CHAPTER 5

Security Models and Architecture

This chapter presents the following:

- Computer hardware architecture
- Operating system architectures
- Trusted computing base and security mechanisms
- Protection mechanisms within an operating system
- Various security models
- Assurance evaluation criteria and ratings
- Certification and accreditation processes
- Attack types

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—a "blueprint" essentially—and the architecture of a computer and operating system, which fulfills this blueprint.

A ***security policy*** is a statement that outlines how entities access each other, what operations different entities can carry out, what level of protection is required for a system or software product, and what actions should be taken when these requirements are not met. The policy outlines the expectations that the hardware and software must meet to be considered in compliance. A ***security model*** 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 circumstance 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. These are not the organizational security policies that contain management's directives. The systems' security policies, and the models they use, should enforce the higher-level organizational security policy that is in place. A system policy dictates the level of security that should be provided by the individual device or operating system.

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. As stated in previous chapters, information security consists of the following main attributes:

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

These main attributes branch off into more granular security attributes, such as authenticity, accountability, nonrepudiation, and dependability. How does a company know which of these it needs, to what degree they are needed, and whether 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 products 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, internal users sharing trade secrets, 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 regard to securing data. Companies, and the management that runs them, can be held accountable if any one of the many issues previously mentioned goes 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 to provide. 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 that are 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 of the parts of a computer system that are necessary for it to function, including the operating system, memory chips, logic circuits, storage devices, input and output devices, 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 consists 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.

NOTE This chapter interweaves the hardware and operating system architectures and their components to show you how they work together.

Central Processing Unit

The CPU seems complex. How does it work? Response: Black magic. It uses eye of bat, tongue of goat, and some transistors.

The ***central processing unit (CPU)*** is the brain of a computer. In the most general description possible, it fetches instructions from memory and executes them. Although a CPU is a piece of hardware, it has its own instruction sets (provided by the operating system) that are necessary to carry out its tasks. Each CPU type has a specific architecture and set of instructions that it can carry out. The operating system has to be designed to be able to work within this CPU architecture. This is why one operating system may be able to work on a Pentium processor but not a SPARC processor.

NOTE Scalable Processor Architecture (SPARC) is a type of Reduced Instruction Set Computing (RISC) chip developed by Sun Microsystems. SunOS, Solaris, and some Unix operating systems have been developed to work on this type of processor.

The chips within the CPU cover only a couple of square inches, but contain over 40 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 that enable the CPU to keep status information of the data that needs to be processed. A ***register*** is a temporary storage location. Accessing memory to get information on what instructions and data need to be executed is a much slower process than accessing a register, which is a component of the CPU itself. So when the CPU is done with one task, it asks the registers, "Okay, what do I have to do now?" And the registers hold the information that tells the CPU what its next job is.

The actual execution of the instructions is done by the ***arithmetic logic unit (ALU)***. 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 memory addresses are passed to the CPU registers, as shown in Figure 5-1. When the control unit indicates that the CPU can process them, the instructions and data memory addresses are passed to the CPU for actual processing, number crunching, and data manipulation. The results are sent back to the requesting process's memory address.

An operating system and applications are really just made up of lines and lines of instructions. These instructions contain empty variables, which are populated at run

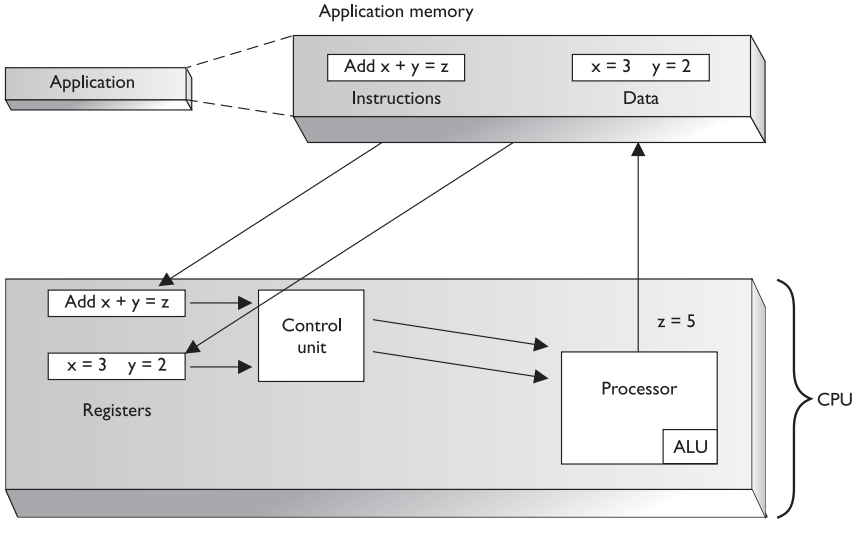


Figure 5-1 Instruction and data addresses are passed to the CPU for processing.

time. The empty variables hold the actual data. There is a difference between instructions and data. The instructions have been written to carry out some type of functionality on the data. For example, let's say that you open a Calculator application. In reality, this program is just lines of instructions that allow you to carry out addition, subtraction, division, and other types of mathematical functions that will be executed on the data that you provide. So, you type in 3 + 5. The 3 and the 5 are the data values. Once you click the = button, the Calculator program tells the CPU that it needs to take the instructions on how to carry out addition and apply these instructions to the two data values 3 and 5. The ALU carries out this instruction and returns the result of 8 to the requesting program. This is when you see the value 8 in the Calculator's field. To users, it seems as though the Calculator program is doing all of this on its own, but it is incapable of this. It depends upon the CPU and other components of the system to carry out this type of activity.

The ***control unit*** manages and synchronizes the system while different applications' code and operating system instructions are being executed. The control unit is the component that fetches the code, interprets the code, and oversees the execution of the different instruction sets. 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 illustrated in Figure 5-2. The CPU's time has to be sliced up into individual units and assigned to processes. It is this time slicing that fools the applications and users into thinking that the system is actually carrying out several different functions at one time. While the operating system can carry out several different functions at one time (multitasking), in reality the CPU is executing the instructions in a serial fashion (one at a time).

A CPU has several different types of registers, containing information about the instruction set and data that needs to be executed. ***General registers*** are used to hold variables and temporary results as the ALU works through its execution steps. The

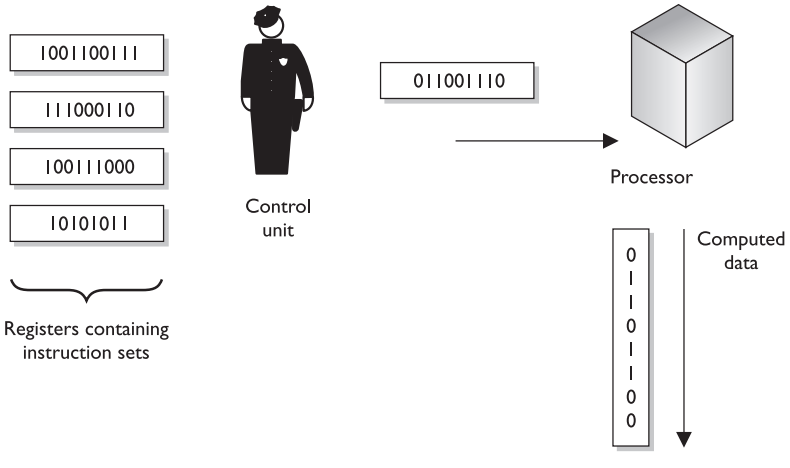


Figure 5-2 The control unit works as a traffic cop, indicating when instructions are sent to the processor.

general registers are like the ALU's scratch pad to use while it is working. ***Special registers*** (dedicated registers) hold information such as the program counter, stack pointer, and program status word (PSW). The ***program counter*** register contains the memory address of the next instruction that needs to be fetched. After that instruction is executed, the program counter is updated with the memory address of the next instruction set that needs to be processed. It is similar to a boss and secretary relationship. The secretary keeps the boss on schedule and points her to the necessary tasks that she needs to carry out. This allows the boss to just concentrate on carrying out the tasks instead of having to worry about the "busy work" that is being done in the background.

Before we get into what a stack pointer is, we first need to know what a stack is. Each process has its own ***stack***, which is a memory segment that the process can read from and write to. Let's say that you and I need to communicate through a stack. What I do is put all of the things that I need to say to you in a stack of papers. The first paper tells you how you can respond to me when you need to, which is called a return pointer. The next paper has some instructions that I need you to carry out. The next piece of paper has the data that I need you to use when you carry out these instructions. So, I write down on individual pieces of paper all that I need you to do for me and *stack* them up. When I am done, I signal to you that I need you to read my stack of papers. You take the first page off of my stack and carry out my request. Then you take the second page and carry out that request. You continue to do this until you are at the bottom of my stack, which contains my return pointer. You look at this return pointer (which is my memory address) to know where to send the results of all the instructions I asked you to carry out. This is how processes communicate to other processes and to the CPU. One process stacks up its information that it needs to communicate to the CPU. The CPU has to keep track of where it is in the stack, which is the purpose of the stack pointer. Once the first item on the stack is executed, then the ***stack pointer*** moves down to direct the CPU where the next piece of data is located.

NOTE The traditional way of explaining how a stack works is to use the analogy of stacking up trays in a cafeteria. When people are done eating, they place their trays on a stack of other trays, and when the cafeteria employees need to get the trays for cleaning, they take the last tray that was put on top and work down the stack. This analogy is used to explain how a stack works in the mode of "last in first off." The process that is being communicated to takes the last piece of data the requesting process laid down off the top of the stack and works down the stack.

The ***program status word (PSW)*** holds different condition bits. One of the bits indicates whether the CPU should be working in ***user mode*** (also called ***problem state***) or ***privileged mode*** (also called ***kernel*** or ***supervisor mode***). The crux of this chapter is to teach you how operating systems protect themselves. They need to protect themselves from applications, utilities, and user activities if they are going to provide a stable and safe environment. One of these protection mechanisms is implemented through the use of these different execution modes. When an *application* needs the CPU to carry out its instructions, the CPU works in user mode. This mode has a lower privilege level and many of the CPU's instructions and functions are not available to the requesting application. The reason for the extra caution is that the developers of the operating system do not know who developed the application or

how it is going to react, so the CPU works in a lower privileged mode when executing these types of instructions. By analogy, if you are expecting visitors who are bringing their two-year-old boy, you move all of the breakables that someone under three feet can reach. No one is ever sure what a two-year-old toddler is going to do, but it usually has to do with breaking something. An operating system and CPU are not sure what applications are going to attempt, which is why this code is executed in a lower priviledge.

If the PSW has a bit value that indicates that the instructions that need to be executed are to be carried out in privileged mode, this means that a trusted process (an operating system process) made the request and can have access to the functionality that is not available in user mode. An example would be if the operating system needed to communicate with a peripheral device. This is a privileged activity that applications cannot carry out. When these types of instructions are passed to the CPU, the PSW is basically telling the CPU, "The process that made this request is an alright guy. We can trust him. Go ahead and carry out this task for him."

Memory addresses of the instructions and data that need to be processed are held in registers until needed by the CPU. The CPU is connected to an ***address bus***, which is a hardwired connection to the RAM chips in the system and the individual input/output (I/O) devices. Memory is cut up into sections that have individual addresses that are associated with them. I/O devices (CD-ROM, USB device, hard drive, floppy drive, and so on) are also allocated specific unique addresses. If the CPU needs to access some data, either from memory or from an I/O device, it sends down the address of where the needed data is located. The circuitry associated with the memory or I/O device recognizes the address that the CPU sent down the address bus and instructs the memory or device to read the requested data and put it on the ***data bus***. So the address bus is used by the CPU to indicate the location of the instructions that need to be processed, and the memory or I/O device responds by sending the data that resides at that memory location through the data bus. This process is illustrated in Figure 5-3.

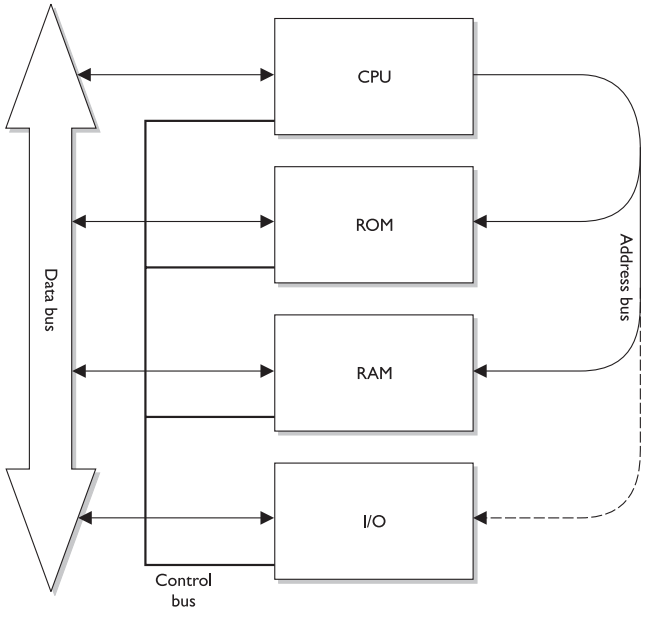


Figure 5-3 Address and data buses are separate and have specific functionality.

Once the CPU is done with its computation, it needs to return the results to the requesting program's memory. So, the CPU sends the requesting program's address down the address bus and sends the new results down the data bus with the command `write`. This new data is then written to the requesting program's memory space.

The address and data buses can be 8, 16, 32, or 64 bits wide. Most systems today use a 32-bit address bus, which means that the system can have a large address space (2^{32}). Systems can also have a 32-bit data bus, which means that the system can move data in parallel back and forth between memory, I/O devices, and the CPU. (A *32-bit data bus* means that the size of the chunks of data that a CPU can request at a time is 32 bits.)

Multiprocessing

Some specialized computers have more than one CPU, for increased performance. An operating system must be developed specifically to be able to understand and work with more than one processor. If the computer system is configured to work in ***symmet-***

Processor Evolution

The following table provides the different characteristics of the various processors that have been used over the years.

Name	Date	Transistors	Microns	Clock Speed	Data Width	MIPS
8080	1974	6000	6	2 MHz	8 bits	0.64
80286	1982	134,000	1.5	6 MHz	16 bits	1
Pentium	1993	3,100,000	0.8	60 MHz	32 bits, 64-bit bus	100
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits, 64-bit bus	1700

The following list defines the terms of measure used in the preceding table:

- **Microns** Indicates the width of the smallest wire on the CPU chip (a human hair is 100 microns thick).
- **Data width** Indicates how much data the ALU can accept and process; 64-bit bus refers to the size of the data bus. So, modern systems fetch 64 bits of data at a time, but the ALU works only on instruction sets in 32-bit sizes.
- **MIPS** Millions of instructions per second, which is a basic indication of how fast a CPU can work (but other factors are involved, such as clock speed).
- **Clock speed** Indicates the speed at which the processor can execute instructions. An internal clock is used to regulate the rate of execution, which is broken down into cycles. A system that runs at 100 MHz means that there are 100 million clock cycles per second. Processors working at 4 GHz are now available, which means that the CPU can execute 4 thousand million cycles per second.

ric mode, this means that the processors are handed work as needed. It is like a load-balancing environment. When a process needs instructions to be executed, a scheduler determines which processor is ready for more work and sends it on. If the system is configured to work in ***asymmetric mode***, this usually means that the computer has some type of time-sensitive application that needs its own personal processor. So, the system scheduler will send instructions from the time-sensitive application to CPU 1 and send all the other instructions (from the operating system and other applications) to CPU 2.

Operating System Architecture

An operating system provides an environment for applications and users to work within. Every operating system is a complex beast, made up of various layers and modules of functionality. It has the responsibility of managing the underlying hardware components, memory management, I/O operations, file system, process management, and providing system services. We next look at each of these responsibilities of every operating system, but you must realize that whole books are written on just these individual topics, so the discussion here is only topical.

Process Management

Well just look at all of these processes squirming around like little worms. We need some real organization here!

Operating systems, utilities, and applications in reality are just lines and lines of instructions. They are static lines of code that are brought to life when they are initialized and put into memory. Applications work as individual units, called processes, and the operating system has several different processes carrying out various types of functionality. A ***process*** is the set of instructions actually running. A program is not considered a process until it is loaded into memory and activated by the operating system. When a process is created, the operating system assigns resources to it, such as a memory segment, CPU time slot (interrupt), access to system application programming interfaces (APIs), and files to interact with. The *collection* of the instructions and the assigned resources is referred to as a process.

The operating system has many processes, which are used to provide and maintain the environment for applications and users to work within. Some examples of the functionality that individual processes provide include displaying data onscreen, spooling print jobs, and saving data to temporary files. Today's operating systems provide ***multiprogramming***, which means that more than one program (or process) can be loaded into memory at the same time. This is what allows you to run your antivirus software, word processor, personal firewall, and e-mail client all at the same time. Each of these applications runs as one or more processes.

NOTE Many resources state that today's operating systems provide multiprogramming and multitasking. This is true, in that multiprogramming just means that more than one application can be loaded into memory at the same time. But in reality, multiprogramming was replaced by multitasking, which means that more than one application can be in memory at the same time *and* the operating system can deal with requests from these different applications *simultaneously*.

Earlier operating systems wasted their most precious resource—CPU time. For example, when a word processor would request to open a file on a floppy drive, the CPU would send the request to the floppy drive and then wait for the floppy drive to initialize, for the head to find the right track and sector, and finally for the floppy drive to send the data via the data bus to the CPU for processing. To avoid this waste of CPU time, multitasking was developed, which enabled more than one program to be loaded into memory at one time. Instead of sitting idle waiting for activity from one process, the CPU could execute instructions for other processes, thereby speeding up the necessary processing required for all the different processes.

As an analogy, if you (CPU) put bread in a toaster (process) and just stand there waiting for the toaster to finish its job, you are wasting time. On the other hand, if you put bread in the toaster and then, while it's toasting, feed the dog, make coffee, and come up with a solution for world peace, you are being more productive and not wasting time.

Operating systems started out as cooperative and then evolved into preemptive multitasking. ***Cooperative multitasking***, used in Windows 3.1 and early Macintosh systems, required the processes to voluntarily release resources that they were using. This was not necessarily a stable environment, because if a programmer did not write his code properly to release a resource when his application was done using it, the resource would be committed indefinitely to his application and thus unavailable to other processes. With ***preemptive multitasking***, used in Windows 9*x*, NT, 2000, XP, and in Unix systems, the operating system controls how long a process can use a resource. The system can suspend a process that is using the CPU and allow another process access to it through the use of *time sharing*. So, in operating systems that used cooperative multitasking, the processes had too much control over resource release, and when an application hung, it usually affected all the other applications and sometimes the operating system itself. Operating systems that use preemptive multitasking run the show, and one application does not negatively affect another application as easily.

Different operating system types work within different process models. For example, Unix and Linux systems allow their processes to create new children processes, which is referred to as *forking*. Let's say that you are working within a shell of a Linux system. That shell is the command interpreter and an interface that enables the user to interact with the operating system. The shell runs as a process. When you type in a shell the command `cat file1 file2 | grep stuff`, you are telling the operating system to concatenate (`cat`) the two files and then search (`grep`) for the lines that have the value of `stuff` in them. When you press the ENTER key, the shell forks two children processes—one for the `cat` command and one for the `grep` command. Each of these children processes takes on the characteristics of the parent process, but has its own memory space, stack, and program counter values.

A process can run in ***running state*** (CPU is executing its instructions and data), ***ready state*** (waiting to send instructions to the CPU), or ***blocked state*** (waiting for input data, such as keystrokes from a user). These different states are illustrated in Figure 5-4. When a process is blocked, it is waiting for some type of data to be sent to it. In the preceding example of typing the command `cat file1 file2 | grep stuff`, the `grep` process cannot actually carry out its functionality of searching until the first pro-

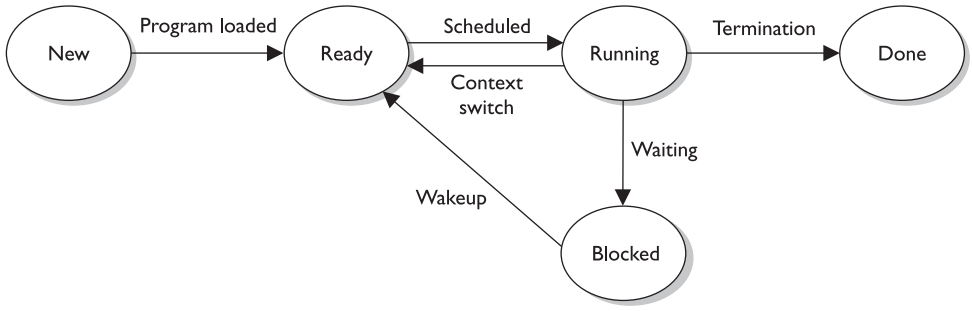


Figure 5-4 Processes enter and exit different states.

cess (`cat`) is done combining the two files. The `grep` process will put itself to ***sleep*** and will be in the blocked state until the `cat` process is done and sends the `grep` process the input it needs to work with.

NOTE Not all operating systems create and work in the process hierarchy as do Unix and Linux systems. Windows systems do not fork new children processes, but instead creates new processes that work independently. This is deeper than what you need to know for the CISSP exam, but life is not just about this exam—right?

The operating system is responsible for creating new processes, assigning them resources, synchronizing their communication, and making sure that nothing insecure is taking place. The operating system keeps a ***process table***, which has one entry per process. The table contains each individual process's state, stack pointer, memory allocation, program counter, and status of open files in use. The reason the operating system documents all of this status information is that the CPU needs all of it loaded into its registers when it needs to interact with, for example, process 1. When process 1's CPU time slice is over, all of the current status information on process 1 is stored in the process table so that when its time slice is open again, all of this status information can be put back into the CPU registers. So, when it is process 2's time with the CPU, its status information is transferred from the process table to the CPU registers, and transferred back again when the time slice is over. These steps are shown in Figure 5-5.

How does a process know when it can communicate with the CPU? This is taken care of by using ***interrupts***. An operating system fools us, and applications, into thinking that it and the CPU are carrying out all tasks (operating system, applications, memory, I/O, and user activities) simultaneously. In fact, this is impossible. Most CPUs can do only one thing at a time. So the system has hardware and software interrupts. When a device needs to communicate with the CPU, it has to wait for its interrupt to be called upon. The same thing happens in software. Each process has an interrupt assigned to it.

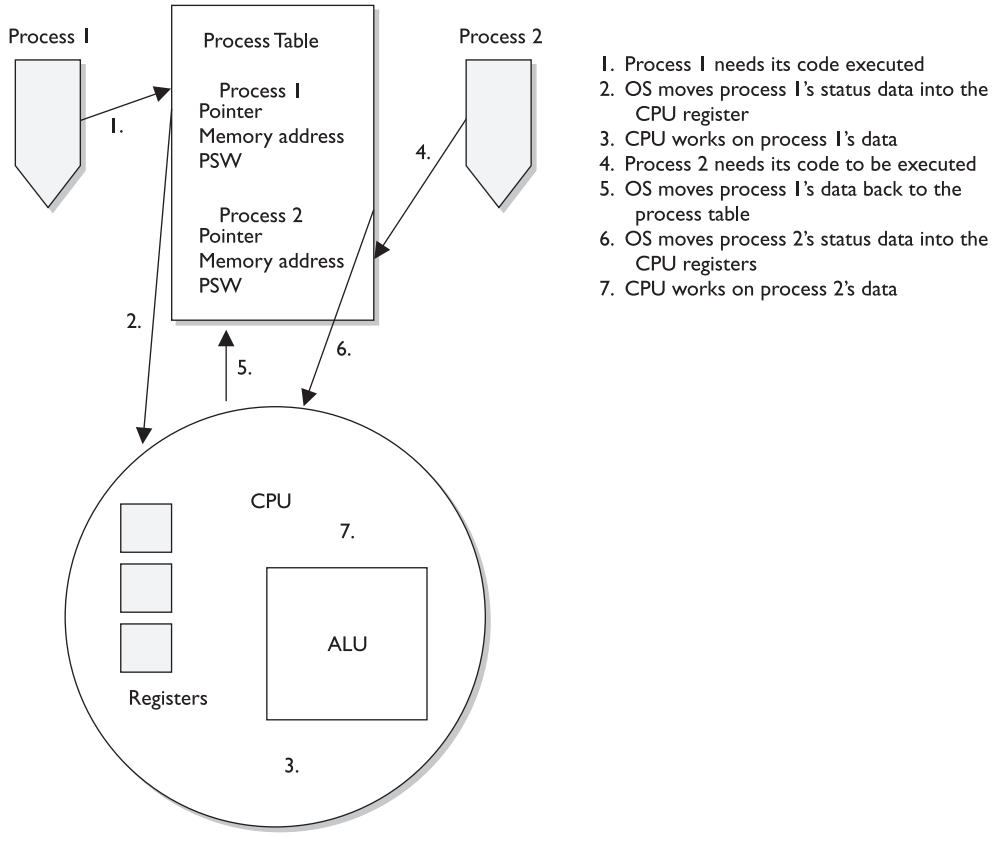


Figure 5-5 A process table contains process status data that the CPU requires.

It is like pulling a number at a customer service department in a store. You can't go up to the counter until your number has been called out.

When a process is interacting with the CPU and an interrupt takes place (another process has requested access to the CPU), the current process's information is stored in the process table, and the next process gets its time to interact with the CPU.

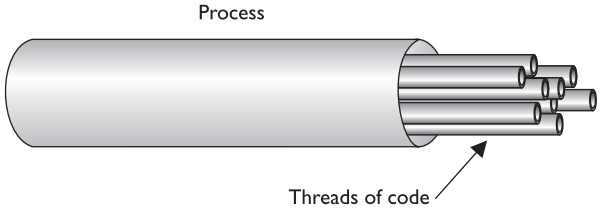
NOTE Some critical processes cannot afford to have their functionality interrupted by another process. The operating system is responsible for setting the priorities for the different processes. When one process needs to interrupt another process, the operating system compares the priority levels of the two processes to determine if this interruption should be allowed.

Thread Management

What are all of these hair-like things hanging off of my processes? Response: Threads.

As described earlier, a process is a program in memory; more precisely, a process is the program's instructions and all the resources that are assigned to the process by the operating system. It is just easier to group all of these instructions and resources to-

gether and control it as one entity, which is a process. When a process needs to send something to the CPU for processing, it generates a thread. A ***thread*** is made up of an individual instruction set and the data that needs to be worked on by the CPU.



Most applications have several different functions. Word processors can open files, save files, open other programs (such as an e-mail client), and print documents. Each one of these functions requires a thread (instruction set) to be dynamically generated. So, for example, if Tom chooses to print his document, the word processor process generates a thread that contains the instructions of how this document should be printed (font, colors, text, margins, and so on). If he chooses to send a document via e-mail through this program, another thread is created that tells the e-mail client to open and what file needs to be sent. Threads are dynamically created and destroyed as needed. Once Tom is done printing his document, the thread that was generated for this functionality is destroyed.

A program that has been developed to carry out several different tasks at one time (display, print, interact with other programs) is capable of running several different threads simultaneously. An application with this capability is referred to as a ***multithreaded*** application. An operating system that can understand and deal with the requests of several different threads, through the use of interrupts, is a multithreading system.

NOTE Each thread shares the same resources of the process that created it. So, all the threads created by a word processor work in the same memory space and have access to all the same files and system resources.

Definitions

The concepts of how computer operating systems work can be overwhelming at times. For test purposes make sure that you understand the following definitions:

- **Multiprogramming** An operating system can load more than one program in memory at one time.
- **Multitasking** An operating system can handle requests from several different processes loaded into memory at the same time.
- **Multithreading** An application has the ability to run multiple threads simultaneously. An operating system can handle requests from several different threads at the same time.
- **Multiprocessing** The computer has more than one CPU.

Process Scheduling

Scheduling and synchronizing various processes and their activities is part of process management, which is a responsibility of the operating system. Several components need to be considered during the development of an operating system, which will dictate how process scheduling will take place. A scheduling policy is created to govern how threads will interact with other threads. Different operating systems can use different schedulers, which are basically algorithms that control the timesharing of the CPU. As stated earlier, the different processes are assigned different priority levels (interrupts), which dictate which processes overrule other processes when CPU time allocation is required. The operating system creates and deletes processes as needed, and oversees them changing state (ready, blocked, running). The operating system is also responsible for controlling deadlocks between processes that are attempting to use the same resources.

When a process makes a request for a resource (memory allocation, printer, secondary storage devices, disk space, etc.), the operating system creates certain data structures and dedicates the necessary processes for the activity to be completed. Once the action takes place (a document is printed, a file is saved, or data is retrieved from the drive), the process needs to tear down these built structures and release the resources back to the resource pool so that they are available for other processes. If this does not happen properly, a ***deadlock*** situation may occur or a computer may not have enough resources to process other requests (resulting in a denial of service). A deadlock situation may occur when each process in a set of processes is waiting for an event to take place and that event can only be caused by another process in the set. Because each process is waiting for its required event, none of the processes will carry out their events—so the processes just sit there staring at each other.

One example of a 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 in deadlock because they do not have the resources they need to finish the function they are trying to carry out. This situation does not take place as often as it used to, as a result of better programming. Also, operating systems now have the intelligence to detect this activity and either release committed resources or control allocation of resources so that they are 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 resource is unavailable for a certain period of time, the operating system kills the process that is "holding on to" that resource. This action releases the resource from the process that had committed it and restarts the process so that it is "clean" and available to be used by other applications. Other operating systems might require a program to request all the resources it needs *before* it actually starts executing instructions, or require a program to release its currently committed resources before it may acquire more.

Process Activity

Process 1, go into your room and play with your toys. Process 2, go into your room and play with your set of toys. No intermingling and no fighting!

Computers can run different applications and processes at the same time. The processes 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 processes. 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 through the use of 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.

To protect processes from each other, operating systems can implement process isolation. ***Process isolation*** is necessary to ensure that processes do not "step on each other's toes," communicate in an insecure manner, or negatively affect each other's productivity. Older operating systems did not enforce process isolation as well as systems do today. This is why in earlier operating systems, when one of your programs hung, all other programs, and sometimes the operating system itself, hung. With process isolation, if one process hangs for some reason, it will not affect the other software running. (Process isolation is required for preemptive multitasking.) There are different methods for carrying out process isolation:

- Encapsulation of objects
- Time multiplexing of shared resources
- Naming distinctions
- Virtual mapping

When a process is ***encapsulated***, no other process understands or interacts with its internal programming code. When process A needs to communicate with process B, process A just needs to know how to communicate with process B's interface. An interface defines how communication must take place between two processes. As an analogy, think back to how you had to communicate with your third-grade teacher. You had to call her Mrs. SoandSo, say please and thank you, and speak respectfully to get whatever it was you needed. The same thing is true for software components that need to communicate with each other. They have to know *how* to communicate properly with each other's interfaces. The interfaces dictate the type of requests that a process will accept and the type of output that will be provided. So, two processes can communicate with each other, even if they are written in different programming languages, as long as they know how to communicate with each other's interface. Encapsulation provides *data hiding*, which means that outside software components will not know how a process works and will not be able to manipulate the process's internal code. This is an integrity mechanism and enforces modularity in programming code.

Time multiplexing was already discussed, although we did not use this term. ***Time multiplexing*** is a technology that allows processes to use the same resources. As stated earlier, a CPU has to be shared between many processes. Although it seems as though all applications are running (executing their instructions) simultaneously, the operating system is splitting up time shares between each process. Multiplexing means that

there are several data sources and the individual data pieces are piped into one communication channel. In this instance, the operating system is coordinating the different requests from the different processes and piping them through the one shared CPU. An operating system has to provide proper time multiplexing (resource sharing) to ensure that a stable working environment exists for software and users.

Naming distinctions just means that the different processes have their own name or identification value. Processes are usually assigned process identification (PID) values, which the operating system and other processes use to call upon them. If each process is isolated, that means that each process has its own unique PID value.

Virtual mapping is different from physical mapping of memory. An application is written such that basically it thinks that it is the only program running on an operating system. When an application needs memory to work with, it tells the operating system's memory manager how much memory it needs. The operating system carves out that amount of memory and assigns it to the requesting application. The application uses its own address scheme, which usually starts at 0, but in reality, the application does not work in the *physical* address space that it thinks it is working in. Rather, it works in the address space that the memory manager assigns to it. The physical memory is the RAM chips in the system. The operating system chops up this memory and assigns portions of it to the requesting processes. Once the process is assigned its own memory space, then it can address this portion however it needs to, which is called virtual address mapping. Virtual address mapping allows the different processes to have their own memory space; the memory manager ensures that no processes improperly interact with another process's memory. This provides integrity and confidentiality.

Memory Management

To provide a safe and stable environment, an operating system must exercise proper memory management, which is one of the most important tasks that it carries out, because basically everything happens in memory. It is similar to how we are dependent upon oxygen and gravity to work. If either is not properly managed by nature, we are in trouble.

The goals of memory management are to

- Provide an abstraction level for programmers
- Maximize performance with the limited amount of memory available
- Protect the operating system and applications loaded into memory

Abstraction means that the details of something are hidden. Developers of applications do not know the amount or type of memory that will be available in each and every system their code will be loaded on. If a developer had to be concerned with this type of detail, then her application would be able to work only on the one system that maps to all of her specifications. To allow for portability, the memory manager hides all of the memory issues and just provides the application with a memory segment.

Every computer has a memory hierarchy. There are small amounts of memory that are very fast and expensive (registers, cache) and larger amounts of memory that are slower and less expensive (RAM, hard drive). The portion of the operating system that keeps track of how these different types of memory are used is lovingly called the mem-

ory manager. Its jobs are to allocate and deallocate different memory segments, enforce access control to ensure that processes are interacting only with their own memory segments, and swap memory contents from RAM to the hard drive.

The memory manager has five basic responsibilities:

- **Relocation**
  - Swap contents from RAM to the hard drive as needed (explained later in the "Virtual Memory" section)
  - Provide pointers for applications if their instructions and memory segment have been moved to a different location in main memory
- **Protection**
  - Limit processes to interact only with the memory segments that are assigned to them
  - Provide access control to memory segments
- **Sharing**
  - Use complex controls to ensure integrity and confidentiality when processes need to use the same shared memory segments
  - Allow many users with different levels of access to interact with the same application running in one memory segment
- **Logical organization**
  - Allow for the sharing of specific software modules, such as dynamic link library (DLL) procedures
- **Physical organization**
  - Segment the physical memory space for application and operating system processes

NOTE A dynamic link library (DLL) is a set of functions that applications can call upon to carry out different types of procedures. For example, the Windows operating system has a crypt32.dll that is used by the operating system and applications for cryptographic functions. Windows has a set of DLLs, which is just a library of functions to be called upon.

How can an operating system make sure that a process only interacts with its memory segment? When a process creates a thread, because it needs some instructions and data processed, there are two registers used by the CPU. A ***base register*** contains the beginning address that was assigned to the process and a ***limit register*** contains the ending address, as illustrated in Figure 5-6. The thread contains an address of where the instruction and data reside that need to be processed. The CPU compares this address to the base and limit registers to make sure that the thread is not trying to access a memory segment outside of its bounds. So, the base register makes it impossible for a thread to reference a memory address below its allocated memory segment, and the limit register makes it impossible for a thread to reference a memory address above this segment.

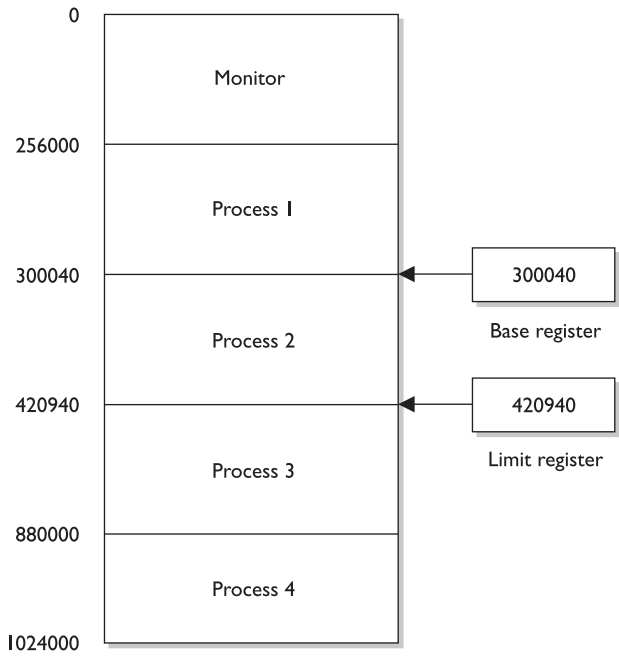


Figure 5-6 Base and limit registers are used to contain a process in its own memory segment.

Memory is also protected through the use of user and privileged modes of execution, as previously mentioned, and covered in more detail later in the "CPU Modes and Protection Rings" section.

Memory Types

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 sections outline the different types of memory that can be used within computer systems.

Random Access Memory

Random access memory (RAM) is a type of temporary storage facility where data and program instructions can temporarily be held and altered. It is used for read/write ac-

> **Hardware Segmentation**
> 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.

tivities 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.

RAM is an integrated circuit made up of millions of transistors and capacitors. The capacitor is where the actual charge is stored, which represents a 1 or 0 to the system. The transistor acts like a gate or a switch. A capacitor that is storing a binary value of 1 has several electrons stored in it, which have a negative charge, whereas a capacitor that is storing a 0 value is empty. When the operating system writes over a 1 bit with a 0 bit, in reality it is just emptying out the electrons from that specific capacitor.

One problem is that these capacitors cannot keep their charge for long. Therefore, a memory controller has to "recharge" the values in the capacitors, which just means that it continually reads and writes the same values to the capacitors. If the memory controller does not "refresh" the value of 1, the capacitor will start losing its electrons and become a 0 or a corrupted value. This explains how ***dynamic RAM (DRAM)*** works. The data being held in the RAM memory cells has to be continually and dynamically refreshed, so that your bits do not magically disappear. This activity of constantly refreshing takes time, which is why DRAM is slower than static RAM.

NOTE When we are dealing with memory activities, we use a time metric of nanoseconds (ns), which is a billionth of a second. So if you look at your RAM chip and it states 70 ns, this means that it takes 70 nanoseconds to read and refresh each memory cell.

Static RAM (SRAM) does not require this continuous-refreshing nonsense; it uses a different technology, by holding bits in its memory cells without the use of capacitors, but it *does* require more transistors than DRAM. Since SRAM does not need to be refreshed, it is faster than DRAM, but since SRAM requires more transistors, it takes up more space on the RAM chip. Manufacturers cannot fit as many SRAM memory cells on a memory chip as they can DRAM memory cells, which is why SRAM is more expensive. So, DRAM is cheaper and slower, and SRAM is more expensive and faster. It always seems to go that way. SRAM has been used in cache, and DRAM is commonly used in RAM chips.

Because life is not confusing enough, we have many other types of RAM. The main reason for the continual evolution of RAM types is that it directly affects the speed of the computer itself. Many people, mistakenly, think that just because you have a fast processor, your computer will be fast. However, memory type and size and bus sizes are also critical components. Think of memory as pieces of paper used by the system to hold instructions. If the system had small pieces of papers (small amount of memory) to read and write from, it would spend most of its time looking for these pieces and lining them up properly. When a computer spends more time moving data from one small portion of memory to another than actually processing the data it is referred to as *thrashing*. This causes the system to crawl in speed and your frustration level to increase.

The size of the data bus also makes a difference in system speed. You can think of a data bus as a highway that connects different portions of the computer. If a ton of data needs to go from memory to the CPU and can only travel over a 4-lane highway, compared to a 64-lane highway, there will be delays in processing. So the processor, memory type and amount, and bus speeds are critical components to system performance.

The following are additional types of RAM that you need to understand:

- **Synchronous DRAM (SDRAM)** Synchronizes itself with the system's CPU and synchronizes signal input and output on the RAM chip. It coordinates its activities with the CPU clock so that the timing of the CPU and the timing of the memory activities are synchronized. This increases the speed of transmitting and executing data.
- **Extended data out DRAM (EDO DRAM)** Is faster than DRAM because DRAM can access only one block of data at a time, whereas EDO DRAM can capture the next block of data while the first block is being sent to the CPU for processing. It has a type of "look ahead" feature that speeds up memory access.
- **Burst EDO DRAM (BEDO DRAM)** Works like (and builds upon) EDO DRAM in that it can transmit data to the CPU as it carries out a read option, but it can send more data at once (burst). It reads and sends up to four memory addresses in a small number of clock cycles.
- **Double data rate SDRAM (DDR SDRAM)** Carries out read operations on the rising and falling cycles of a clock pulse. So instead of carrying out one operation per clock cycle, it carries out two and thus can deliver twice the throughput of SDRAM. Basically, it doubles the speed of memory activities, when compared to SDRAM, with a smaller number of clock cycles. Pretty groovy.

NOTE These different RAM types require different controller chips to interface with them; therefore, the motherboards that these memory types are used on often are very specific in nature.

Well, that's enough about RAM for now. Let's look at other types of memory that are used in basically every computer in the world.

Read-Only Memory

Read-only memory (ROM) is a *nonvolatile* memory type, meaning that when a computer's power is turned off, the data is still held within the memory chips. When data is inserted into ROM memory chips, it cannot be altered. Individual ROM chips are manufactured with the stored program or routines designed into it. The software that is stored within ROM is called *firmware.*

Programmable read-only memory (PROM) is a form of ROM that can be modified after it has been manufactured. PROM can be programmed only one time because the voltage that is used to write bits into the memory cells actually burns out the fuses that connect the individual memory cells. The instructions are "burned into" PROM using a specialized PROM programmer device.

Erasable and programmable read-only memory (EPROM) can be erased, modified, and upgraded. EPROM holds data that can be electrically erased or written to. To erase the data on the memory chip, you need your handy-dandy ultraviolet (UV) light device that provides just the right level of energy. The EPROM chip has a quartz window, which is where you point the UV light. Although playing with UV light devices can be fun for the whole family, we have moved on to another type of ROM technology that does not require this type of activity.

To erase an EPROM chip, you must remove the chip from the computer and wave your magic UV wand, which erases *all* of the data on the chip—not just portions of it. So someone invented ***electrically erasable programmable read-only memory (EEPROM)***, and we all put our UV light wands away for good.

EEPROM is similar to EPROM, but its data storage can be erased and modified electrically by onboard programming circuitry and signals. This activity erases only one byte at a time, which is slow. And because we are an impatient society, yet another technology was developed that is very similar, but works more quickly.

Flash memory is a special type of memory that is used in digital cameras, BIOS chips, memory cards for laptops, and video game consoles. It is a solid-state technology, meaning that it does not have moving parts and is used more as a type of hard drive than memory.

Flash memory basically moves around different levels of voltages to indicate that a 1 or 0 needs to be held in a specific address. It acts as a ROM technology rather than a RAM technology. (For example, you do not lose pictures stored on your memory stick in your digital camera just because your camera loses power. RAM is volatile and ROM is nonvolatile.) When Flash memory needs to be erased and turned back to its original state, a program initiates the internal circuits to apply an electric field. The erasing function takes place in blocks or on the entire chip instead of erasing one byte at a time.

Flash memory is used as a small disk drive in most implementations. Its benefits over a regular hard drive are that it is smaller, faster, and lighter. So let's deploy Flash memory everywhere and replace our hard drives! Maybe one day. Today it is relatively expensive compared to regular hard drives.

References

- **Unix/Linux Internals Course and Links** www.softpanorama.org/Internals
- **Linux Knowledge Base and Tutorial** www.linux-tutorial.info/modules.php?name=Tutorial&pageid=117
- **Fast, Smart RAM, Peter Wayner, Byte.com (June 1995)** www.byte.com/art/9506/sec10/art2.htm

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. When the system assumes (through its programmatic logic) that it will need to access specific information many times throughout its processing activities, it will store the information in cache memory so that it is easily and quickly accessible. Data

in cache can be accessed much more quickly than data stored in real memory. Therefore, any information needed by the CPU very quickly, and very often, is usually stored in cache memory, thereby improving the overall speed of the computer system.

An analogy is how the brain stores information that it uses often. If one of Marge's primary functions at her job is to order parts, which requires telling vendors the company's address, Marge stores this address information in a portion of her brain from which she can easily and quickly access 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 from specialized cache memory.

NOTE Different motherboards have different types of cache. Level 1 (L1) is faster than Level 2 (L2), and L2 is faster than L3. Some processors and device controllers have cache memory built into them. L1 and L2 are usually built into the processors and the controllers themselves.

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 and that sensitive information is not available to unauthorized processes. 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 (logical addresses) to memory segments. The CPU has physical wires connecting it to the memory chips within the computer. Because physical wires connect 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 logical memory addresses. Accessing memory indirectly provides an access control layer between the software and the memory, which is done for protection and efficiency. Figure 5-7 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 that you can talk to him about the land and determine whether you want to meet him 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 access memory

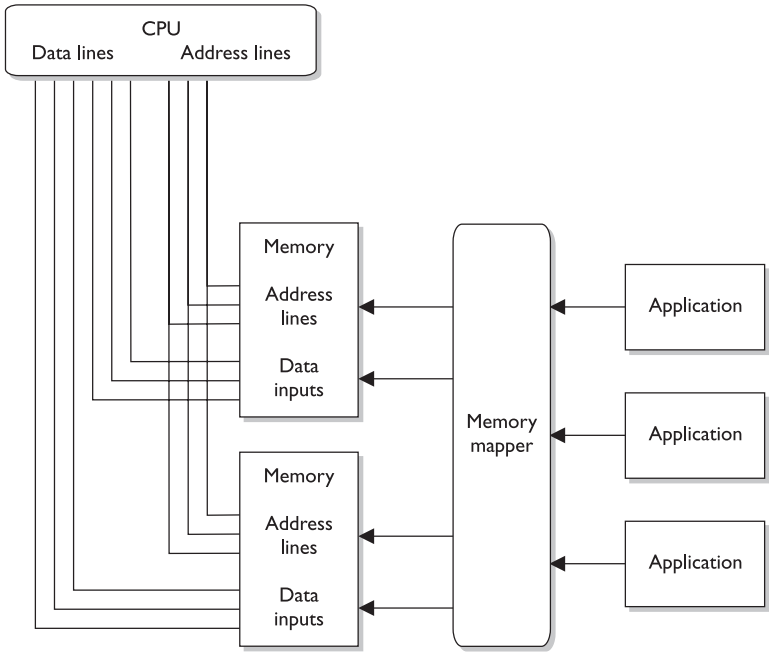


Figure 5-7 The CPU and applications access memory differently.

indirectly by using index tables and pointers, instead of giving them the right to access the memory directly. 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 access the memory allocated to them, as shown in Figure 5-8. This type of memory architecture provides protection and efficiency.

The physical memory addresses that the CPU uses are called ***absolute addresses***. The indexed memory addresses that software uses are referred to as ***logical addresses***. And ***relative addresses*** are based on a known address with an offset value applied. As explained previously, an application does not "know" that it is sharing memory with other applications. When the program needs a memory segment to work with, it tells the memory manager how much memory it needs. The memory manager allocates this much physical memory, which could have the physical addressing of 34,000 to 39,000, for example. But the application is not written to call upon addresses in this numbering scheme. It is most likely developed to call upon addresses starting with 0 and extending to, let's say, 5000. So the memory manager allows the application to use its own addressing scheme—the logical addresses. When the application makes a call to one of these "phantom" logical addresses, the memory manager must map this address to the actual physical address. (It's like two people using their own naming scheme. When Bob asks Diane for a ball, Diane knows he really means a stapler. Don't judge Bob and Diane, it works for them.)

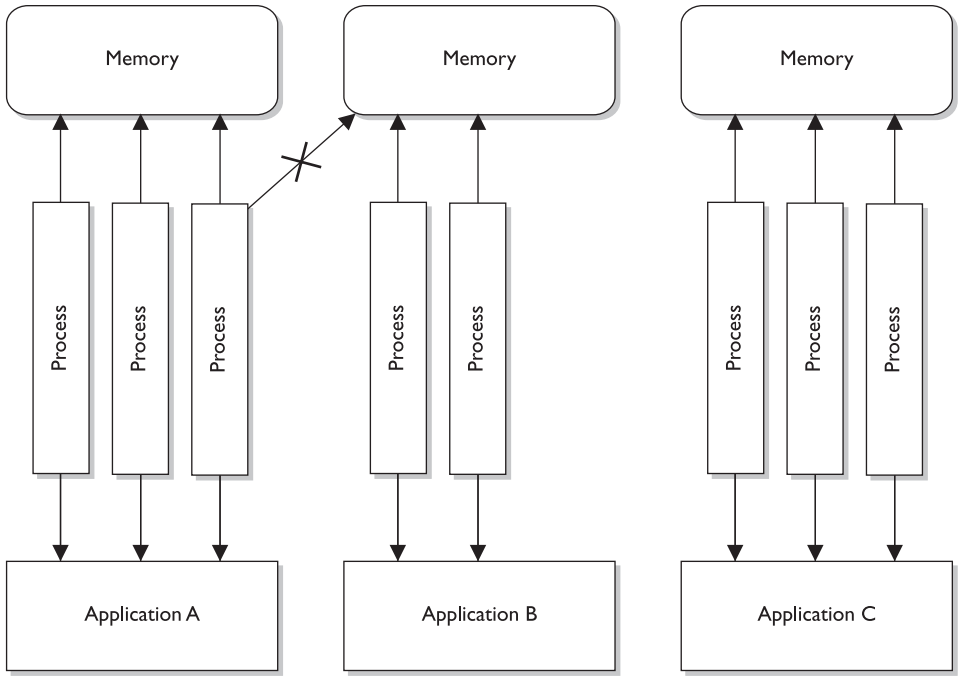


Figure 5-8 Applications, and the processes they use, access their own memory segments only.

The mapping process is illustrated in Figure 5-9. When an application needs its instructions and data to be processed by the CPU, the physical addresses are loaded into the base and limit registers. When a thread indicates the instruction that needs to be processed, it provides a logical address. The memory manager maps the logical address to the physical address, so the CPU knows where the instruction is located. The thread will actually be using a relative address, because the application uses the address space of 0 to 5000. When the thread indicates that it needs the instruction at the memory address 3400 to be executed, the memory manager has to work from its mapping of logical address 0 to the actual physical address and then figure out the physical address for the logical address 3400. So the logical address 3400 is relative to the starting address 0.

As an analogy, if I know that you use a different number system than everyone else in the world, and you tell me that you need 14 cookies, I would need to know where to start in *your* number scheme to figure out how many cookies to really give you. So, if you inform me that in "your world" your numbering scheme starts at 5, I would map 5 to 0 and know that the offset is a value of 5. So when you tell me you want 14 cookies (the relative number), I take the offset value into consideration. I know that you start at the value 5, so I map your logical address of 14 to the physical number of 8. (But I would not give you 8 cookies, because you made me work too hard to figure all of this out. I will just eat them myself.)

So the application is working in its "own world" using its "own addresses," and the memory manager has to map these values to reality, which means the absolute address values.

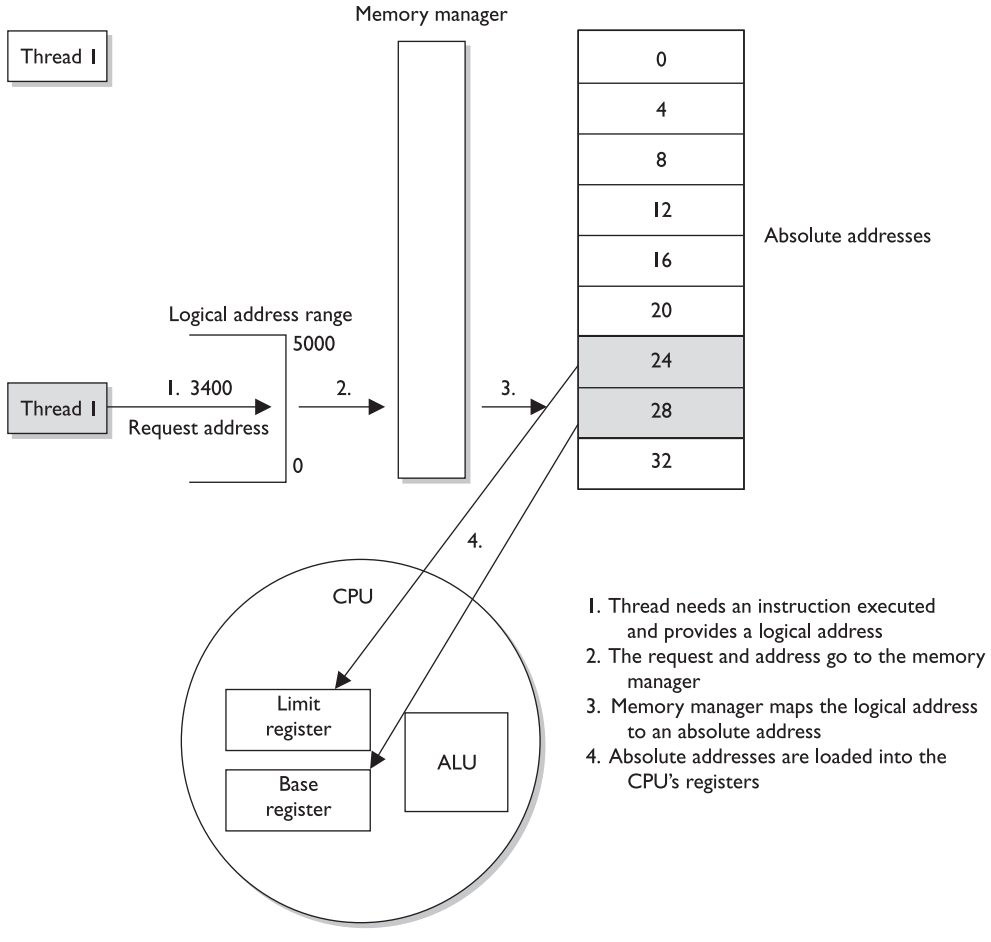


Figure 5-9 The CPU uses absolute addresses and software uses logical addresses.

Memory Leaks

Oh great, the memory leaked all over me. Does someone have a mop?

When an application makes a request for a memory segment, it is allocated a specific memory amount by the operating system. When the application is done with the memory, it is supposed to tell the operating system to release the memory so that it is available to other applications. This is only fair. But some applications are written poorly and do not indicate to the system that this memory is no longer in use. If this happens enough times, the operating system could become "starved" for memory, which would drastically affect the system's performance.

When a memory leak is identified in the hacker world, this opens the door to new denial-of-service (DoS) attacks. For example, when it was uncovered that a Unix application and a specific version of a Telnet protocol contained memory leaks, hackers amplified the problem. They continually sent requests to systems with these vulnerabilities. The systems would allocate resources for these network requests, which in turn would cause more and more memory to be allocated and not returned. Eventually the systems would run out of memory and freeze.

NOTE Memory leaks can be caused by operating systems, applications, and software drivers.

There are two main countermeasures to protect against memory leaks: develop better code that releases memory properly, and use a ***garbage collector***. A garbage collector is software that runs an algorithm to identify unused committed memory and then tells the operating system to mark that memory as "available." There are different types of garbage collectors that work with different operating systems, programming languages, and algorithms.

Virtual Memory

My RAM is overflowing! Can I use some of your hard drive space? Response: No, I don't like you.

Secondary storage is considered nonvolatile storage media and includes such things as the computer's hard drive, floppy disks, or CD-ROMs. When RAM and secondary storage are combined, the result is ***virtual memory***. The system uses hard drive space to extend its RAM memory space. *Swap space* is the reserved hard drive space that is used to extend RAM capabilities. Windows systems use the pagefile.sys file to reserve this space. When a system fills up its volatile memory space, it writes data from memory onto the hard drive. When a program requests access to this data, it is brought from the hard drive back into memory in specific units, called *page frames*. 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 physical disk read/write access has to take place. There are internal control blocks, maintained by the operating system, to keep track of what page frames are residing in RAM, and what is available "offline," ready to be called into RAM for execution or processing, if needed. The payoff is that it seems as though the system can hold an incredible amount of information and program instructions in memory, as shown in Figure 5-10.

A security issue with using virtual swap space is that when the system is shut down, or processes that were using the swap space are terminated, the pointers to the pages are reset to "available" even though the actual data that was written to disk is still physically there. This data could conceivably be compromised and captured. On a very secure operating system, there are routines to wipe the swap spaces after a process is done with it, before it is used again. The routines should also erase this data before a system shutdown, at which time the operating system would no longer be able to maintain any control over what happens on the hard drive surface.

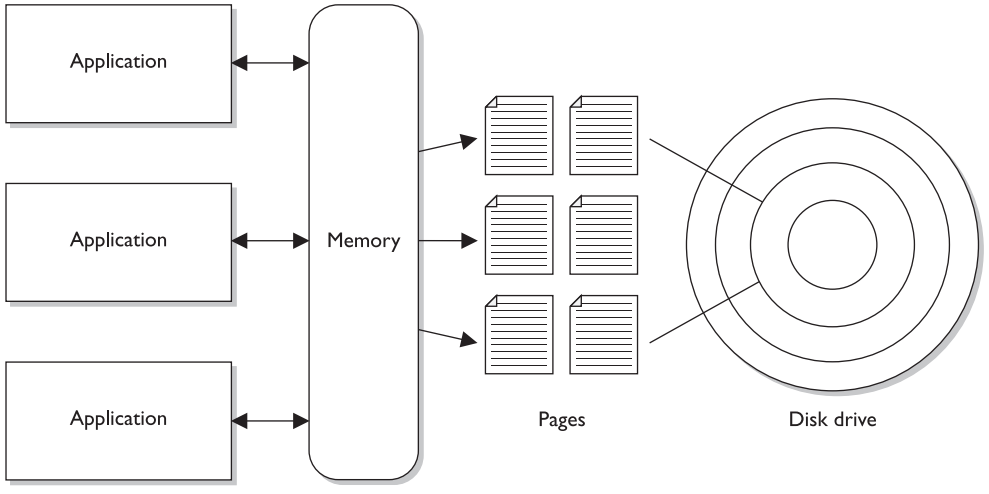


Figure 5-10 Data from RAM is moved to the hard drive in page frames.

NOTE If a program, file, or data is encrypted and saved on the hard drive, it will be decrypted when it is used by the controlling program. While this unencrypted data is sitting in RAM, the system could write out that data to the swap space on the hard drive, in its unencrypted state. Attackers have figured out how to gain access to this space in unauthorized manners.

References

- **"Introduction to Virtual Memory," by Tuncay Basar, Kyung Kim, and Bill Lemley** www.cne.gmu.edu/itcore/virtualmemory/vmintro.html
- **Memory Hierarchy** http://courses.ece.uiuc.edu/ece411/lectures/LecturesSpring05/Lectures_2.07.pdf
- **"Virtual Memory," by Prof. Bruce Jacob, University of Maryland at College Park (2001)** www.ee.umd.edu/~blj/papers/CS-chapter.pdf
- **LabMice.net links to articles on memory leaks** http://labmice.techtarget.com/troubleshooting/memoryleaks.htm

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 may be accessing memory segments, sending instructions to the CPU for processing, and accessing secondary storage devices at the same time. Each user application

(e-mail client, antivirus program, web browser, word processor, personal firewall, and so on) may also 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 system's overall 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 protection. 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, because the inner rings only permit the most trusted components and processes to operate within them. Although operating systems may vary in the number of protection rings they use, processes that execute within the inner rings are usually referred to as existing in privileged, or supervisor, mode. The processes working in the outer rings are said to execute in user mode.

NOTE The actual ring architecture that is used by a system is dictated by the processor and the operating system. The hardware chip (processor) is constructed to provide a certain number of rings, and the operating system must be developed to also work in this ring structure. This is one reason why an operating system platform may work with an Intel chip but not an Alpha chip, for example. 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 services or system calls. The different rings are illustrated in Figure 5-11. 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 or exception error and attempt to shut down the offending application.

Protection rings support the availability, integrity, and confidentiality requirements of multitasking operating systems. The most commonly used architecture provides 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 user activity

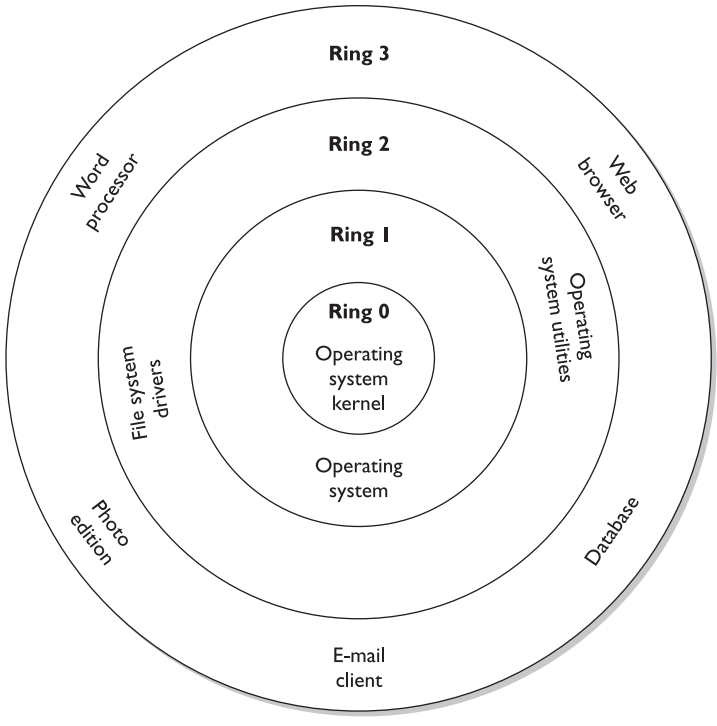


Figure 5-11 More trusted processes operate within lower-numbered rings.

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 cannot directly communicate with objects in higher rings. 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 that are 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 more sensitive tasks.

When the operating system executes instructions for processes in rings 0 and 1, it operates in supervisor 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.

If CPU execution modes and protection rings are new to you, think of protection rings as buckets. The operating system has to work within the structure and confines provided by the CPU. The CPU provides the operating system with different buckets, labeled 0 through 3. The operating system must logically place processes into the different buckets, based upon the trust level the operating system has in those processes. Since the operating system kernel is the most trusted component, it and its processes go into bucket 0. The remaining operating system processes go into bucket 1 and all user applications go into bucket 3.

NOTE Many operating systems today do not use the second protection ring very often, if at all.

So, when a process from bucket 0 needs its instructions to be executed by the CPU, the CPU checks the bucket number (ring number) and flips a bit indicating that this process can be fully trusted. This means that this process can interact with all of the functionality the CPU provides to processes. Some of the most privileged activities are I/O and memory access attempts. When another process, this time from bucket 3, needs its instructions processed by the CPU, the CPU first looks at what bucket this process came from. Since this process is from bucket 3, the CPU knows that the operating system has the least amount of trust in this process and therefore flips a bit that restricts the amount of functionality that is available to this process.

The CPU dictates how many buckets (rings) there are, and the operating system will be developed to use either two or all of them.

Operating System Architecture

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

Operating systems can be developed by using several types of architecture. The architecture is the framework that dictates how the operating system's services and functions are placed and how they interact. This section looks at the monolithic, layered, and client/server structures.

A ***monolithic operating system architecture*** is commonly referred to as "The Big Mess" because of its lack of structure. The operating system is mainly made up of various procedures that can call upon each other in a haphazard manner. Figure 5-12 illustrates a monolithic architecture. In these types of systems, modules of code can call upon each other as needed. The communication between the different modules is not as structured and controlled as in a layered architecture, and data hiding is not provided. MS-DOS is an example of a monolithic operating system.

A ***layered operating system*** architecture separates system functionality into hierarchical layers. For example, a system that followed a layered architecture was strangely enough called THE. THE had five layers of functionality. Layer 0 controlled access to the processor and provided multiprogramming functionality; layer 1 carried out memory management; layer 2 provided interprocess communication; layer 3 dealt with I/O devices; and layer 4 was where the applications resided. The processes at the different layers each had interfaces to be used by processes in layers below and above them.

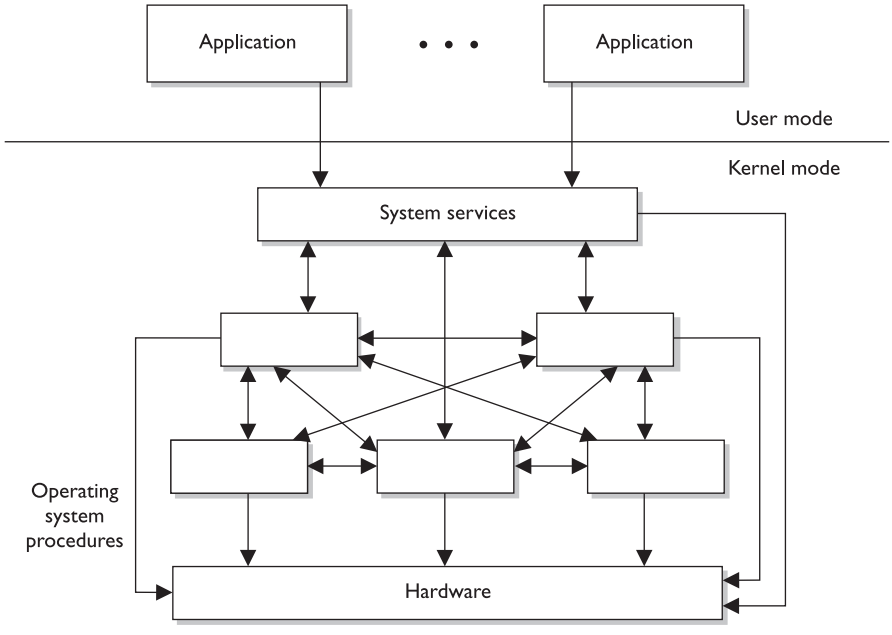


Figure 5-12 A monolithic architecture does not provide much structure.

This is different from a monolithic architecture, in which the different modules can communicate with any other module. Layered operating systems provide ***data hiding***, which means that instructions and data (packaged up as procedures) at the various layers do not have direct access to the instructions and data at any other layers. Each procedure at each layer has access only to its own data and a set of functions that it requires to carry out its own tasks. If a procedure can access more procedures than it really needs, this opens the door for more successful compromises. For example, if an attacker is able to compromise and gain control of one procedure, and this procedure has direct access to all other procedures, the attacker could compromise a more privileged procedure and carry out more devastating activities.

A monolithic operating system provides only one layer of security. In a layered system, each layer should provide its own security and access control. If one layer contains the necessary security mechanisms to make security decisions for all the other layers, then that one layer knows too much about (and has access to) too many objects at the different layers. This directly violates the data-hiding concept. Modularizing software and its code increases the assurance level of the system, because if one module is compromised, it does not mean that all other modules are now vulnerable. Examples of layered operating systems are THE, VAX/VMS, Multics, Unix (although THE and Multics are no longer in use).

NOTE Do not confuse client/server *operating system* architecture with client/server *network* architecture, which is the traditional association for "client/server." In a network, an application works in a client/server model because it provides distributed computing capabilities. The client portion of the application resides on the workstations and the server portion is usually a back-end database or server.

Windows operating systems work within a client/server architecture, which means that portions of software and functionality that were previously in the monolithic kernel are now at the higher levels of the operating system. The operating system functions are divided into several different processes that run in user mode, instead of kernel mode.

The goal of a client/server architecture is to move as much code as possible from having to work in kernel mode (privileged mode) so that the system has a leaner kernel, referred to as the *microkernel*. In this model, the requesting process is referred to as the client, and the process that fulfills the request is called the server. The server processes can be file system server, memory server, I/O server, or process server. These servers are commonly called *subsystems*. The client is either a user process or another operating system process.

For example, in Windows 2000, when the Paint process needs to display a graphic on the screen, it sends its request to the Win32 subsystem. The Paint process is the client and the Win32 subsystem is the server. The microkernel just controls client-to-server communication, instead of providing the functionality to the requesting processes itself. The microkernel is the "traffic cop," directing communication, ensuring that it is safe, and dealing with problems as they come up. The Windows operating system architecture is illustrated in Figure 5-13.

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.

NOTE Remember that a thread is a portion of a process. When the thread is generated it shares the same domain (resources) as its process.

These domains have to be identified, separated, and strictly enforced. An operating system and CPU may work in privileged mode and user mode. The reason to even use these different modes, which are dictated by the protection ring, is to define different domains. When a process's instructions are being executed in privileged mode, the process has a much larger domain to work with (or more resources to access); thus, it can carry out more activities. When an operating system process works in privileged mode,

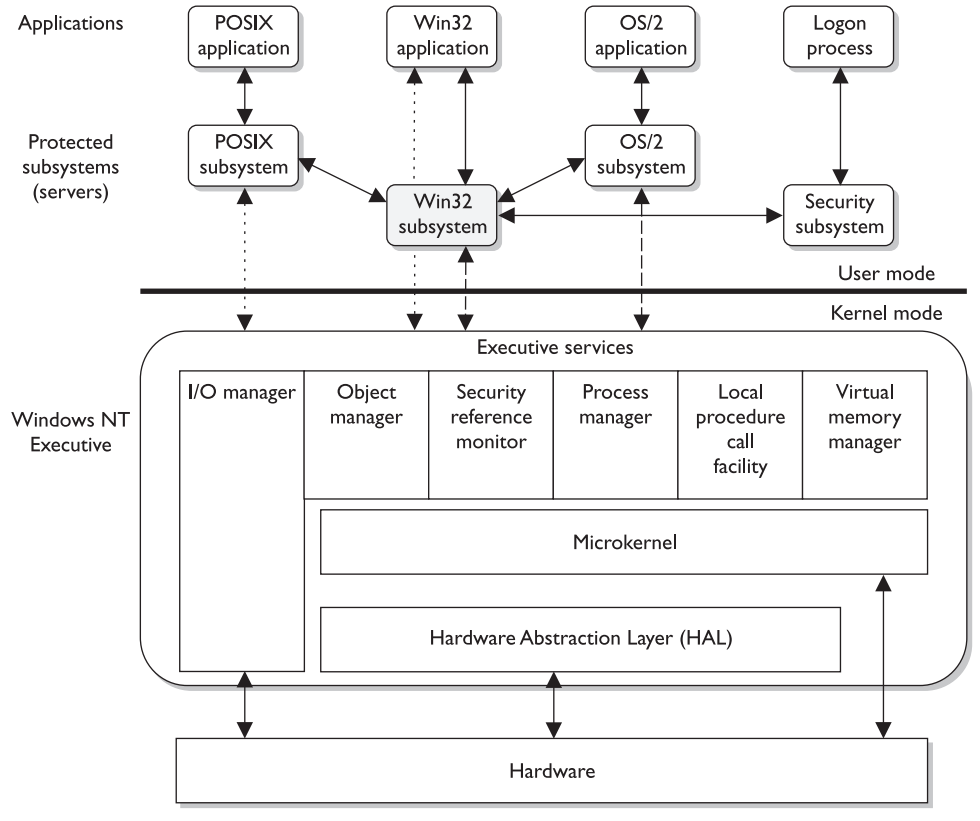


Figure 5-13 Subsystems fulfill the requests of the client processes.

it can access more memory segments, 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 that segment must be accessed in an indirect and controlled fashion.

A process 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 processes 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 execution domains.

An execution 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 domain. This concept is depicted in Figure 5-14.

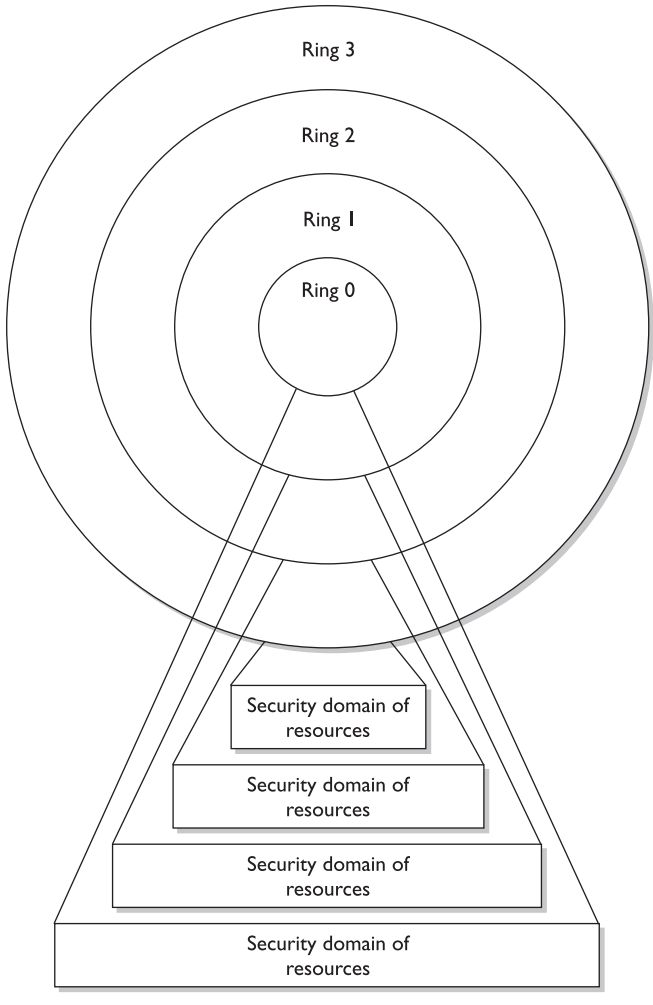


Figure 5-14 The higher the level of trust, the larger the number of available resources.

Layering and Data Hiding

Although, academically, there are three main types of architectures for operating systems, the terms ***layering*** and ***data hiding*** are commonly used when talking about protection mechanisms for operating systems—even ones that follow the client/server architecture, because it also uses layering and data hiding to protect itself.

A layered operating system architecture *mainly* addresses how functionality is laid out and is available to the users and programs. It provides its functionality in a hierarchy, whereas a client/server architecture provides functionality in more of a linear fashion. A request does not have to go through various layers in a client/server architecture. The request just goes to the necessary subsystem. But in terms of security, both architectures use layer and data hiding to protect the operating system itself.

It is almost too bad that we have so many terms—execution domains, protection rings, layering, data hiding, protection domains, CPU modes, etc.—because in reality they all are different ways to describe the same thing that takes place within every operating system today. When people are first learning these topics, many of these concepts seem discreet and totally unrelated. But in reality, these concepts have to work together in a very orchestrated manner for the whole operating system to work and provide the level of protection that it does.

As previously discussed, the operating system and CPU work within the same architecture, which provides protection rings. A process's protection domain (execution domain) is determined by the protection ring that it resides within. When a process needs the CPU to execute instructions, the CPU works in a specific mode (user or privileged) depending upon what protection ring the process is in. Layering and data hiding are provided by placing the different processes in different protection rings and controlling how communication takes place from the less trusted and the more trusted processes.

So, layering is a way to provide buffers between the more trusted and less trusted processes. The less trusted processes cannot directly communicate with the more trusted processes, but rather must submit their requests to an operating system service. This service acts as a broker or a bouncer that makes sure that nothing gains unauthorized access to the more trusted processes. This architecture protects the operating system overall, including all the applications and user activities that are going on within it.

Virtual Machines

I would like my own simulated environment so that I can have my own world. Response: No problem. Just slip on this straightjacket first.

If you have been into computers for a while, you might remember computer games that did not have the complex, life-like graphics of today's games. Pong and Asteroids were what we had to play with when we were younger. In those simpler times, the games were 16-bit and were written to work in a 16-bit MS-DOS environment. When our Windows operating systems moved from 16-bit to 32-bit, the 32-bit operating systems were written to be backward compatible, so someone could still load and play a 16-bit game in an environment that the game did not understand. The continuation of this little life pleasure was available to users because the operating systems created ***virtual machines*** for the games to run in.

A virtual machine is a simulated environment. When a 16-bit application needs to interact with the operating system, it has been developed to make system calls and interact with the computer's memory in a way that would only work within a 16-bit operating system—not a 32-bit system. So, the virtual machine simulates a 16-bit operating system, and when the application makes a request, the operating system converts the 16-bit request into a 32-bit request (this is called *thunking*) and reacts to the request appropriately. When the system sends a reply to this request, it changes the 32-bit reply into a 16-bit reply so that the application understands it.

Although not many people run 16-bit games anymore, we do use virtual machines for other purposes. The product VMWare creates individual virtual machines so that a user can run multiple operating systems on one computer at the same time. The Java

Virtual Machine (JVM), used by basically every web browser today, creates virtual machines (called sandboxes) in which Java applets run. This is a protection mechanism, because the sandbox contains the applet and does not allow it to interact with the operating system and file system directly. The activities that the applet attempts to carry out are screened by the JVM to see if they are safe requests. If the JVM determines that an activity is safe, then the JVM carries out the request on behalf of the applet.

NOTE Malware has been written that escapes the "walls of the sandbox" so that it can carry out its deeds without being under control of the JVM. These compromises, as well as Java and the JVM, will be covered in more detail in Chapter 11.

References

- **Operating Systems – OS System Architectures, by Arno Jacobsen, University of Toronto** www.eecg.toronto.edu/~jacobsen/os/week7-os-architecture.pdf
- ***The Design of PARAS Microkernel*, Chapter 2, "Operating System Models," by Rajkumar Buyya (1998)** www.gridbus.org/~raj/microkernel/chap2.pdf
- **Chapter 12, "Windows NT/2000," by M.I. Vuskovic** http://medusa.sdsu.edu/cs570/Lectures/chapter12.pdf
- **Answers.com definitions of *virtual machine*** www.answers.com/topic/virtual-machine

Additional Storage Devices

Besides the memory environment discussed previously, there are many types of physical storage devices that need to be discussed, along with the ramifications of security compromises that could affect them. Many, if not all, of the various storage devices used today enable the theft or compromise of data in an organization. As their sizes have become smaller, their capacities have grown larger. Floppy disks, while small in relative storage capacity (about 1.44MB of data), have long been known to be a source of viruses and data theft. A thief who has physical access to a computer with an insecure operating system can use a basic floppy disk to boot the system.

Many PCs and Unix workstations have a BIOS that allows the machine to be booted from devices other than the floppy disk, such as a CD-ROM or even a USB thumb drive. Possible ways to harden the environment include password-protecting the BIOS, so that a nonapproved medium cannot take over the machine, and controlling access to the physical environment of the computer equipment.

There have been many instances in which removable storage units have come up missing. Two noteworthy incidents occurred in July 2004, at which time both Los Alamos National Laboratory and Sandia National Laboratories reported missing storage media containing classified information. This raised enough of a concern at Los Alamos that the military research facility was totally shut down, with no employees al-

lowed to enter, while a thorough search and investigation was performed. Sandia National Laboratories reported it was missing a computer floppy disk marked classified, which it later located.

Rewritable CD/DVDs, mini-disks, optical disks—virtually any portable storage medium—can be used to compromise security. Current technology headaches for the security professional include USB thumb drives and USB-attachable MP3 players that are capable of storing multiple gigabytes of data. The first step in prevention is to update existing security policies (or implement new ones) to include the new technologies. Even cellular phones can be connected to computer ports for data, sound, image, and video transmission that could be out of bounds of an outdated security policy. Technologies such as Bluetooth, FireWire, and Blackberry all have to be taken into account when addressing security concerns and vulnerabilities.

Input/Output Device Management

Some things come in, some things go out. Response: We took a vote and would like you to go out.

We have covered a lot of operating system responsibilities up to now, and we are not stopping yet. An operating system also has to control all input/output devices. It sends commands to them, accepts their interrupts when they need to communicate with the CPU, and provides an interface between the devices and the applications.

I/O devices are usually considered block or character devices. A block device works with data in fixed-size blocks, each block with its own unique address. A disk drive is an example of a block device. A character device, such as a printer, network interface card, or mouse, works with streams of characters, without using any fixed sizes. This type of data is not addressable.

When a user chooses to print a document, open a stored file on a word processor, or save files to a jump drive, these requests go 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 with a device controller, which may be a circuit card that fits into an expansion slot. The controller is an electrical component with its own software that provides 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. If the command is to extract data from the hard drive, the controller takes the bits and puts them into the necessary block size and carries out a checksum activity to verify the integrity of the data. If the integrity is successfully verified, then the data is put into memory for the CPU to interact with.

Operating systems need to access and release devices and computer resources properly. 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 in Windows NT 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 method 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.

Why Does My Video Card Need to Have Its Own RAM?

The RAM on a video card is really just a type of buffer. The application or operating system writes the pixel values into this RAM space instead of writing to the system's RAM. The pixel values are then displayed to the user on the monitor screen. Graphic-intensive games work better with video cards with a lot of RAM, because storing this display information on the system's RAM takes too long for the read and write procedures. This results in delayed reactions between the user's interaction commands and what is displayed on the screen. We never seemed to have these problems when we all played Pong.

Interrupts

When an I/O device has completed whatever task that was asked of it, it needs to inform the CPU that the necessary data is now in memory for processing. The device's controller sends a signal down a bus, which is detected by the interrupt controller. (This is what it means to use an interrupt. The device signals the interrupt controller and is basically saying, "I am done and need attention now.") If the CPU is busy *and* the device's interrupt is not a higher priority than whatever job is being processed, then the device has to wait. The interrupt controller sends a message to the CPU, indicating what device needs attention. The operating system has a table (called the interrupt vector) of all the I/O devices connected to it. The CPU compares the received number with the values within the interrupt vector so that it knows which I/O device needs its services. The table has the memory addresses of the different I/O devices. So when the CPU understands that the hard drive needs attention, it looks in the table to find the correct memory address. This is the new program counter value, which is the initial address of where the CPU should start reading from.

One of the main goals of the operating system software that controls I/O activity is to be device independent. This means that a developer can write an application to read (open a file) or write (save a file) to any device (floppy disk, jump drive, hard drive, CD-ROM drive). This level of abstraction frees application developers from having to write different procedures to interact with the various I/O devices. If a developer had to write an individual procedure of how to write to a CD-ROM drive, *and* how to write to a floppy disk, *and* how to write to a jump drive, *and* how to write to a hard disk, and so on, each time a new type of I/O device was developed, all of the applications would have to be patched or upgraded.

There are different ways that operating systems can carry out software I/O procedures. We will look at the following methods:

- Programmed I/O
- Interrupt-driven I/O
- I/O using DMA

- Premapped I/O
- Fully mapped I/O

Programmable I/O If an operating system is using *programmable I/O*, this means that the CPU sends data to an I/O device and polls the device to see if it is ready to accept more data. If the device is not ready to accept more data, the CPU wastes time by waiting for the device to become ready. For example, the CPU would send a byte of data (a character) to the printer and then ask the printer if it is ready for another byte. The CPU sends the text to be printed one byte at a time. This is a very slow way of working and wastes precious CPU time. So the smart people figured out a better way: interrupt-driven I/O.

Interrupt-Driven I/O If an operating system is using *interrupt-driven I/O*, this means that the CPU sends a character over to the printer and then goes and works on another process's request. When the printer is done printing the first character, it sends an interrupt to the CPU. The CPU stops what it is doing, sends another character to the printer, and moves to another job. This process (send character—go do something else—interrupt—send another character) continues until the whole text is printed. Although the CPU is not waiting for each byte to be printed, this method does waste a lot of time dealing with all the interrupts. So we excused those smart people and brought in some smarter people, who came up with I/O using DMA.

I/O Using DMA *Direct memory access (DMA)* is a way of transferring data between I/O devices and the system's memory without using the CPU. This speeds up data transfer rates significantly. When used in I/O activities, the DMA controller feeds the characters to the printer without bothering the CPU. This method is sometimes referred to as *unmapped I/O*.

Premapped I/O *Premapped I/O* and fully mapped I/O (described next) does not pertain to performance, as do the earlier methods, but provides two approaches that can directly affect security. In a premapped I/O system, the CPU sends the physical memory address of the requesting process to the I/O device, and the I/O device is trusted enough to interact with the contents of memory directly. So the CPU does not control the interactions between the I/O device and memory. The operating system trusts the device to behave properly. Scary.

Fully Mapped I/O Under *fully mapped I/O*, the operating system does not fully trust the I/O device. The physical address is not given to the I/O device. Instead, the device works purely with logical addresses and works on behalf (under the security context) of the requesting process. So the operating system does not trust the device to interact with memory directly. The operating system does not trust the process or device and acts as the broker to control how they communicate with each other.

Beam Me Up, Scotty!

A cofounder of Intel, Gordon Moore, came up with Moore's Law, which states that the number of electronic components (transistors, capacitors, diodes) will double every 18 months. No one really paid attention to this until it came to be true every 18 months or so. Now he is right up there with Nostradamus.

So chip makers have continually put an amazing amount of stuff on one tiny piece of silicon, but we will soon hit the wall and have to look for other types of technologies. Today, people are looking at quantum physics to make up tomorrow's processor. In quantum physics, particles are not restricted to holding just two states (1 or 0), but instead can hold these states simultaneously and other states in between. Freaky.

And other individuals are looking to our own personal DNA, which holds the instructions for our bodies. There have already been experiments in which DNA was used for computation processes. It can carry out complex computations in parallel instead of serially like today's processors. But it sounds kind of gross.

You won't find a quantum or DNA computer at Best Buy any time soon, but one day maybe.

References

- **Chapter 12, "I/O Management and Disk Scheduling," by Joseph Kee-Yin** www.comp.hkbu.edu.hk/~jng/comp2320/2320-C11.ppt
- **I/O Management, by Sotirios Terzis (1/12/2000)** www.cs.tcd.ie/Sotirios.Terzis/3BA3/L9/
- **Introduction to Operating Systems, by B. Ramamurthy (1/25/2002)** www.cse.buffalo.edu/faculty/bina/cse421/spring02/jan25.pdf

System Architecture

Designing a system from the ground up is a complicated task that has many intricate and abstract goals that have to be achieved through mathematics, logic, design, programming code, and implementation. Fundamental design decisions 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 should 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 activi-

ties? 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 5-15.

The same type of questions have to be answered when building an operating system. 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, then a broad level of protection is provided, because it provides a baseline of security for all the components that work on top of the hardware layer. If the protection mechanisms are closer to the user (in the higher levels of the operating system architecture), the security is more detail oriented and focused than mechanisms that work at the lower levels of the architecture.

No matter where the security mechanism is placed within the operating system architecture, the more complex the security mechanism becomes, the less assurance it usually provides. The reason is that greater complexity of the mechanism demands more technical understanding from the individuals who install, test, maintain, and use it. The more complex the security mechanism, the harder it is to fully test it under all possible conditions. On the other hand, simplistic mechanisms may not be able to provide the desired richness of functionality and options, although they are easier to install, maintain, use, and test. So the trade-offs 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 how complex each mechanism is, they need to build and integrate the mechanisms in such a way that they have a proper relationship with other parts of the system.

First, the design team needs to decide what system mechanisms to trust and place in protection ring 0. Then, the team needs to specify how these modules of code 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 must perform in a predictable and secure manner and not adversely affect other trusted or untrusted mechanisms. In return, these trusted components have access to more privileged services, have

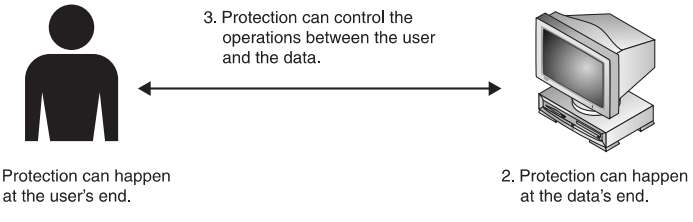


Figure 5-15 Security can take place in three main areas.

> **Security Architecture**
> The security architecture is one component of a product's overall architecture and is developed to provide guidance during the design of the product. It should outline the level of assurance that is required and potential impacts that this level of security could have during the development stages and on the product overall. As the software development project moves from architecture, to design, to specifications, and then code development, the security architecture and requirements become more granular with each step.

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 not all components 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 within the TCB need to be identified and their accepted capabilities need to be defined. For example, a system that does not require a high level of trust 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 that requires a higher level of trust may permit only two subjects to access all files on a computer system, and permit only one of those subjects to actually modify all the files. This subset is much smaller and the rules being enforced are more stringent and detailed.

If developers want to develop a system that achieves an Orange Book assurance rating of D (very low), then what makes up the TCB 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 ensure that all components of the TCB carry out their tasks properly. These TCB components need to enforce strict rules that dictate how subjects and objects interact. The developers also need to ensure that these components are identified, audited, and react in a predictable manner, 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 is an evaluation criteria and is addressed in the section "The Orange Book" later in the chapter.)

Trusted Computing Base

The term "trusted computing base," which originated from the Orange Book, does not address the level of security that a system provides, but rather the level of trust that a system provides, albeit from a security sense. This is 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 become successful. However, if a system meets a certain criteria, it is looked upon as providing a certain level of trust, meaning that it will react predictably in different types of situations.

The TCB does not address *only* operating system components, 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 or 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 processes overwrite other processes' data. Then there are components that do not enforce the security policy but must behave properly and not violate the trust of a system. Examples of the ways in which a component could violate the system's security policy include an application that attempts to make a direct call to a piece of hardware instead of using the proper system 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.

A TCB is usually a very abstract concept for people who are not system designers, and many books or papers do not make it any easier to fully understand what this term means. If an operating system is using a TCB, this means that the system has a hardened kernel that compartmentalizes system processes. The kernel is made up of hardware, software, and firmware, so in a sense the kernel is the TCB. But the TCB can include other components, such as trusted commands, programs, and configuration files that can directly interact with the kernel. For example, when installing a Unix system, the administrator can choose to install the TCB during the setup procedure. If the TCB is enabled, then the system has a trusted path, a trusted shell, and system integrity–checking capabilities. A ***trusted path*** is a communication channel between the user, or program, and the kernel. The TCB provides protection resources to ensure that this channel cannot be compromised in any way. A ***trusted shell*** means that someone who is working in that shell cannot "bust out of it" and other processes cannot "bust into" it.

There are several Unix commands that are part of the TCB, such as `setuid` (set process ID) and `setguid` (set process group ID). Only a privileged user, as in root, should be able to change process ID information.

Earlier operating systems (MS-DOS, Windows 3.11, and Novell Netware release 3, for example) did not have TCBs. Windows 95 has a TCB, but it can be used only while working in 32-bit mode. Windows NT was the first version of Windows that truly incorporated the idea of a TCB. As of this writing, Microsoft is using the words "Trustworthy Computing" a lot, but this is not just a Microsoft concept. Several vendors came

together to develop a better TCB by agreeing to improve methods on how to protect software from being compromised. Microsoft is just the first vendor to implement it, in its Windows 2003 product. Microsoft is basically building upon its current TCB and calling it the Next-Generation Secure Computing Base (NGSCB). (It must be better, it has a longer name!) And Microsoft renamed its security kernel "nexus." How NGSCB actually works is beyond the scope of this book and the exam, but basically the operating system is doing what it is supposed to be doing, only better—better isolation between trusted and untrusted processes, better memory management, more secure I/O operations, and better software authentication mechanisms. So we are all one step closer to more secure systems and world peace.

Every operating system has specific components that would cause the system grave danger if they were compromised. The TCB provides extra layers of protection around these mechanisms to help ensure that they are *not* compromised, so that the system will always run in a safe and predictable manner.

How Does the TCB Do Its Magic?

The processes within the TCB are the components that protect the system overall. So the developers of the operating system must make sure that these processes have their own ***execution domain***. This means that they reside in ring 0, their instructions are executed in privileged state, and no less trusted processes can directly interact with them. The developers need to ensure that the operating system maintains an isolated execution domain, so that their processes cannot be compromised or tampered with. The resources that the TCB processes use must also be isolated, so that tight access control can be provided and all access requests and operations can be properly audited. So basically, the operating system tells all the other processes that they cannot play with the TCB processes and that they cannot play with the TCB's resources.

The four basic functions of the TCB are process activation, execution domain switching, memory protection, and I/O operations. We have really covered all of these things already throughout previous sections without using this exact terminology, so here's a recap. ***Process activation*** deals with the activities that have to take place when a process is going to have its instructions and data processed by the CPU. As described earlier, the CPU fills its registers with information about the requesting process (program counter, base and limit addresses, user or privileged mode, etc.). A process is "activated" when its interrupt is called upon, enabling it to interact with the CPU. A process is "deactivated" when its instructions are completely executed by the CPU or when another process with a higher priority calls upon the CPU. When a process is deactivated, the CPU's registers have to be filled with new information about the new requesting process. The data that is getting switched in and out of the registers may be sensitive in nature, so the TCB components must make sure all of this is taking place securely.

For an analogy, suppose that five people all want to ask your advice at the same time. There is only one of you, so each person has a certain amount of time that they can spend with you. Person 1 gives you her papers, so that you can read through them and figure out her problem so that you can give her a proper answer. As you are reading Person 1's papers, Person 2 comes barging into the room claiming that he has an emergency. So you put Person 1's papers down on your desk and start reading Person 2's

papers to figure out his problem. Eventually you have five people around you, and you can only spend five seconds with each of them at a time. As you talk to one individual, you must grab the right file from your desk. These files have sensitive information, so you have to make sure to keep all the files straight and make sure that the people cannot read one another's files. This is what the TCB and the CPU are doing as they are dealing with several different process requests at one time. Instead of reading stacks of paper, the CPU reads instructions and data on a memory segment stack.

Execution domain switching takes place when a process needs to call upon a process in a higher protection ring. As explained earlier, less trusted processes run in user mode and cannot carry out activities such as communicating with hardware or directly sending requests to the kernel. Therefore, a process running in user mode (ring 3) must make a request to an operating system service, which works in ring 1. The less trusted process will have its information loaded into the CPU's registers and then when the CPU sees that an operating system service has been called it switches domains and security context. This means that the information of the operating system service process is loaded into the CPU's registers and the CPU carries out those instructions in privileged mode. So, execution domain switching refers to when the CPU has to go from executing instructions in user mode to privileged mode and back. All of this has to happen properly or a less trusted process might be executed in privileged mode and have direct access to system resources.

Memory protection and I/O operations have been discussed in previous sections, so just realize that these operations are the responsibility of the components within the TCB.

The TCB is responsible for carrying out the TCB memory protection and I/O operations securely. It does this by compartmentalizing into discreet units, which are the processes that make up the kernel. This ensures that if a kernel process is compromised, it does not mean that all the processes are now under the control of the attacker.

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. During the evaluation process, the tests must show how the TCB is protected from accidental or intentional tampering and compromising activity. For systems to achieve a higher trust level rating, 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 product to another. It also gives vendors guidelines on what expectations are put upon their systems and provides a common assurance rating metric so that when one group talks about a C2 rating, everyone else understands what that term means.

The Orange Book is one of these evaluation criteria. It 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 that 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 level of protection, and operate in an expected manner under many different situations. When the operating system makes calls to hardware, it anticipates that data will be returned in a specific data format and 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, receive the required data in return, and 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? Response: Anyone inside the security perimeter.

As stated previously, not every process and resource falls within the TCB, so some of these components fall outside of an imaginary boundary referred to as the ***security perimeter***. A security perimeter is a boundary that divides the trusted from the untrusted. For the system to stay in a secure and trusted state, precise communication standards must be developed to ensure that when a component within the TCB needs to communicate with a component outside the TCB, the communication cannot expose the system to 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 allow less trusted components access to critical system resources. The processes within the TCB must also be careful about the commands and information they accept 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. Communication between trusted components and untrusted components needs to be controlled to ensure that the system stays stable and safe.

NOTE The TCB and security perimeter are not physical entities, but conceptual constructs used by the operating system developers to delineate between trusted and untrusted 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 programs), the processes that are within the TCB, and how the security mechanisms and processes will interact with each other. They have defined the security perimeter that separates the trusted and untrusted components. 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 that mediates all access subjects have to objects, both 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 require subjects (programs, users, or processes) to be fully authorized prior to accessing an object (file, program, or resource). A subject must not be allowed to use a requested resource until the subject has proven that it has been granted access privileges to use the requested object. The reference monitor is an access control concept, not an actual physical component, which is why it is normally referred to as the "reference monitor concept" or an "abstract machine."

The ***security kernel*** is made up of hardware, software, and firmware components that fall within the TCB and implements and enforces the reference monitor concept. The security kernel mediates 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 three main requirements of the security kernel:

- It must provide isolation for the processes carrying out the reference monitor concept, and the processes must be tamperproof.
- It must be invoked for every access attempt and must be impossible to circumvent. Thus, the security kernel must be implemented in a complete and foolproof way.
- 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 in which an abstract machine mediates all access 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.

The following is a quick analogy to show you the relationship between the processes that make up the kernel, the kernel itself, and the reference monitor concept. Individuals make up a society. The individuals represent the processes, and the society

represents the kernel. For a society to have a certain standard of living, its members need to interact 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 that 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.

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** http://citeseer.ist.psu.edu/299300.html
- **Implementing Complete Mediation** www.cs.cornell.edu/html/cs513-sp99/NL05.html

Security Policy

As previously stated, 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 that dictates how sensitive information and resources are 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 supposed 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.

Chapter 3 examined security policies in depth, but those policies were directed toward the company itself. The security policies being addressed here are for operating systems, devices, 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 oversees the communication between subjects and objects. The policy must indicate which subjects can access individual objects, and which actions are acceptable and unacceptable. The security policy provides the framework for the system's security architecture.

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 obtained through formal evaluations require a defined subset of subjects and objects, explicit domains, and the isolation of processes so that their access can be controlled and the activities performed on them can be audited.

Let's regroup. We know that a system's trust is defined by how it enforces its own security policy. When a system is tested against a specific 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 allow access to sensitive resources. The reference mon-

itor is a concept that says all subjects must have proper authorization to access objects, and this concept is implemented by the security kernel. The security kernel comprises 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 the security kernel to work correctly, the individual processes need to be isolated from each other and domains need to be defined to dictate which objects are available to which 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 previously stated, the concepts covered in the previous sections are abstract ideas that will be manifested in 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 obtained by our physical hard work and discipline.

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 that it can interact directly with a system resource, 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 that need complete system privileges are located in the kernel; other, less privileged processes call upon them to process sensitive or delicate operations.

As an example of least privilege access control, the system backup program may have read access on the files, but it does not need to be able to modify the files. Similarly, the restore program would be allowed to write files to the disk, but not to read them.

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 policymakers into a set of rules that a computer system must follow.

The reason this chapter has repeatedly mentioned the security policy and its importance is that 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 the point at which an administrator is able to uncheck a box

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.

on the 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 that are 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 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 there, specifications are developed per operating system type (Unix, Windows, Macintosh, and so on), 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 the 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 ACLs and give administrators some degree of control. This mechanism presents the network administrator with a GUI that enables the administrator to choose (via check boxes, for example) 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, such as the Bell-LaPadula model, enforce rules to provide confidentiality protection. Other models, such as the Biba model, enforce rules to provide integrity protection. 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 without regard to how they will be accomplished. A model is a framework that gives the policy form and solves security access problems for particular situations. Several security models have been developed to enforce security policies. The following sections provide overviews of each model.

Formal Models

Using models in software development has not become as popular as once imagined, primarily because vendors are under pressure to get products to market as soon as possible. Using formal models takes more time during the architectural phase of development, extra time that many vendors feel they cannot afford. Formal models are definitely used in the development of systems that cannot allow errors or security breaches, such as air traffic control systems, spacecraft software, railway signaling systems, military classified systems, and medical control systems. This does not mean that these models, or portions of them, are not used in industry products, but rather that industry vendors do not always follow these models in the purely formal and mathematical way that the models were designed for.

State Machine Models

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

In ***state machine models***, to verify the security of a system, the state is used, which means that 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 access objects only 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 at one moment of time. There are many activities that can alter this state, which are referred to as ***state transitions***. The developers of an operating system that will implement the state machine model need to look at all the different state transitions that are possible and assess whether a system that starts up in a secure state can be put into an insecure state by any of these events. 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.

The state machine model is used to describe the behavior of a system to different inputs. It provides mathematical constructs that represent sets (subjects and objects) and sequences. When an object accepts an input, this modifies a state variable. A simplistic example of a state variable is (Name, Value), as shown in Figure 5-16. This variable is part of the operating system's instruction set. When this variable is called upon to be used, it can be populated with (Color, Red) from the input of a user or program. Let's say that the user enters a different value, so now the variable is (Color, Blue). This is a simplistic example of a state transition. Some state transitions are this simplistic, but the complexity comes in when the system has to decide if this transition should be allowed. To allow this transition, the object's security attributes and the access rights of the subject must be reviewed and allowed by the operating system.

Developers who implement the state machine model must identify all the initial states (default variable values) and outline how these values can be changed (inputs that will be accepted) so that the various number of final states (resulting values) still ensure that the system is safe. The outline of how these values can be changed are often implemented through condition statements: "if *condition* then *update*."

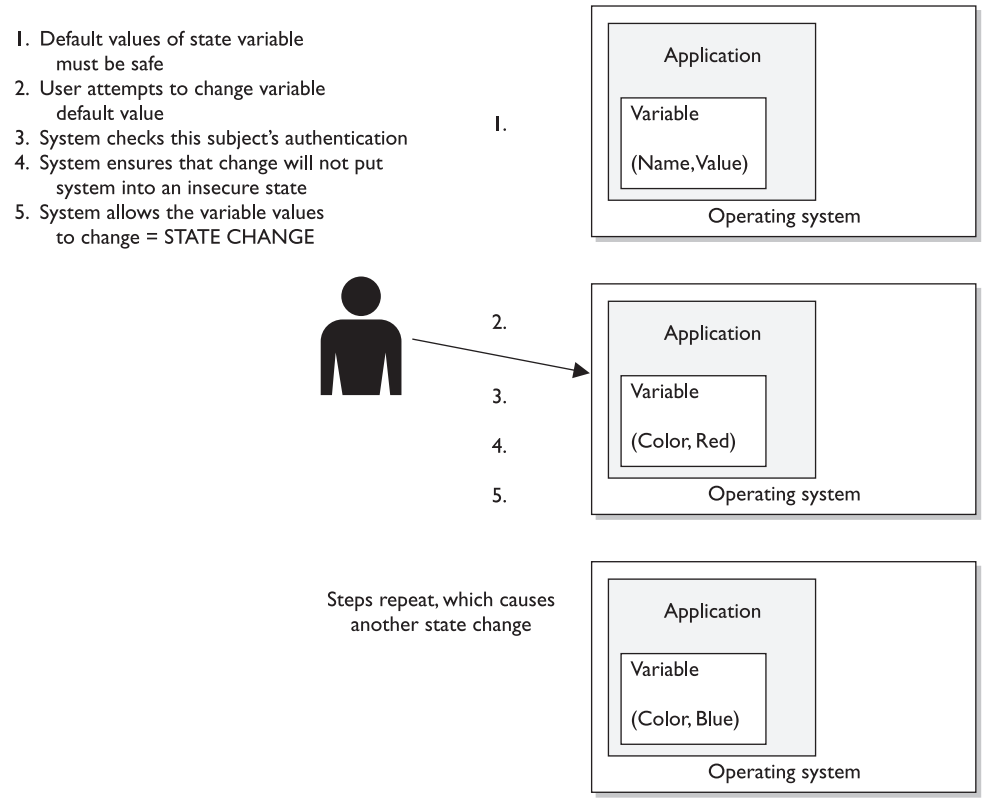


Figure 5-16 A simplistic example of a state change.

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, the system must be able to "save itself" and not make itself vulnerable. When an operating system displays an error message to the user or reboots or freezes, it is executing a safety measure. 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 stay in this insecure state, it reacts in one of these fashions. So if an application or system freezes on you, know that it is simply the system trying to protect itself and your data.

There are several points to consider when developing a product that uses a state machine model. Initially, the developer must define what and where the state variables are. In a computer environment, all data variables could independently be considered state variables, and an inappropriate change to one could conceivably change or corrupt the system or another process's activities. Next, the developer must define a secure state for each state variable. The next step is to define and identify the

allowable state transition functions. These functions will describe the allowable changes that can be made to the state variables.

After the state transition functions are defined, they must be tested to verify that the overall machine state will not be compromised and that these transition functions will keep the integrity of the system (computer, data, program, or process) intact at all times.

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 outlined 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 system, and the system processes data with different classifications. The level at which information is classified determines the handling procedures that should be used. The Bell-LaPadula model is a state machine model that enforces 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 and then specific rules are applied to control how subject-to-object interactions can take place.

This model uses subjects, objects, access operations (read, write, and read/write), and security levels. Subjects and objects can reside at different security levels and will have relationships and rules dictating the acceptable activities between them. This model, when properly implemented and enforced, has been mathematically proven to provide a very secure and effective operating system. 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 (subject) could read a data element (object) from a specific database and write data into that database. The Bell-LaPadula model focuses on ensuring that subjects are properly authenticated—by having the necessary security clearance, need to know, and formal access approval—before accessing an object.

Three main rules are used and enforced in the Bell-LaPadula model: the simple security rule, the *-property (star 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. For example, if Bob is given the security clearance of secret, this rule states that he cannot read data classified as top secret. If the organization wanted Bob to be able to read top-secret data, it would have given him that clearance in the first place.

The ***-property rule*** (star 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 5-17. 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. So, for a subject to be able to read and write to an object, the clearance and classification must be equal.

These three rules indicate what states the system can go into. Remember that a state is the values of the variables in the software at a snapshot in time. If a subject has performed a read operation on an object at a lower security level, the subject now has a variable that is populated with the data that was read, or copied into its variable. If a subject has written to an object at a higher security level, the subject has modified a variable within that object's domain.

NOTE In access control terms, the word *dominate* means to be higher than or equal to. So if you see a statement such as "A subject can only perform a read operation if the access class of the subject dominates the access class of an object," this just means that the subject must have a clearance that is higher than or equal to the object.

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 2000 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 was at the beginning of the day. This is the definition of the ***Basic Secu-***

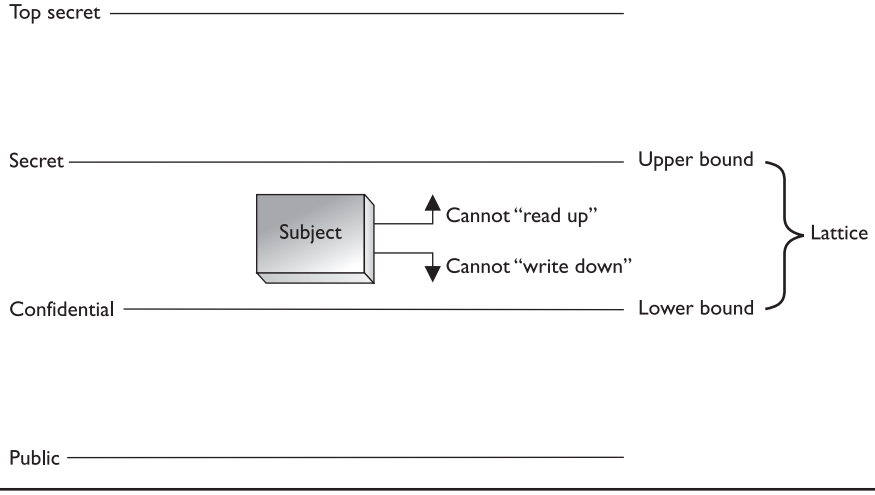


Figure 5-17 In the Bell-LaPadula model, each subject has a lattice of rights.

rity Theorem used in computer science, which states that if a system initializes in a secure state and all allowed state transitions are secure, then every subsequent state will be secure no matter what inputs occur.

NOTE The *tranquility principle*, which is also used in this model, means that subjects and objects cannot change their security levels once they have been instantiated (created).

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 and what operations can be carried out.

NOTE Ensuring that information does not flow from a higher security level to a lower level is referred to as *controlling unauthorized downgrading of information*, which would take place through a "write down" operation. An actual compromise occurs if and when a user at a lower security level reads this data.

So What Does This Mean and Why?

Chapter 4 discussed mandatory access control (MAC) systems versus discretionary access control (DAC) systems. All MAC systems are based on the Bell-LaPadula model, because it allows for multilevel security to be integrated into the code. Subjects and objects are assigned labels. The subject's label contains its clearance label (top secret, secret, or confidential) and the object's label contains its classification label (top secret, secret, or confidential). 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. In our scenario, 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, which makes sure that subjects cannot accidentally or intentionally share confidential information by writing to an object at a lower

Rules to Know

The main rules of the Bell-LaPadula model that you need to understand are

- **Simple security rule** A subject cannot read data within an object that resides at a higher security level ("No read up" rule).
- ***- property rule** A subject cannot write to an object at a lower security level ("No write down" rule).
- **Strong star property rule** For a subject to be able to read and write to an object, the subject's clearance and the object's classification must be equal.

security level. As an example, suppose that a busy and clumsy general (who has top secret clearance) in the army opens up a briefing letter (which has a secret classification) that will go to all clerks at all bases 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 military staff clerk tried to read a memo that was available only to generals and above, the Bell-LaPadula model would stop this activity. 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.

NOTE It is important that MAC operating systems and MAC databases follow these rules. In Chapter 11, we will look at how databases can follow these rules by the use of polyinstantiation.

Biba Model

The ***Biba model*** was developed after the Bell-LaPadula model. It is a state machine model and is very similar to the Bell-LaPadula model. Biba addresses the *integrity* of data within applications. The Bell-LaPadula model uses a lattice of security levels (top secret, secret, sensitive, and so on). These security levels were developed mainly to ensure that sensitive data was only available to authorized individuals. The Biba model is not concerned with security levels and confidentiality, so it does not base access decisions upon this type of lattice. The Biba model uses a lattice of integrity 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:

- ***-integrity axiom** A subject cannot write data to an object at a higher integrity level (referred to as "no write up").
- **Simple integrity axiom** A subject cannot read data from a lower integrity level (referred to as "no read down").

The name "simple integrity axiom" might sound a little goofy, but this rule protects the subject and data at a higher integrity level from being corrupted by data at a lower integrity level. This is all about trusting the source of the information. Another way to look at it is that trusted data is "clean" data and untrusted data (from a lower integrity level) is "dirty" data. Dirty data should not be mixed with clean data, because that could ruin the integrity of the clean data.

The simple integrity axiom applies not only to subjects creating the data, but also to processes. A process of lower integrity should not be writing to trusted data of a higher integrity level. The areas of the different integrity levels are compartmentalized within the application that is based on the Biba model.

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, IDSs, 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 you are first learning about the Bell-LaPadula and Biba models, they may seem very similar, and the reasons for their differences may be somewhat confusing. The Bell-LaPadula model was written for the U.S. government, and the government is very paranoid about leakage of its secret information. In its 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 its 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 the firm would use software that employs 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 its application. Which model to use is something that was decided upon and implemented when the application was being designed. The assurance ratings are what consumers use to determine if a system is right for them. So, even if the accountants are using an application that uses the Biba model, they would not necessarily know (and we're not going to tell them).

If you don't have enough rules to understand so far, here's another one. The ***invocation property*** in the Biba model states that a subject cannot invoke (call upon) a subject at a higher integrity level. Well, how is this different from the other two Biba rules? The *-integrity axiom (no write up) dictates how subjects can *modify* objects. The simple integrity axiom (no read down) dictates how subjects can *read* objects. The invocation property dictates how one subject can communicate with and initialize other subjects at run time. An example of a subject invoking another subject is when a process sends a request to a procedure to carry out some type of task. So the rules may seem similar, but they really lay down the parameters for different types of activities that can take place within an application.

Bell-LaPadula vs. 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. 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 for how 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 the reading and writing directions per model.

References

- **Module 5, "Security Policies and Security Models"** www.radium.ncsc.mil/tpep/library/ramp-modules/mod_05.pdf
- **Security Models** www.iwar.org.uk/comsec/resources/security-lecture/showb1a7.html
- **Course study materials for Introduction to Security, University of Cambridge, Dr. Markus Kuhn, principle lecturer (academic year 2003–2004)** www.cl.cam.ac.uk/Teaching/2003/IntroSecurity/slides.pdf
- **Chapter 3.3, "Models of OS Protection," by Fred Cohen** www.all.net/books/ip/Chap3-3.html

Clark-Wilson Model

The ***Clark-Wilson model*** was developed after Biba and takes some different approaches to protecting the integrity of information. This model uses the following elements:

- **Users** Active agents
- **Transformation procedures (TPs)** Programmed abstract operations, such as read, write, and modify
- **Constrained data items (CDIs)** Can be manipulated only by TPs
- **Unconstrained data items (UDIs)** Can be manipulated by users via primitive read and write operations
- **Integrity verification procedures (IVPs)** Run periodically to check the consistency of CDIs with external reality

Although this list may look overwhelming, it is really quite straightforward. When an application uses the Clark-Wilson model, it separates data into one subset that needs to be highly protected, which is referred to as a constrained data item (CDI) and another subset that does not require a high level of protection, which is called an unconstrained data item (UDI). Users cannot modify critical data (CDI) directly. Instead, the subject (user) must be authenticated to a piece of software, and the software procedures (TPs) will carry out the operations on behalf of the user. For example, when Kathy needs to update information held within her company's database, she will not be allowed to do so without a piece of software controlling these activities. First, Kathy must authenticate to a program, which is acting as a front end for the database, and then the program will control what Kathy can and cannot do to the information in the database. This is referred to as *access triple*: subject (user), program (TP), and object (CDI). A user cannot modify CDI without using a TP.

So, Kathy is going to input data, which is supposed to overwrite some original data in the database. The software (TP) has to make sure that this type of activity is secure and will carry out the write procedures for Kathy. Kathy (and any type of subject) is not trusted enough to manipulate objects directly.

The CDI must have its integrity protected by the TPs. The UDI does not require such a high level of protection. For example, if Kathy did her banking online, the data on her

bank's servers and databases would be split into UDI and CDI categories. The CDI category would contain her banking account information, which needs to be highly protected. The UDI data could be her customer profile, which she can update as needed. TPs would not be required when Kathy needed to update her UDI information.

In some cases, a system may need to change UDI data into CDI data. For example, when Kathy updates her customer profile via the web site to show her new correct address, this information will need to be moved into the banking software that is responsible for mailing out bank account information. The bank would not want Kathy to interact directly with that banking software, so a piece of software (TP) is responsible for copying that data and updating this customer's mailing address. At this stage, the TP is changing the state of the UDI data to CDI. These concepts are shown in Figure 5-18.

Remember that this is an integrity model, so it must have something that ensures that specific integrity rules are being carried out. This is the job of the IVP. The IVP ensures that all critical data (CDI) follows the application's defined integrity rules. What usually turns people's minds into spaghetti when they are first learning about models is that models are theoretical and abstract. Thus, when they ask the common question, "What are these defined integrity rules that the CDI must comply with?" they are told, "Whatever the vendor chooses them to be."

A model is made up of constructs, mathematical formulas, and other PhD kinds of stuff. The model provides the framework that can be used to build a certain characteristic into software (confidentiality, integrity, and so on). So the model does not stipulate what integrity rules the IVP must enforce; it just provides the framework, and the

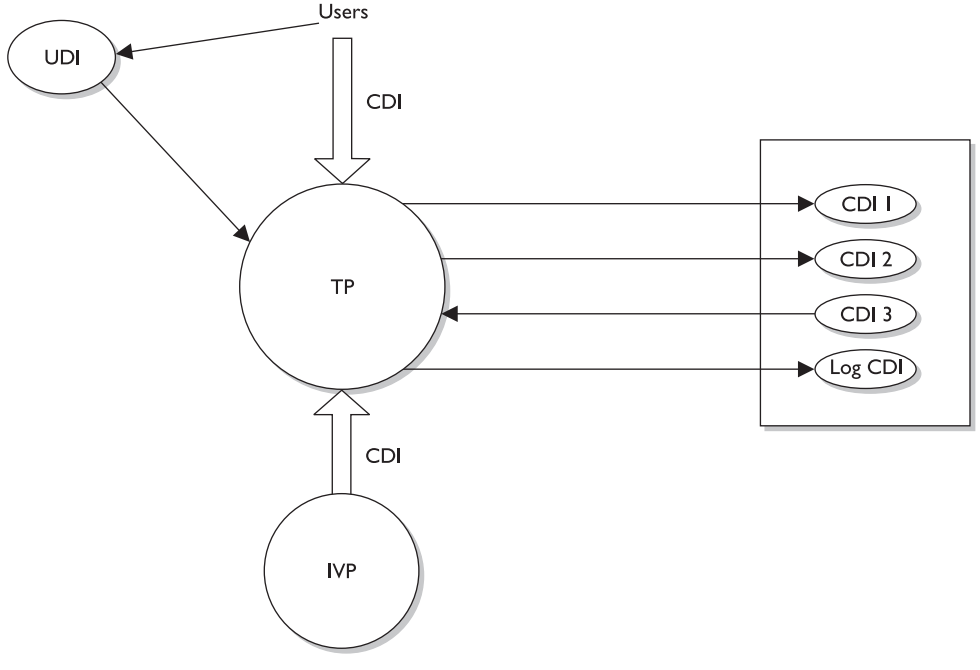
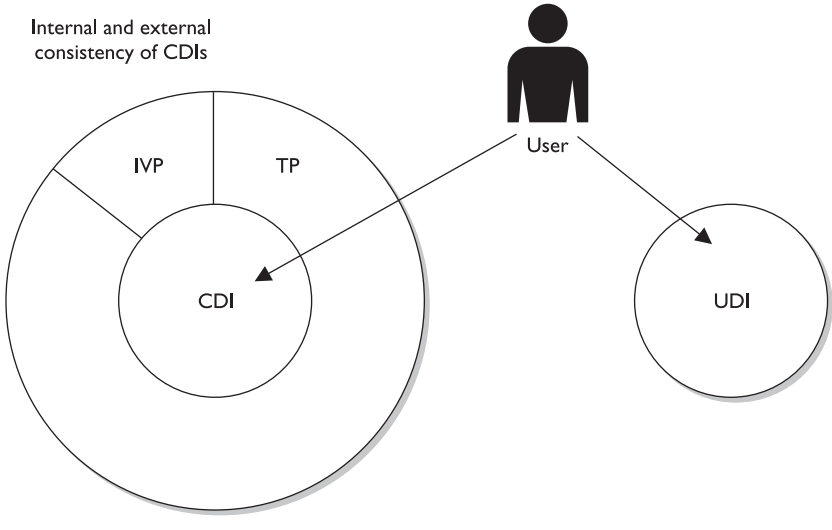


Figure 5-18 Subjects cannot modify CDI without using TP.

vendor chooses the integrity rules. The vendor implements integrity rules that its customer base needs the most. So if a vendor is developing an application for a financial institution, the UDI could be customer profiles that they are allowed to update and the CDI could be the bank account information, usually held on a mainframe. The UDI data does not need to be as highly protected and can be located on the same system or another system. A user can have access to UDI data without the use of a TP, but when the user needs to access CDI, they must use TP. So the vendor who develops the product will determine what type of data is considered UDI and what type of data is CDI and develop the TPs to control and orchestrate how the software enforces the integrity of the CDI values.

In a banking application, the IVP would ensure that the CDI represents the correct value. For example, if Kathy has $2000 in her account and then deposits $50, the CDI for her account should now have a value of $2050. The IVP ensures the consistency of the data. So after Kathy carries out this transaction and the IVP validates the integrity of the CDI (new bank account value is correct), then the CDI is considered to be in a *consistent state*. TPs are the only components that are allowed to modify the state of the CDIs. In our example, TPs would be software procedures that carry out deposit, withdraw, and transfer functionalities. Using TPs to modify CDIs is referred to as a ***well-formed transaction***.



A well-formed transaction is a series of operations that are carried out to transfer the data from one consistent state to the other. If Kathy transfers money from her checking account to savings account, this transaction is made up of two operations: subtract money from one account and add it to a different account. By making sure that the new values in her checking and savings accounts are accurate and their integrity is intact, the IVP maintains internal and external consistency. The Clark-Wilson model also outlines how to incorporate ***separation of duties*** into the architecture of an application. If we follow our same example of banking software, if a customer needs to withdraw over $10,000, the application may require a supervisor to log in and authenticate this trans-

action. This is a countermeasure to potential fraudulent activities. The model provides the rules that the developers must follow to properly implement and enforce separation of duties through software procedures.

Goals of Integrity Models

The following are the three main goals of integrity models:

- Prevent unauthorized users from making modifications
- Prevent authorized users from making improper modifications (separation of duties)
- Maintain internal and external consistency (well-formed transaction)

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

Internal and external consistency is provided by the IVP, which ensures that what is stored in the system as CDI properly maps to the input value that modified its state. So if Kathy has $2500 in her account and she withdraws $2000, the resulting value in the CDI is $500.

To summarize, the Clark-Wilson model enforces the three goals of integrity by using access triple (subject, software [TP], object), separation of duties, and auditing. This model enforces integrity by using well-formed transactions (through access triple) and separation of user duties.

NOTE The access control matrix was covered in Chapter 4. This is another commonly used model in operating systems and applications.

Information Flow Model

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

The Bell-LaPadula model focuses on preventing information from flowing from a high security level to a low security level. The Biba model focuses on preventing information from flowing from a low integrity level to a high integrity level. Both of these models were built upon the ***information flow model***. Information flow models can deal with any kind of information flow, not only from one security (or integrity) level to another.

In the information flow model, data is thought of as being held in individual and discreet compartments. In the Bell-LaPadula model, these compartments are based on security levels. Remember that MAC systems (which you learned about in Chapter 4) are based on the Bell-LaPadula model. MAC systems use labels on each subject and object. The subject's label indicates the subject's clearance and need to know. The object's label indicates the object's classification and categories. If you are in the army and have a top-secret clearance, this does not mean that you can access all of the army's top-secret information. Information is *compartmentalized*, based on two factors—classification and need to know. Your clearance has to dominate the object's classification and your security profile must contain one of the categories listed in the object's label, which enforces need to know. So Bell-LaPadula is an information flow model that

ensures that information cannot flow from one compartment to another in a way that threatens the confidentiality of the data. Biba compartmentalizes data based on integrity levels. It is an information flow model that controls information flow in a way that is intended to protect the integrity of the most trusted information.

How can information flow within a system? The answer is *many* ways. Subjects can access files. Processes can access memory segments. When data is moved from the hard drive's swap space into memory, information flows. Data is moved into and out of registers on a CPU. Data is moved into different cache memory storage devices. Data is written to the hard drive, thumb drive, CD-ROM drive, and so on. Properly controlling all of these ways of how information flows can be a very complex task. This is why the information flow model exists—to help architects and developers make sure that their software does not allow information to flow in a way that can put the system or data in danger. One way that the information flow model provides this type of protection is by ensuring that covert channels do not exist in the code.

Covert Channels

I have my decoder ring, cape, and pirate's hat on. I will communicate to my spy buddies with this tribal drum and a whistle.

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. This type of information path was not developed for communication; thus, the system does not properly protect this path, because the developers never envisioned information being passed in this way. Receiving information in this manner clearly violates the system's security policy.

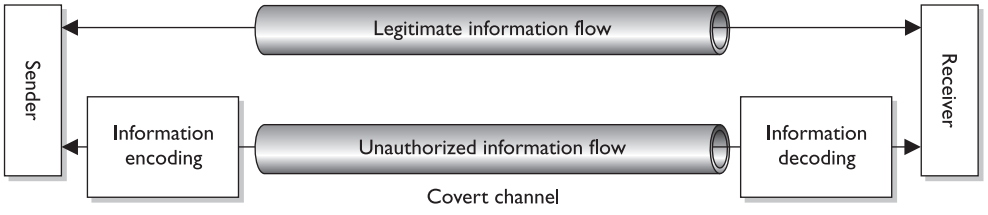
The channel to transfer this unauthorized data is the result of one of the following conditions:

- Oversight in the development of the product
- Improper implementation of access controls
- Existence of a shared resource between the two entities
- Installation of a Trojan horse

Other Types of Covert Channels

Although we are looking at covert channels within programming code, covert channels can be used in the outside world as well. Let's say that you are going to attend one of my lectures. Before the lecture begins, you and I agree on a way of communicating that no one else in the audience will understand. I tell you that if I twiddle a pen between my fingers in my right hand, that means there will be a quiz at the end of class. If I twiddle a pen between my fingers in my left hand, there will be no quiz. It is a covert channel, because this is not a normal way of communicating and it is secretive. (In this scenario, I would twiddle the pen in both hands to confuse you and make you stay after class to take the quiz all by yourself. Shame on you for wanting to be forewarned about a quiz!)

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 may entail accessing the hard drive or using excessive CPU cycles. For example, if one process writes 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 may watch for this "signal" and, once it receives it, carry 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.



In a ***covert storage channel***, one 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 are not supposed to be sharing sensitive data. For example, suppose that an attacker figures out that two processes with different trust levels can both view the pagefile.sys file. Process 1 writes some type of confidential information to the pagefile.sys file, and the attacker can make process 2 read it. This would go against the information flow rules of the system and directly negate the security policy.

Information flow models produce rules on how to ensure that covert channels do not exist. But there are many ways that information flows within a system, so identifying and rooting out covert channels is usually more difficult than one would think at first glance.

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 get rid of them all. The number of acceptable covert channels usually depends on the assurance rating of a system. A system that has an Orange Book rating of B2 has fewer covert channels than a C1 system, because a B2 rating represents a higher assurance level of providing a particular protection level when compared to the C1 rating. There is not much a user can do to counteract these channels; instead, the channels must be addressed when the system is constructed and developed.

NOTE The Orange Book and its ratings are fully covered later in the section "The Orange Book."

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.

References

- **"Secure Databases: An Analysis of Clark-Wilson Model in a Database Environment," by Xiaocheng Ge, Fiona Polack, and Régine Laleau** www-users.cs.york.ac.uk/~fiona/PUBS/CAiSE04.pdf
- **"Access Control: Theory and Practice"** www.cs.purdue.edu/homes/ninghui/courses/Fall03/lectures/lecture11_6.pdf
- **"New Thinking About Information Technology Security," by Marshall D. Abrams, PhD and Michael V. Joyce (first published in *Computers & Security*, Vol. 14, No. 1, pp. 57–68)** www.acsac.org/secshelf/papers/new_thinking.pdf

Noninterference Model

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

Multilevel security properties can be expressed in many 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 rather 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 the 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.

Let's say that Tom and Kathy are both working on a multilevel mainframe at the same time. Tom has the security clearance of secret and Kathy has the security clearance of top secret. Since this is a central mainframe, the terminal Tom is working at has the context of secret, and Kathy is working at her own terminal, which has a context of top secret. This model states that nothing that Kathy does at her terminal should directly or indirectly affect Tom's domain (available resources and working environment). So whatever commands she executes or whichever resources she interacts with should not affect Tom's experience of working with the mainframe in any way. This sounds simple enough, until you actually understand what this model is *really* saying.

It seems very logical and straightforward that when Kathy executes a command, it should not affect Tom's terminal. But the real intent of this model is to address covert channels and inference attacks. The model looks at the shared resources that the different users of a system will use and tries to identify how information can be passed from a process working at a higher security clearance to a process working at a lower security

clearance. Since Tom and Kathy are working on the same system at the same time, they will most likely have to share some type of resources. So the model is made up of rules to ensure that Kathy cannot pass data to Tom through covert storage or timing channels.

The other security breach that this model addresses is the inference attack. An ***inference attack*** occurs when someone has access to some type of information and can infer (or guess) something that he does not have the clearance level or authority to know. For example, let's say that Tom is working on a file that contains information about supplies that are being sent to Russia. He closes out of that file and one hour later attempts to open the same file. During this time, this file's classification has been elevated to top secret, so when Tom attempts to access it, he is denied. Tom can infer that some type of top-secret mission is getting ready to take place with Russia. He does not have clearance to know this, thus it would be an inference attack or "leaking information." (Inference attacks are further explained in Chapter 11.)

Lattice Model

A lattice is a mathematical construct that is built upon the notion of a group. The most common definition of the ***lattice model*** is "a structure consisting of a finite partially ordered set together with least upper and greatest lower bound operators on the set." Further explanation of this statement is usually shown in the following type of representation:

Let S be a set and $\leq$ be a relation on S that satisfies

- $\forall s \in S : s \leq s$ (reflexive)
- $\forall s, t \in S : (s \leq t) \wedge (t \leq s) \Rightarrow s = t$ (antisymmetric)
- $\forall s, t, u \in S : (s \leq t) \wedge (t \leq u) \Rightarrow s \leq u$ (transitivity)
- $\forall s, t \in S : \exists u \in S, u = lub\ (s, t)$
- $\forall s, t \in S : \exists w \in S, w = glb\ (s, t)$ (greatest lower bound)

There are two things wrong with this type of explanation. First, "a structure consisting of a finite partially ordered set together with least upper and greatest lower bound operators on the set" can only be understood by someone who understands the model in the first place. This is similar to the common definition of metadata: "data about data." Only *after* you really understand what metadata is does this definition make any sense to you. So this definition of lattice model is not overly helpful.

The problem with the mathematical explanation is that it is in weird alien writings that only people who obtain their master's or PhD degree in mathematics can understand. This model needs to be explained in everyday language, so that even Homer Simpson can understand it. So let's give it a try.

The MAC model was explained in Chapter 4 and then built upon in this chapter. In this model, the subjects and objects have labels. Each subject's label contains the clearance and need-to-know categories that this subject can access. Suppose that Kathy's security clearance is top secret and she has been formally granted access to the

compartments named Iraq and Korea, based on her need to know. So Kathy's security label states the following: TS {Iraq, Korea}. Table 5-1 shows the different files on the system in this scenario. The system is based on the MAC model, which means that the operating system is making access decisions based on security label contents.

Kathy attempts to access File B; since her clearance is greater than File B's classification, she can read this file but not write to it. (Remember, under Bell-LaPadula, a higher-level subject can read down, but not write up). This is where the "*partially ordered set* together with *least upper* and *greatest lower bound* operators on the set" comes into play. A set is a subject (Kathy) and an object (file). It is a partially ordered set because all of the access controls are not completely equal. The system has to decide between read, write, full control, modify, and all the other types of access permissions used in this operating system. So, "partially ordered" means that the system has to apply the most restrictive access controls to this set, and "least upper bound" means that the system looks at one access control's statement (Kathy can read the file) and the other access control's statement (Kathy cannot write to the file) and takes the least *upper* bound value. Since no write is more restrictive than read, Kathy's *least upper bound* access to this file is read and her greatest lower bound is no write. Figure 5-19 illustrates the bounds of access. This is just a confusing way of saying, "The most that Kathy can do with this file is to read it. The least she can do is to *not* write to it."

Let's figure out the least upper bound and greatest lower bound access levels for Kathy and File C. Kathy's clearance equals File C's classification. Under the Bell-LaPadula model, this is when the strong star property would kick in. (Remember that the strong star property states that a subject can read and write to an object of the same security level.) So the least upper bound is write and the greatest lower bound is read.

If we look at File D's security label, we see that it has a category that Kathy does not have in her security label, which is Iran. This means that Kathy does not have the necessary need to know to be able to access this file. Kathy's least upper bound and greatest lower bound access permission is no access.

So why does this model state things in a very confusing way when in reality it describes pretty straightforward concepts? First, I am describing this model in the most simplistic and basic terms possible so that you can get the basic meaning of the purpose of the model. These seemingly straightforward concepts build in complexity when you think about all the subject-to-object communications that go on within an operating system during any one second. Also, this is a formal model, which means that it can be proven mathematically to provide a specific level of protection if all of its rules are followed properly. Learning these models is similar to learning the basics of chemistry. A student first learns about the components of an atom (protons, neutrons, and electrons) and how these elements interact with each other. This is the easy piece. Then the student gets into organic chemistry and has to understand how these components work

Kathy's Security Label	File B's Security Label	File C's Security Label	File D's Security Label
Top Secret {Iraq, Korea}	Secret {Iraq}	Top Secret {Iraq, Korea}	Secret {Iraq, Korea, Iran}

Table 5-1 Security Access Control Elements

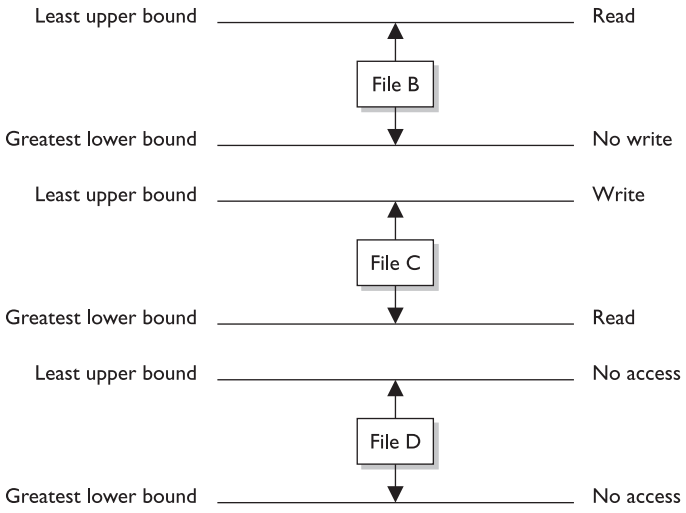


Figure 5-19 Bounds of access through the lattice model

together in complex organic systems (weak and strong attractions, osmosis, and ionization). The student then goes to quantum physics to learn that the individual elements of an atom actually have several different subatomic particles (quarks, leptons, and mesons). In this book, you are just learning the basic components of the models. There is a lot more complexity once you start looking under the covers.

Brewer and Nash Model

A wall separates our stuff, so that you can't touch my stuff. Response: Your stuff is green and smells funny. I don't want to touch it.

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 conflicts of interest by users' access attempts. For example, if a large marketing company provides marketing promotions and materials for two banks, an employee working on a project for Bank A should not look at the information the marketing company has on its other bank customer, Bank B. Such action could create a conflict of interest because the banks are competitors. If the marketing company's project manager for the Bank A project could view information on Bank B's new marketing campaign, he may try to trump its promotion to please his more direct customer. The marketing company would get a bad reputation if it allowed its internal employees to behave so irresponsibly.

This marketing company could implement a product that tracks the different marketing representatives' access activities and disallows certain access requests that would present this type of conflict of interest. In Figure 5-20, 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 authorizations, activities, and previous access requests.

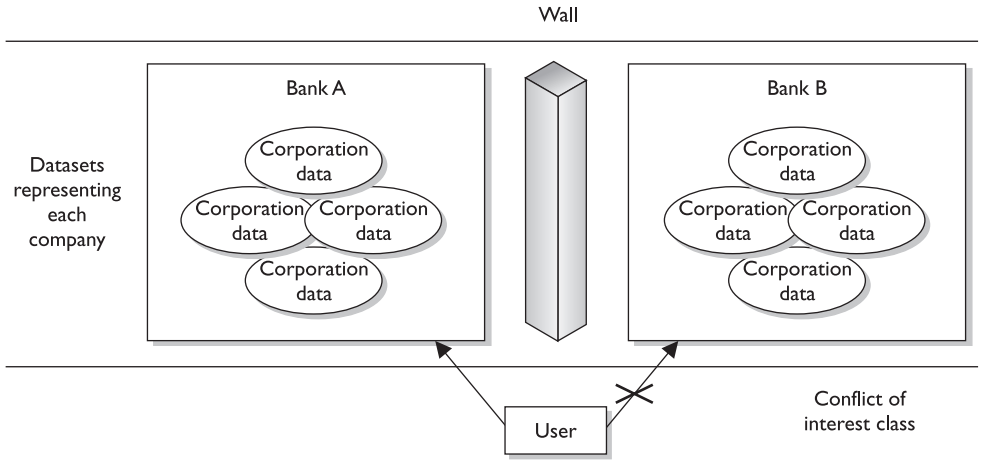


Figure 5-20 The Chinese Wall model provides dynamic access controls.

The Chinese Wall model is also based on an information flow model. No information can flow between subjects and objects in a way that would result in a conflict of interest. The model states that a subject can write to an object if, and only if, the subject cannot read another object that is in a different dataset. So if we stay with our example, the project manager could not write to any objects within the Bank A dataset if he currently has read access to any objects in the Bank B data set.

This is only one example of how this model can be used. There are other industries that have their own possible conflicts of interest. If you were Martha's stockbroker, you should not be able to read a dataset that indicates that a stock's price is getting ready to go down *and* be able to write to Martha's account indicating that she should sell the stock she has.

Graham-Denning Model

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. This model has eight primitive protection rights, or rules of how these types of functionalities should take place securely, which are outlined here:

- How to securely create an object
- How to securely create a subject
- How to securely delete an object
- How to securely delete a subject

- How to securely provide the read access right
- How to securely provide the grant access right
- How to securely provide the delete access right
- How to securely provide transfer access rights

These things may sound insignificant, but when we are talking about building a secure system they are very critical.

Security Models Recap

All of these different models can get your head spinning. Most people are not familiar with all of them, which can make it all even harder to absorb. The following are the core concepts of the different models.

Bell-LaPadula Model This confidentiality model describes the allowable information flows and formalizes the military security policy. It is the first mathematical model of a multilevel security policy that defines the concept of a secure state and necessary modes of access.

- **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 can perform read and write functions only to the objects at its same security level.

Biba Model This model protects the integrity of the information within a system and the activities that take place. It addresses the first goal of integrity.

- **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 This integrity model is implemented to protect the integrity of data and to ensure that properly formatted transactions take place. It addresses all three goals of integrity.

- Subjects can access objects only through authorized programs (access triple).
- Separation of duties is enforced.
- Auditing is required.

Access Control Matrix Model This is a model in which access decisions are based on objects' ACLs and subjects' capability tables.

Information Flow Model This is a model in which 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 This model states that commands and activities performed at one security level should not be seen by or affect subjects or objects at a different security level.

Brewer and Nash Model This model allows for dynamically changing access controls that protect against conflicts of interest. Also known as the Chinese Wall model.

Graham-Denning Model This model shows how subjects and objects should be created and deleted. It also addresses how to assign specific access rights.

References

- **Various Papers on Models** http://citeseer.ist.psu.edu/context/20585/0
- **Module 8, "Mandatory Access Control and Labels"** www.radium.ncsc.mil/tpep/library/ramp-modules/mod_08.txt
- **"A Lattice Model of Secure Information Flow," by Dorothy E. Denning (first published in *Communications of the ACM*, Vol. 19, No. 5, pp. 236–243 (May 1976)** www.cs.georgetown.edu/~denning/infosec/lattice76.pdf
- **Course syllabus for Security, University of Cambridge, Dr. Ross Anderson, principle lecturer (Jan. 1999)** www.cl.cam.ac.uk/Teaching/1998/Security/
- **Access Control and Security Policy Models** http://www.tml.hut.fi/Opinnot/T-110.402/2003/Luennot/titu20031024.pdf

Security Modes of Operation

A system can operate in different 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.

These modes are used in MAC systems, which hold one or more classifications of data. Several things come into play when determining the mode that the operating system should be working in:

- The types of users who will be directly or indirectly connecting to the system
- The type of data (classification levels, compartments, and categories) that are processed on the system
- The clearance levels, need to know, and formal access approvals that the users will have

The following sections describe the different security modes that operating systems can be developed and configured to work in.

Dedicated Security Mode

Our system only holds secret data and we can all access it.

A system is operating in a ***dedicated security mode*** if *all* users have a clearance for and a formal need to know about *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 (NDAs) 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 holds top-secret data, only users with that clearance can 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

Our system only holds secret data, but only some of us can access all of it.

A system is operating in ***system high-security mode*** when all users have a security clearance to access the information but not necessarily a need to know for all the information processed on the system. So, unlike in the dedicated security mode, in which 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, even though a user has the necessary security level to access an object, the user may still be restricted if they do not have a need to know pertaining to that specific object.

Compartmented Security Mode

Our system has various classifications of data, and each individual has the clearance to access all of the data, but not necessarily the need to know.

A system is operating in ***compartmented security mode*** when all users have the clearance to access all the information processed by the system in a system high-security configuration, but might not have the need to know and formal access approval. This means that if the system is holding secret and top-secret data, all users must have at least a top-secret clearance to gain access to this system. This is how compartmented and multilevel security modes are different. Both modes require the user to have a valid need to know, NDA, and formal approval, but compartmented security mode requires the user to have a clearance that dominates (above or equal to) any and all data on the system, whereas multilevel security mode just requires the user to have clearance to access the data she will be working with.

In compartmented security mode, users are restricted from accessing 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 it. This would be enforced by having on all objects security labels that reflect the sensitivity (classification level, classification

category, and handling procedures) of the information. In this mode, users can access a compartment, of data only, enforced by mandatory access controls.

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

Our system has various classifications of data, and each individual has the clearance and need to know to access only individual pieces of data.

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 not all of the users have the clearance or formal approval to access all the information being processed by the system. So all users must have formal approval, NDA, need to know, and the necessary clearance to access the data that they need to carry out their jobs. In this mode, the user cannot access all of the data on the system, only what she is cleared to access.

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 simultaneously.

Guards

Software and hardware guards allow the exchange of data between trusted (high assurance) and less trusted (low assurance) systems and environments. Let's say that you are working on a MAC system (working in dedicated security mode of secret) and you need the system to communicate with a MAC database (working in multilevel security mode, which goes up to top secret). These two systems provide different levels of protection. As previously stated, if a system with lower assurance could directly communicate with a system of higher assurance, then security vulnerabilities and compromises could be introduced. So, a software guard can be implemented, which is really just a front-end product that allows interconnectivity between systems working at different security levels. (There are different types of guards, and they can carry out filtering, processing requests, data blocking, and data sanitization.) Or a hardware guard can be implemented, which is a system with two NICs connecting the two systems that need to communicate. The guard is an add-on piece that provides a level of strict access control between different systems.

The guard accepts requests from the system of lower assurance, reviews the request to make sure it is allowed, and then submits the request to the system of higher assurance. The goal is to ensure that information does not flow from a high security level to a low security level in an unauthorized manner.

Guards can be used to connect different MAC systems working in different security modes and to connect different networks working at different security levels. In many cases, the less trusted system can send messages to the more trusted system but can only receive acknowledgments in return. This is common when e-mail messages need to go from less trusted systems to more trusted classified systems.

Security Modes Recap

Many times it is easier to understand these different modes when they are laid out in a clear and simplistic format. Pay attention to the italic words, because they emphasis the differences between the various modes.

Dedicated Security Mode All users must have…

- Proper clearance for all information on the system
- Formal access approval for all information on the system
- Signed NDA for all information on the system
- Valid need to know for *all* information on the system

All users can access all data.

System High-Security Mode All users must have…

- Proper clearance for all information on the system
- Formal access approval for all information on the system
- Signed NDA for all information on the system
- Valid need to know for *some* information on the system

All users can access some data, based on their need to know.

Compartmented Security Mode All users must have…

- Proper clearance for the highest level of data classification on the system
- Formal access approval for all information they will access on the system
- Signed NDA for all information they will access on the system
- Valid need to know for *some* of the information on the system

All users can access some data, based on their need to know and formal access approval.

Multilevel Security Mode All users must have…

- Proper clearance for all information they will access on the system
- Formal access approval for all information they will access on the system
- Signed NDA for all information they will access on the system
- Valid need to know for *some* of the information on the system

All users can access some data, based on their need to know, clearance, and formal access approval.

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
- **Chapter 8, "Automated Information System Security"** www.dss.mil/isec/chapter8.htm
- ***National Industrial Security Program Operating Manual (NISPOM)*, U.S. Department of Defense** www.tscm.com/Nispom.html

Trust and Assurance

I trust that you will act properly, thus I have a high level of assurance in you. Response: You are such a fool.

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

The TCB comprises 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 their 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 components are put through an evaluation process and assigned an assurance rating, which represents the level of trust and assurance that the testing team has in the product. Customers then use this rating to determine which system best fits their security needs.

Assurance and trust are similar in nature, but slightly different with regard to product ratings. In a trusted system, all protection mechanisms work together to process sensitive data for many types of uses, and will provide the necessary level of protection per classification level. Assurance looks at the same issues but in more depth and detail. Systems that provide higher levels of assurance have been tested extensively and have had their designs thoroughly inspected, their development stages reviewed, and their technical specifications and test plans evaluated. 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 the car 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 (TCSEC), commonly known as the Orange Book (addressed shortly), the lower assurance level ratings look at a

system's protection mechanisms and testing results to produce an assurance rating, but the higher assurance level ratings look more at the system design, specifications, development procedures, supporting documentation, and testing results. The protection mechanisms in the higher assurance level systems may not necessarily be much different from those in the lower assurance 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 levels to systems. Two reasons explain why more than one type of assurance evaluation process exist: methods and ideologies have evolved over time, and various parts of the world look at computer security differently and rate some aspects of security differently. Each method will be explained and compared.

Why Put a Product Through Evaluation?

Submitting a product to be evaluated against the Orange Book, Information Technology Security Evaluation Criteria, or Common Criteria is no walk in the park for a vendor. In fact, it is a really painful and long process, and no one wakes up in the morning thinking, "Yippee! I have to complete all of the paperwork that the National Computer Security Center requires so that my product can be evaluated!" So, before we go through these different criteria, let's look at *why* anyone would even put themselves through this process.

If you were going shopping to buy a firewall, how would you know what level of protection each provides and which is the best product for your environment? You could listen to the vendor's marketing hype and believe the salesperson who informs you that a particular product will solve all of your life problems in one week. Or you could listen to the advice of an independent third party who has fully tested the product and does not have any bias toward the product. If you choose the second option, then you join a world of people who work within the realm of assurance ratings in one form or another.

In the United States, the National Computer Security Center (NCSC) is an organization within the National Security Agency (NSA) that is responsible for evaluating computer systems and products. It has a group, Trusted Product Evaluation Program (TPEP), which oversees the testing by approved evaluation entities of commercial products against a specific set of criteria.

So, a vendor creates a product and submits it to an approved evaluation entity that is compliant with the TPEP guidelines. The evaluation entity has groups of testers who will follow a set of criteria (for many years it was the Orange Book but now is moving toward the Common Criteria) to test the vendor's product. Once the testing is over, the product is assigned an assurance rating. So, instead of having to trust the marketing

hype of the financially motivated vendor, you as a consumer can take the word of an objective third-party entity that fully tested the product.

This evaluation process is very time consuming and expensive for the vendor. Not every vendor puts its product through this process, because of the expense and delayed date to get it to market. Typically, a vendor would put its product through this process if its main customer base will be making purchasing decisions based on assurance ratings. In the United States, the Department of Defense *is* the largest customer, so major vendors put their main products through this process with the hope that the Department of Defense (and others) will purchase their products.

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 assurance rating that the criteria presents as a metric when comparing different products. It also provides direction for manufactures so that 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 is used to evaluate whether a product contains the security properties the vendor claims it does and whether the product is appropriate for a specific application or function. The Orange Book is used to review the functionality, effectiveness, and assurance of a product during its evaluation, and it uses classes that were devised to address typical patterns of security requirements.

TCSEC provides a classification system that is divided into hierarchical divisions of assurance levels:

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

Classification A represents the highest level of assurance and D represents the lowest level of assurance.

Each division can have one or more numbered classes with a corresponding set of requirements that must be met for a system to achieve that particular rating. The classes with higher numbers offer 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 include 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** Documentation must be provided, including test, design, and 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's 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.

So, when a vendor submits a product for evaluation, it submits it to the NCSC. The group that oversees the processes of evaluation is called the Trusted Products Evaluation Program (TPEP). Successfully evaluated products are placed on the Evaluated Products List (EPL) with their corresponding rating. When consumers are interested in certain products and systems, they can check the appropriate EPL to find out their assigned security levels.

Isn't the Orange Book Dead?

We are moving from the Orange Book to the Common Criteria in the industry, so a common question is, "Why do I have to study this Orange Book stuff?" Just because we are going through this transition, does not make the Orange Book unimportant. It was the first evaluation criteria and was used for 20 years. Many of the basic terms and concepts that have carried through originated in the Orange Book. And we still have several products with these ratings that eventually will go through the Common Criteria evaluation process.

The CISSP exam is moving steadily from the Orange Book to the Common Criteria all the time, but don't count the Orange Book out yet.

As a follow-on observation, many people are new to the security field. It is a booming market, which means a flood of not-so-experienced people will be jumping in and attempting to charge forward without a real foundation of knowledge. To some readers, this book will just be a nice refresher and something that ties already known concepts together. To other readers, many of these concepts are new and more challenging. If a lot of this stuff is new to you—you are new to the market. That is okay, but knowing how we got where we are today is very beneficial because it broadens your view of *deep* understanding—instead of just memorizing for an exam.

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, which are described next. 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 that users can ensure that their data will not be accessed and corrupted by others. The system architecture must supply a protected execution domain so that 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 that the system was built to include protection mechanisms, test documentation (test plan and results), a facility manual, so that companies know how to install and configure the system correctly, and user manuals.

The type of environment that would require this rating is one in which 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 that proper protection can be applied to the resource and 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 attempts to compromise it harder to accomplish.

The type of environment that would require systems with a C2 rating is one in which users 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's and object's security labels to ensure that 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.

This security rating 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 that the subject communicates directly with the application or operating system, and no trapdoors exist. 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 type of environment that would require B2 systems is one that processes sensitive data that requires a higher degree of security. This type of environment would require systems that are relatively resistant to penetration and compromise.

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, so must the skill level of the individuals who need to test, maintain, and configure it; 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.

The type of 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 that 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 of the formality in the way the A1 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 in which 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.

The type of environment that would require A1 systems is the most secure of secured environments. This type of 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 rather are combined and rated as a whole.

References

- **Trusted Computer System Evaluation Criteria (Orange Book), U.S. Department of Defense (Dec. 26, 1985)** www.boran.com/security/tcsec.html
- **Lecture notes for Cryptography and Network Security, Part 7, "Trusted Computer Systems," William Stallings, lecturer, Dr. Lawrie Brown, author** williamstallings.com/Extras/Security-Notes/lectures/trusted.html

Rainbow Series

Why are there so many colors in the rainbow? Response: Because there are so many product types that need to be evaluated.

The Orange Book mainly addresses government and military requirements and expectations for their computer systems. Many people within the security field have pointed out several deficiencies in 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 list summarizes a majority of the troubling issues that security practitioners have expressed about the Orange Book:

- It looks specifically at the operating system and not at other issues like networking, databases, and so on.
- It focuses mainly on one attribute of security, confidentiality, and not at integrity and availability.
- It works with government classifications and not the protection classifications that commercial industries use.
- It 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, whereas 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 operating systems 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 the cover of each is a different color.

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

References

- **Rainbow Series** csrc.ncsl.nist.gov/secpubs/rainbow/
- **Rainbow Series and related documents from the Federation of American Scientists** www.fas.org/irp/nsa/rainbow.htm

Red Book

The Orange Book addresses single-system security, but networks are a combination of systems, and each 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 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 the protocol header, routing information, and 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 notarization.
- **Denial-of-service prevention**
  - **Continuity of operations** Ensures that the 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 sending false messages.

- **Selective routing** Routes messages in a way to avoid specific threats. Mechanisms include network configuration and routing tables.

Assurance is derived by comparing how things actually work to a theory of how things should work. Assurance is also derived by testing configurations in many different scenarios, evaluating engineering practices, and validating and verifying security claims.

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 and products by many European countries. The United States looked to the Orange Book and Rainbow Series, and Europe employed ITSEC to evaluate and rate computer systems. (Today, everyone is migrating to the Common Criteria, explained in the next section.)

There are two main attributes of a system's protection mechanisms when they are evaluated under ITSEC: functionality and assurance. When the functionality of a system's protection mechanisms is being evaluated, the services that are provided to the subjects (access control mechanisms, auditing, authentication, and so on) are examined and measured. Protection mechanism 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's protection mechanisms deliver what its vendor says they deliver. Assurance, on the other hand, is the degree of confidence in the protection mechanisms, and their 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' protection mechanisms to provide the same type of functionalities and have very different assurance levels. This is because the underlying mechanisms providing the functionality can be developed, engineered, and implemented differently. System A and System B may have protection mechanisms that provide the same type of functionality for authentication, in which case both products would get the same rating for functionality. But System A's developers could have been sloppy and careless when developing their authentication mechanism, in which case their product would receive a lower assurance rating. ITSEC actually separates these two attributes (functionality and assurance) and rates them separately, whereas TCSEC clumps them together and assigns them one rating (D through A1).

The following list shows the different types of functionalities and assurance items that are tested during an evaluation.

- **Security functional requirements**
  - Identification and authentication
  - Audit
  - Resource utilization
  - Trusted paths/channels
  - User data protection
  - Security management
  - Product access
  - Communications
  - Privacy
  - Protection of the product's security functions
  - Cryptographic support
- **Security assurance requirements**
  - Guidance documents and manuals
  - Configuration management
  - Vulnerability assessment
  - Delivery and operation
  - Life-cycle support
  - Assurance maintenance
  - Development
  - Testing

Consider again our example of two systems that provide the same functionality (pertaining to the protection mechanisms) but have very different assurance levels. Using the TCSEC approach, the difference in assurance levels will be hard to distinguish because the functionality and assurance level are rated together. Under the ITSEC approach, the functionality is rated separately from the assurance, so the difference in assurance levels will be more noticeable. In the ITSEC criteria, classes F1 to F10 rate the functionality of the security mechanisms, whereas E0 to E6 rate the assurance of those mechanisms.

So a difference between ITSEC and TCSEC is that TCSEC bundles functionality and assurance into one rating, whereas ITSEC evaluates these two attributes separately. The other differences are that ITSEC was developed to provide more flexibility than TCSEC, and ITSEC addresses integrity, availability, and confidentiality whereas TCSEC addresses only confidentiality. ITSEC also addresses networked systems, whereas TCSEC deals with stand-alone systems.

Table 5-2 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 that 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

Table 5-2 ITSEC and TCSEC Mapping

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 criteria for operating systems and other products, which it refers to individually 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.

The ratings are based on effectiveness and correctness. Effectiveness means that the TOE meets the security claims that the vendor has specified. This analysis looks at the construction and operational vulnerabilities and the ease of use, to ensure that the proper security settings do not get in the way of productivity. Correctness deals with how the TOE was built and implementation issues. This type of analysis looks at the architectural design, how the security mechanisms enforce the policy, and the operational documentation and environment.

References

- **Criteria and Methods of Evaluations of Information Systems** www.cordis.lu/infosec/src/crit.htm
- **ITSEC home page** www.iwar.org.uk/comsec/resources/standards/itsec.htm

Common Criteria

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

The Orange Book and the Rainbow Series provide evaluation schemes that are too rigid for the business world. 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 resulted 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). The Common Criteria 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 set of criteria is that it 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 evaluates all systems by how they compare to the Bell-LaPadula model. The Common Criteria provides more flexibility by evaluating a product against a protection profile, which is structured to address a real-world security need. So while the Orange Book says, "Everyone march in this direction in this form using this path," the Common Criteria asks, "Okay, what are the threats we are facing today and what are the best ways of battling them?"

Under the Common Criteria model, an evaluation is carried out on a product and is assigned an ***Evaluation Assurance Level (EAL)***. The thorough and stringent testing increases in detailed-oriented tasks as the assurance levels increase. The Common Criteria has seven assurance levels. The range is 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

NOTE When a system is "formally verified" this means that it is based on a model that can be mathematically proven.

The Common Criteria uses ***protection profiles*** in its evaluation process. This is a mechanism that is used to describe a real-world need of a product that is not currently on the market. 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 protection profile describes the environmental assumptions, the objectives, and the functional and assurance level expectations. Each relevant threat is listed along with how it is to be controlled by specific objectives. The protection profile 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, that person can write a protection profile describing the product that would be a solution for this real-world problem. The protection 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 then 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 the product requirements.

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

A protection profile contains the following five sections:

- **Descriptive elements** Provides the name of the profile and a description of the security problem that is to be solved.
- **Rationale** Justifies the profile and gives 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** Establishes a protection boundary, 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** Identifies the specific requirements that the product or system must meet during the development phases, from design to implementation.
- **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, the company has no guarantee that the next version of that software will have 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 that are not recommended, the level of security the company was hoping to achieve can easily go down the drain. So all of this rating stuff is a formalized method of reviewing a system being evaluated in a lab. When the product is implemented in a real environment, factors other than its rating need to be addressed and assessed to ensure that it is properly protecting resources and the environment.

NOTE When a product is assigned an assurance rating, this means it has the *potential* of providing this level of protection. The customer has to properly configure the product to actually obtain this level of security. The vendor should provide the necessary configuration documentation, and it is up to the customer to keep the product properly configured at all times.

References

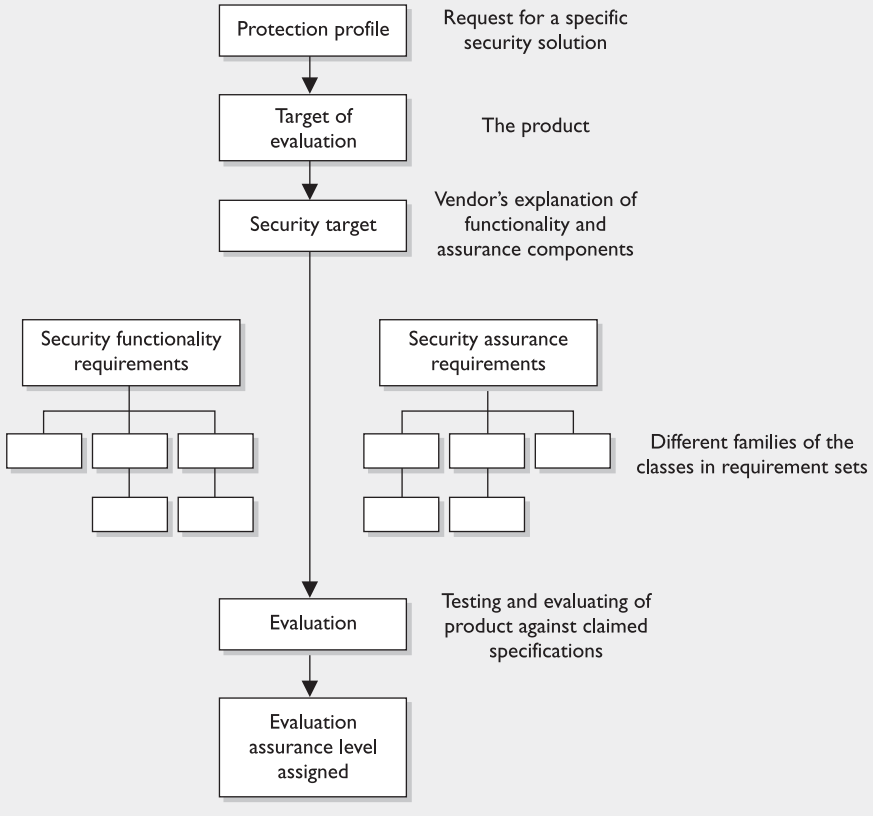
- **Computer Security Resource Center (CSRC) Common Criteria for IT Security Evaluation** csrc.nist.gov/cc/
- **Common Criteria** www.radium.ncsc.mil/tpep/library/ccitse/index.html
- **Common Criteria overview, Rycombe Consulting** www.rycombe.com/cc.htm

Certification vs. Accreditation

We have gone through the different types of evaluation criteria that a system can be appraised against to receive a specific rating. This is a very formalized process, following which the evaluated system or product will 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 protection. However, once a consumer buys this product and sets it up in their environment, security is not guaranteed. 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.

Different Components of the Common Criteria

The different components of the Common Criteria are shown and described here:



- **Protection profile** Description of needed security solution.
- **Target of evaluation** Product proposed to provide needed security solution.
- **Security target** Vendor's written explanation of the 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—EALs** Functional and assurance requirements are bundled into packages for reuse. This component describes what must be met to achieve specific EAL ratings.

Certification

How did you certify this product? Response: It came in a very pretty box. Let's keep it.

Certification is the comprehensive technical evaluation of the security components and their compliance for the purpose of accreditation. A certification process may use safeguard evaluation, risk analysis, verification, testing, and auditing techniques to assess the appropriateness of a specific system. For example, suppose that Dan is the security officer for a company that just purchased new systems to be used to process its confidential data. He wants to know if these systems are appropriate for these tasks and if they are going to provide the necessary level of protection. He also wants to make sure that they are compatible with his current environment, do not reduce productivity, and do not open doors to new threats—basically he wants to know if these are the right products for his company. He could pay a company that specializes in these matters to perform the necessary procedures to certify the systems or it can be carried out internally. The evaluation team will perform tests on the software configurations, hardware, firmware, design, implementation, system procedures, and physical and communication controls.

The goal of a certification process is to ensure that a system, product, or network is right for the customer's purposes. Customers will rely upon a product for slightly different reasons, and environments will have various threat levels. So a particular product is not necessarily the best fit for every single customer out there. (Of course, vendors will try to convince you otherwise.) The product has to provide the right functionality and security for the individual customer, which is the whole purpose of a certification process.

The certification process and corresponding documentation will indicate the good, the bad, and the ugly about the product and how it works within the given environment. Dan will take these results and present them to his management for 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 whether to accept the product and whether 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 and maintaining this system.

NOTE Certification is a technical review that assesses the security mechanisms 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.

Open vs. 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. Response: But, no one wants to play with you.

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 created by different vendors. 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 communicating easily on the same network is because these platforms are open. If a software vendor creates a closed system, it is restricting its potential sales to proprietary environments.

NOTE In Chapter 11, we will look at the standards that support interoperability, including CORBA, DCOM, J2EE, and more.

No More Pencil Whipping

Many organizations are taking the accreditation process more seriously than they did in the past. Unfortunately, sometimes when a certification process is completed and the documentation is sent to management for review and approval, management members just blindly sign the necessary documentation without really understanding what they are signing. Accreditation means that management is accepting the *risk* that is associated with allowing this new product to be introduced into the organization's environment. When large security compromises take place, the buck stops at the individual who signed off on the offending item. So as these management members are being held more accountable for what they sign off on and as more regulations make executives personally responsible for security, the pencil whipping of accreditation papers is decreasing, but has not stopped.

Closed Systems

I only want to play with you and him. Response: Just play with 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. 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 it seems.

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.

NOTE Carnegie Mellon University estimates that there are 5 to 15 bugs in every 1000 lines of code. Windows 2000 has 40–60 million lines of code.

Maintenance Hooks

In the programming world, ***maintenance hooks*** are a type of backdoor. They are instructions within software that only the developer knows about and can invoke. They are placed in the software to give the developer easy access to the code. They 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 the software goes into production, they can cause major security issues.

The maintenance hook 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 commands by using a specific sequence of keystrokes. Once this is done successfully, the developer can be inside the application looking directly at the code or configuration files. 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 taking place. Although this sounds nice and healthy, if an attacker finds out about this maintenance hook, he can take more sinister actions. So all maintenance hooks need to be removed from software before it goes into production.

NOTE Many would think that since security is more in the minds of people today, that maintenance hooks would be a thing of the past. This is not true. Developers are still using maintenance hooks, because of their lack of understanding or care of security issues, and many maintenance hooks still reside in older software that organizations are using.

Countermeasures

Because maintenance hooks are usually inserted by programmers, they are the ones who usually have to take them out before the programs go into production. Code reviews and unit and quality assurance testing should always be looking out for backdoors, in case the programmer overlooked extracting them. Because maintenance hooks 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 its product, it usually develops and releases a patch to reduce this vulnerability. Because most vendors sell their software without including the associated source code, it may be very difficult for companies who have purchased software to identify backdoors. The following lists some preventative measures against backdoors:

- Use a host intrusion detection system to watch for an attacker using a backdoor into the system.
- Use file system encryption to protect sensitive information.
- Implement auditing to detect any type of backdoor use.

Time-of-Check/Time-of-Use Attack

Specific attacks can take advantage of the way a system processes requests and performs tasks. A ***time-of-check/time-of-use (TOC/TOU) attack*** deals with the sequence of steps that a system uses to complete a task. This type of attack takes advantage of the dependency on the timing of events that take place in a multitasking operating system.

As stated previously, operating systems and applications are in reality just lines and lines of instructions. An operating system has to carry out instruction 1, then instruction 2, then instruction 3, and so on. This is how it is written. If an attacker can get in between instruction 2 and 3 and manipulate something, then she can control the result of these activities.

An example of a TOC/TOU attack is if process 1 validates the authorization of a user to open a noncritical text file and process 2 carries out the `open` command. If the attacker can change out this noncritical text file with a password file while process 1 is carrying out its task, she has just obtained access to this critical file. (It is a flaw within the code that allows this type of compromise to take place.)

NOTE This type of attack is also referred to as an ***asynchronous attack***. Asynchronous describes a process in which the timing of each step may vary. The attack gets in between these steps and modifies something. Race conditions are also considered TOC/TOU attacks by some in the industry.

A ***race condition*** is when two different processes need to carry out their tasks on one resource. The processes 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 can manipulate the processes so that process 2 does its task first, she can control the outcome of the processing procedure. Let's say that process 1's instructions are to add 3 to a value and process 2's instructions are to divide by 15. If process 2 carries out its tasks before process 1, the outcome would be different. So if an attacker can make process 2 do its work before process 1, she can control the result.

Looking at this issue from a security perspective, there are several types of race condition attacks that are quite concerning. If a system splits up the authentication and authorization steps, an attacker could be authorized before she is even authenticated. For example, in the normal sequence, process 1 verifies the authentication before allowing a user access to a resource, and process 2 authorizes the user to access the resource. If the attacker makes process 2 carry out its tasks before process 1, then she can access a resource without the system making sure she has been authenticated properly.

So although the terms "race condition" and "TOC/TOU attack" are sometimes used interchangeably, in reality they are two different things. A race condition is an attack in which an attacker makes processes execute out of sequence to control the result. A TOC/TOU attack is when an attacker jumps in between two tasks and modifies something to control the result.

Countermeasures

It would take a dedicated attacker with great precision to perform these types of attacks, but it is possible and has been done. To protect against race condition attacks, it is best to *not* split up critical tasks that can have their sequence altered. This means that the system should use atomic operations where only one system call is used to check authentication and then grant access in one task. This would not give the processor the opportunity to switch to another process in between two tasks. Unfortunately, using these types of atomic operations is not always possible.

To avoid TOC/TOU attacks, it is best if the operating system can apply software locks to the items that it will use when it is carrying out its "checking" tasks. So if a user requests access to a file, while the system is validating this user's authorization, it should put a software lock on the file that is being requested. This ensures that the file cannot

be deleted and replaced with another file. Applying locks can be carried out easily on files, but it is more challenging to apply locks to database components and table entries to provide this type of protection.

Buffer Overflow

My cup runneth over and so does my buffer.

Today, many people know the term buffer overflow and the basic definition, but it is important for security professionals to understand what is going on beneath the covers.

A ***buffer overflow*** takes place when too much data is accepted as input to an application or operating system. A buffer is an allocated segment of memory. A buffer can be overflowed arbitrarily with too much data, but for it to be of any use to an attacker, the code that is inserted into the buffer must be of a specific length followed up by commands the attacker wants to be executed. So, the purpose of a buffer overflow may be either to make a mess, by shoving arbitrary data into various memory segments, or to accomplish a specific task, by pushing into the memory segment a carefully crafted set of data that will accomplish a specific task. This task could be to open a command shell with administrative privilege or execute malicious code.

Let's take a deeper look at how this is accomplished. Software may be written to accept data from a user, web site, database, or another application. The accepted data needs something to happen to it, because it has been inputted for some type of manipulation or calculation or to be used as a parameter to be passed to a procedure. A procedure is code that can carry out a specific type of function on the data and return the result to the requesting software, as shown in Figure 5-21.

When a programmer writes a piece of software that will accept data, this data will be stored in a variable. When this software calls upon a procedure to carry out some type of functionality, it stacks the necessary instructions and data in a memory segment for the procedure to read from. (Memory stacks were explained earlier in the chapter, but we will go over them again in this section.)

The data that is accepted from an outside entity is placed in a variable. This variable has to have a place to live in memory, which is called a buffer. A buffer is like a memory container for data. The buffer needs to be the right size to accept the inputted data. So if the input is supposed to be one character, the buffer should be one byte in size. If a programmer does not ensure that only one byte of data is being inserted into the software, then someone can input several characters at once and overflow that specific buffer.

The buffers hold data, which are placed on a memory stack. You can think of a buffer as a small bucket to hold water (data). We have several of these small buckets stacked on top of one another (memory stack), and if too much water is poured into the top bucket, it spills over into the buckets below it (buffer overflow) and overwrites the instructions and data on the memory stack.

What Is a Stack and How Does It Work?

If you are interacting with an application that calculates mortgage rates, you have to put in the parameters that need to be calculated—years of loan, percentage of interest rate, and amount of loan. These parameters are passed into empty variables and put in a

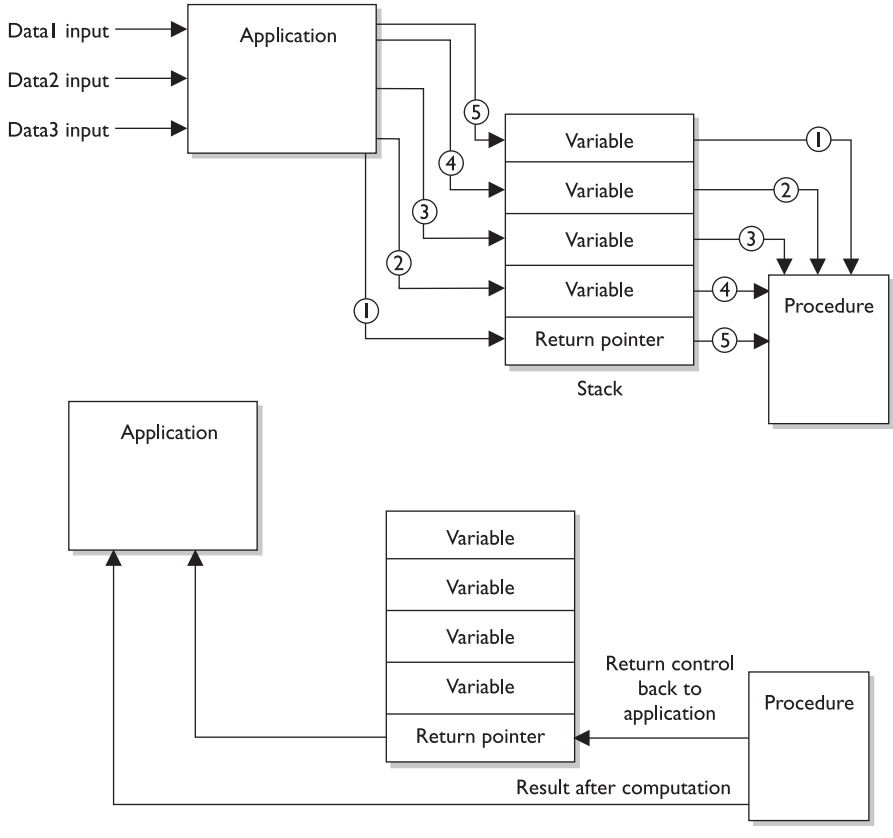


Figure 5-21 A memory stack has individual buffers to hold instructions and data.

linear construct (memory stack), which acts like a queue for the procedure to pull from when it carries out this calculation. The first thing your mortgage rate application lays down on the stack is its return pointer. This is a pointer to the requesting application's memory address that tells the procedure to return control to the requesting application after the procedure has worked through all the values on the stack. The mortgage rate application then places on top of the return pointer the rest of the data you have inputted and sends a request to the procedure to carry out the necessary calculation, as illustrated in Figure 5-21. The procedure takes the data off the stack starting at the top, so it is first in, last off (FILO). The procedure carries out its functions on all the data and returns the result and control back to the requesting mortgage rate application once it hits the return pointer in the stack.

So the stack is just a segment in memory that allows for communication between the requesting application and the procedure or subroutine. The potential for problems comes into play when the requesting application does not carry out proper ***bounds checking*** to ensure that the inputted data is of an acceptable length. Look at the following C code to see how this could happen:

```
#include <stdio.h>
      int main(int argc, char **argv) {
            char buf1 [5] = "1111";
            char buf2 [7] = "222222";
            strcpy (buf2, "3333333333");
            printf ("%s\n", buf2);
printf ("%s\n", buf1);
      return 0;}
```

CAUTION You do not need to know C programming for the CISSP exam. We are digging deep into this topic because buffer overflows are so common and have caused grave security breaches over the years. For the CISSP exam, you just need to understand the overall concept of a buffer overflow.

Here we are setting up a buffer (buf1) to hold four characters and a NULL value, and second buffer (buf2) to hold six characters and a NULL value. (The NULL values indicate the buffer's end place in memory.) If we viewed these buffers, we would see the following:

```
Buf2
\0 2 2 2 2 2 2
Buf1
\0 1 1 1 1
```

The application then accepts ten 3s into buf2, which can only hold six characters. So the six variables in buf2 are filled and then the four variables in buf1 are filled, overwriting the original contents of buf1. This took place because the `strcpy` command did not make sure that the buffer was large enough to hold that many values. So now if we looked at the two buffers, we would see the following:

```
Buf2
\0 3 3 3 3 3 3
Buf1
\0 3 3 3 3
```

But what gets even more interesting is when the actual return pointer is written over, as shown in Figure 5-22. In a carefully crafted buffer overflow attack, the stack is filled properly so that the return pointer can be overwritten and control is given to the malicious instructions that have been loaded onto the stack instead of back to the requesting application. This allows the malicious instructions to be executed in the security context of the requesting application. If this application is running in a privileged mode, then the attacker has more permissions and rights to carry out more damage.

The attacker must know the size of the buffer to overwrite and must know the addresses that have been assigned to the stack. Without knowing these addresses, she could not lay down a new return pointer to her malicious code. The attacker must also write this dangerous payload to be small enough so that it can be passed as input from one procedure to the next.

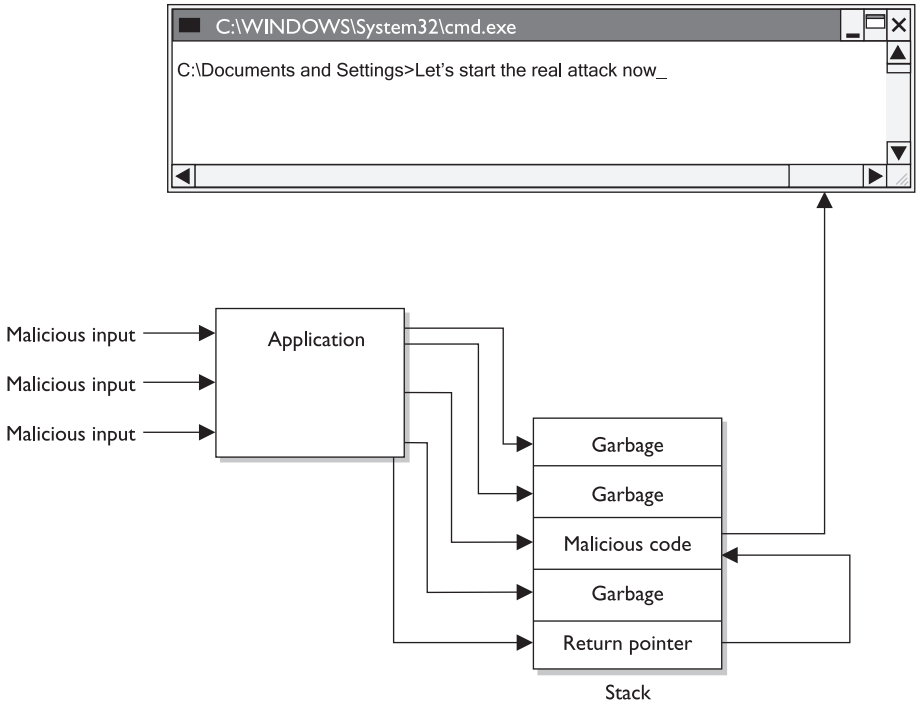


Figure 5-22 Illustration of a buffer overflow attack

Windows' core is written in the C language and has layers and layers of object-oriented code on top of it. When a procedure needs to call upon the operating system to carry out some type of task, it calls upon a system service via an API call. The API works like a doorway to the operating system's functionality.

The C language is susceptible to buffer overflow attacks because it allows for direct pointer manipulations to take place. Specific commands can provide access to low-level memory addresses without carrying out bounds checking. The C functions that do perform the necessary boundary checking include `sprintf( )`, `strcat( )`, `strcpy( )`, and `vsprintf( )`.

An operating system has to be written to work with specific CPU architectures. These architectures dictate system memory addressing, protection mechanisms, and modes of execution, and work with specific instruction sets. This means that a buffer overflow attack that works on an Intel chip will not necessarily work on a Motorola or a SPARC processor. These different processors set up the memory address of the stacks differently, so the attacker may have to craft different buffer overflow code for different platforms. This is usually not an obstacle since most of the time the code is already written and available via different hacking web sites.

Countermeasures

Buffer overflows are in the source code of various applications and operating systems. They have been around since programmers started developing software. This means

that it is very difficult for a user to identify and fix them. When a buffer overflow is identified, the vendor usually sends out a patch. So keeping systems current on updates, hotfixes, and patches is usually the best countermeasure. There are also some products that are installed on systems that can watch for input values that could result in buffer overflows. But the best countermeasure is proper programming. This means to use bounds checking. If an input value is only supposed to be nine characters, then the application should only accept nine characters and no more. Some languages are more susceptible to buffer overflows than others. So programmers should understand these issues, use the right languages for the right purposes, and carry out code review to identify buffer overflow vulnerabilities.

Summary

The architecture of a computer system is very important and comprises many topics. The system has to ensure 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, model, and architecture 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 of 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

- Two systems 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 two 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-numbered rings 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 supervisor 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 drive, floppy drive, tape backup, or a jump drive.
- Virtual storage combines RAM and secondary storage so that 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 usually provide.
- Not all system components fall under the trusted computing base (TCB), which includes only those system components 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 provide 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 outside it.
- 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 be invoked 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, encapsulation of objects, time multiplexing of shared resources, naming distinctions, and virtual mapping.
- A security policy is a set of rules that dictates 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 security at different layers of their architectures, which is called layering. This separates the processes and provides more protection for them individually.
- Data hiding occurs when processes work at different layers and have layers of access control between them. Processes need to know how to communicate only with each other's interfaces.
- 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.
- A closed system is often proprietary to the manufacturer or vendor, whereas the open system allows for more interoperability.
- The Bell-LaPadula model deals only with confidentiality, and the Biba and Clark-Wilson models deal only 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 model provides an upper bound and a lower bound of authorized access for subjects.
- 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 read and write to objects at its own security level.
- 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. This model also illustrates how to provide separation of duties, functionality, and auditing tasks within software.
- 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 Orange Book, also called Trusted Computer System Evaluation Criteria (TCSEC), was developed to evaluate systems built to be used mainly by the military. Its use was expanded to evaluate other types of products.
- In the Orange Book, D classification means that a system provides minimal protection 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 the B division deals with mandatory protection (security labels).
- In the Orange Book, the A classification means that the system's design and level of protection are verifiable and provide 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 security labels for all subjects and devices, the existence of a trusted path, routine covert channel analysis, and provision of separate administrator functionality.
- 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.
- ITSEC evaluates the assurance and functionality of a system's protection mechanisms separately, whereas 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 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 or product and its security components. Accreditation is management's formal approval and acceptance of the security provided by a system.
- 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 that another process can read it.
- A maintenance hook is developed to let a programmer into the application quickly for maintenance. 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.
- TOC/TOU stands for time-of-check/time-of-use. This is a class of asynchronous attacks.
- 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.

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 bounds 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.** Compromise of 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.** Temporal isolation
   - **B.** Maintenance hook
   - **C.** Race conditions
   - **D.** Process multiplexing
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.** Difficult to modify without physical access
   - **B.** Requires a smaller memory segment
   - **C.** Does not need to enforce the security policy
   - **D.** 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 security kernel?
   - **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 privileged 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 is the purpose of base and limit registers?

A. Countermeasure buffer overflows

B. Time sharing of system resources, mainly the CPU

C. Process isolation

D. TCB enforcement

13. A guard is commonly used with a classified system. What is the main purpose of implementing and using a guard?

A. Ensure that less trusted systems only receive acknowledgements and not messages

B. Ensure proper information flow

C. Ensure that less trusted and more trusted systems have open architectures and interoperability

D. Allow multilevel and dedicated mode systems to communicate

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 processes 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 concepts of a security state and mode of access, and to outline rules of access?
    A. Biba
    B. Bell-LaPadula
    C. Clark-Wilson
    D. State machine
25. Which of the following is a true statement pertaining to memory addressing?
    A. The CPU uses absolute addresses. Applications use logical addresses. Relative addresses are based on a known address and an offset value.
    B. The CPU uses logical addresses. Applications use absolute addresses. Relative addresses are based on a known address and an offset value.
    C. The CPU uses absolute addresses. Applications use relative addresses. Logical addresses are based on a known address and an offset value.
    D. The CPU uses absolute addresses. Applications use logical addresses. Absolute addresses are based on a known address and an offset value.

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 that 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. This is true of media (hard drives) and memory segments.
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 hooks 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 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 or through logical means.
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 security kernel is the software component that enforces access control for the operating system. 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 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.** The CPU has base and limit registers that hold the starting and ending memory addresses that a process is allowed to work within. This ensures that the process is isolated from other processes in that it cannot interact with another process's memory segment.
13. **B.** The guard accepts requests from the less trusted entity, reviews the request to make sure it is allowed, and then submits the request on behalf of the less trusted system. The goal is to ensure that information does not flow from a high security level to a low security level in an unauthorized manner.
14. **D.** The TCB contains and controls all protection mechanisms within the system, whether they are software, hardware, or firmware.
15. **C.** The security perimeter is a boundary between items that are within the TCB and items that are outside 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 concepts of a security state and mode of access and to outline 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.
18. **A.** It is difficult to add useful and effective security at the end of developing a product or to add 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 make 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 criteria. 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. **A.** The physical memory addresses that the CPU uses are called absolute addresses. The indexed memory addresses that software uses are referred to as logical addresses. And relative addresses are based on a known address with an offset value applied.