a higher integrity level. Biba has two main rules to provide this type of protection. The first rule, referred to as "no write up," states that a subject cannot write data to an object at a higher *integrity* level. The second rule, referred to as "no read down," states that a subject cannot read data from a lower integrity level. This second rule might sound a little goofy, but it is protecting the subject and data at a higher integrity level from being corrupted by data in a lower integrity level. An analogy would be if you were writing an article for the *New York Times* about the security trends over the last year, the amount of money businesses lost, and the cost/benefit ratio of implementing firewalls, intrusion detection systems, and vulnerability scanners. You do not want to get your data and numbers from any old Web site without knowing how those figures were calculated and the sources of the information. Your article (data at a higher integrity level) can be compromised if mixed with unfounded information from a bad source (data at a lower integrity level).

When first learning about the Bell-LaPadula and Biba models, they seem very similar, and the reasons for their differences may bring about some confusion. The Bell-LaPadula model was written for the U.S. government, and the government is very paranoid about the leakage of their secret information. In their model, a user cannot write to a lower level because that user might let out some secrets. Similarly, a user at a lower level cannot read anything at a higher level because that user might learn some secrets. However, not everyone is so worried about confidentiality and has such big, important secrets to protect. The commercial industry is more concerned about the integrity of its data. An accounting firm is more worried about keeping their numbers straight and making sure decimal points are not dropped or extra zeros are not added in a process carried out by an application. The accounting firm is more concerned about the integrity of this data and is usually under little threat of someone trying to steal these numbers, so they would employ the Biba model. Of course, the accounting firm does not look for the name Biba on the back of a product or make sure it is in the design of their application. It is something that was decided upon and implemented when the application was being developed. The security ratings are what consumers use to determine if a system is right for them. So even if the accountants are using a system using the Biba model, they would not know (and we're not going to tell them).

Bell-LaPadula versus Biba

The Bell-LaPadula model is used to provide confidentiality. The Biba model is used to provide integrity. The Bell-LaPadula and Biba models are informational flow models because they are most concerned about data flowing from one level to another. Bell-LaPadula uses security levels and Biba uses integrity levels.

Confusion of the Different Rules

It is important for CISSP test takers to know the rules of Biba and Bell-LaPadula. Their rules sound very similar, simple and *-rules—one writing one way and one reading another way. A tip to remember them is that if the word "simple" is used, the rule is talking about reading. If the rule uses * or "star", it is talking about writing. So now you just need to remember reading and writing in which direction for the different models.

Clark-Wilson Model

The ***Clark-Wilson model*** was developed after Biba and takes some different approaches to protecting the integrity of information by focusing on preventing authorized users from making unauthorized modification of data, or commit fraud and errors within commercial applications.

As stated earlier, military institutions are usually more concerned about confidentiality, and the commercial sector is usually more concerned with the integrity of the data they process. In the Clark-Wilson model, users cannot access and manipulate objects directly, but must access the object through a program. This provides another layer of protection between the subject and the object and further restricts the type of actions that can take place on that object, thus protecting the integrity of the object.

Let's say Emily is a user of an application that has been built using the Clark-Wilson model. She cannot access files directly, but must open and manipulate files through the program built upon the Clark-Wilson model. The programs have their own set of restrictions dictating what actions the users can and cannot perform on objects. This streamlines the way objects are protected and reduces the methods that could cause the data to be corrupted or changed in an undesirable manner.

NOTE Ensuring that subjects can only access objects through the use of an application is referred to as ***access triple***. A subject has to go through an application to access an object: subject – application – object.

This model also enforces ***separation of duties***, which divides an operation into different parts and requires different users or rules to perform each part. This ensures that a critical task cannot be carried out by one entity. Auditing is also required in this model to track the information coming in from the outside of the system.

So the Clark-Wilson model prevents authorized users from making modifications by requiring them to go through programs to modify objects. It also prevents authorized users from making improper modifications by enforcing separation of duties, and maintains an audit log for external transactions.

Goals of Integrity

There are three main goals of integrity:

- Prevent unauthorized users from making modifications
- Prevent authorized users from making improper modifications
- Maintain internal and external consistency

Clark-Wilson addresses each of these goals in its model. Biba only addresses the first goal.

Information Flow Model

Now, which way is the information flowing in this system? Answer: Not to you.

The Bell-LaPadula model is concerned about information that can flow from a high security level to a low security level. The Biba model is concerned about information that can flow from a high integrity level to a low integrity level. Both of these were built upon the ***information flow model***. Information flow models can deal with any kind of information flow, not only the direction of the flow. This type of model looks at insecure informational flow that can happen at the same level and between objects along with the flow between different levels. When the information flow model is used, a system is secure if there is no illegal information flow permitted.

Basically, information can flow from one security level to another or from one object to another in the same security level until a restricted operation is attempted. In most systems, once a restricted operation is attempted, the system looks at an access control matrix to see if this action has been specifically permitted for this specific subject or object.

Noninterference Model

Stop touching me. Stop touching me. You are interfering with me!

Multilevel security properties can be expressed in other ways, one being *noninterference*. This concept is implemented to ensure that any actions that take place at a higher security level do not affect, or interfere, with actions that take place at a lower level. This type of model does not concern itself with the flow of data, but with what a subject knows about the state of the system. So if an entity at a higher security level performs an action, it cannot change the state for the entity at the lower level.

If a lower-level entity was aware of a certain activity that took place by an entity at a higher level and the state of system changed for this lower-level entity, the entity might be able to deduce too much information about the activities of the higher state, which in turn is a way of leaking information.

Users at a lower security level should not be aware of the commands executed by users at a higher level and should not be affected by those commands in any way.

Brewer and Nash Model

The ***Brewer and Nash model***, also called the Chinese Wall model, was created to provide access controls that can change dynamically depending upon a user's previous actions. The main goal of the model is to protect against users accessing data that could be seen as conflicts of interest. For example, if a large marketing company provided marketing promotions and materials for several banks, one individual working on a project for Bank A should not be looking at information the marketing company has for Bank B. So this marketing company could implement a product that tracked the different marketing representatives' access activities and disallow certain access requests that would present a conflict of interest. In Figure 5-15, we see that when a representative accesses Bank A's information the system automatically makes Bank B's information off limits. If the representative accessed Bank B's data, Bank A's information would be off limits. These access controls change dynamically depending upon the user's activities and previous access requests.

Graham-Denning and Harrison-Ruzzo-Ullman Models

Remember that these are all models, thus they are not very specific in nature. Each individual vendor must decide upon how it is going to actually meet the rules outlined in the chosen model. Bell-LaPadula and Biba don't define how the security and integrity ratings are defined and modified, nor do they provide a way to delegate or transfer access rights. The ***Graham-Denning model*** addresses these issues and defines a set of basic rights in terms of commands that a specific subject can execute on an object. The ***Harrison-Ruzzo-Ullman model*** outlines how access rights can be changed and how subjects and objects should be created and deleted.

These newer models provide more granularity and direction for vendors on how to actually meet the goals outlined in the earlier models.

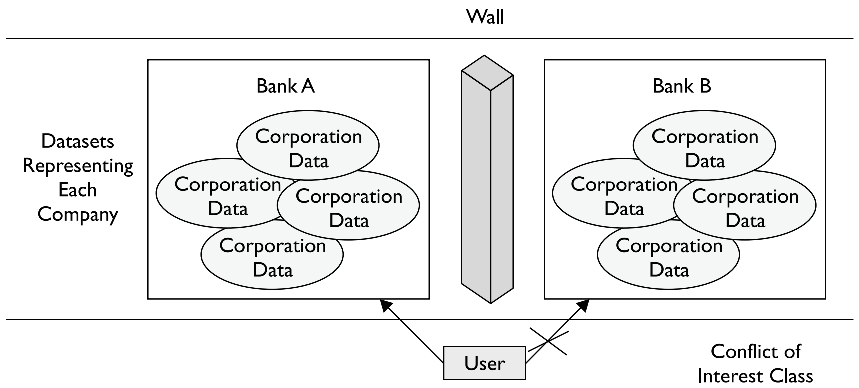


Figure 5-15 The Chinese Wall model provides dynamic access controls.

Security Models

- **Bell-LaPadula model** A model that protects the confidentiality of the information within a system.
  - **Simple security rule** A subject cannot read data at a higher security level (no read up).
  - ***-property rule** A subject cannot write data to an object at a lower security level (no write down).
  - **Strong star property rule** A subject that has read and write capabilities can only perform those functions at the same security level.
- **Biba model** A model that protects the integrity of the information within a system.
  - **Simple integrity axiom** A subject cannot read data at a lower integrity level (no read down).
  - ***-integrity axiom** A subject cannot modify an object in a higher integrity level (no write up).
- **Clark-Wilson model** An integrity model implemented to protect the integrity of data and to ensure that properly formatted transactions take place.
  - Subjects can only access objects through authorized programs (access triple).
  - Separation of duties is enforced.
  - Auditing is required.
- **Information flow model** Information is restricted in its flow to only go to and from entities in a way that does not negate the security policy.
- **Noninterference model** Commands and activities performed at one security level should not be seen or affect subjects or objects at a different security level.
- **Brewer and Nash model** A model that allows for dynamically changing access controls that protect against conflicts of interest
- **Graham-Denning model** A model that creates rights for subjects, which correlate to the operations that can be execute on objects.
- **Harrison-Ruzzo-Ullman model** A model that allows for access rights to be changed and specifies how subjects and objects should be created and deleted.

References

Access Control Models: **www.cccure.org/Documents/HISM/087-089.html**

Models of OS Protection: **all.net/books/ip/Chap3-3.html**

Computer Security Models: **infoeng.ee.ic.ac.uk/~malikz/surprise2001/spc99e/article1**

Database Security: **www.wi-inf.uni-essen.de/~ifs/summerschool**

Security Modes of Operation

A system can operate at different types of modes depending on the sensitivity of the data being processed, the clearance level of the users, and what those users are authorized to do. The mode of operation describes the security conditions under which the system actually functions. There are four modes of operations a system can function under and the following sections describe each one.

Dedicated Security Mode

A system is operating in a ***dedicated security mode*** if all users have the clearance and formal need-to-know to *all* data processed within the system. All users have been given formal access approval for *all* information on the system and have signed nondisclosure agreements pertaining to this information. The system can handle a single classification level of information.

Many military systems have been designed to handle only one level of security, which works in dedicated security mode. This requires everyone who uses the system to have the highest level of clearance required by any and all data on the system. If a system held top-secret data, only users with this clearance could use the system. Other military systems work with multiple security levels, which is done by compartmentalizing the data. These types of systems can support users with high and low clearances simultaneously.

System-High Security Mode

A system is operating in ***system-high security mode*** when all users have a security clearance or authorization to access the information but not necessarily a need-to-know for all the information processed on the system. So the difference between the dedicated security mode and the system-high security mode is that in the dedicated security mode, all users have a need-to-know pertaining to *all* data on the system; in system-high security mode, all users have a need-to-know pertaining to *some* of the data.

This mode also requires all users to have the highest level of clearance required by any and all data on the system. However, just because a user has the necessary security level to access an object, the user could be restricted because they do not have a need-to-know pertaining to that specific object.

Compartmented Security Mode

A system is operating in ***compartmented security mode*** when all users have the clearance to access all the information processed by the system, but might not have the need-to-know and formal access approval. In compartmented security mode, users are restricted from being able to access some information because they do not need to access it to perform the functions of their jobs and they have not been given formal approval to access this data. In this mode, users can access a segment, or compartment, of data only.

The objective is to ensure that the minimum possible number of people learn of information at each level. Compartments are categories of data with limited number of subjects cleared to access data at each level. ***Compartmented mode workstations (CMWs)*** enable users to process multiple compartments of data at the same time, if they have the necessary clearance.

Multilevel Security Mode

A system is operating in ***multilevel security mode*** when it permits two or more classification levels of information to be processed at the same time when all the users do not have the clearance or formal approval to access all the information being processed by the system.

The Bell-LaPadula model is an example of a multilevel security model because it handles multiple information classifications at a number of different security levels within one system.

References

Data Protection Measures: **www.tpub.com/ans/51.htm**

Personal Computer Security: **www.cs.nps.navy.mil/curricula/tracks/security/notes/chap02_17.html**

Physical Model of Operations: **chacs.nrl.navy.mil/publications/CHACS/1997/jifi_web/node24.html**

Defense Security Service, NISPOM Chapter 8: **www.dss.mil/isec/ch8nispom/toc.htm**

Trust and Assurance

I trust that you will act properly, thus I have a high level of assurance in you.

We discussed the TCB concept in the section "Trusted Computing Base" and explained that no system is really secure because with enough resources almost any system can be compromised in one way or another; however, systems can provide levels of trust. The trust level tells the customer how much he can expect out of this system, what level of security it will provide, and the ***assurance*** that the system will act in a correct and predictable manner in each and every computing situation.

The TCB is made up of all the protection mechanisms within a system (software, hardware, firmware). All of these mechanisms need to work in an orchestrated way to enforce all the requirements of a security policy. When evaluated, these mechanisms are tested, their designs are inspected, and the supporting documentation is reviewed and evaluated. How the system is developed, maintained, and even delivered to the customer are all under review when the trust for a system is being gauged. All of these different evaluation components are put through an evaluation process to assign the correct level of trust and assurance. Customers then use this assignment, or rating, to determine which system best fits their security needs.

Assurance and trust are similar in nature, but slightly different when we are looking at rating computer systems. A trusted system means that all protection mechanisms work together to process sensitive data for many types of uses and still keeps the same secure computing environment. Assurance looks at the same issues but with more depth and detail. Systems that provide higher levels of assurance means that their designs were thoroughly inspected, the development stages were reviewed, the technical specifications and test plans were evaluated, and the system was tested extensively. You can buy a car and you can trust it, but you have a much deeper sense of assurance of that trust if you know how it was built, what it was built with, who built it, what tests it was put through, and how it performed in many different situations.

In the Trusted Computer Security Evaluation Criteria, commonly known as the Orange Book, which we will address shortly, the lower security level ratings look at a system's performance and testing results to produce a security rating, but the higher security level ratings look more at the system design, specifications, development procedures, supporting documentation, and the testing results. The protection mechanisms in the higher security level systems may not necessarily be much different from those in the lower security level systems, but the way they were designed and built is under much more scrutiny. With this extra scrutiny comes higher levels of assurance of the trust that can be put into a system.

Systems Evaluation Methods

A ***security evaluation*** examines the security-relevant parts of a system, meaning the TCB, access control mechanisms, reference monitor, kernel, and protection mechanisms. The relationship and interaction between these components are also evaluated. There are different methods of evaluating and assigning assurance and trust levels to systems. The reason that there is more than one type of security evaluation process is that the methods and ideologies have evolved over time and because various parts of the world look at computer security differently and rate some aspects of security differently than others. Each method will be explained and then at the end of the section they will be compared to each other to show the differences and strengths, and look at where security evaluation methods are heading.

The Orange Book

The U.S. Department of Defense developed the ***Trusted Computer System Evaluation Criteria (TCSEC)***, which is used to evaluate operating systems, applications, and different products. This evaluation criteria is published in a book with an orange cover, which is called appropriately the ***Orange Book***. (We like to keep things simple in security!) Customers use the security rating that the criteria presents so they have a metric to use when comparing different systems. It also provides direction for manufactures so they know what specifications to build to, and provides a one-stop evaluation process so customers do not need to have individual components within the systems evaluated.

The Orange Book evaluates products to assess if they contain the security properties they claim and evaluate if the product is appropriate for a specific application or function.

The Orange Book looks at the functionality, effectiveness, and assurance of a system during its evaluation, and it uses classes that were devised to address typical patterns of security requirements.

TCSEC provides a graded classification of systems that is divided into hierarchical divisions of security levels:

A	Verified protection
B	Mandatory protection
C	Discretionary protection
D	Minimal security

The classification A represents the highest level of security and D represents the lowest level of security.

Each division can have one or more numbered classes and each has a corresponding set of requirements that must be met for a system to achieve that particular rating. The classes with higher numbers indicate a greater degree of trust and assurance. So B2 would offer more trust than B1, and C2 would offer more trust than C1.

The criteria includes four main topics: security policy, accountability, assurance, and documentation, but these actually break down into seven different areas:

- **Security policy** The policy must be explicit and well defined and enforced by the mechanisms within the system.
- **Identification** Individual subjects must be uniquely identified.
- **Labels** Access control labels must be associated properly with objects.
- **Documentation** This includes the test, design, specification documents, user guides, and manuals.
- **Accountability** Audit data must be captured and protected to enforce accountability.
- **Life cycle assurance** Software, hardware, and firmware must be able to be tested individually to ensure that each enforces the security policy in an effective manner throughout their lifetimes.
- **Continuous protection** The security mechanisms and the system as a whole must perform predictably and acceptably in different situations continuously.

These categories are evaluated independently, but the rating that is assigned at the end does not specify these different objectives individually. The rating is a sum total of these items.

Each division and class incorporates the requirements of the ones below it. This means that C2 must meet its criteria requirements and all of C1 requirements, and B3 has its requirements to fulfill along with those of C1, C2, B1, and B2. Each division or class ups the ante on security requirements and is expected to fulfill the requirements of all the classes and divisions below it.

When a vendor submits a product for evaluation, the product is submitted to the National Computer Security Center (NCSC). The process of evaluation is called the Trusted Products Evaluation Program (TPEP). Successfully evaluated products are placed on the Evaluated Product List (EPL) with their corresponding rating. When consumers are interested in certain products and systems, they can check the EPLs to find out the security level that are assigned to these products.

Division D: Minimal Protection

There is only one class in this division. It is reserved for systems that have been evaluated but fail to meet the criteria and requirements of the higher divisions.

Division C: Discretionary Protection

The C rating category has two individual assurance ratings within it; these ratings are described below. The higher the number of the assurance rating, the more protection that is provided.

C1: Discretionary Security Protection

Discretionary access control is based on individuals and/or groups. It requires a separation of users and information, and identification and authentication of individual entities. Some type of access control is necessary so users can ensure that their data will not be accessed and corrupted by others. The system architecture must supply a protected execution domain so privileged system processes are not adversely affected by lower-privileged processes. There must be specific ways of validating the system's operational integrity. The documentation requirements include design documentation, which shows the way the system was built to include protection mechanisms, test documentation (test plan and results), a facility manual so companies know how to install and configure the system's correctly, and user manuals.

The environment that would require this rating would be where users are processing information at the same sensitivity level; thus, strict access control and auditing measures are not required. It would be a trusted environment with low security concerns.

C2: Controlled Access Protection

Users need to be identified individually to provide more precise access control and auditing functionality. Logical access control mechanisms are used to enforce authentication and the uniqueness of each individual's identification. Security-relevant events are audited and these records must be protected from unauthorized modification. The architecture must provide resource, or object, isolation so proper protection can be applied to the resource and that actions taken upon it can be properly audited. The ***object reuse*** concept must also be invoked, meaning that any medium holding data must not contain any remnants of information after it is released for another subject to use. If a subject uses a segment of memory, that memory space must not hold any information after the subject is done using it. The same is true for storage media, objects being populated, temporary files being created—all data must be efficiently erased once the subject is done with that medium.

This class requires a more granular method of providing access control. The system must enforce strict logon procedures and provide decision-making capabilities when subjects request access to objects. A C2 system cannot guarantee that it will not be compromised, but it supplies a level of protection that would make compromising attempts harder to accomplish.

The environment that would require systems with a C2 rating would be one that contains users that are trusted, but a certain level of accountability is required. C2, overall, is regarded to be the most reasonable class for commercial applications, but the level of protection is still relatively weak.

Division B: Mandatory Protection

Mandatory access control is enforced by the use of security labels. The architecture is based on the Bell-LaPadula security model and evidence of reference monitor enforcement must be available.

B1: Labeled Security

Each data object must contain a classification label and each subject must have a clearance label. When a subject attempts to access an object, the system must compare the subject and object's security labels to ensure the requested actions are acceptable. Data leaving the system must also contain an accurate security label. The security policy is based on an informal statement and the design specifications are reviewed and verified.

It is intended for environments that require systems to handle classified data.

NOTE Security labels are not required until security rating B; thus, C2 does not require security labels but B1 does.

B2: Structured Protection

The security policy is clearly defined and documented, and the system design and implementation are subjected to more thorough review and testing procedures. This class requires more stringent authentication mechanisms and well-defined interfaces among layers. Subjects and devices require labels, and the system must not allow covert channels. A trusted path for logon and authentication processes must be in place, which means there are no trapdoors. A trusted path means that the subject is communicating directly with the application or operating system. There is no way to circumvent or compromise this communication channel. There is a separation of operator and administration functions within the system to provide more trusted and protected operational functionality. Distinct address spaces must be provided to isolate processes, and a covert channel analysis is conducted. This class adds assurance by adding requirements to the design of the system.

The environment that would require B2 systems could process sensitive data that requires a higher degree of security. This environment would require systems that are relatively resistant to penetration and compromise.

(A trusted path means that the user can be sure that he is talking to a genuine copy of the operating system.)

B3: Security Domains

In this class, more granularity is provided in each protection mechanism, and the programming code that is not necessary to support the security policy is excluded. The design and implementation should not provide too much complexity because as the complexity of a system increases, the ability of the individuals who need to test, maintain, and configure it reduces; thus, the overall security can be threatened. The reference monitor components must be small enough to test properly and be tamperproof. The security administrator role is clearly defined, and the system must be able to recover from failures without its security level being compromised. When the system starts up and loads its operating system and components, it must be done in an initial secure state to ensure that any weakness of the system cannot be taken advantage of in this slice of time.

An environment that requires B3 systems is a highly secured environment that processes very sensitive information. It requires systems that are highly resistant to penetration.

Division A: Verified Protection

Formal methods are used to ensure all subjects and objects are controlled with the necessary discretionary and mandatory access controls. The design, development, implementation, and documentation are looked at in a formal and detailed way. The security mechanisms between B3 and A1 are not very different, but the way that the system was designed and developed is evaluated in a much more structured and stringent procedure.

A1: Verified Design

The architecture and protection features are not much different from systems that achieve a B3 rating, but the assurance of an A1 system is higher than a B3 system because the formality in the way the system was designed, the way the specifications were developed, and the level of detail in the verification techniques. Formal techniques are used to prove the equivalence between the TCB specifications and the security policy model. More stringent change configuration is put in place with the development of an A1 system, and the overall design can be verified. In many cases, even the way that the system is delivered to the customer is under scrutiny to ensure that there is no way of compromising the system before it reaches its destination.

An environment that would require A1 systems is the most secure of secured environments. This environment deals with top-secret information and cannot adequately trust anyone using the systems without strict authentication, restrictions, and auditing.

NOTE TCSEC addresses confidentiality, but not integrity. Functionality of the security mechanisms and the assurance of those mechanisms are not evaluated separately, but combined and rated as a whole.

References

Trusted Computer System Evaluation Criteria: **www.boran.com/security/tcsec.html**

Trusted Computer Systems: **williamstallings.com/Extras/Security-Notes/lectures/trusted.html**

Rainbow Series

Why are there so many colors in the rainbow? Answer: Because there are so many security topics in the computing world that needed names.

The Orange Book mainly addresses government and military requirements and expectations from their computer systems. Many people within the security field have pointed out several deficiencies of the Orange Book, particularly when it is being applied to systems that are to be used in commercial areas instead of government organizations. The following summarizes a majority of the troubling issues security practitioners have expressed about the use of the Orange Book:

- The Orange Book looks specifically at the operating system and not other issues like networking, databases, and so on.
- The Orange Book focuses mainly on one attribute of security, confidentiality, and not at integrity, availability, and authenticity.
- The Orange Book works with government classifications and not the protection classifications that commercial industries use.
- The Orange Book has a relatively small number of ratings, which means many different aspects of security are not evaluated and rated independently.

The Orange Book places great emphasis on controlling which users can access a system and virtually ignores controlling what those users do with the information once they are authorized. Authorized users can, and usually do, cause more damage to data than outside attackers. Commercial organizations have expressed more concern about the integrity of their data while the military organizations stress that their top concern is confidentiality. Because of these different goals, the Orange Book is a better evaluation tool for government and military systems.

Because the Orange Book focuses on the operating system, many other areas of security were left out. The Orange Book provides a broad framework for building and evaluating trusted systems, but it leaves many questions about topics other than just an operating system unanswered. So more books were written to extend the coverage of the Orange Book into other areas of security. These books provide detailed information and interpretations of certain Orange Book requirements and describe the evaluation processes. These books are collectively called the ***Rainbow Series*** because each book has a different color cover.

For an explanation of each book and its usage, please refer to the following references.

References

Rainbow Series Library: **www.radium.ncsc.mil/tpep/library/rainbow**

Rainbow Series: **csrc.ncsl.nist.gov/secpubs/rainbow**

Rainbow Series and Related Documents: **www.fas.org/irp/nsa/rainbow.htm**

Red Book

The Orange Book addresses single-system security, but networks are a combination of systems and the network needs to be secure without having to fully trust each and every system connected to it. The ***Trusted Network Interpretation (TNI)***, also called the ***Red Book*** because of the color of its cover, addresses security evaluation topics for networks and network components. It addresses isolated local area networks and wide area internetwork systems.

Like the Orange Book, the Red Book does not supply specific details about how to implement security mechanisms; instead, it provides a framework for securing different types of networks. A network has a security policy, architecture, and design, as does an operating system. Subjects accessing objects on the network need to be controlled, monitored, and audited. In a network, the subject could be a workstation and an object could be a network service on a server.

The Red Book rates confidentiality and integrity of data and operations that happen within a network and the network products. Data and labels need to be protected from unauthorized modification, and the integrity of information as it is transferred needs to be ensured. The source and destination mechanisms used for messages are evaluated and tested to ensure that modification is not allowed.

Encryption and protocols are components that provide a lot of the security within a network, and the Red Book measures their functionality, strength, and assurance.

The following is a brief overview of the security items addressed in the Red Book:

- Communication integrity:
  - **Authentication** Protects against masquerading and playback attacks. Mechanisms include digital signatures, encryption, timestamp, and passwords.
  - **Message integrity** Protects protocol header, routing information, and the packet payload from being modified. Mechanisms include message authentication and encryption.
  - **Nonrepudiation** Ensures that a sender cannot deny sending a message. Mechanisms include encryption, digital signatures, and notary.
- Denial-of-service prevention:
  - **Continuity of operations** Ensures that network is available even if attacked. Mechanisms include fault tolerant and redundant systems and the capability to reconfigure network parameters in case of an emergency.
  - **Network management** Monitors network performance and identifies attacks and failures. Mechanisms include components that enable network administrators to monitor and restrict resource access.

- Compromise protection:
  - **Data confidentiality** Protects data from being accessed in an unauthorized method during transmission. Mechanisms include access controls, encryption, and physical protection of cables.
  - **Traffic flow confidentiality** Ensures that unauthorized entities are not aware of routing information or frequency of communication via traffic analysis. Mechanisms include padding messages, sending noise, or false messages.
  - **Selective routing** Routes messages in a way to avoid specific threats. Mechanisms include network configuration and routing tables.

Assurance is derived from a theory of how things should work and then compared to how they actually perform. Assurance is also derived by testing configurations in many different scenarios, evaluating engineering practices, and validating and verifying security claims. The Red Book ratings available are the following:

- **None**
- **C1** Minimum
- **C2** Fair
- **B2** Good

TCSEC was introduced in 1985 and retired in December 2000. It was the first methodical and logical set of standards developed to secure computer systems. It was greatly influential to several countries who based their evaluation standards on the TCSEC guidelines. TCSEC was finally replaced with the Common Criteria.

Information Technology Security Evaluation Criteria

The ***Information Technology Security Evaluation Criteria (ITSEC)*** was the first attempt at establishing a single standard for evaluating security attributes of computer systems by many European countries. ITSEC is only used in Europe. The United States looked to the Orange Book and Rainbow Series, and Europe employed the ITSEC to evaluate and rate computer systems. (Actually, today everyone is migrating to the Common Criteria, explained in the next section.)

There are two main attributes of a system when it is evaluated under ITSEC: functionality and assurance. When the functionality of a system is being evaluated, the services that are provided to the users (access control mechanisms, auditing, authentication, and so on) are examined and measured. System functionality can be very diverse in nature because systems are developed differently just to provide different functionality to users. Nonetheless, when functionality is evaluated, it is tested to see if the system delivers what it says it delivers. Assurance, on the other hand, is more abstract and harder to test. Assurance is the degree of confidence in a security component, and its effectiveness and capability to perform consistently. Assurance is generally tested by examining development practices, documentation, configuration management, and testing mechanisms.

It is possible for two systems that provide the same functionality to have very different assurance levels. This is because of the underlying mechanisms providing the functionality, and the way it was developed, engineered, and implemented. ITSEC actually separates these two attributes and rates them separately, whereas TCSEC clumps them together and assigns them one rating (D through A1).

When we look back at our example of two systems that provide the same functionality but have very different assurance levels, using the TCSEC approach will make this individual factor hard to distinguish. Under the ITSEC approach, the functionality is rated separately from the assurance. In the ITSEC criteria, classes F1 to F10 rate the functionality of the system, whereas E0 to E6 rate the assurance of a system.

So the fundamental difference between ITSEC and TCSEC is that TCSEC bundles functionality and assurance, whereas ITSEC evaluates these two attributes separately.

The following is a general mapping of the two evaluation schemes to show you their relationship to each other:

ITSEC		TCSEC
E0	=	D
F1 + E1	=	C1
F2 + E2	=	C2
F3 + E3	=	B1
F4 + E4	=	B2
F5 + E5	=	B3
F5 + E6	=	A1
F6	=	Systems the provide high integrity
F7	=	Systems that provide high availability
F8	=	Systems that provide data integrity during communication
F9	=	Systems that provide high confidentiality (like cryptographic devices)
F10	=	Networks with high demands on confidentiality and integrity

As you can see, a majority of the ITSEC ratings can be mapped to the Orange Book ratings, but then ITSEC took a step farther and added F6 through F10 for specific needs consumers might have that the Orange Book does not address.

ITSEC is a criteria for operating systems and other products and refers to them as the target of evaluation (TOE). So if you are reading literature discussing the ITSEC rating of a product and it states that the TOE has a rating of F1 and E5, you know that the TOE is the product that was evaluated and that it has a low functionality rating and a high assurance rating.

References

Criteria and Methods of Evaluations of Information Systems: **www.cordis.lu/infosec/src/crit.htm**

ITSEC: **www.iwar.org.uk/comsec/resources/standards/itsec.htm**

Common Criteria

"The TCSEC is too hard, the ITSEC is too soft, but the Common Criteria is just right," said the baby bear.

The Orange and Red Books provided evaluation schemes that were too rigid. ITSEC attempted to provide a more flexible approach by separating the functionality and assurance attributes and considering the evaluation of entire systems. However, this flexibility added complexity because evaluators could mix and match functionality and assurance ratings, which ended up in too many classifications to keep straight. Because we are a species that continues to try to get it right, the next attempt for an effective and usable evaluation criteria was the ***Common Criteria***.

In 1990, the International Organization for Standardization (ISO) identified the need of international standard evaluation criteria to be used globally. The Common Criteria project started in 1993 when several organizations came together to combine and align existing and emerging evaluation criteria (TCSEC, ITSEC, Canadian Trusted Computer Product Evaluation Criteria [CTCPEC], and the Federal Criteria). It was developed through a collaboration among national security standards organizations within the United States, Canada, France, Germany, the United Kingdom, and the Netherlands.

The benefit of having a worldwide recognized and accepted criteria helps consumers by reducing the complexity of the ratings and eliminating the need to understand the definition and meaning of different ratings within various evaluation schemes. This also helps manufacturers because now they can build to one specific set of requirements if they want to sell their products internationally, instead of having to meet several different ratings with varying rules and requirements.

The Orange Book evaluated all systems by how they compared to the Bell-LaPadula model. The Common Criteria provides more flexibility by evaluating a product against a protection profile, which is structured to address specific security problems. So while the Orange Book said, "Everyone march this direction in this form using this path," the Common Criteria says, "Okay, what are the threats we are facing today and what are the best ways of battling them?"

An evaluation is carried out on a product and is assigned an ***evaluation assurance level (EAL)***. The thoroughness and stringent testing increases in detailed-oriented tasks as the levels increase. The Common Criteria has seven assurance levels. The ranges go from EAL1, where functionality testing takes place, to EAL7, where thorough testing is performed and the system design is verified. The different EAL packages are listed here:

- **EAL 1** Functionally tested
- **EAL 2** Structurally tested
- **EAL 3** Methodically tested and checked
- **EAL 4** Methodically designed, tested, and reviewed
- **EAL 5** Semiformally designed and tested
- **EAL 6** Semiformally verified design and tested
- **EAL 7** Formally verified design and tested

The Common Criteria uses ***protection profiles*** to evaluate products. The protection profile contains the set of security requirements, their meaning and reasoning, and the corresponding EAL rating that the intended product will require. The profile describes the environmental assumptions, the objectives, and functional and assurance level expectations. Each relevant threat is listed along with how it is to be controlled by specific objectives. It also justifies the assurance level and requirements for the strength of each protection mechanism.

The protection profile provides a means for a consumer, or others, to identify specific security needs; this is the security problem that is to be conquered. If someone identifies a security need that is not currently being addressed by any current product, this person can write a profile of the product that would be a solution for this real-world problem. The profile goes on to provide the necessary goals and protection mechanisms to achieve the necessary level of security and a list of the things that can go wrong during this type of system development. This list is used by the engineers who develop the system and the same list is used by the evaluators to make sure the engineers dotted every *i* and crossed every *t*.

The Common Criteria was developed to stick to evaluation classes but also to retain some degree of flexibility. Protection profiles were developed to describe the functionality, assurance, description, and rationale of its findings.

Like other evaluation criteria before it, Common Criteria works to answer two basic and general questions about products being evaluated: what does it do (functionality), and how sure are you of that (assurance)? This system sets up a framework for consumers to be able to clearly specify their security issues and problems; developers to specify their security solution to those problems; evaluators to unequivocally determine what the product actually accomplishes.

A protection profile contains the following five sections:

- **Descriptive elements** Name of profile and a description of the security problem that is to be solved.
- **Rationale** Justification of the profile and a more detailed description of the real-world problem to be solved. The environment, usage assumptions, and threats are illustrated along with guidance on the security policies that can be supported by products and systems that conform to this profile.
- **Functional requirements** A protection boundary is established, meaning the threats or compromises that are within this boundary to be countered. The product or system must enforce the boundary established in this section.
- **Development assurance requirements** The development phases from design to implementation have specific requirements established that the product or system must meet.
- **Evaluation assurance requirements** Establishes the type and intensity of the evaluation.

The evaluation process is just one leg of determining the functionality and assurance of a product. Once a product achieves a specific rating, it only applies to that particular version and only to certain configurations of that product. So if a company buys a firewall product because it has a high assurance rating, it doesn't mean the next version of that software automatically inherits that rating. The next version will need to go though its own evaluation review. If this same company buys the firewall product and installs it with configurations not recommended, the level of security they were hoping to achieve can easily go down the drain. So all of this rating stuff is a formalized method of a system being evaluated in a lab. When the product is implemented into a real environment, other factors than just its rating need to be addressed and assessed to ensure that it is properly protecting resources and the environment.

Different Components of the Common Criteria

The different components of the Common Criteria are shown in Figure 5-16 and described here:

- **Protection profile** Description of needed security solution.
- **Target of evaluation** Product proposed to provide needed security solution.
- **Security target** Written by vendor explaining security functionality and assurance mechanisms that meet the needed security solution. In other words, "This is what our product does and how it does it."
- **Packages—evaluation assurance levels (EALs)** Functional and assurance requirements are bundled into packages for reuse. This component describes what must be met to achieve specific EAL ratings.

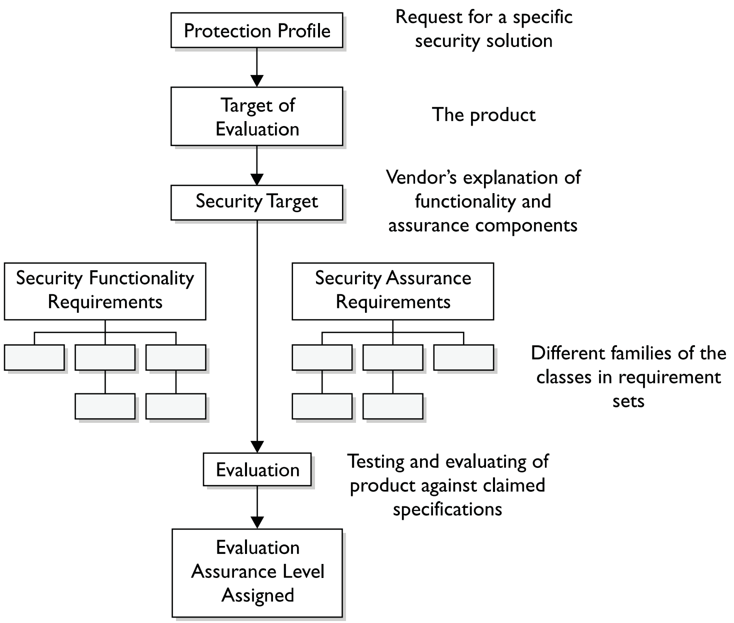

Protection Profile
Request for a specific security solution
Target of Evaluation
The product
Security Target
Vendor's explanation of functionality and assurance components
Security Functionality Requirements
Security Assurance Requirements
Different families of the classes in requirement sets
Evaluation
Testing and evaluating of product against claimed specifications
Evaluation Assurance Level Assigned

Figure 5-16 Relationship of the different components

References

CSRC, Common Criteria: **csrc.nist.gov/cc**

Common Criteria, The Standard for Information Security: **www.commoncriteria.org**

Common Criteria overview: **www.rycombe.com/cc.htm**

Certification versus Accreditation

We have gone through the different types of evaluation criteria a system can be appraised against to receive a specific rating. This is a very formalized process, and the evaluated system or product will then be placed on an EPL indicating what rating it achieved. Consumers can check this listing and compare the different products and systems to see how they rank against each other in the property of security. However, once a consumer buys this product and sets it up in his environment, it does not guarantee security. Security is made up of system administration, physical security, installation, and configuration mechanisms within the environment, and other security issues. To fairly say a system is secure, all of these items need to be taken into account. The rating is just one piece in the puzzle of security.

Certification

How did you certify this product? Answer: It came in a very pretty box.

Certification is the comprehensive technical evaluation of the security components and their compliance for the purpose of accreditation. A certification process can use safeguard evaluation, risk analysis, verification, testing, and auditing techniques to assess the appropriateness of a specific system, which processes a certain classification level of information within a particular environment. For example, if Dan were the security officer for a company that just purchased new systems to be used to process their confidential data, he would like to know if these systems are appropriate for these tasks and if they are going to provide the necessary level of protection. He could pay a company that specializes in these matters to perform the necessary procedures to certify the systems. The company will perform tests on the software configurations, hardware, firmware, design, implementation, system procedures, and the physical and communication controls. The certification will indicate the good, bad, and the ugly about the security protection level and the mechanisms that support it within these systems and how they work within the given environment. If the outcome of this process looks good to Dan, he will take it to his management to start the accreditation process.

Accreditation

Accreditation is the formal acceptance of the adequacy of a system's overall security and functionality by management. The certification information is presented to management, or the responsible body, and it is up to the management to ask questions, review the reports and findings, and decide upon the acceptance of the safeguards and if any corrective action needs to take place. Once satisfied with the system's overall security as it is presented, management makes a formal accreditation statement. By doing this, management is stating that it understands the level of protection the system will provide in its current environment and understands the security risks associated with installing this system.

To summarize, certification is a technical review that assesses the security mechanisms and controls and evaluates their effectiveness. Accreditation is management's official acceptance of the information in the certification process findings.

Because software, systems, and environments continually change and evolve, the certification and accreditation should also continue to take place. Any major addition of software, change to the system, or modification of the environment should initiate a new certification and accreditation cycle.

7799 Standards

The ***British Standard 7799*** is a risk-based method for assessing, evaluating, and managing risks. It takes a holistic approach to security, instead of just focusing on the technical issues. It is a standard and a framework for developing a security program. This can provide guidelines for organizations all around the world on how to develop and implement a security program and how to improve currently deployed programs. Companies can be certified against these standards, which evaluate and "grade" their overall security posture.

The British Standard is divided into ten sections:

- Security policy
- Security organization
- Assets classification and control
- Personnel security
- Physical and environmental security
- Computer and network management
- System access control
- System development and maintenance
- Business continuity planning
- Compliance

The British Standard 7799 was developed in England (who would of thought) and became a de facto standard. A de facto standard just means that the industry basically stated, "Hey, that's a good idea. Let's all do that." A de facto standard is not dictated by a standards board. Then a standards board did get into the act. The International Organization for Standardization (ISO), which develops standards for 145 countries, formalized the approach a bit and gave it a more international name, ISO I7799, which is also referred to as the Code of Practice for Information Security Management.

Open versus Closed Systems

Computer systems can be developed to integrate easily with other systems and products (open systems) or can be developed to be more proprietary in nature and work with only a subset of other systems and products (closed systems). The following sections describe the difference between these approaches.

Open Systems

I want to be able to work and play well with others.

Systems that are described as *open* are built upon standards, protocols, and interfaces that have published specifications, which enable third-party vendors to develop add-on components and devices. This type of architecture provides interoperability between products by different vendors of different operating systems, applications, and hardware devices. This interoperability is provided by all the vendors involved who follow specific standards and provide interfaces that enable each system to easily communicate with other systems and allow add-ons to hook into the system easily.

A majority of the systems in use today are open systems. The reason that an administrator can have Windows NT 4.0, Windows 2000, Macintosh, and Unix computers on the same network communicating easily is because these platforms are open. If a software vendor creates a closed system, they are restricting their sales to proprietary environments instead of to the whole world.

Closed Systems

I only want to work and play with you and him.

Systems that are referred to as *closed* use an architecture that does not follow industry standards. Interoperability and standard interfaces are not employed to enable easy communication between different types of systems and add-on features. Closed systems are proprietary, meaning that the system can only communicate with like systems.

A closed architecture can provide more security to the system because it does not have as many doorways in, and it operates in a more secluded environment than open environments. Because a closed system is proprietary, there are not as many tools to thwart the security mechanisms and not as many people who understand its design, language, and security weaknesses to exploit. However, more security brings less functionality. A majority of the systems today are built with open architecture to enable them to work with other types of systems, easily share information, and take advantage of the functionality that third-party add-ons bring. However, this opens the doors to more hacks, cracks, and attacks. You can't have your cake and eat it too.

A Few Threats to Security Models and Architectures

Now that we have talked about how everything is supposed to work, let's take a quick look at some of the things that can go wrong when designing a system.

Software almost always has bugs and vulnerabilities. The rich functionality demanded by users brings about deep complexity, which usually opens the doors to problems in the computer world. Also, vulnerabilities are always around because attackers continually find ways of using system operations and functionality in a negative and destructive way. Just like there will always be cops and robbers, there will always be attackers and security professionals. It is a game of trying to outwit each other and seeing who will put the necessary effort into winning the game.

Covert Channels

A ***covert channel*** is a way for an entity to receive information in an unauthorized manner. It is an information flow that is not controlled by a security mechanism or the mechanism has been successfully compromised. This type of information path is usually not used for communication; thus, the system does not properly protect this path because the developers never envisioned information being passed this way. For an entity to receive information in this manner violates the security policy of the system.

There are two types of covert channels: timing and storage. In a ***covert timing channel***, one process relays information to another by modulating its use of system resources. The modulation of system resources can be accessing the hard drive, using excessive CPU cycles, or head placement on a hard drive track. For example, if one process wrote to the hard drive 30 times within 30 seconds this could mean something to another process that is programmed to look for this type of activity. The second process watches out for this "signal" and once it receives it, the second process carries out whatever evil activity it is programmed to do. You can think of it as a type of Morse code, but with the use of some type of system resource.

A ***covert storage channel*** is when a process writes data to a storage location and another process directly, or indirectly, reads it. The problem occurs when the processes are at different security levels, and therefore not supposed to be sharing sensitive data. Maybe an attacker figures out that two processes with different trust levels can view the pagefile.sys file. Process 1 writes some type of confidential information to the pagefile.sys file, and process 2 reads it. This would go against the information flow of the system and directly negate the security policy.

The most common covert channel in use today is the Loki attack. This attack uses the ICMP protocol for communication purposes. This protocol was not developed to be used in this manner, it is only supposed to send status and error messages. But someone developed a tool (Loki) that will allow an attacker to write data right behind the ICMP header. This allows the attacker to communicate to another system through a covert channel. It is usually very successful because most firewalls are configured to allow ICMP traffic in and out of their environments. This is a covert channel because it is using something for communication purposes that was not developed for this type of communication functionality.

NOTE An overt channel is a channel of communication that was developed specifically for communication purposes. Processes should be communicating through overt channels, not covert channels.

Countermeasures

Because all operating systems have some type of covert channels, it is not always feasible to attempt to get rid of them all. The number of acceptable covert channels usually depends on the security rating of a system. A system that has a B2 rating has fewer convert channels than a C1 system. There is not much a user can do to countermeasure these channels; instead, they are addressed when constructing and developing a system.

In the Orange Book, covert channels in operating systems are not addressed until the security level B2 and above because these are the systems that would be holding data sensitive enough for others to go through all the necessary trouble to access data in this fashion.

Detecting a covert channel is very difficult to do; the following are some points about countermeasures:

- It is unlikely that intrusion detection systems will detect this type of attack within an operating system, but it's always worth a try.
- A network and host intrusion detection system can be more successful in identifying covert channels within the network.
- Auditing should be enabled to try and detect a covert channel use pattern, although it can be hard to detect within an operating system or product. Auditing for covert channel violations within the network could be much more successful.

Backdoors

In the programming world, ***backdoors*** are also called ***maintenance hooks***. They are instructions within software that only the developer knows about and can invoke. They are placed in the software for easy access. They also allow the developer to view and edit the code without having to go through regular access controls. During the development phase of the software, these can be very useful, but if they are not removed before going into production, they can cause major security issues.

The backdoor is usually initiated by a random sequence of keystrokes that provides access into the software without having to go through normal access control and security checks and mechanisms.

An application that has a maintenance hook enables the developer to execute specific commands by using a specific sequence of keystrokes. Once this is done successfully, the developer is inside the application looking directly at the code. She might do this to watch problem areas within the code, check variable population, export more code into the program, or fix problems that she sees that are taking place. Although this sounds nice and healthy, if an attacker finds out about this maintenance hook, more sinister actions may be taken. So all backdoors need to be removed from software before it goes into production.

Countermeasures

Because backdoors are usually inserted by programmers, they are the ones who usually have to take them out before the programs and systems go into production. Code reviews and unit and integration testing should always be looking out for backdoors in case the programmer overlooked extracting them. Because backdoors are within the code of an application or system, there is not much a user can do to prevent their presence, but when a vendor finds out that a backdoor exists in their product, they usually develop and release a patch to reduce this vulnerability. Because most vendors sell their software without selling the associated source code with it, it may be very difficult for companies who have purchased software to identify backdoors. The following lists some preventative measures against backdoors:

- A host intrusion detection system can be used to watch for an attacker using a backdoor into the system.
- File system permissions can be set to try and protect configuration files and sensitive information from being modified.
- Strict access control can be added to prevent access to the system in the first place.
- File system encryption can be used to protect sensitive information.
- Auditing should be in place to detect any type of backdoor use.

Asynchronous Attack

Specific attacks can take place on a system that take advantage of the way a system processes requests and performs tasks. An ***asynchronous attack*** deals with the sequences of steps a system uses to complete a task. For example, if a system uses an autoexec.bat file, the system first goes through a boot process and checks to see if an autoexec.bat file exists on the system. If one does not exist, it will continue with its bootup procedures, but if it does exist, it will open the file and process it line by line. There is a timing difference of when the system checks to see if an autoexec.bat file exists and actually executing the contents of the file. There are actually two different and distinct steps here. A ***time-of-check versus time-of-use (TOC/TOU)*** attack could replace the autoexec.bat with a different autoexec.bat that compromises the system before the system actually initiates it.

A similar but different issue is called a ***race condition***. When two different processes need to carry out their tasks on a resource, they need to follow the correct sequence. Process 1 needs to carry out its work before process 2 accesses the same resource and carries out its tasks. If process 2 goes before process 1, the outcome could be very different. If an attacker could manipulate the processes so that process 2 did its thing first, she is controlling the outcome of the processing procedure.

Countermeasures

It would take a dedicated attacker with great precision to perform these types of attacks, but it is possible and has been done. The following can be used to protect against this type of attack:

- A host intrusion detection system can be used to watch for this type of suspicious behavior.
- File system permissions and encryption can be set to protect sensitive files.

- Strict access control measures can be used to prevent an intruder from accessing the system in the first place.
- Auditing should be in place to catch patterns or indications of TOC/TOU attacks.

NOTE This attack would be extremely hard to detect because it deals with the sequence of steps processes take when carrying out their tasks. There are few, if any, protection tools that look at these issues to this level of granularity. It is best addressed during the development of the product.

Buffer Overflows

Buffer overflows happen when programs do not check the length of data that is inputted into a program. The programmer can set the value of a required input to be 80 characters, but does not ensure that only up to 80 are actually inputted from a user or another program. If more than 80 characters are entered, this extra data overflows its allotted memory segment and is executed by the CPU. The extra data can launch another program or be a set of code that was devised to perform disruptive behavior in the system. This is sometimes referred to as "smashing the stack." A buffer overflow is usually aimed at systems that let the extra code be executed with privileged rights, meaning the software executes in a more privileged mode and has access to more critical resources.

Countermeasures

The first and best countermeasure to buffer overflows is proper programming and good coding practices. There are also several types of tools available to help prevent buffer overflows. Some tools monitor dynamic link library (DLL) usage, and other tools provide a wrapper around the kernel that monitors calls and watches for known buffer overflow attacks.

There is not much a user can do to prevent buffer overflows, because they are part of the program code. But when vendors discover these issues, they develop and distribute patches to reduce this type of vulnerability. The following can be used to protect against buffer overflows:

- A host intrusion detection system can be used to watch for this type of suspicious behavior.
- File system permissions and encryption can be set to protect sensitive files.
- Strict access control measures can be used to prevent an intruder from accessing the system in the first place.
- Auditing should be in place to catch patterns or indications of buffer overflow attacks.

Summary

The architecture of a computer system is very important and covers many topics. The system has to make sure that memory is properly segregated and protected, ensure that only authorized subjects access objects, ensure that untrusted processes cannot perform activities that would put other processes at risk, control the flow of information, and define a domain of resources for each subject. It also must ensure that if the computer experiences any type of disruption, it will not result in an insecure state. Many of these issues are dealt with in the system's security policy, and the security model is built to support the requirements of this policy.

Once the security policy, architecture, and model have been developed, the computer operating system, or product, must be built, tested, evaluated, and rated. An evaluation is done by comparing the system to a predefined criteria. The rating that is assigned to the system depends upon how it fulfills the requirements in the criteria. Customers use this rating to understand what they are really buying and how much they can trust this new product. Once the customer buys the product, it must be tested within their own environment to make sure it meets their company's needs, which takes place through certification and accreditation processes.

Quick Tips

- A system can have the exact same hardware, software components, and applications, but provide different levels of protection because of the different security policies and security models that the systems were built upon.
- A CPU contains a control unit, which controls the timing of the execution of instructions and data, and an ALU, which performs mathematical functions and logical operations.
- Most systems use protection rings. The more privileged processes run in the lower ring numbers and have access to all or most of the system resources. Applications run in higher-numbered rings and have access to a smaller amount of resources.
- Operating system processes are executed in privileged or supervisory mode, and applications are executed in user mode, also known as "problem" state.
- Secondary storage is nonvolatile and can be a hard drive, CD-ROM, floppy drive, tape backup, or a zip drive.
- Virtual storage combines RAM and secondary storage so the system seems to have a larger bank of memory.
- A deadlock situation occurs when two processes are trying to access the same resource at the same time or when a process commits a resource and does not release it.
- Security mechanisms can focus on different issues, work at different layers, and vary in complexity.
- The more complex a security mechanism is, the less amount of assurance it can provide.
- Not all system components fall under the TCB, only those that enforce the security policy directly and protect the system. These components are within the security perimeter.

- Components that make up the TCB are hardware, software, and firmware that provides some type of security protection.
- A security perimeter is an imaginary boundary that has trusted components within it (those that make up the TCB) and untrusted components on the outside of the boundary.
- The reference monitor concept is an abstract machine that ensures that all subjects have the necessary access rights before accessing objects. Therefore, it mediates all accesses to objects by subjects.
- The security kernel is the mechanism that actually enforces the rules of the reference monitor concept.
- The security kernel must isolate processes carrying out the reference monitor concept, must be tamperproof, must invoke the reference monitor for each access attempt, and must be small enough to be properly tested.
- A security domain is all the objects available to a subject.
- Processes need to be isolated, which can be done through segmented memory addressing.
- A security policy is a set of rules that dictate how sensitive data is to be managed, protected, and distributed. It provides the security goals that the system must accomplish.
- The level of security a system provides depends upon how well it enforces the security policy.
- A multilevel security system processes data at different classifications (security levels), and users with different clearances (security levels) can use the system.
- Processes should be assigned least privilege so that they have just enough system privileges to fulfill their tasks and no more.
- Some systems provide functionality at different layers of the system, which is called layering. This separates the processes and provides more protection for them individually.
- Data hiding is when the processes at different layers do not know about each other, and therefore do not have a way to communicate to each other. This provides more protection for the data.
- When a class of objects is assigned permissions, it is called abstraction.
- A security model maps the abstract goals of a security policy to computer system terms and concepts. It gives the security policy structure and provides a framework for the system.
- The Bell-LaPadula model deals only with confidentiality, and the Biba and Clark-Wilson models deal with integrity.
- A state machine model deals with the different states a system can enter. If a system starts in a secure state, all state transitions take place securely, and the system shuts down and fails securely, the system will never end up in an insecure state.

- A lattice provides an upper bound and a lower bound of authorized access.
- An information flow security model does not permit data to flow to an object in an insecure manner.
- The Bell-LaPadula model has a simple security rule, which means that a subject cannot read data from a higher level (no read up). The *-property rule means that a subject cannot write to an object at a lower level (no write down). The strong star property rule dictates that a subject can only perform an operation at the exact same classification level, nothing higher, and nothing lower.
- The Biba model does not let subjects write to objects at a higher integrity level (no write up), and it does not let subjects read data at a lower integrity level (no read down). This is done to protect the integrity of the data.
- The Bell-LaPadula model is used mainly in military systems. The Biba and Clark-Wilson models are used in the commercial sector.
- The Clark-Wilson model dictates that subjects can only access objects through applications, that separation of duties is enforced, and auditing is in place.
- If a system is working in a dedicated security mode, it only deals with one level of data classification and all users must have this level of clearance to be able to use the system.
- Compartmented and multilevel security modes enable the system to process data classified at different classification levels.
- Trust means that a system uses all of its protection mechanisms properly to process sensitive data for many types of users. Assurance is the level of confidence you have in this trust and that the protection mechanisms behave properly in all circumstances predictably.
- The lower ratings in the evaluation criteria review the performance of a system and its testing results. The higher ratings look at this information and the system design, development procedures, and documentation.
- The Orange Book, also called Trusted Computer System Evaluation Criteria (TCSEC), was developed to evaluate systems built to be used mainly by the military.
- In the Orange Book, D classification means a system provides minimal security and is used for systems that were evaluated but failed to meet the criteria of higher divisions.
- In the Orange Book, the C division deals with discretionary protection and division B deals with mandatory protection (security labels).
- In the Orange Book, the A division means the system's design and level of protection is verifiable and provides the highest level of assurance and trust.
- In the Orange Book, C2 requires object reuse protection and auditing.
- In the Orange Book, B1 is the first rating that requires security labels.

- In the Orange Book, B2 requires all subjects and devices to have security labels, there must be a trusted path and covert channel analysis, and separate administrator functionality is provided.
- In the Orange Book, B3 requires that a security administrator role is defined, trusted recovery takes place, and the system monitors events and notifies security personnel.
- In the Orange Book, C1 outlines access control to be based on individuals and/or groups. It requires a separation of users and information, and identification and authentication of individual entities.
- The Orange Book deals mainly with stand-alone systems, so a range of books were written to cover many other topics in security. These books are called the Rainbow Series.
- The Red Book, or Trusted Network Interpretation (TNI), provides guidelines for networks and network components.
- The Information Technology Security Evaluation Criteria (ITSEC) was an attempt by European countries to develop and use one set of evaluation criteria instead of several.
- The ITSEC evaluates the assurance and functionality of a system separately, whereas the TCSEC combines the two into one rating.
- The Common Criteria was developed to provide a globally recognized evaluation criteria and is in use today. It combines sections of the TCSEC, ITSEC, CTCPEC, and the Federal Criteria.
- The Common Criteria uses protection profiles and ratings from EAL1 to EAL7.
- Certification is the technical evaluation of a system and its security components. Accreditation is management's formal approval and acceptance of the security provided by a system.
- An open system provides better interoperability with other systems and products. A closed system works within a proprietary environment, which lowers its interoperability and functionality possibilities.
- A covert channel is an unintended communication path that transfers data in a way that violates the security policy. There are two types: timing and storage covert channels.
- A covert timing channel enables a process to relay information to another process by modulating its use of system resources.
- A covert storage channel enables a process to write data to a storage medium so another process can read it.
- A maintenance hook is developed to let a programmer into the application quickly for maintenance or adding functionality. This should be removed before the application goes into production or it can cause a serious security risk.

- An execution domain is where instructions are executed by the CPU. The operating system's instructions are executed in a privileged mode, and applications' instructions are executed in user mode.
- Process isolation ensures that multiple processes can run concurrently and the processes will not interfere with each other or affect each other's memory segments.
- The only processes that need complete system privileges are located in the system's kernel.
- A single state machine processes data of a single security level. A multistate machine processes data at two or more security levels without risk of compromising the system's security.
- Strong typing indicates that there is strong enforcement of abstract data types.
- TOC/TOU stands for time-of-check versus time-of-use. This is a class of asynchronous attacks.
- The Biba model is based on a hierarchical lattice of integrity levels.
- The Biba model addresses the first goal of integrity, which is to prevent unauthorized users from making modifications.
- The Clark-Wilson model addresses all three integrity goals: prevent unauthorized users from making modifications, prevent authorized users from making improper modifications, and maintain internal and external consistency.
- In the Clark-Wilson model, users can only access and manipulate objects through programs. It uses access triple, which is subject-program-object.
- ITSEC was developed for European countries. It was not an international evaluation criterion.

Questions

Please remember that these questions are formatted and asked in a certain way for a reason. You must remember that the CISSP exam is asking questions at a conceptual level. Questions may not always have the perfect answer, and the candidate is advised against always looking for the perfect answer. The candidate should look for the best answer in the list.

1. What flaw creates buffer overflows?
   A. Application executing in privileged mode
   B. Inadequate memory segmentation
   C. Inadequate protection ring use
   D. Insufficient parameter checking
2. The operating system performs all except which of the following tasks?
   A. Memory allocation
   B. Input and output tasks

C. Resource allocation

D. User access to database views

3. If an operating system allows sequential use of an object without refreshing it, what security issue can arise?

A. Disclosure of residual data

B. Unauthorized access to privileged processes

C. Data leakage through covert channels

D. Compromising the execution domain

4. What is the final step in authorizing a system for use in an environment?

A. Certification

B. Security evaluation and rating

C. Accreditation

D. Verification

5. What feature enables code to be executed without the usual security checks?

A. Antivirus software

B. Maintenance hook

C. Timing channel

D. Ready state

6. If a component fails, a system should be designed to do which of the following?

A. Change to a protected execution domain

B. Change to a problem state

C. Change to a more secure state

D. Release all data held in volatile memory

7. What security advantage does firmware have over software?

A. It is difficult to modify without physical access.

B. It requires a smaller memory segment.

C. It does not need to enforce the security policy.

D. It is easier to reprogram.

8. Which is the first level of the Orange Book that requires classification labeling of data?

A. B3

B. B2

C. B1

D. C2

9. Which of the following best describes the reference monitor concept?
    - A. A software component that monitors activity and writes security events to an audit log
    - B. A software component that determines if a user is authorized to perform a requested operation
    - C. A software component that isolates processes and separates privilege and user modes
    - D. A software component that works in the center protection ring and provides interfaces between trusted and untrusted objects
10. The Information Technology Security Evaluation Criteria was developed for which of the following?
    - A. International use
    - B. U.S. use
    - C. European use
    - D. Global use
11. A security kernel contains which of the following?
    - A. Software, hardware, and firmware
    - B. Software, hardware, and system design
    - C. Security policy, protection mechanisms, and software
    - D. Security policy, protection mechanisms, and system design
12. What characteristic of a trusted process does not allow users unrestricted access to sensitive data?
    - A. Process isolation enforcement
    - B. Security domain enforcement
    - C. Need-to-know enforcement
    - D. TCB enforcement
13. The Orange Book states that a system should uniquely identify each user for accountability purposes and ________________.
    - A. Require the user to perform object reuse operations
    - B. Associate this identity with all auditable actions taken by that individual
    - C. Associate this identity with all processes the user initiates
    - D. Require that only that user have access to his specific audit information
14. The trusted computing base (TCB) controls which of the following?
    - A. All trusted processes and software components
    - B. All trusted security policies and implementation mechanisms

C. All trusted software and design mechanisms

D. All trusted software and hardware components

15. What is the imaginary boundary that separates components that maintain security from components that are not security related?

A. Reference monitor

B. Security kernel

C. Security perimeter

D. Security policy

16. Which model deals only with confidentiality?

A. Bell-LaPadula

B. Clark-Wilson

C. Biba

D. Reference monitor

17. What is the best description of a security kernel from a security point of view?

A. Reference monitor

B. Resource manager

C. Memory mapper

D. Security perimeter

18. When is security of a system most effective and economical?

A. If it is designed and implemented from the beginning of the development of the system

B. If it is designed and implemented as a secure and trusted front end

C. If it is customized to fight specific types of attacks

D. If the system is optimized before security is added

19. In secure computing systems, why is there a logical form of separation used between processes?

A. Processes are contained within their own security domains so that each does not make unauthorized accesses to other objects or their resources.

B. Processes are contained within their own security perimeter so that they can only access protection levels above them.

C. Processes are contained within their own security perimeter so that they can only access protection levels equal to them.

D. The separation is hardware and not logical in nature.

20. What type of attack is taking place when a higher level subject writes data to a storage area and a lower level subject reads it?

A. TOC/TOU

B. Covert storage attack

C. Covert timing attack

D. Buffer overflow

21. What type of rating does the Common Criteria give to products?

A. PP

B. EPL

C. EAL

D. A–D

22. Which best describes the *-integrity axiom?

A. No write up in the Biba model

B. No read down in the Biba model

C. No write down in the Bell-LaPadula model

D. No read up in the Bell-LaPadula model

23. Which best describes the simple security rule?

A. No write up in the Biba model

B. No read down in the Biba model

C. No write down in the Bell-LaPadula model

D. No read up in the Bell-LaPadula model

24. Which of the following was the first mathematical model of a multilevel security policy used to define the concept of a security state, modes of access, and outlines rules of access?

A. Biba

B. Bell-LaPadula

C. Clark-Wilson

D. State machine

25. Which of the following is not a characteristic of the Bell-LaPadula model?

A. Confidentiality model

B. Integrity model

C. Developed and used by the U.S. DoD

D. First mathematical multilevel security model

Answers

1. **D.** A buffer overflow takes place when too much data is accepted as input. Programmers should implement the correct security controls to ensure that this does not take place. This means they need to perform bounds checking and parameter checking to ensure only the allowed amount of data is actually accepted and processed by the system.
2. **D.** The operating system has a long list of responsibilities, but implementing database views is not one of them. This is the responsibility of the database management software.
3. **A.** If an object has confidential data and this data is not properly erased before another subject can access it, this leftover or residual data can be accessible. This can compromise the data and system's security by disclosing this confidential information.
4. **C.** Certification is a technical review of a product, and accreditation is management's formal approval of the findings of the certification process. This question asked you which step was the final step of authorizing a system before it is to be used in an environment, and that is what accreditation is all about.
5. **B.** Maintenance hooks get around the system's or application's security and access control checks by allowing whoever knows the key sequence to access the application and most likely its code. Maintenance hook should be removed from any code before it gets into production.
6. **C.** The state machine model dictates that a system should start up securely, carry out secure state transitions, and even fail securely. This means that if the system encounters something it deems as unsafe, it should change to a more secure state for self-preservation and protection.
7. **A.** Firmware is a type of software that is held in a ROM or EROM chip. It is usually used to allow the computer to be able to communicate with some type of peripheral device. The system's BIOS instructions are also held in firmware on the motherboard. In most situations firmware cannot be modified unless someone has physical access to the system. This is different from other types of software that may be modified remotely.
8. **C.** These assurance ratings are from the Orange Book. B levels and on up require security labels to be used, but the question asks which is the first level to require this. B1 comes before B2 and B3, thus it is the correct answer.
9. **B.** A reference monitor is the abstract machine that holds all of the rules of access for the system. The security kernel is the active entity that enforces the reference monitor's rules. They control the access attempts of any and all subjects, a user is just one example of a subject.
10. **C.** In ITSEC, the I does not stand for international, it stands for information. This is a criteria that was developed to be used by European countries to evaluate and rate their security products.

11. **A.** The security kernel makes up the main component of the TCB, which is made up of software, hardware, and firmware. The security kernel performs a lot of different activities to protect the system; enforcing the reference monitor's access rules is just one of those activities.
12. **C.** A system that enforces need-to-know does not allow subjects to access objects unless they have been granted the formal approval, which is based on a need-to-know. This question is targeting MAC-based systems that would use security labels, clearances, and classifications also in its access criteria.
13. **B.** Proper security implementations include tracking individuals and their actions. The users need to be identified uniquely to be able to track their individual activities. If all users logged in and authenticated to a system as user001, the system could never be able to distinguish which user actually carried out specific actions.
14. **D.** The TCB contains and controls all protection mechanisms within the system, whether they be software, hardware, or firmware.
15. **C.** The security perimeter is a boundary between items that are within the TCB and the ones that are not part of the TCB. It is just a mark of delineation between these two groups of items.
16. **A.** The Bell-LaPadula model was developed for the U.S. government with the main goal of keeping sensitive data unreachable to those who were not authorized to access and view it. This model was the first mathematical model of a multilevel security policy used to define the concept of a security state, modes of access, and outlines rules of access. The Biba and Clark-Wilson models do not deal with confidentiality, but with integrity instead.
17. **A.** The security kernel is a portion of the operating system's kernel and enforces the rules outlined in the reference monitor. It is the enforcer of the rules and is invoked each time a subject makes a request to access an object. A portion of the kernel is the resource manager, not the security kernel. A memory mapper is a distinct function of the kernel also, and the security perimeter delineates between what is within the TCB and what is not.
18. **A.** It is difficult to add useful and effective security at the end of developing a product or adding security as a front end to an existing product. Adding security at the end of a project is usually more expensive because it will break items and the team will need to go back to the drawing board and redesign and recode portions of the product.
19. **A.** Processes are assigned their own variables, system resources, and memory segments, which makes up their domain. This is done so that they do not corrupt each other's data or processing activities.
20. **B.** A covert channel is being used when something is using a resource for communication purposes and that is not the reason this resource was created. A process can write to some type of shared media or storage place that another process will be able to access. The first process writes to this media and the second process reads it. This action goes against the security policy of the system.

21. **C.** The Common Criteria uses a different assurance rating system than the previously used systems. It has packages of specifications that must be met for a product to obtain the corresponding rating. These ratings and packages are called evaluation assurance levels (EALs). Once a product achieves any type of rating, customers can view this information on an Evaluated Products List (EPL).
22. **A.** The *-integrity axiom (or star integrity axiom) indicates that a subject of a lower integrity level cannot write to an object of a higher integrity level. This rule is put into place to protect the integrity of the data that resides at the higher level.
23. **D.** The simple security rule is implemented to ensure that any subject at a lower security level cannot view data that resides at a higher level. The reason this type of rule is put into place is to protect the confidentiality of the data that resides at the higher level. This rule is used in the Bell-LaPadula model. Remember that if you see "simple" in a rule it pertains to reading, and * or "star" pertains to writing.
24. **B.** This is a formal definition of the Bell-LaPadula model, which was created and implemented to protect government and military confidential information.
25. **B.** The Bell-LaPadula security model was the first mathematical state machine model that provided multilevel security systems. The model was developed because the U.S. DoD had concerns about the systems it was depending upon to keep its military secrets and confidential information. Bell-LaPadula is a confidentiality model and does not address integrity.