Chapter 9
Security Vulnerabilities, Threats, and Countermeasures

THE CISSP TOPICS COVERED IN THIS CHAPTER INCLUDE:

- ✓ **Domain 3.0: Security Architecture and Engineering**
  - 3.1 Research, implement and manage engineering processes using secure design principles
    - 3.1.10 Shared responsibility
  - 3.5 Assess and mitigate the vulnerabilities of security architectures, designs, and solution elements
    - 3.5.1 Client-based systems
    - 3.5.2 Server-based systems
    - 3.5.5 Industrial control systems (ICS)
    - 3.5.7 Distributed systems
    - 3.5.8 Internet of Things (IoT)
    - 3.5.9 Microservices (e.g., application programming interface (API))
    - 3.5.10 Containerization
    - 3.5.12 Embedded systems
    - 3.5.13 High-Performance Computing systems
    - 3.5.14 Edge computing systems
    - 3.5.15 Virtualized systems

Security professionals must pay careful attention to the IT system and ensure that their higher-level protective controls are not built on a shaky foundation. After all, the most secure firewall configuration in the world won't do much good if the underlying system has a fundamental security flaw that allows malicious individuals to bypass the firewall altogether.

In this chapter, we'll cover those underlying security concerns by surveying a field known as *computer architecture*: the physical design of computers from various components.

The Security Architecture and Engineering domain addresses various concerns and issues, including secure design elements, security architecture, vulnerabilities, threats, and associated countermeasures. Additional elements of this domain are discussed in various chapters:

- Chapter 1, “Security Governance Through Principles and Policies”
- Chapter 6, “Cryptography and Symmetric Key Algorithms”
- Chapter 7, “PKI and Cryptographic Applications”
- Chapter 8, “Principles of Security Models, Design, and Capabilities”
- Chapter 9, “Security Vulnerabilities, Threats, and Countermeasures”
- Chapter 10, “Physical Security Requirements”
- Chapter 14, “Controlling and Monitoring Access”
- Chapter 16, “Managing Security Operations”
- Chapter 20, "Software Development Security"
- Chapter 21, “Malicious Code and Application Attacks”

Please be sure to review all of these chapters to have a complete perspective on the topics of this domain.

Shared Responsibility

Shared responsibility is the security design principle that indicates that organizations do not operate in isolation. Instead, they are intertwined with the world in numerous ways. We all use the same basic technology, we follow the same communication protocol specifications, we use the same Internet, we use common foundations of operating systems and programming languages, and most of our IT/IS is implemented using *off-the-shelf solutions* (whether commercial or open source). Thus, we are automatically integrated with the rest of the world and share the responsibility of establishing and maintaining security.

We must realize this shared responsibility and take our role in this situation seriously. Here are several aspects of this concept to ponder:

- Everyone in an organization has some level of security responsibility. It is the job of the CISO and security team to establish security and maintain it. It is the job of the regular employees to perform their tasks within the confines of security. It is the job of the auditor to monitor the environment for violations.
- Organizations are responsible to their stakeholders to make good security decisions to sustain the organization. Otherwise, the needs of the stakeholders may be violated.
- When working with third parties, especially with cloud providers, each entity needs to understand their portion of the shared responsibility of performing work operations and maintaining security. This is often referenced as the cloud-shared responsibility model, which is discussed further in Chapter 16.
- As we become aware of new vulnerabilities and threats, we should consider it our responsibility (if not our duty) to responsibly disclose that information to the proper vendor or to an information sharing center (also known as a threat intelligence source or service).

Automated indicator sharing (AIS) is an initiative by the Department of Homeland Security (DHS) to facilitate the open and free exchange of indicators of compromise (IoCs) and other cyberthreat information between the U.S. federal government and the private sector in an automated and timely manner (described as "machine speed"). An *indicator* is an observable along with a hypothesis about a threat. An *observable* is an identified fact of occurrence, such as the presence of a malicious file, usually accompanied by a hash.

AIS uses *Structured Threat Information eXpression (STIX)* and *Trusted Automated eXchange of Intelligence Information (TAXII)* to share threat indicators. STIX is a standardized language expressing structured information about cyberthreats and a common framework for organizations to share and analyze threat intelligence. TAXII defines protocols and services for automated sharing of structured threat information.

AIS is managed by the National Cybersecurity and Communications Integration Center (NCCIC). For more information on the AIS program, please visit us-cert.gov/ais.

Because we participate in shared responsibility, we must research, implement, and manage engineering processes using secure design principles.

Data Localization and Data Sovereignty

Data localization refers to storing and processing data within a specific country or region's physical borders or geographical boundaries. This concept is often driven by regulatory requirements or government policies that mandate certain data, especially sensitive or personal information, to be kept within the jurisdiction's borders where it was generated or where the data subject resides. Data localization aims to exert greater control

over data privacy, security, and compliance with local laws. It can involve restrictions on cross-border data transfer and may influence how businesses structure their data storage and processing infrastructure to adhere to these regulations.

Data sovereignty refers to the concept that digital data is subject to the laws and regulations of the country or region in which it is located or originates. It emphasizes that governments have authority over the data collected within their jurisdiction, and organizations must comply with local data protection and privacy laws. Data sovereignty is closely related to concerns about privacy, security, and compliance, and it often influences decisions regarding where data is stored, processed, and managed to align with legal and regulatory requirements in a given geographical area.

While data localization and data sovereignty are related concepts, they have distinct focuses and implications:

- *Focus:*
  - *Data localization:* Primarily centers on the physical location of data storage and processing. It mandates that certain types of data, especially sensitive or personal information, must be kept within specific geographical borders.
  - *Data sovereignty:* Encompasses a broader set of principles related to the authority and control that a country or region asserts over the data generated within its jurisdiction. This includes where the data is stored and concerns about legal jurisdiction and compliance with local data protection laws.
- *Scope:*
  - *Data localization:* Primarily addresses the geographical aspect of data storage and processing. It often involves regulations specifying that data should reside on servers physically located within the borders of a specific country or region.

    - *Data sovereignty:* Encompasses a broader range of concerns, including legal jurisdiction, regulatory compliance, and the overarching authority that a government has over data collected within its boundaries. It extends beyond mere storage location to include control and governance aspects.
- *Drivers:*
    - *Data localization:* Often driven by specific regulations or laws that mandate the physical presence of data within a particular jurisdiction. This can be motivated by data privacy, security, or economic concerns.
    - *Data sovereignty:* Driven by legal, political, and cultural considerations. Governments may enact data sovereignty regulations to protect the privacy of their citizens, ensure compliance with local laws, and assert control over data handling within their borders.
- *Implications:*
    - *Data localization:* The primary implication is that organizations must establish data centers or use cloud services within the specified jurisdiction to comply with local regulations. This may impact the efficiency of cross-border data flow and increase operational costs.
    - *Data sovereignty:* Besides impacting data storage location, data sovereignty considerations may influence how organizations handle data governance, legal compliance, and interactions with third-party service providers. It can require a more comprehensive approach to data management and legal compliance.

While data localization is a subset of data sovereignty, focusing specifically on where data is stored and processed, data sovereignty encompasses a broader set of principles related to data governance, jurisdiction, and legal compliance within a specific geopolitical boundary. Both concepts, however, underscore the importance of aligning data practices with local regulations and legal frameworks.

Data portability refers to the ability of individuals to easily and securely move their personal data from one system, service, or application to another. It allows users to transfer their data between different platforms, promoting user control and facilitating competition among service providers. Data portability empowers individuals by allowing them to choose services based on their preferences while maintaining access and control over their personal information. This concept is often associated with data protection and privacy regulations emphasizing user rights and data control.

Assess and Mitigate the Vulnerabilities of Security Architectures, Designs, and Solution Elements

Computer architecture is an engineering discipline concerned with designing and constructing computing systems at a logical level. Technical mechanisms that can be implemented via computer architecture are the controls that system designers can build into their systems. These include layering (see Chapter 1, "Security Governance Through Principles and Policies"), abstraction (see Chapter 1), data hiding (see Chapter 1), trusted recovery (see Chapter 18, "Disaster Recovery Planning"), process isolation (later in this chapter), and hardware segmentation (later in this chapter).

The more complex a system, the less assurance it provides. More complexity means more areas for vulnerabilities exist, and more areas must be secured against threats. More vulnerabilities and more threats mean that the subsequent security the system provides is less trustworthy. See Chapter 8 for more on "keep it simple."

Hardware

The term *hardware* encompasses any tangible part of a computer that you can reach out and touch, from the keyboard and monitor to its CPU(s), storage media, and memory chips. Remember that although the physical portion of a storage device (such as a hard disk or flash memory) may be considered hardware, the contents of those devices—the collections of 0s and 1s that make up the software and data stored within them—may not.

Processor

The *central processing unit (CPU)*, generally called the *processor* or the *microprocessor*, is the computer's nerve center—it is the chip (or chips in a multiprocessor system) that governs all major operations and either directly performs or coordinates the complex symphony of calculations that allows a computer to perform its intended tasks. Surprisingly, the CPU can perform only a limited set of computational and logical operations, despite the complexity of the tasks it allows the overall computer system to perform. The operating system and compilers or interpreters are responsible for translating high-level programming languages into simple instructions that a CPU understands. This limited range of functionality is intentional—it allows a CPU to perform computational and logical operations at blazing speeds.

Execution Types

As computer processing power increased, users demanded more advanced features to enable these systems to process information

at greater rates and to manage multiple functions simultaneously:

At first blush, the terms *multitasking, multicore, multiprocessing, multiprogramming*, and *multithreading* may seem nearly identical. However, they describe very different ways of approaching the "doing two things at once" problem. We strongly advise you to review the distinctions between these terms until you feel comfortable with them.

Multitasking In computing, *multitasking* means handling two or more tasks simultaneously. In the past, most systems did not truly multitask because they relied on the OS to simulate multitasking by carefully structuring the sequence of commands sent to the CPU for execution (see multiprogramming). A single-core multitasking system is able to juggle more than one task or process at any given time. However, with that single-core CPU, it only executes a single process at any given moment. This is similar to juggling three balls, where your hands usually touch only one ball at any given instant, but the coordination of movements keeps all three balls moving.

Multicore Today, most CPUs are *multicore*. This means that the CPU is now a chip containing two, four, eight, dozens, or more independent execution cores that can operate simultaneously and/or independently. There are even some specialty chips with over 10,000 cores.

Multiprocessing In a *multiprocessing* environment, a multiprocessor system harnesses the power of more than one processor to complete the execution of a multithreaded application. See the section "Large-Scale Parallel Data Systems," later in this chapter.

Some multiprocessor systems may assign or dedicate a process or execution thread to a specific CPU (or core). This is called *affinity*.

Multiprogramming *Multiprogramming* is similar to multitasking. It involves the pseudo-simultaneous execution of two tasks on a single processor coordinated by the OS as a way to increase operational efficiency. For the most part, multiprogramming is a way to batch or serialize multiple processes so that when one process stops to wait on a peripheral, its state is saved, and the next process in line begins to process. The first program does not return to processing until all other processes in the batch have had their chance to execute, and they, in turn, stop for a peripheral. This methodology causes significant delays in completing a task for any single program. However, across all processes in the batch, the total time to complete all tasks is reduced.

Multithreading *Multithreading* permits multiple concurrent tasks to be performed within a single process. Unlike multitasking, where multiple tasks consist of multiple processes, multithreading enables multiple tasks to operate within a single process. A thread is a self-contained sequence of instructions that can execute in parallel with other threads that are part of the same parent process. Multithreading is often used in applications where frequent context switching between multiple active processes causes excessive overhead and reduces efficiency; switching between threads incurs far less overhead and is, therefore, more efficient.

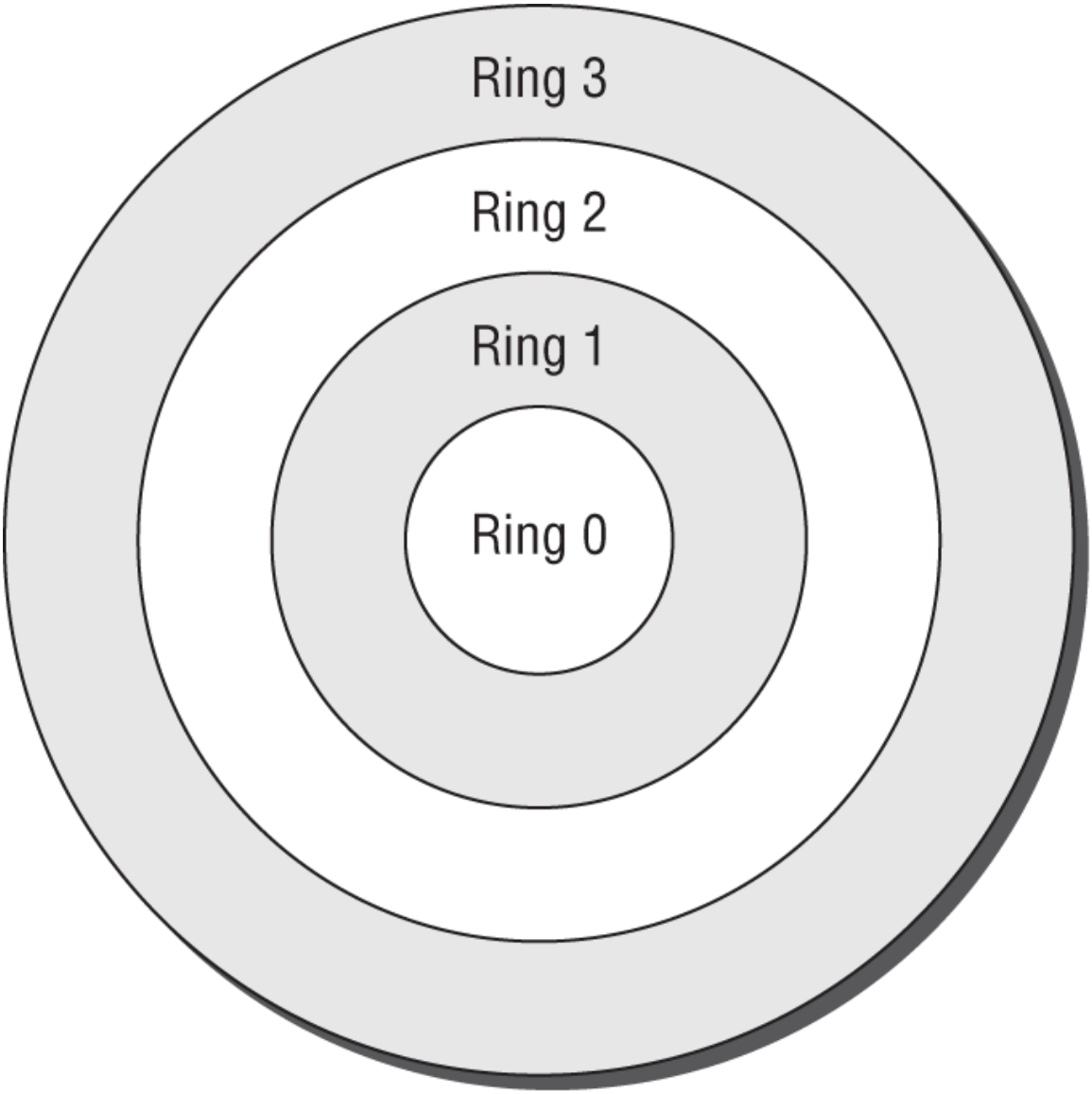
Protection Mechanisms

When a computer is running, it operates a runtime environment that represents the combination of the OS and whatever applications may be active. Within that runtime environment, it's

necessary to integrate security controls to protect the OS's integrity, manage which users are allowed to access specific data items, authorize or deny operations requested against such data, and so forth. How running computers implement and handle security at runtime may be broadly described as a collection of protection mechanisms, such as protection rings and operational states.

PROTECTION RINGS

From a security standpoint, *protection rings* organize code and components in an OS (as well as applications, utilities, or other code that runs under the OS's control) into concentric rings, as shown in Figure 9.1. The deeper inside the circle, the higher the privilege level associated with the code that occupies a specific ring. Though the original Multics implementation allowed up to seven rings (numbered 0 through 6), most modern OSs use a four-ring model (numbered 0 through 3).



Ring 0: OS Kernel/Memory (Resident Components)
Ring 1: Other OS Components
Ring 2: Drivers, Protocols, etc.
Ring 3: User-Level Programs and Applications

Rings 0–2 run in supervisory or privileged mode.
Ring 3 runs in user mode.

FIGURE 9.1 The four-layer protection ring model

As the innermost ring, 0 has the highest level of privilege and can basically access any resource, file, or memory location. The part of an OS that always remains resident in memory (so that it can run on demand at any time) is called the *kernel*. It occupies ring 0 and can preempt code running at any other ring. The remaining parts of the OS—those that come and go as various tasks are requested, operations are performed, processes are switched, and so forth—occupy ring 1. Ring 2 is also somewhat privileged in that it's where I/O drivers and system utilities reside; these are able to access peripheral devices, special files, and so forth that applications and other programs cannot themselves access directly. Those applications and programs occupy the outermost ring, ring 3.

The essence of the ring model lies in priority, privilege, and memory segmentation. Any process that wants to execute must get in line (a pending process queue). The process associated with the lowest ring number always runs before processes associated with higher-numbered rings. Processes in lower-numbered rings can access more resources and interact with the OS more directly than those in higher-numbered rings. Those processes that run in higher-numbered rings must generally ask a handler or driver in a lower-numbered ring for services they need (aka system call), sometimes called a *mediated-access model*. In practice, many modern OSs use only two rings or divisions: one for system-level access (rings 0 through 2), often called *kernel mode* or *privileged mode*, and one for user-level programs and applications (ring 3), often called *user mode*.

From a security standpoint, the ring model enables an OS to protect and insulate itself from users and applications. It also permits the enforcement of strict boundaries between highly privileged OS components (such as the kernel) and less privileged parts of the OS (such as other parts of the OS, plus drivers and utilities).

The ring that a process occupies determines its access level to system resources. Processes may access objects directly only if they reside within their own ring or within some outside ring. Before any such request can be honored, the called ring must

check to make sure that the calling process has the proper credentials and authorization to access the data and to perform the operation(s) involved in satisfying the request.

Rings Compared to Levels

Many of the protecting ring concept's features also apply to a multilayer or multilevel system. The top of a layered or multilevel system is the same as the center ring (i.e., ring 0) of a protection ring scheme. Likewise, the bottom of a layered or multilevel system is the same as the outer ring of a protection ring scheme. Levels, layers, domains, and rings are similar in terms of protection and access concepts.

PROCESS STATES

Process states or *operating states* are various forms of execution in which a process may run. Where the OS is concerned, it can be in one of two modes at any given moment: operating in a privileged, all-access mode known as *supervisor state* (aka Kernel mode) or operating in what's called the *problem state* associated with user mode, where privileges are low and all access requests must be checked against credentials for authorization before they are granted or denied. The latter is called the problem state not because problems are guaranteed to occur but because the unprivileged nature of user access means that problems can occur and the system must take appropriate measures to protect security, integrity, and confidentiality.

Processes line up for execution in an OS in a processing queue, where they will be scheduled to run as a processor becomes available. Most OSs allow processes to consume processor time only in fixed increments or chunks; if a process consumes its entire processing time (called a *time slice*) without completing, it returns to the processing queue for another time slice the next time its turn comes around. Also, the process scheduler usually selects the highest-priority process for execution, so reaching the front of the line doesn't always guarantee access to the CPU

(because a process may be preempted at the last instant by another process with higher priority).

According to whether a process is running, it can operate in one of several states:

Ready In the *ready state*, a process is ready to resume or begin processing as soon as it is scheduled for execution. If the CPU is available when the process reaches this state, it will transition directly into the running state; otherwise, it sits in the ready state until its turn comes up.

Running The *running state* or *problem state* is when a process executes on the CPU and keeps going until it finishes, its time slice expires, or it is blocked for some reason (usually because it has generated an interrupt for I/O). If the time slice ends and the process isn't completed, it returns to the ready state; if the process is paused while waiting for I/O, it goes into the waiting state.

Waiting The *waiting state* is when a process is ready for continued execution but is waiting for I/O to be serviced before it can continue processing. Once I/O is complete, then the process typically returns to the ready state, where it waits in the process queue to be assigned time again on the CPU for further processing.

Supervisory The *supervisory state* is used when the process must perform an action that requires privileges that are greater than the problem state's set of privileges, including modifying system configuration, installing device drivers, or modifying security settings. Basically, any function not occurring in the user mode (ring 3) or problem state takes place in the supervisory mode. This state is not shown in Figure 9.2, but it effectively replaces the running state when a process is run with higher-level privileges.

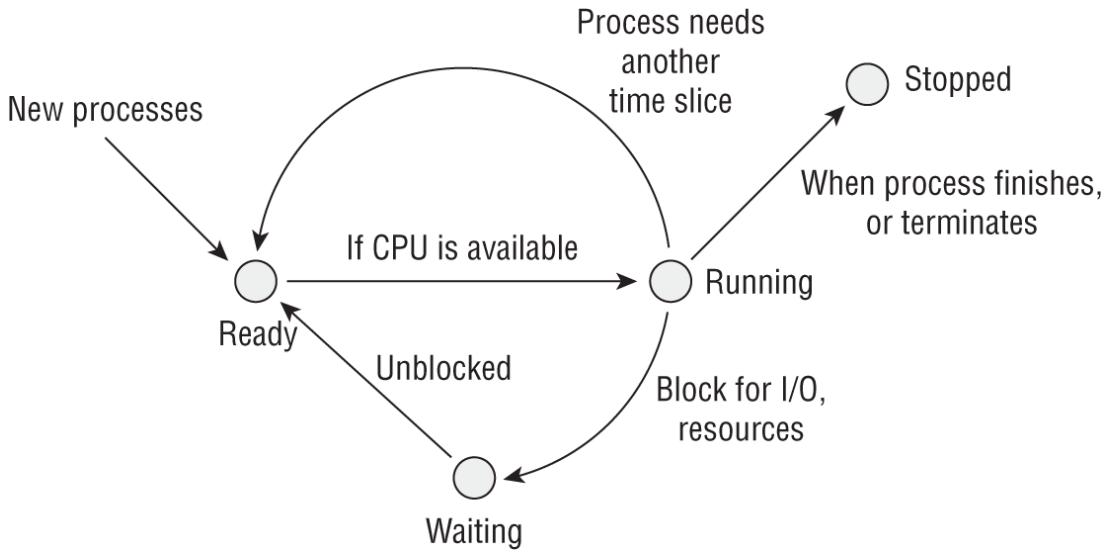


FIGURE 9.2 The life cycle of an executed process

Stopped When a process finishes or must be terminated (because an error occurs, a required resource is not available, or a resource request can't be met), it goes into a *stopped state*. At this point, the OS can recover all memory and other resources allocated to the process and reuse them for other processes as needed.

Figure 9.2 shows a diagram of how these various states relate to one another. New processes always transition into the ready state. When the OS decides which process to run next, it checks the ready queue and takes the highest-priority job that's ready to run.

Memory

The second major hardware component of a system is *memory*, the storage bank for information that the computer needs to keep readily available. There are many different kinds of memory, each suitable for different purposes, and we'll take a look at each in the sections that follow.

Read-Only Memory

Read-only memory (ROM) works like the name implies—it's memory the system can read but can't change (no writing allowed). The contents of a standard ROM chip are burned in at the factory, and the end user simply cannot alter it. ROM chips

often contain “bootstrap” information that computers use to start up prior to loading an OS from disk. This includes the *power-on self-test (POST)* series of diagnostics that run each time you boot a PC.

ROM's primary advantage is that it can't be modified. This attribute makes ROM extremely desirable for orchestrating a computer's innermost workings.

There is a type of ROM that may be altered to some extent. It is known as programmable read-only memory (PROM), and its several subtypes:

Programmable Read-Only Memory (PROM) A basic *programmable read-only memory (PROM)* chip is similar to a ROM chip in functionality, but with one exception. During manufacturing, a PROM chip's contents aren't “burned in” at the factory as with standard ROM chips. Instead, a PROM incorporates special functionality that allows an end user to burn in the chip's contents later. Once data is written to a PROM chip, no further changes are possible.

Erasable Programmable Read-Only Memory (EPROM) EPROM combines the relatively high cost of PROM chips and software developers' inevitable desire to tinker with their code once it's written. You then have the rationale for the development of *erasable PROM (EPROM)*. There are two main subcategories of EPROM: UVEPROM and EEPROM (see the next item). *Ultraviolet EPROMs (UVEPROMs)* can be erased with UV light. These chips have a small window that, when illuminated with a special ultraviolet light, causes the contents of the chip to be erased. After this process is complete, end users can burn new information into the UVEPROM as if it had never been programmed before.

Electronically Erasable Programmable Read-Only Memory (EEPROM) A more flexible, friendly alternative to UVEPROM is *electronically erasable PROM (EEPROM)*, which uses electric voltages delivered to the pins of the chip to force erasure.

Flash Memory *Flash memory* is a derivative concept from EEPROM. It is a nonvolatile form of storage media that can be electronically erased and rewritten. The primary difference between EEPROM and flash memory is that EEPROM can be erased and written at the byte level, whereas flash memory can be erased and written in blocks or pages. The most common type of flash memory is NAND flash. It is widely used in memory cards, thumb drives, mobile devices, and SSDs (solid-state drives).

Random Access Memory

Random access memory (RAM) is readable and writable memory that contains information a computer uses during processing. RAM retains its contents only when power is continuously supplied to it. Unlike with ROM, when a computer is powered off, all data stored in RAM disappears. For this reason, RAM is useful only for temporary storage. Critical data should never be stored solely in RAM; a backup copy should always be kept on another storage device to prevent its disappearance in the event of a sudden loss of electrical power. The following are types of RAM:

Real Memory *Real memory* (also known as *main memory* or *primary memory*) is typically the largest RAM storage resource available to a computer. It is normally composed of a number of dynamic RAM chips and, therefore, must be refreshed by the CPU periodically (see the sidebar "Dynamic vs. Static RAM" for more information on this subject).

Cache RAM Computer systems contain many caches that improve performance by taking data from slower devices and temporarily storing it in faster devices when repeated use is likely; this is *cache RAM*. The processor normally contains an onboard cache of extremely fast memory used to hold data on which it will operate. This can be referred to as L1, L2, L3, and even L4 cache (with the L being short for level). Many modern CPUs include up to three levels of on-chip cache, with some caches (usually L1 and/or L2) dedicated to a single processor core, whereas L3 may be a shared cache

between cores. Some CPUs can involve L4 cache, which may be located on the mainboard/motherboard or on the GPU (graphics processing unit). Likewise, real memory often contains a cache of information pulled or read from a storage device.

Many peripherals also include onboard caches to reduce the storage burden they place on the CPU and OS. Many storage devices, such as hard disk drives (HDDs), solid-state drives (SSDs), and some thumb drives, contain caches to assist with improving read and write speed. However, these caches must be flushed to the permanent or secondary storage area before disconnection or power loss to avoid data loss of cache resident data.

Dynamic vs. Static RAM

There are two main types of RAM: dynamic RAM and static RAM. Most computers contain a combination of both types and use them for different purposes.

To store data, dynamic RAM uses a series of capacitors, tiny electrical devices that hold a charge. These capacitors either hold a charge (representing a 1 bit in memory) or do not hold a charge (representing a 0 bit). However, because capacitors naturally lose their charges over time, the CPU must spend time refreshing the contents of dynamic RAM to ensure that 1 bits don't unintentionally change to 0 bits, thereby altering memory contents.

Static RAM uses more sophisticated technology—a logical device known as a *flip-flop*, which to all intents and purposes, is simply an on/off switch that must be moved from one position to another to change a 0 to 1, or vice versa. More importantly, static memory maintains its contents unaltered as long as power is supplied and imposes no CPU overhead for periodic refresh operations.

Dynamic RAM is cheaper than static RAM because capacitors are cheaper than flip-flops. However, static RAM runs much faster than dynamic RAM. This creates a trade-off for system designers, who combine static and dynamic RAM modules to strike the right balance of cost versus performance.

Registers

The CPU also includes a limited amount of onboard memory, known as *registers*, that provide it with directly accessible memory locations that the brain of the CPU, the *arithmetic-logical unit (ALU)*, uses when performing calculations or processing instructions. The size and number of registers vary, but typical CPUs have 8 to 32 registers and are often either 32 or 64 bits in size. In fact, any data that the ALU is to manipulate must be loaded into a register unless it is directly supplied as part

of the instruction. The main advantage of this type of memory is that it is part of the ALU itself and, therefore, operates in lockstep with the CPU at typical CPU speeds.

Memory Addressing

When using memory resources, the processor must have some means of referring to various locations in memory. The solution to this problem is known as *memory addressing*, and several different addressing schemes are used in various circumstances. The following are five of the most common addressing schemes:

Register Addressing As you learned in the previous section, registers are small memory locations directly in the CPU. When the CPU needs information from one of its registers to complete an operation, it uses a *register address* (for example, "register 1") to access its contents.

Immediate Addressing *Immediate addressing* is not a memory addressing scheme per se but rather a way of referring to data that is supplied to the CPU as part of an instruction. For example, the CPU might process the command "Add 2 to the value in register 1." This command uses two addressing schemes. The first is immediate addressing—the CPU is being told to add the value 2 and does not need to retrieve that value from a memory location —it's supplied as part of the command. The second is register addressing; it's instructed to retrieve the value from register 1.

Direct Addressing In *direct addressing*, the CPU is provided with an actual address of the memory location to access. The address must be located on the same memory page as the instruction being executed. Direct addressing is more flexible than immediate addressing since the contents of the memory location can be changed more readily than reprogramming the immediate addressing's hard-coded data.

Indirect Addressing *Indirect addressing* uses a scheme similar to direct addressing. However, the memory address supplied to the CPU as part of the instruction doesn't contain

the actual value that the CPU is to use as an operand. Instead, the memory address contains another memory address. The CPU reads the indirect address to learn the address where the desired data resides and then retrieves the actual operand from that address.

Base+Offset Addressing *Base+offset addressing* uses a value stored in one of the CPU's registers or pointers as the base location to begin counting. The CPU then adds the offset supplied with the instruction to that base address and retrieves the operand from that computed memory location.

A *pointer* is a basic element or object in many programming languages that is used to store a memory address. Basically, a pointer holds the address of something stored in memory so that when the program reads the pointer, it points to the location of the data actually needed by the application. Effectively, a pointer references a memory location. The act of accessing a pointer to read that memory location is known as *dereferencing*. Pointers can store the memory address used in direct, indirect, or base addressing. Another potential issue is a *race condition*, which occurs when a system or device tries to perform two or more operations at the same time. This can cause null pointer errors in which an application dereferences a pointer that it expects to be valid but is really null (or corrupted), resulting in a system crash.

Secondary Memory

Secondary memory is a term commonly used to refer to magnetic, optical, or flash-based media or other storage devices that contain data not immediately available to the CPU. For the CPU to access secondary memory data, the OS must first read the data and store it in real memory.

Virtual memory is a special type of secondary memory that is used to expand the addressable space of real memory. The most common type of virtual memory is the *pagefile* or *swapfile* that most OSs manage as part of their memory management functions. This specially formatted file contains previously stored RAM data but not recently used in real memory. When the OS needs to access addresses stored in the pagefile, it checks to see whether the page is memory-resident (in which case it can access it immediately) or whether it has been swapped to disk, in which case it reads the data from disk back into real memory (this process is called *paging*).

Virtual memory's primary drawback is that the paging operations that occur when data is exchanged between primary and secondary memory are relatively slow. The need for virtual memory is reduced with larger banks of actual physical RAM, and the performance hit of virtual memory can be reduced by using flash media (NVMe [nonvolatile memory express] SSDs, SSDs, and USB flash drives) to host the virtual memory paging file.

Data Storage Devices

Data storage devices are used to store information that may be used by a computer at any time after it's written.

Primary vs. Secondary

Primary memory, also known as *primary storage*, is the RAM that a computer uses to keep necessary information readily available to the CPU while the computer is running. Secondary memory (or *secondary storage*) includes all the familiar long-term storage devices you use often. Secondary storage consists of magnetic, optical, and flash media such as HDDs, SSDs, flash drives, magnetic tapes, CDs, DVDs, and flash memory cards.

Volatile vs. Nonvolatile

The volatility of a storage device is simply a measure of how likely it is to lose its data when power is turned off or cycled. Devices designed to retain their data (such as magnetic media, flash media, ROMs, and optical media) are classified as

nonvolatile, whereas devices such as static or dynamic RAM modules, which lose their data when power is removed, are classified as *volatile*.

Random vs. Sequential

Storage devices may be accessed in one of two fashions. *Random access storage* devices allow an OS to read (and sometimes write) immediately from any point within the device by using some type of addressing system. Almost all primary storage devices are random access devices. You can use a memory address to access information stored at any point within a RAM chip without reading the data that is physically stored before it. Most secondary storage devices are also random access.

Sequential storage devices, on the other hand, do not provide this flexibility. They require you to read (or speed past) all the data physically stored before the desired location. A common example of a sequential storage device is a magnetic tape drive.

Memory Security Issues

Memory stores and processes your data—some of which may be extremely sensitive. Any memory devices that may retain sensitive data should be purged before they are allowed to leave your organization for any reason. This is especially true for secondary memory and ROM/PROM/EPROM/EEPROM devices designed to retain data even after the power is turned off.

However, memory data retention issues are not limited to secondary memory (i.e., storage devices). It is technically possible that the electrical components used in volatile primary memory could retain some of their charge for a limited period of time after power is turned off. A technically sophisticated individual could theoretically retrieve portions of the data stored on such devices.

A memory compromise called the cold boot attack freezes memory chips to delay the decay of resident data when the system is turned off or the RAM is pulled out of the motherboard (see en.wikipedia.org/wiki/Cold_boot_attack). There are even

attacks and tools that focus on memory image dumps or system crash dumps to extract encryption keys (see www.passware.com/kit-forensic).

Storage Media Security

There are several concerns when it comes to the security of secondary storage devices:

- Data may remain on secondary storage devices even after it has been erased. This condition is known as *data remanence*. Utilities are available that can retrieve files from a disk even after they have been deleted or reformatted. If you truly want to remove data from a secondary storage device, you must use a specialized utility designed to overwrite all traces of data on the device (commonly called *sanitizing*) or damage or destroy it beyond possible repair. (See Chapter 5, "Protecting Security of Assets.")

  SSDs are large-capacity flash memory secondary storage devices. Many SSDs include additional reserved memory blocks, which can be used in place of bad blocks. As blocks are written to or erased, they deteriorate at a predictable failure rate. Many SSD manufacturers counter this failure rate with two main techniques: reserved blocks and wear leveling. When a block stops working reliably, it is marked as bad, and a reserve block is then used in its place. This is similar to an HDD's bad sectors. Wear leveling attempts to perform write and erase events evenly across the entire drive's capacity of blocks to maximize use lifetime.

- A traditional zeroization wipe is less effective for SSDs because bad blocks/cells are likely not overwritten.
- Secondary storage devices are also prone to theft. Economic loss is not the major factor (after all, how much does a backup tape or a hard drive cost?), but the loss of

confidential information poses the great risks. For this reason, it is important to use full-disk encryption to reduce the risk of an unauthorized entity gaining access to your data. Many HDDs, SSDs, and flash devices offer on-device native encryption.

- Removable media pose a significant information disclosure risk, so securing them often requires encryption technologies.

Emanation Security

Many electrical devices emanate electrical signals or radiation that can be intercepted and may contain confidential, sensitive, or private data. Obvious examples of emanation devices are wireless networking equipment and mobile phones, but many other devices are vulnerable to emanation interception that you might not expect, including monitors, network cables, modems, and internal or external media drives (hard drives, USB thumb drives, CDs, and so on). With the right equipment, adversaries can intercept electromagnetic or radio frequency signals (collectively known as *emanations*) from these devices and interpret them to extract confidential data.

There are many valid uses of emanations, such as Wi-Fi, Bluetooth, GPS, and mobile phone signals.

The types of countermeasures and safeguards used to protect against emanation attacks are derived from TEMPEST. *TEMPEST* (Telecommunications Electronics Material Protected from Emanating Spurious Transmissions) was originally a government research study aimed at protecting electronic equipment. It refers to a set of standards and guidelines used to minimize the unintentional electromagnetic signals that could be emitted by electronic equipment and potentially intercepted, leading to unauthorized access to sensitive information. The term TEMPEST has become less common, and the corresponding

security discipline is now commonly known as EMSEC. Emission Security (EMSEC) involves implementing various measures to prevent unauthorized individuals from obtaining valuable information that could be derived through intercepting and analyzing compromising emanations from cryptographic equipment, automated information systems (AISs), and telecommunications systems. It has since expanded to a general study of monitoring emanations and preventing their interception.

Simply because of the kinds of electronic components from which they're built, many computer hardware devices emit electromagnetic (EM) radiation during normal operation. The process of communicating with other machines or peripheral equipment creates emanations that can be intercepted. These emanation leaks can cause serious security issues but are generally easy to address.

TEMPEST/EMSEC attacks read the electronic emanations that devices produce (known as *Van Eck radiation*) from a distance (this process is known as *Van Eck phreaking*). TEMPEST/EMSEC eavesdropping or Van Eck phreaking countermeasures include the following:

Faraday Cage A *Faraday cage* is a box, mobile room, or entire building designed with an external metal skin, often a wire mesh that fully surrounds an area on all sides. This metal skin acts as an EM-absorbing capacitor that prevents electromagnetic signals (emanations) from exiting or entering the area that the cage encloses.

White Noise

White noise simply means broadcasting false traffic to mask and hide the presence of real emanations. White noise can consist of a real signal from another source that is not confidential, a constant signal at a specific frequency, a randomly variable signal, or even a jam signal that causes interception equipment to fail. Although this is similar to jamming devices, the purpose is to convolute the signal only

for the eavesdropper, not the authorized user, rather than stopping even valid uses of emanations.

White noise describes any random sound, signal, or process that can drown out meaningful information. This can vary from audible frequencies to inaudible electronic transmissions, and it may even involve the deliberate act of creating line or traffic noise to disguise origins or disrupt listening devices.

Control Zone A third type of TEMPEST/EMSEC countermeasure, a *control zone*, is simply implementing a Faraday cage and white noise generation to protect a specific area in an environment; the rest of the environment is unaffected. A control zone can be a room, a floor, or an entire building.

In addition to the TEMPEST/EMSEC derived countermeasure technologies concepts, shielding, access control, and antenna management can be helpful against emanation eavesdropping. *Shielding* of cables (networking and otherwise) may be sufficient to reduce or block emanation access. This may be an element included in the manufacture of equipment, such as shielded twisted pair (STP) cabling, or may be accomplished by using shielding conduits or just replacing copper network cables with fiber-optic cables.

Input and Output Devices

Input and output devices can present security risks to a system. Security professionals should be aware of these risks and ensure that appropriate controls are in place to mitigate them.

Monitors

TEMPEST/EMSEC eavesdropping technology can compromise the security of data displayed on a monitor. This may be accomplished by eavesdropping on the video cable or the monitor

itself. It is arguable that the biggest risk with any monitor is still shoulder surfing or cameras. Don't forget shoulder surfing is a concern for desktop displays, laptop displays, tablets, and mobile phones.

Printers

Printers also represent a security risk that is easy to overlook. Depending on the physical security controls used at your organization, it may be much easier to walk out with sensitive information in printed form than to walk out with a flash drive or magnetic media. If printers are shared, users may forget to retrieve their sensitive printouts, leaving them vulnerable to prying eyes. Many modern printers also store data locally, often on a hard drive, and some retain copies of printouts indefinitely. Printers are usually exposed on the network for convenient access and are often not designed to be secure systems.

Concerns should also apply to *multifunction printers (MFPs)*, especially those that include fax capabilities and that are network-attached (whether wired or wireless). In 2018, researchers discovered that it is still possible to take control of a computer system over a *public switched telephone network (PSTN)* line using ancient *AT commands* supported by telephone modems and fax modems/machines. See the researcher's DEFCON 26 presentation PDF at media.defcon.org/DEF%20CON%2026/DEF%20CON%2026%20presentations/DEFCON-26-Yaniv-Balmas-What-The-FAX.pdf. If you don't always need fax capabilities, don't leave the telephone line plugged in. If you do need always-available fax capabilities, use a stand-alone fax machine.

Keyboards/Mice

Keyboards, mice, and similar input devices are not immune to security vulnerabilities. All of these devices are vulnerable to TEMPEST/EMSEC monitoring, whether wired or wireless. Also, keyboards are vulnerable to less sophisticated bugging. A simple device can be placed inside a keyboard or along its connection cable to intercept all the keystrokes that take place and transmit

them to a remote receiver using a radio signal. This has the same effect as TEMPEST/EMSEC monitoring but can be done with much less expensive gear. Additionally, if your keyboard and mouse are wireless, including Bluetooth, their radio signals can be intercepted.

POTS Telephone Modems

With the advent of ubiquitous broadband and wireless connectivity, telephone modems are becoming a scarce legacy computer component. If your organization is still using older equipment, there is a chance that a modem is part of the hardware configuration. Modems allow users to create uncontrolled access points into your network. If improperly configured, they can create extremely serious security vulnerabilities that allow an outsider to bypass all your perimeter protection mechanisms and directly access your network resources. Modems create an alternate egress channel that insiders can use to funnel data outside your organization. But keep in mind that these vulnerabilities can be exploited only if the modem is connected to an operational telephone landline.

The same risk of creating a security perimeter bypass exists when systems have both wired and wireless NICs. Systems should typically be restricted to using only one method/means of connection at a time. For example, if a cable is connected to the system's RJ45 jack, then the wireless interface should be deactivated. It may also be worth considering that for devices that exit and enter the premises a geofencing type system be used, where wireless connection devices are deactivated as the equipment enters the facility. See more on this in Chapter 11, “Secure Network Architecture and Components.”

You should seriously consider an outright ban on telephone modems in your organization's security policy unless you truly need them for business reasons. In those cases, security officials

should know the physical and logical locations of all modems on the network, ensure that they are correctly configured, and make certain that appropriate protective measures are in place to prevent their illegitimate use.

Firmware

Firmware (also known as *microcode*) is a term used to describe software that is stored in a ROM or an EEPROM chip. This type of software is changed infrequently (actually, never, if it's stored on a true ROM chip as opposed to an EEPROM or flash chip) and often drives the basic operation of a computing device.

Many hardware devices, such as printers and modems, need some limited set of instructions and processing power to complete their tasks while minimizing the burden placed on the OS itself. In many cases, these “mini” OSs are entirely contained in firmware chips onboard the devices they serve. Many devices commonly use firmware as their primary OS, including mobile devices, Internet of Things (IoT) equipment, edge computing devices, fog computing devices, and industrial control systems.

Basic input/output system (BIOS) is the legacy basic low-end firmware or software embedded in a motherboard's EEPROM or flash chip. The BIOS contains the OS-independent primitive instructions that a computer needs to start up and load the OS from disk. The BIOS identifies and initiates the basic system hardware components, such as the hard drive, optical drive, and video card, so that the bootstrapping process of loading an OS can begin. In most modern systems, the BIOS has been replaced by UEFI.

Unified Extensible Firmware Interface (UEFI) provides support for all of the same functions as BIOS with many improvements, such as support for larger hard drives (especially for booting), faster boot times, enhanced security features, and even the ability to use a mouse when making system changes (BIOS was limited to keyboard control only). UEFI also includes a CPU-independent architecture, a flexible pre-OS environment with networking support, measured boot, boot attestation (aka secure boot), and

backward and forward compatibility. It also runs CPU-independent drivers (for system components, drive controllers, and hard drives).

The process of updating the UEFI, BIOS, or firmware is known as *flashing*. If bad actors or malware can alter the UEFI, BIOS, or firmware of a system, they may be able to bypass security features or initiate otherwise prohibited activities. There have been a few examples of malicious code embedding itself into UEFI, BIOS, or firmware. There is also an attack known as *phlashing,* in which a malicious variation of official BIOS or firmware is installed that introduces remote control or other malicious features into a device.

Boot attestation or *secure boot* is a feature of UEFI that aims to protect the local OS by preventing the loading or installing of device drivers or an OS that is not signed by a preapproved digital certificate. Secure boot thus protects systems against a range of low-level or boot-level malware, such as certain rootkits and backdoors. Secure boot intends that only drivers and OSs that pass attestation (the verification and approval process accomplished through the validation of a digital signature) are allowed to be installed and loaded on the local system. Unfortunately, there are now forms of malware which can bypass or abuse UEFI security features on specific systems.

Measured boot is an optional feature of UEFI that takes a hash calculation of every element involved in the booting process. The hashes are performed by and stored in the Trusted Platform Module (TPM). If foul play is detected in regard to booting, the hashes of the most recent boot can be accessed and compared against known-good values to determine which (if any) of the boot components have been compromised. Measured boot does not interrupt or stop the process of booting; it just records the hash IDs of the elements used in the boot. Thus, it is like a security camera. It does not prevent a malicious action; it just records whatever occurs in its area of view.

Client-Based Systems

Client-based vulnerabilities place the user, their data, and their system at risk of compromise and destruction. A client-side attack is any attack that is able to harm a client. Generally, when attacks are discussed, it's assumed that the primary target is a server or a server-side component. A client-side or client-focused attack is one where the client itself, or a process on the client, is the target. A common example of a client-side attack is a malicious website that transfers malicious mobile code (such as an applet) to a vulnerable browser running on the client. Client-side attacks can occur over any communications protocol, not just Hypertext Transfer Protocol (HTTP). Another potential vulnerability that is client-based is the risk of poisoning of local caches.

Mobile Code

Applets are code objects sent from a server to a client to perform some action. In fact, applets are actually self-contained miniature programs that execute independently of the server that sent them—that is, mobile code. The arena of the web is undergoing constant flux. The use of applets is not as common today as it was in the early 2010s. However, applets are not absent from the web, and many browsers still support them (or still have add-ons present that support them). Thus, even when your organization does not intentionally use applets in your internal or public web design, your web browsers could encounter them while surfing the public web.

Imagine a web server that offers a variety of financial tools to web users. One of these tools might be a mortgage calculator that processes a user's financial information and provides a monthly mortgage payment based on the loan's principal and term and the borrower's credit information. Instead of processing this data and returning the results to the client system, the remote web server might send an applet to the local system that enables it to perform those calculations itself. This provides a number of benefits to both the remote server and the end user:

- The processing burden is shifted to the client, freeing up resources on the web server to process requests from more users.
- The client is able to produce data using local resources rather than waiting for a response from the remote server. In many cases, this results in a quicker response to changes in the input data.
- In a properly programmed applet, the web server does not receive any data provided to the applet as input, therefore maintaining the security and privacy of the user's financial data.

However, applets introduce security concerns. Any time you execute someone else's code, you are at risk of the code being malicious or having exploitable flaws. Security administrators must take steps to ensure that code sent to systems on their network is safe and properly screened for malicious activity. Also, unless the code is analyzed line by line, the end user can never be certain that the applet doesn't contain a Trojan horse, backdoor, rootkit, ransomware, or some other malware component. For example, the mortgage calculator might indeed transmit sensitive financial information to the web server without the end user's knowledge or consent.

Two historical examples of applet types are Java applets and ActiveX controls. *Java* is a platform-independent programming language developed by Sun Microsystems (now owned by Oracle). *ActiveX* controls were Microsoft's response to Sun's Java. Java is still in use for internal development and business software, but its use on the Internet is rare. ActiveX is now a legacy technology and is both EOL and EOS. It was only supported by internet Explorer. Most modern internet-capable systems no longer support these applet forms, but it is important to ensure that before assuming a system is secure.

JavaScript is the most widely used web scripting language in the world and is embedded into (included inside of) HTML documents using `<script></script>` enclosure tags. JavaScript is usually dependent on its HTML host document, but it can

operate as a stand-alone script file (both client-side and server-side). However, it is generally not considered an applet—it is usually embedded code. It is automatically downloaded along with the primary web documents from most web servers you access. JavaScript enables dynamic web pages and supports web applications as well as a plethora of client-side activities and page behaviors.

Most browsers support JavaScript via a dedicated JavaScript engine. Most of the implementations use sandbox isolation to restrict JavaScript to web-related activities while minimizing its ability to perform general-purpose programming tasks. Also, most browsers default to enforcing the same-origin policy. The same-origin policy prohibits JavaScript code from accessing content from another origin. The origin is typically defined by a combination of protocol (i.e., HTTP versus HTTPS), domain/IP address, and port number. If other content has any one of these origin elements different from the origin of the JavaScript code, the code will be blocked from accessing that content.

However, there are ways of abusing JavaScript. Threat actors can create believable fake websites that look and act like a valid site, including duplicating the JavaScript dynamic elements. But since the JavaScript code is in the HTML document sent to the browser, a malicious actor could alter that code to perform harmful actions, such as copying or cloning credentials and distributing them to the attacker. Threat actors have also found means to breach the sandbox isolation and even violate same-original policies from time to time, so JavaScript should be considered a threat. Whenever you allow code from an unknown and thus untrusted source to execute on your system, you are putting your system at risk of compromise. XSS and XSRF/CSRF exploit methods can be used to compromise JavaScript support in browsers.

Here are some responses to these risks:

- Keep browsers updated (client side).
- Implement secure JavaScript subsets/libraries (server side).

- Use a content security policy (CSP) that attempts to rigidly enforce same-origin restrictions for most browser-side active technologies (integrated into browsers and referenced by HTML header values).

As with most web applications, insertion attacks are common, so watch out for the injection of odd or abusive JavaScript code in the input being received by a web server.

As a client, you may gain some benefit by being behind a web application firewall (WAF) or next-generation firewall (NGFW). We do not recommend deactivating JavaScript outright—that would cause most of the web to stop functioning in your browser. Instead, the use of security-focused add-ons, browser helper objects (BHOs), and extensions may reduce the risk of JavaScript. However, the use of add-ons and extensions can also put your system at greater risks to other threats.

For more on web-related vulnerabilities, attacks, and countermeasures, see Chapter 21, "Malicious Code and Application Attacks."

Local Caches

There are many types of local caches, including DNS cache, ARP cache, and temporary internet files. See Chapter 11 for details about DNS cache and ARP cache abuses.

Temporary internet files or the *internet files cache* is the temporary storage of files downloaded from Internet sites that are being held by the client's utility (typically a browser) for current and possibly future use. This cache mostly contains website content, but other Internet services can also use a file cache. A variety of exploitations, such as the split-response attack, can cause the client to download content that was not an intended element of a requested web page and store it in the cache. DOM XSS may be able to access and use locally cached files to execute malicious code or exfiltrate data (see Chapter 21). Mobile code scripting attacks could also be used to plant false content in the cache. Once files have been poisoned in the cache,

then even when a legitimate web document calls on a cached item, the malicious item will be activated.

Client utilities should be managing the local files cache, but those utilities might not always be doing the best job. Often, the defaults are for efficiency and performance, not for security. Consider reconfiguring the cache to only retain files for a short period of time, minimize the cache size, and deactivate the preloading of content. Keep in mind that these changes can reduce browsing performance when on slower or high-latency connections. You may want to configure the browser to delete all cookies and cache upon exit. Although you can typically perform a manual cache wipe, you would have to remember to do that. Another option is to use an automated tool that can be configured to wipe temporary Internet files on a schedule or upon a targeted program close.

Additional coverage of client-based endpoint security concerns is in Chapter 11.

Server-Based Systems

An important area of server-based concern, which may include clients as well, is the issue of *data flow control*. Data flow is the movement of data between processes, between devices, across a network, or over communication channels. Management of data flow ensures not only efficient transmission with minimal delays or latency, but also reliable throughput using hashing and confidentiality protection with encryption. Data flow control also ensures that receiving systems are not overloaded with traffic, especially to the point of dropping connections or being subject to a malicious or even self-inflicted denial of service. When data overflow occurs, data may be lost or corrupted or may trigger a need for retransmission. These results are undesirable, and data flow control is often implemented to prevent these issues from occurring. Data flow control may be provided by networking devices, including routers and switches, as well as network applications and services.

A *load balancer* spreads or distributes network traffic load across several network links or devices. A load balancer may be able to provide more control over data flow. The purpose of load balancing is to obtain more optimal infrastructure utilization, minimize response time, maximize throughput, reduce overloading, and eliminate bottlenecks. Although load balancing can be used in a variety of situations, a common implementation is spreading a load across multiple members of a server farm or cluster. A load balancer might use a variety of techniques to perform load distribution, including random choice, round robin, load/utilization monitoring, and preferencing. See Chapter 12, "Secure Communications and Network Attacks," for more on load balancing.

A denial-of-service (DoS) attack can severely deter data flow control. It is important to monitor for DoS attacks and implement mitigations. See Chapter 17, "Preventing and Responding to Incidents," for a discussion of these attacks and potential defenses.

For more on server protections, see Chapter 18.

Large-Scale Parallel Data Systems

Parallel data systems or *parallel computing* is a computation system designed to perform numerous calculations simultaneously. However, parallel data systems often go far beyond basic multiprocessing capabilities. They often include the concept of dividing up a large task into smaller elements, and then distributing each subelement to a different processing subsystem for parallel computation. This implementation is based on the idea that some problems can be solved efficiently if broken into smaller tasks that can be worked on concurrently. Parallel data processing can be accomplished by using distinct CPUs or multicore CPUs, virtual systems, or any combination of these. *Large-scale parallel data systems* must also be concerned with performance, power consumption, and reliability/stability issues.

Within the arena of multiprocessing or parallel processing there are several divisions. The first division is between symmetric multiprocessing (SMP) and asymmetric multiprocessing (AMP).

The scenario where a single computer contains multiple processors that are treated equally and controlled by a single OS is called *symmetric multiprocessing (SMP)*. In SMP, processors share not only a common OS but also a common data bus and memory resources. In this type of arrangement, systems may use a large number of processors. The collection of processors works collectively on a single or primary task, code, or project.

In *asymmetric multiprocessing (AMP)*, the processors are often operating independently of one another. Usually, each processor has its own OS and/or task instruction set, as well as a dedicated data bus and memory resources. Under AMP, processors can be configured to execute only specific code or operate on specific tasks (or specific code or tasks are allowed to run only on specific processors; this might be called affinity in some circumstances).

A variation of AMP is *massive parallel processing (MPP)*, where numerous AMP systems are linked together to work on a single primary task across multiple processes in multiple linked systems. Some computationally intensive operations, such as those that support the research of scientists and mathematicians, require more processing power than a single OS can deliver. MPP may best serve such operations. MPP systems house hundreds or even thousands of processors, each of which has its own OS and memory/bus resources. Some MPPs have over 10 million execution cores. When the software that coordinates the entire system's activities and schedules them for processing encounters a computationally intensive task, it assigns responsibility for the task to a single processor (not so different from the Master Control Program [MCP] in the popular movie “Tron”). This processor breaks the task into manageable parts and distributes them to other processors for execution. Those processors return their results to the coordinating processor, where they are assembled and returned to the requesting application. MPP systems are extremely powerful (not to mention extremely

expensive!) and are used in a great deal of computing or computational-based research.

Both types of multiprocessing provide unique advantages and are suitable for different types of situations. SMP systems are adept at processing simple operations at extremely high rates, whereas MPP systems are uniquely suited for processing very large, complex, computationally intensive tasks that lend themselves to decomposition and distribution into a number of subordinate parts.

The arena of large-scale parallel data systems is still evolving. It is likely that many management issues are yet to be discovered and solutions to known issues are still being sought. Large-scale parallel data management is likely a key tool in managing big data and will often involve cloud computing (see Chapter 16), grid computing, or peer-to-peer computing solutions.

Grid Computing

Grid computing is a form of parallel distributed processing that loosely groups a significant number of processing nodes to work toward a specific processing goal. Members of the grid can enter and leave the grid at random intervals. Often, members join the grid only when their processing capacities are not being taxed for local workloads. When a system is otherwise in an idle state, it could join a grid group, download a small portion of work, and begin calculations. When a system leaves the grid, it saves its work and may upload completed or partial work elements back to the grid. Many interesting uses of grid computing have developed, including projects seeking out intelligent aliens, performing protein folding, predicting the weather, modeling earthquakes, planning financial decisions, and solving for primes.

The biggest security concern with grid computing is that the content of each work packet is potentially exposed to the world. Many grid computing projects are open to the world, so there is no restriction on who can run the local processing application and participate in the grid's project. This also means that grid members could keep copies of each work packet and examine the

contents. Thus, grid projects will not likely be able to maintain secrecy and are not appropriate for private, confidential, or proprietary data.

Grid computing can also vary greatly in computational capacity from moment to moment. Work packets are sometimes not returned, returned late, or returned corrupted. This requires significant reworking and causes instability in the speed, progress, responsiveness, and latency of the project as a whole and with individual grid members. Time-sensitive projects might not be given sufficient computational time to finish by a specific chronological deadline.

Grid computing often uses a central primary core of servers to manage the project, track work packets, and integrate returned work segments. If the central servers are overloaded or go offline, complete failure or crashing of the grid can occur. However, when central grid systems are inaccessible, grid members usually complete their current local tasks and then regularly poll to discover when the central servers come back online. There is also a potential risk that a compromise of the central grid servers could be leveraged to attack grid members or trick grid members into performing malicious actions instead of the intended purpose of the grid community.

Peer to Peer

Peer-to-peer (P2P) technologies are networking and distributed application solutions that share tasks and workloads among peers. This is similar to grid computing; the primary differences are that there is no central management system, and the services are usually provided in real time rather than as a collection of computational power. Common examples of P2P include many VoIP services, BitTorrent (for data/file distribution), and tools for streaming audio/music distribution.

Security concerns with P2P solutions include a perceived inducement to pirate copyrighted materials, the ability to eavesdrop on distributed content, a lack of central

control/oversight/management/filtering, and the potential for services to consume all available bandwidth.

Industrial Control Systems

An *industrial control system (ICS)* is a form of computer-management device that controls industrial processes and machines, also known as *operational technology (OT)* (and also industrial IoT [IIOT]; see the section "Internet of Things"). ICSs are used across a wide range of industries, including manufacturing, fabrication, electricity generation and distribution, water distribution, sewage processing, and oil refining. There are several forms of ICS, such as *programmable logic controllers (PLCs)*, *distributed control systems (DCSs)*, and *supervisory control and data acquisition (SCADA)*.

PLC units are effectively single-purpose or focused-purpose digital computers. They are typically deployed for managing and automating various industrial electromechanical operations, such as controlling systems on an assembly line or a large-scale digital light display (such as a giant display system in a stadium or on a Las Vegas Strip marquee).

DCS units are typically found in industrial process plants where gathering data and implementing control over a large-scale environment from a single location is essential. An important aspect of DCS is that the controlling elements are distributed across the monitored environment, such as a manufacturing floor or a production line, and the centralized monitoring location sends commands out of those localized controllers while gathering status and performance data. A DCS might be analog or digital, depending on the task or the device being controlled. For example, a liquid flow value DCS would be an analog system, whereas an electric voltage regulator DCS would likely be a digital system.

A SCADA system can operate as a stand-alone device, be networked together with other SCADA systems, or be networked with traditional IT systems. SCADA typically includes a human-machine interface (HMI) to enable personnel to oversee, manage,

and control the complex industrial machine and technology systems. SCADA is used to monitor and control a wide range of industrial processes. SCADA can communicate with PLCs and DCS solutions.

A DCS focuses on processes and is state driven, whereas SCADA focuses on data-gathering and is event driven. DCS is more suited to operating on a limited scale, whereas SCADA is suitable for managing systems over large geographic areas.

Legacy SCADA systems were designed with minimal human interfaces. Often, they used mechanical buttons and knobs or simple LCD screen interfaces (similar to what you might have on a business printer or a GPS navigation device). However, modern networked SCADA devices may have more complex remote-control software interfaces.

A PLC is used to control a single device in a stand-alone manner. DCS was used to interconnect several PLCs within a limited physical range to gain centralized control, management, and oversight through networking. SCADA expanded this to large-scale physical areas to interconnect multiple DCSs and individual PLCs. For example, a PLC can control a single transformer, a DCS can manage a power station, and SCADA can oversee a power grid.

Modbus is a widely used communication protocol in industrial automation and control systems. Developed by Modicon (now part of Schneider Electric) in 1979, Modbus has become a de facto standard for connecting and managing devices within industrial environments. It provides a common language for different devices to exchange data and commands, facilitating communication in SCADA systems, PLCs, and other industrial applications. The open standard nature of Modbus fosters interoperability, a crucial requirement in ICS and SCADA environments where diverse devices from different manufacturers must seamlessly interact.

In theory, the static design of PLC, DCS, and SCADA and their minimal human interfaces should make the system fairly resistant to compromise or modification. Thus, little security was built into these industrial control devices, especially in the past. But there have been several well-known compromises of industrial control systems in recent years; for example, Stuxnet delivered the first-ever-known rootkit to a SCADA system located in a nuclear facility. Many SCADA vendors have started implementing security improvements into their solutions to prevent or at least reduce future compromises. However, in practice, SCADA and ICS systems are still often poorly secured, vulnerable, and infrequently updated, and older versions not designed for security are still in widespread use.

Generally, typical security management and hardening processes can be applied to ICS, DCS, PLC, and SCADA systems to improve on whatever security is or isn't present in the device from the manufacturer. Common important security controls include isolating networks, limiting access physically and logically, restricting code to only essential applications, changing default credentials, and logging all activity. While confidentiality and integrity are often important considerations, the primary concern for OT/ICS is ensuring the availability of real-time control signals. Immediate and continuous availability of control signals is critical for the proper functioning of industrial processes. Otherwise, catastrophic consequences could result.

The ISA99 standards development committee has established and is maintaining guidelines for securing ICS, DCS, PLC, and SCADA systems. Much of their work is integrated into the International Electrotechnical Commission's (IEC) 62443 series of standards. To learn more about these standards, visit www.isa.org and iecee.org. NIST maintains ICS security standards in SP 800-82rs (nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf). North American Electric Reliability Corporation (NERC) maintains its own security guides for ICS (www.nerc.com), which are similar to those of the European Reference Network for Critical Infrastructure Protection (ERNCIP) (erncip-project.jrc.ec.europa.eu).

Distributed Systems

A *distributed system* or a *distributed computing environment (DCE)* is a collection of individual systems that work together to support a resource or provide a service. Often, a DCE is perceived by users as a single entity rather than numerous individual servers or components. DCEs are designed to support communication and coordination among their members to achieve a common function, goal, or operation. Some DCE systems are composed of homogenous members; others are composed of heterogeneous systems. Distributed systems can be implemented to provide resiliency, reliability, performance, and scalability benefits. Most DCEs exhibit numerous duplicate or concurrent components, are asynchronous, and allow for fail-soft or independent failure of components. A DCE is also known as (or at least described as) concurrent computing, parallel computing, and distributed computing. DCE solutions are implemented as client-server architectures (see the previous client and server sections as well as endpoint coverage in Chapter 11), as a three-tier architecture (such as basic web applications), as multitiered architectures (such as advanced web applications), and as peer-to-peer architectures (such as BitTorrent and most cryptocurrency blockchain ledgers as discussed in Chapter 7). DCE solutions are often employed for scientific and medical research projects, education projects, and industrial applications requiring extensive computational resources.

DCE forms the backbone of a wide range of modern Internet, business, and communication technologies that you might regularly use, including DNS, single-sign-on, directory services, massively multiplayer online role-playing games (MMORPGs), mobile networks, and most websites. DCE also makes possible a plethora of advanced technologies such as service-oriented architecture (SOA), software-defined networking (SDN), microservices, and so on.

A DCE typically includes an *interface definition language (IDL)*. An IDL is a language used to define the interface between client and server processes or objects in a distributed system. IDL enables the creation of interfaces between objects when those

objects are in varying locations or are using different programming languages; thus, IDL interfaces are language-independent and location-independent. There are numerous examples of DCE IDLs or frameworks, such as remote procedure calls (RPCs), the Common Object Request Broker Architecture (CORBA), and the Distributed Component Object Model (DCOM).

There are some security issues inherent with DCE. The primary security concern is the interconnectedness of the components. This configuration could allow for error or malware propagation, and if an adversary compromises one component, it may grant them the ability to compromise other components in the collective through pivoting and lateral movement. Other common issues to consider and address include the following:

- Access by unauthorized users
- Masquerading, impersonation, and spoofing attacks of users and/or devices
- Security control bypass or deactivating
- Communication eavesdropping and manipulation
- Insufficient authentication and authorization
- A lack of monitoring, auditing, and logging
- Failing to enforce accountability

The issues in this list are not unique to DCE, but they are especially problematic in a distributed system.

Since distributed systems include members that may be distributed geographically, they have a larger potential attack surface than that of a single system. Thus, it is important to consider the collective threats and risks of the individual member components of a DCE as well as the communications interconnections between them. To secure DCE, encryption is needed for storage, transmission, and processing (such as homomorphic encryption). Also, strong multifactor authentication should be implemented. If a strict homogeneous

component set is not maintained, heterogenous systems introduce their own risks, whether different OSs are in use or just different versions or patch levels of the same OS. The more varied the DCE components, the more challenging it is to maintain consistent security configuration, enforcement, monitoring, and oversight. If the DCE is so large or broadly distributed as to cross international boundaries, then data sovereignty issues need to be addressed.

High-Performance Computing (HPC) Systems

High-performance computing (HPC) systems are computing platforms designed to perform complex calculations or data manipulations at extremely high speeds. Super computers and MPP solutions are common examples of HPC systems. HPC systems are used when real-time or near-real-time processing of massive data is necessary for a particular task or application. These applications can include scientific studies, industrial research, medical analysis, societal solutions, and commercial endeavors.

Many of the products and services we use today, including mobile devices and their apps, IoT devices, ICS solutions, streaming media, voice assistants, 3D modeling and rendering, and AI/ML calculations, all depend on HPC to exist. As the population of Internet and computing devices increases, as the datasets being collected continue to increase exponentially, and as new uses of that data and those devices are conceived, HPC will be in even greater demand in the future.

An HPC solution is composed of three main elements: compute resources, network capabilities, and storage capacity. Each element must be able to provide equivalent capabilities to optimize overall performance. If storage is too slow, then data cannot be fed to the application processing on the compute resources. If networking capacity is not sufficient, then users of a resource will experience latency or even a benign denial of service (DoS).

A *benign DoS* occurs when a service is running on insufficient resources, when there has been an unforeseen popularity or traffic spike, or when something about the supporting system fails, such as drive loss, network link drop, or a corrupted configuration. This type of DoS occurs through no direct or intentional malign action on the part of an adversary. It is due to innocent events, unexpected conditions, or mistakes on the part of the owners/operators. For more on DoS, see Chapter 17.

Real-Time Operating Systems

A concept related to HPC is that of the real-time OS (RTOS). Often HPCs implement RTOS compute capability or otherwise attempt to achieve real-time processing and operations.

A *real-time operating system (RTOS)* is designed to process or handle data as it arrives on the system with minimal latency or delay. An RTOS is usually stored on read-only memory (ROM) and is designed to operate in a hard real-time or soft real-time condition. A hard real-time solution is for mission-critical operations where delay must be eliminated or minimized for safety, such as autonomous cars. A soft real-time solution is used when some level of modest delay is acceptable under typical or normal conditions, as it is for most consumer electronics, such as the delay between a digitizing pen and a graphics program on a computer.

RTOSs can be event driven or time-sharing. An event-driven RTOS will switch between operations or tasks based on preassigned priorities. A time-sharing RTOS will switch between operations or tasks based on clock interrupts or specific time intervals. An RTOS is often implemented when scheduling or timing is the most critical part of the task to be performed.

A security concern using RTOSs is that these systems are often focused and single-purpose, leaving little room for security. They

often use custom or proprietary code, which may include unknown bugs or flaws that attackers could discover. An RTOS might be overloaded or distracted with bogus datasets or process requests by malware. When deploying or using RTOSs, use isolation and communication monitoring to minimize abuses.

Internet of Things

Smart devices offer the user a plethora of customization options, typically through installing apps, and may take advantage of on-device or in-the-cloud machine learning (ML) processing. The products that can be labeled “smart devices” are constantly expanding and already include smartphones, tablets, music players, home assistants, extreme sport cameras, virtual reality/augmented reality (VR/AR) systems, and fitness trackers.

The *Internet of Things (IoT)* is a class of Internet-connected smart devices that provide automation, remote control, or AI processing to appliances or devices. IoT may often perform functions similar to an embedded system, but they are different. An IoT device is almost always a separate and distinct hardware device that is used on its own or in conjunction with an existing system (such as a smart ioT thermostat for a heating, ventilation, and air-conditioning [HVAC] system). An embedded system is one where the computer control component has been integrated into the larger mechanism's structure, design, and operation, often even built into the same chassis or case.

The security issues related to IoT are often about access and encryption. An IoT device was often not designed with security as a core concept or even an afterthought. This has resulted in numerous home and office network security breaches. Additionally, once an attacker has remote access to or through an IoT device, they may be able to access other devices on the compromised network. When electing to install IoT equipment, evaluate the security of the device as well as the security reputation of the vendor. If the device does not have the ability to meet or accept your existing security baseline, then don't compromise your security just for a flashy gadget.

One possible secure implementation is to deploy a distinct network segment for the IoT equipment, which is kept separate and isolated from the primary network. This configuration is often known as *three dumb routers* (see www.grc.com/sn/sn-545.pdf or www.pcper.com/reviews/General-Tech/Steve-Gibsons-Three-Router-Solution-IOT-Insecurity). Other standard security practices are beneficial to IoT, including keeping systems patched, limiting physical and logical access, monitoring all activity, and implementing firewalls and filtering.

Wearable technology or *wearables* are offshoots of smart devices and IoT devices that are specifically designed to be worn by an individual. The most common examples of wearable technology are smart watches and fitness trackers. There are an astounding number of available options, with a wide range of features and security capabilities. When selecting a wearable device, consider the security implications. Is the data being collected in a cloud service that is secured for private use or is it made publicly available? What alternative uses is the collected data going to be used for? Is the communication between the device and the collection service encrypted? And can you delete your data and profile from the service completely if you stop using the device?

Although we often associate smart devices and IoT with home or personal use, they concern every organization. This is partly because of the use of mobile devices by employees within the company's facilities and even on the organizational network.

Another concern is that many IoT or networked automation devices are often used in the business environment. This includes environmental controls, such as HVAC management, air quality control, debris and smoke detection, lighting controls, door automation, personnel and asset tracking, and consumable inventory management and auto-reordering (such as coffee, snacks, printer toner, paper, and other office supplies). Thus, both smart devices and IoT devices are potential elements of a

modern business network that need appropriate security management and oversight. For some additional reading on the importance of proper security management of smart devices and IoT equipment, see “NIST Cybersecurity for IOT Program” at www.nist.gov/itl/applied-cybersecurity/nist-cybersecurity-iot-program.

A common IoT device deployed in a business environment is a sensor. Sensors can measure just about anything, including temperature, humidity, light levels, dust particles, movement, acceleration, and air/liquid flow. Sensors can be linked with cyber-physical systems to automatically adjust or alter operations based on the sensor's measurements, such as turning on the air conditioning when the temperature rises above a threshold. Sensors can also be linked to ICS, DCS, and SCADA solutions.

IoT devices—in fact, almost all hardware and software—often have insecure or weak defaults. Never assume defaults are good enough. Always evaluate acquired IoT products' setting and configuration options and make changes that optimize security and support business functions. This is especially relevant to default passwords, which must always be changed and verified.

Industrial Internet of Things (IIoT) is a derivative of IoT that focuses more on industrial, engineering, manufacturing, or infrastructure level oversight, automation, management, and sensing. IIoT is an evolution of ICS and DCS that integrates cloud services to perform data collection, analysis, optimization, and automation. Examples of IIOT include edge computing and fog computing, discussed next.

Edge and Fog Computing

Edge computing is a philosophy of network design where data and the compute resources are located as close as possible to optimize bandwidth use while minimizing latency. In edge computing, the intelligence and processing are contained within each device. Thus, rather than sending data to a master processing entity, each device can process its own data locally. The architecture of edge computing performs computations

closer to the data source, which is at or near the edge of the network. This is distinct from performing processing in the cloud on data transmitted from remote locations. Edge computing is often implemented as an element of IIOT solutions, but edge computing is not limited to this type of implementation.

Edge computing can be viewed as the next evolution of computing concepts. Originally, computing was accomplished on core mainframe computers where applications were executed on the central system but where controlled or manipulated via thin clients. Then the distributed concept of client/server moved computing out to endpoint devices. This allowed for the execution of decentralized applications (i.e., not centrally controlled) that ran locally on the endpoint system. From there, virtualization led to cloud computing. Cloud computing is a type of centralized application execution on remote data center systems, controlled remotely by endpoints. Finally, edge computing is the use of devices that are close to or at the endpoint where applications are centrally controlled, but the actual execution is as close to the user or network edge as possible.

One potential use for edge devices is the deployment of mini-web servers by ISPs to host static or simple pages for popular sites that are located nearer to the bulk of common visitors than the main web servers. This speeds up the initial access to the front page of a popular organization's web presence, but then subsequent page visits are directed to and served by core or primary web servers that may be located elsewhere. Other examples of edge computing solutions include security systems, motion-detecting cameras, image recognition systems, IoT and IIOT devices, self-driving cars, optimized content delivery network (CDN) caching, medical monitoring devices, and videoconferencing solutions.

Fog computing is another example of advanced computation architecture, which is also often used as an element in an IIOT deployment. *Fog computing* relies on sensors, IoT devices, or even edge computing devices to collect data, and then transfers it back to a central location for processing. The fog computing

processing location is positioned in the LAN. Thus, with fog computing, intelligence and processing are centralized in the LAN. The centralized computing power processes information gathered from the fog of disparate devices and sensors.

In short, edge computing performs processing on the distributed edge systems, whereas fog computing performs centralized processing of the data collected by the distributed sensors. Both edge and fog computing can often take advantage of or integrate the use of microcontrollers, embedded devices, static devices, cyber-physical systems, and IoT equipment.

Embedded Devices and Cyber-Physical Systems

An *embedded system* is any form of computing component added to an existing mechanical or electrical system for the purpose of providing automation, remote control, and/or monitoring. The embedded system is typically designed around a limited set of specific functions in relation to the larger product to which it is attached. It may consist of the same components found in a typical computer system or a microcontroller (an integrated chip with onboard memory and peripheral ports).

Microcontrollers

A *microcontroller* is similar to, but less complex, than a system on a chip, or SoC (see Chapter 11). A microcontroller may be a component of an SoC. A microcontroller is a small computer consisting of a CPU, various input/output capabilities, RAM, and often nonvolatile storage in the form of flash or ROM/PROM/EEPROM. Examples include Raspberry Pi, Arduino, and a field-programmable gate array (FPGA).

Embedded systems can be a security risk because they are generally static systems, meaning that even the administrators who deploy them have no real means to alter the device's operations to address security vulnerabilities. Some embedded

systems can be updated with vendor patches, but patches are often released months after a known exploit is found in the wild. It is essential that embedded systems be isolated from the Internet and from a private production network to minimize exposure to remote exploitation, remote control, or malware compromise.

Security concerns for embedded systems include the fact that most are designed to minimize cost and extraneous features. This often leads to a lack of security and difficulty with upgrades or patches. Because an embedded system may be in control of a mechanism in the physical world, a security breach could cause harm to people and property.

Static Systems

Another concept similar to that of embedded systems is *static systems* (aka *static environments*). A static environment is a set of conditions, events, and surroundings that don't change. In theory, a static environment doesn't offer new or surprising elements once understood. A static IT environment is any system that is intended to remain unchanged by users and administrators. The goal is to prevent, or at least reduce, the possibility of a user implementing change that could result in reduced security or functional operation. This is also known as a nonpersistent environment or a stateless system, as opposed to a persistent environment or stateful system, which allows changes and retains them between access events and reboots.

Examples of static systems include the check-in kiosk at the airport, an ATM, and often the complimentary guest computer at a hotel or library. Those guest computers are configured to provide the user with a temporary desktop environment to perform a restricted range of tasks. However, when the user terminates their session due to timeout or logging out, the system discards all the previous session's information and changes and restores a pristine version of the environment for the next user. Static systems can be implemented in a variety of ways, including using local VMs or remotely accessed VDI (virtual desktop infrastructure).

In technology, static environments are applications, OSs, hardware sets, or networks configured for a specific need, capability, or function, then set to remain unaltered. However, although the term *static* is used, no truly static systems exist. There is always the chance that a hardware failure, a hardware configuration change, a software bug, a software-setting change, or an exploit may alter the environment, resulting in undesired operating parameters or actual security intrusions.

Sometimes, the phrase *static OS* refers to the concept of a static system/environment or indicates a slight variation. That variation is that the OS itself is beyond the ability of the user to change, but the user can install or use applications. Often, those applications may be limited, restricted, or controlled to avoid allowing an application to alter the otherwise static OS. Some potential examples of static OSs would be smart TVs, gaming systems/consoles, or mobile devices where only applications from a vendor-controlled app store can be installed.

Cyber-Physical Systems

Cyber-physical systems refer to devices that offer a computational means to control something in the physical world. In the past, these might have been referred to as embedded systems, but the category of cyber-physical seems to focus more on the physical world results rather than the computational aspects. Cyber-physical devices and systems are essentially key elements in robotics and sensor networks. Basically, any computational device that can cause a movement to occur in the real world is considered a robotic element, whereas any such device that can detect physical conditions (such as temperature, light, movement, and humidity) is a sensor. Examples of cyber-physical systems include prosthetics to provide human augmentation or assistance, collision avoidance in vehicles, air traffic control coordination, precision in robot surgery, remote operation in hazardous conditions, and energy conservation in vehicles, equipment, mobile devices, and buildings.

Security Concerns of Embedded and Static Systems

Embedded, static, network-enabled, and cyber-physical are usually more limited or constrained based on their design or hardware capabilities compared to typical endpoint, server, and networking hardware. These constraints can have security implications.

Some embedded systems run on replaceable or rechargeable batteries. Others only receive a small amount of power from a USB plug or special power adapter/converter. These power limitations can restrict the speed of operations, which in turn can limit the execution of security components. If additional power is consumed, the device might overheat. This could result in slower performance, crashing, or destruction.

Most embedded systems use less-capable CPUs. This is due to cost and power savings or limitations. Fewer computing capabilities means fewer functions, which means fewer security operations.

Many embedded systems have limited network capabilities. These network capabilities could be limited to wired only or wireless only. Within wireless, the device could be limited to a specific Wi-Fi version, frequency, speed, and/or encryption. Some devices using wireless are limited to special communication protocols, such as Zigbee or Bluetooth Low Energy (BLE).

Many embedded systems are unable to process high-end encryption. The crypto on these special devices is often limited and may use older algorithms or poor keys, or just lack good key management. Some devices are known to have preshared and/or hard-coded encryption keys.

Some embedded systems are difficult to patch, whereas others might not even offer patching or upgrading. Without update and patch management, vulnerable code will remain at risk.

Some embedded systems do not use authentication to control subjects or restrict updates. Some devices use hard-coded credentials. These should be avoided. Only use equipment that

allows for customized credentials, and choose devices that support mutual-certificate authentication.

Some embedded systems have a limited transmission range due to low-power antennae. This can restrict the device's usefulness or require signal boosting to compensate.

Due to the low cost of some embedded systems, they might not include necessary security features. Other devices that do include needed security components may be too costly to be considered.

Similar to supply chain issues, when an embedded system is used, the organization automatically trusts the device vendor and the cloud service behind it. This implied trust may be misguided. Always thoroughly investigate vendors before relying on their product, and even then, segregate embedded systems in their own constrained network segments. See the discussion of zero trust in Chapter 8.

Based on these constraints and other concerns, security management of embedded and static systems must accommodate the fact that most are designed to minimize costs and extraneous features. This often leads to a lack of security mechanisms and difficulty with upgrades or patches.

Static environments, embedded systems, cyber-physical systems, high-performance computing (HPC) systems, edge computing devices, fog computing devices, mobile devices, and other limited or single-purpose computing environments need security management. Although they may not have as broad an attack surface and aren't exposed to as many risks as a general-purpose computer, they still require proper security government. Many of the same general security management principles used over servers and endpoints can be applied to embedded, static, and cyber-physical systems.

Network segmentation involves controlling traffic among networked devices. Complete or physical network segmentation occurs when a network is isolated from all outside communications, which means transactions can occur only between devices within the segmented network. You can impose logical network segmentation with switches using virtual local

area networks (VLANs), or through other traffic-control means, including MAC addresses, IP addresses, physical ports, TCP or UDP ports, protocols, or application filtering, routing, and access control management. Network segmentation can isolate embedded devices and static environments to prevent changes and/or exploits from reaching them. See Chapter 11 for more on segmentation.

An application firewall is a device, server add-on, virtual service, or system filter that defines a strict set of communication rules for a service and all users. It's intended to be an application-specific server-side firewall to prevent application-specific protocol and payload attacks. A network firewall is a hardware device designed for general network filtering, typically called an appliance. A network firewall is designed to provide broad protection for an entire network. An internal segmentation firewall (ISFW) is used to create a network division or segment. Every network needs a network firewall. Many application servers need an application firewall. However, the use of an application firewall generally doesn't negate the need for a network firewall. You should use firewalls in a series to complement each other, rather than seeing them as competitive solutions. See Chapters 11 and 17 for more on firewalls.

Security layers exist where devices with different levels of classification or sensitivity are grouped together and isolated from other groups with different levels. This isolation can be absolute or one-directional. For example, a lower level may not be able to initiate communication with a higher level, but a higher level may initiate with a lower level. Isolation can also be logical or physical. Logical isolation requires the use of classification labels on data and packets, which must be respected and enforced by network management, OSs, and applications. Physical isolation requires implementing network segmentation or air gaps between networks of different security levels. See Chapter 5, “Protecting Security of Assets,” to learn more about managing data and asset classification.

Manual updates should be used in static environments to implement only tested and authorized changes. An automated

update system would allow untested updates to introduce unknown security reductions. As with manual software updates, strict control over firmware in a static environment is important. Firmware updates should be implemented on a manual basis, only after thorough testing and review. Firmware version control or oversight of firmware release should focus on maintaining a stable operating platform while minimizing exposure to downtime or compromise.

Even embedded and static systems should be monitored for performance, violations, compliance, and operational status. Some of these types of devices can perform on-device monitoring, auditing, and logging, whereas others may require external systems to collect activity data. Any and all devices, equipment, and computers within an organization should be monitored to ensure high performance and minimal downtime, and to detect and stop violations and abuse.

As with any security solution, relying on a single security mechanism is unwise. Defense in depth uses multiple types of access controls in literal or theoretical concentric circles or layers. This form of layered security helps an organization avoid a monolithic security stance. A monolithic mentality is the belief that a single security mechanism is all that is required to provide sufficient security. With security control redundancy and diversity, a static environment can avoid the pitfalls of a single security feature failing; the environment has several opportunities to deflect, deny, detect, and deter any threat. Unfortunately, no security mechanism is perfect. Each individual security mechanism has a flaw or a work-around just waiting to be discovered and abused by a malicious actor.

Microservices

It is important to evaluate and understand the vulnerabilities in system architectures, especially in regard to technology and process integration. As multiple technologies and complex processes are intertwined in the act of crafting new and unique business functions, new issues and security problems often

surface. As systems are integrated, attention should be paid to potential single points of failure as well as to emergent weaknesses in *service-oriented architecture (SOA)*. An SOA constructs new applications or functions out of existing but separate and distinct software services. The resulting application is often new; thus, its security issues are unknown, untested, and unprotected. All new deployments, especially new applications or functions, must be thoroughly vetted before they can go live into a production network or the public Internet.

Microservices, or microservices architecture, is an architectural style in software development where an application is structured as a collection of small, independently deployable, and loosely coupled services. Each service, often referred to as a microservice, represents a specific business capability and operates as a self-contained unit with its own database and communication mechanisms. Microservices are designed to be scalable, maintainable, and independently deployable, allowing developers to build and enhance different parts of an application without affecting the entire system. This architectural approach promotes flexibility, agility, and the ability to scale applications more efficiently. It is the conversion or transformation of a capability of one application into a microservice that can be called upon by numerous other applications.

Microservices are often created as a means to provide purpose-specific business capabilities through services that are independently deployed. Often, microservices are small, focused on a singular operation, designed with few dependencies, and based on fast short-term development cycles (similar to Agile). It is also common to deploy microservices based on immutable architecture or infrastructure as code.

Microservices are a popular development strategy because they allow large, complex solutions to be broken into smaller, self-contained functions. This design also enables multiple programming groups to work on crafting separate elements or microservices simultaneously.

In the context of microservices, APIs play a crucial role. Each microservice exposes an API that allows communication and

interaction with other services within the architecture. APIs form the communication channels between these microservices, enabling them to interact seamlessly. Microservices expose well-defined APIs that specify how external components or services can request or manipulate data. These APIs facilitate the integration and coordination of various microservices, allowing them to work together cohesively within the larger application.

The microservices architecture, with its emphasis on modularization, scalability, and flexibility, is well suited for dynamic and evolving applications. The use of APIs ensures that each microservice can be developed, deployed, and updated independently, promoting agility and enabling teams to work on different components of the application concurrently. This approach has gained popularity in modern software development due to its ability to enhance maintainability, scalability, and the overall efficiency of complex applications.

Security is a paramount concern in a microservices architecture due to its architecture's distributed and interconnected nature. Each microservice needs to implement robust security measures to protect sensitive data and ensure the integrity of communication between services. Common security considerations include:

- *Authentication and authorization:* Implementing secure authentication mechanisms to verify the identity of users and services, and defining proper authorization policies to control access to microservices.
- *Data encryption:* Employing encryption techniques to secure data both in transit and at rest. This is particularly important when microservices communicate over networks or store sensitive information.
- *API security:* Ensuring that APIs are secure by implementing proper authentication, authorization, and encryption for data exchanged between microservices. API gateways are often used to manage and secure API communication.

- *Monitoring and logging:* Implementing comprehensive monitoring and logging to detect and respond to security incidents. This includes tracking access patterns, detecting anomalies, and logging security-relevant events.
- *Container security:* If microservices are deployed using containerization technologies (e.g., Docker), ensuring the security of containerized environments is crucial. This includes employing secure images and configurations, implementing access controls, regularly updating software components to address vulnerabilities, monitoring container activity for suspicious behavior, and utilizing tools for vulnerability scanning and compliance enforcement.

The microservices architecture, emphasizing modularization and scalability, introduces challenges and opportunities for security. Properly addressing security concerns at the level of individual microservices and the interactions between them is essential to building a resilient and secure application.

Infrastructure as Code

Infrastructure as code (IaC) is a change in how hardware management is perceived and handled. Instead of seeing hardware configuration as a manual, direct hands-on, one-on-one administration hassle, it is viewed as just another collection of elements to be managed like software and code are managed under DevSecOps (development, security, and operations). With IaC, the hardware infrastructure is managed in much the same way that software code is managed, including version control, pre-deployment testing, custom-crafted test code, reasonableness checks, regression testing, and consistency in a distributed environment.

This alteration in hardware management approach has allowed many organizations to streamline infrastructure changes to occur more easily, rapidly, securely, and reliably than before. IaC often uses definition files and rule sets that are machine readable to quickly deploy new settings and manage hardware consistently and efficiently. These files can be treated as software code in

terms of development, testing, deployment, updates, and management. IaC is not just limited to hardware; it can also be used to oversee and manage virtual machines (VMs), storage area networks (SANs), and software-defined networking (SDN).

Immutable Architecture

Immutable architecture is the concept that a server never changes once it is deployed. If there is a need to update, modify, fix, or otherwise alter, a new server is built or cloned from the current one, the necessary changes are applied, and then the new server is deployed to replace the previous one. Once the new server is validated, the older server is decommissioned. VMs are destroyed and the physical hardware/system is reused for future deployments.

The benefits of immutable architecture are reliability, consistency, and a predictable deployment process. It eliminates issues common in mutable infrastructures where midstream updates and changes can cause downtime, data loss, or incompatibility.

The mindset of immutable architecture is often described with the analogy of pets versus cattle or snowflakes versus phoenixes. If a server is treated like a pet, when something goes wrong, everyone marshals to the rescue. However, if a server is treated like cattle, it is removed from the herd when something goes wrong, and another is brought in to replace it. If a server is managed uniquely, then it is a snowflake and requires specific focus and attention, causing an increase in administrative time and attention, not to mention complexity for the environment. If a server is always built from scratch, then when changes are needed, a new system can be created with integrated improvements through automated processes, thus rising from the ashes (of previously decommissioned servers) like a phoenix. This minimizes administrative overhead, reduces deployment time, and maintains consistency in the environment.

Software-Defined Networking (SDN)

A derivative of IaC and DCE is software-defined networking (SDN). SDN is the management of networking as a virtual or software resource even though it technically still occurs over hardware (similar to infrastructure as code [IaC]). Just as DCE is a collection of individual systems that work together to support a resource or provide a service, so too is SDN a collection of hardware and software elements designed to virtualize networking management and control. Please see Chapter 11 for details on SDN.

Virtualized Systems

Virtualization technology is used to host one or more OSs within the memory of a single host computer or to run applications that are not compatible with the host OS. This mechanism allows virtually any OS to operate on any hardware. It also allows multiple OSs to work simultaneously on the same hardware.

Organizations may implement virtualization technologies due to the huge cost savings available. For example, an organization may be able to reduce 100 physical servers to just 10 physical servers, with each physical server hosting 10 virtual servers. This reduces HVAC costs, power costs, and overall operating costs.

The *hypervisor*, also known as the *virtual machine monitor/manager (VMM)*, is the component of virtualization that creates, manages, and operates the virtual machines. The computer running the hypervisor is known as the host OS, and the OSs running within a hypervisor-supported virtual machine are known as guest OSs or virtualized systems.

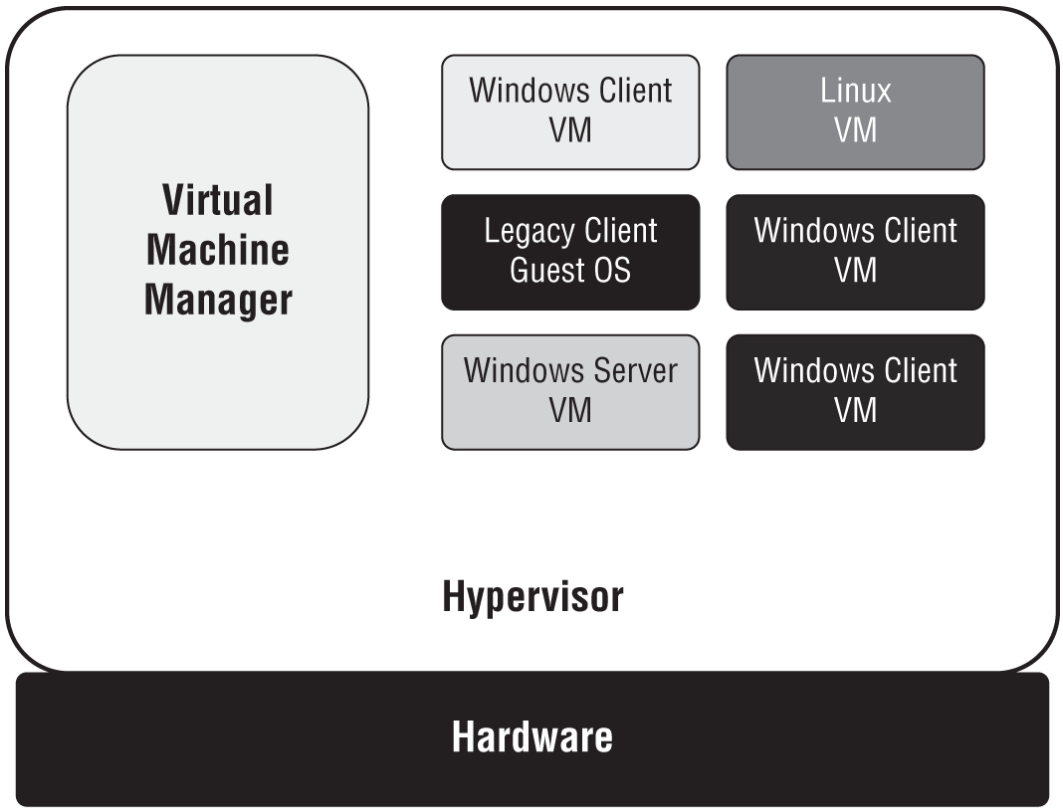
A *type I hypervisor* is a native or bare-metal hypervisor (Figure 9.3, top). In this configuration, there is no host OS; instead, the hypervisor installs directly onto the hardware where the host OS would normally reside. Type 1 hypervisors are often used to support server virtualization. This allows for maximization of the

hardware resources while eliminating any risks or resource reduction caused by a host OS.

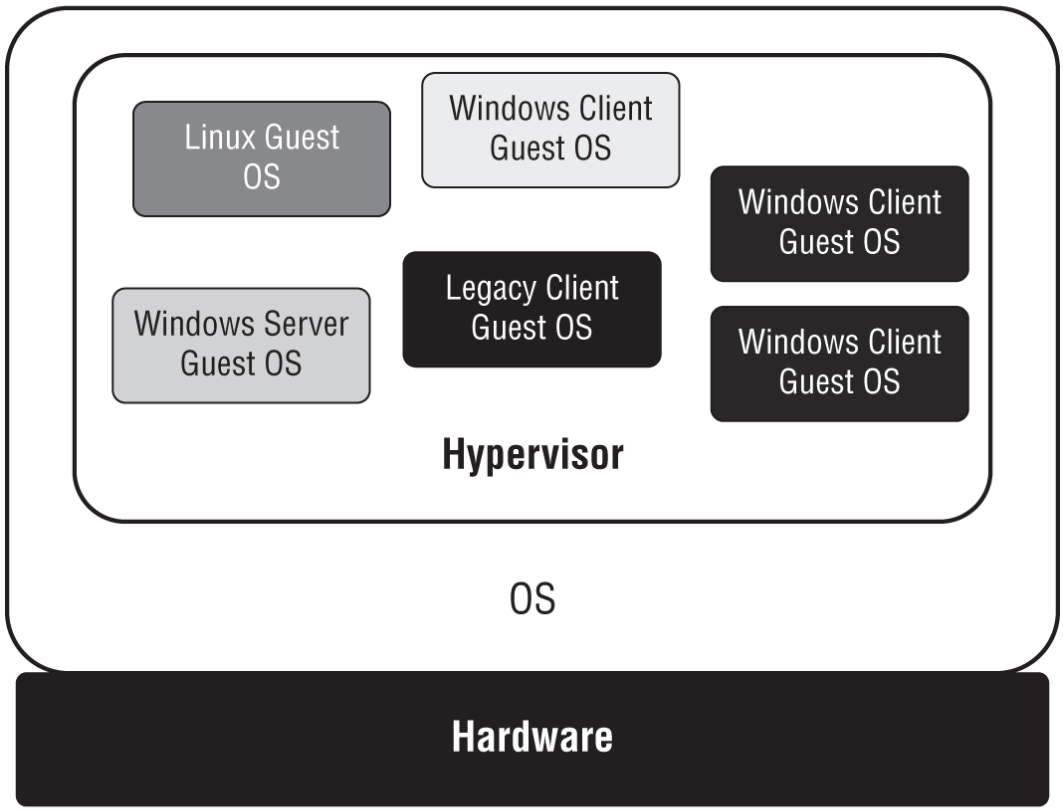
A *type II hypervisor* is a hosted hypervisor (Figure 9.3, bottom). In this configuration, a standard regular OS is present on the hardware, and then the hypervisor is installed as another software application. Type II hypervisors are often used in relation to desktop deployments, where the guest OSs offer safe sandbox areas to test new code, allow the execution of legacy applications, support apps from alternate OSs, and provide the user with access to the capabilities of a host OS.

Cloud computing is a natural extension and evolution of virtualization, the Internet, distributed architecture, and the need for ubiquitous access to data and resources. However, it does have some potential security issues, including privacy concerns, regulation compliance difficulties, use of open versus closed source solutions, adoption of open standards, and whether or not cloud-based data is actually secured (or even securable). See Chapter 16 for details on cloud computing.

Virtualization has several benefits, such as launching individual instances of virtual servers or services as needed, real-time scalability, and running the exact OS version needed for a specific application. Virtualized servers and services are indistinguishable from traditional servers and services from a user's perspective. Recovery from damaged, crashed, or corrupted virtual systems is often quick, simply consisting of replacing the virtual system's main hard drive file with a clean backup version and then relaunching it. Virtualization can also provide a reasonably secure means to continue to operate *end of life (EOL)* and *end of service life (EOSL)/end of support (EOS)* OSs to support legacy business applications.



Type I hypervisor



Type II hypervisor

FIGURE 9.3 Types of hypervisors

Elasticity refers to the flexibility of virtualization and cloud solutions (see Chapter 16) to expand or contract resource utilization based on need. In relation to virtualization, host elasticity means additional hardware hosts can be booted when needed and then used to distribute the workload of the virtualized services over the newly available capacity. As the workload becomes smaller, you can pull virtualized services off unneeded hardware so that it can be shut down to conserve electricity and reduce heat. Elasticity can also refer to the ability of a VM/guest OS to take advantage of any unused hardware resources on the fly as needed, but then release those resources when they are not needed. For example, a hardware host supporting five VM-based guest OSs may have over 30 percent CPU computational capacity unused. If a process-intensive application is launched within one of the VMs, the additional hardware host CPU capacity could be consumed; then once the application completes its intensive work task, the resource would be released. Elasticity has been a common capability of classic stand-alone systems for decades, but now with virtualization, the use of resources is shared among more than just a few processes—it can span across multiple VMs on the same hardware host as well as potentially across numerous hardware hosts.

It is also important to understand scalability in relation to elasticity. These terms are similar, but they are describing different concepts. Elasticity is the expansion or contraction of resources to meet current processing needs (especially in cloud computing), whereas scalability is the ability to take on more work or tasks. Usually, scalability is both a software and hardware characteristic that can handle more tasks or workloads, whereas elasticity is a hardware (physical or virtual) or platform characteristic where resources are optimized to meet the demands of current tasks. Usually, scalability is considered a long-term characteristic, while elasticity is more short-term. An elastic system can adjust resource consumption based on current processing needs, whereas a scalable system can be expanded to

handle more processing over time. Thus, a scalable system must also be elastic, but an elastic system does not need to be scalable.

In relation to security, virtualization offers several benefits. It is often easier and faster to make backups of entire virtual systems than the equivalent native hardware-installed system. Snapshots (aka checkpoints) are backups of virtual machines. Plus, when there is an error or problem, the virtual system can be replaced by a snapshot backup in minutes. Malicious code compromise or infection of virtual systems rarely affects the host OS due to the hypervisor's VM-to-VM and VM-to-host isolation. This allows for safe testing and experimentation.

Virtualization is used for a wide variety of new architectures and system design solutions. Locally (or at least within an organization's private infrastructure), virtualization can be used to host servers, client OSs, limited user interfaces (i.e., virtual desktops), applications, and more.

Virtual Software

A *virtual application* or *virtual software* is a software product deployed in such a way that it is fooled into believing it is interacting with a full host OS. A virtual (or virtualized) application has been packaged or encapsulated so that it can execute but operate without full access to the host OS. A virtual application is isolated from the host OS so that it cannot make any direct or permanent changes to the host OS. Any changes, such as file writes, configuration file or registry modifications, or system setting alterations, are intercepted by the isolation manager and recorded (typically into a single file). This allows the contained software to perceive it has interaction with the OS, without that interaction actually taking place. Thus, the virtualized application executes just like any regularly installed application, but it is only interacting and changing with a virtual representation of the OS, not the actual OS (aka sandboxing).

In many cases, operating an application in a software virtualization tool can effectively transform an installed application into a portable application. This means the

application's encapsulation and file can be moved to another OS (with the same software virtualization product), where it can execute. It may also be possible to place the application's encapsulation onto removable media and be able to execute the software from a portable storage device plugged into another computer system.

Some software virtualization solutions enable applications from one OS to be operated on another. For example, Wine allows some Windows software products to be executed on Linux.

The concept of software virtualization has evolved into its own virtualization derivative concept known as *containerization*, which is covered in a later section, “Containerization.”

Virtualized Networking

The concept of OS virtualization has given rise to other virtualization topics, such as virtualized networks. A *virtualized network* or *network virtualization* is the combination of hardware and software networking components into a single integrated entity. The resulting solution allows for software control over all network functions: management, traffic shaping, address assignment, and so on. A single management console or interface can be used to oversee every aspect of the *virtual network*, a task that required physical presence at each hardware component in the past. Virtualized networks have become a popular means of infrastructure deployment and management by corporations worldwide. They allow organizations to implement or adapt other interesting network solutions, including SDNs, virtual SANs, guest OSs, and port isolation.

Custom *virtual network segmentation* can be used in relation to virtual machines to make guest OSs members of the same network division as that of the host, or guest OSs can be placed into alternate network divisions. A virtual machine can be made a member of a different network segment from that of the host or placed into a network that only exists virtually and does not relate to the physical network media (effectively an SDN; see Chapter 11).

Software-Defined Everything

Virtualization extends beyond just servers and networking. *Software-defined everything (SDx)* refers to a trend of replacing hardware with software using virtualization. SDx includes virtualization, virtualized software, virtual networking, containerization, serverless architecture (Chapter 16), XaaS (Chapter 16), infrastructure as code (IaC), SDN (Chapter 11), virtual storage area network (VSAN) (Chapter 11), software-defined storage (SDS) (Chapter 11), VDI, VMI, SDV, and software-defined data center (SDDC).

The SDx examples that are not defined elsewhere (either in this chapter or in Chapter 11) are discussed here.

Virtual desktop infrastructure (VDI) is a means to reduce the security risk and performance requirements of end devices by hosting desktop/workstation OS virtual machines on central servers that are remotely accessed by users. Thus, VDI is also known as a virtual desktop environment (VDE). Users can connect to the server to access their desktop from almost any system, including from mobile devices. Persistent virtual desktops retain a customizable desktop for the user. Nonpersistent virtual desktops are identical and static for all users. If a user makes changes, the desktop reverts to a known state after the user logs off. (See the discussion of static systems earlier in this chapter under “Static Systems.”)

VDI has been adopted into mobile devices and has already been widely used in relation to tablets and laptop computers. It is a means to retain storage control on central servers, gain access to higher levels of system processing and other resources, and allow lower-end devices access to software and services beyond their hardware's capacity. This has led to *virtual mobile infrastructure (VMI)*, where the OS of a mobile device is virtualized on a central server. Thus, most of the actions and activities of the traditional mobile device no longer occur on the mobile device itself. This remote virtualization allows an organization greater control and security than when using a standard mobile device platform. It

can also enable personally owned devices to interact with the VDI without increasing the risk profile.

A *thin client* is a computer or mobile device with low to modest capability or a virtual interface that is used to remotely access and control a mainframe, virtual machine, VDI, or VMI. Thin clients were common in the 1980s when most computation took place on a central mainframe computer. Today, thin clients are being reintroduced as a means to reduce the expenses of high-end endpoint devices where local computation and storage are not required or are a significant security risk. A thin client can be used to access a centralized resource hosted on-premises or in the cloud. All processing/storage is performed on the server or central system, so the thin client provides the user with display, keyboard, and mouse/touchscreen functionality.

Software-defined visibility (SDV) is a framework to automate the processes of network monitoring and response. The goal is to enable the analysis of every packet and make deep intelligence-based decisions on forwarding, dropping, or otherwise responding to threats. SDV is intended to benefit companies, security entities, and managed service providers (MSPs). The goal of SDV is to automate detection, reaction, and response. SDV provides security and IT management with oversight into all aspects of the company network, both on-premises and in the cloud, with an emphasis on defense and efficiency. SDV is another derivative of IaC.

Software-defined data center (SDDC) or *virtual data center (VDC)* is the concept of replacing physical IT elements with solutions provided virtually, and often by an external third party, such as a cloud service provider (CSP). SDDC is effectively another XaaS (i.e., anything as a service, i.e., a cloud technology) concept, namely *IT as a service (ITaaS)*. It is similar to infrastructure as a service (IaaS); thus, some claim it is nothing more than a marketing or advertising term of misdirection.

To explore SDx further, you will find a wealth of articles at: sdxcentral.com, and you might want to start with “What Is Software Defined Everything – Part 1: Definition of SDx” at

www.sdxcentral.com/cloud/definitions/software-defined-everything-sdx-part-1-definition.

Virtualization Security Management

The primary software component in virtualization is a hypervisor. The hypervisor manages the VMs, virtual data storage, and virtual network components. As an additional layer of software on the physical server, it represents an additional attack surface. If an attacker can compromise a physical host, the attacker can potentially access all of the virtual systems hosted on the physical server. Administrators often take extra care to ensure that virtual hosts are hardened.

Although virtualization can simplify many IT concepts, it's important to remember that many of the same basic security requirements still apply. Virtualization doesn't lessen the security management requirements of an OS. Thus, patch management is still essential. For example, each VM's guest OS still needs to be updated individually. Updating the host system doesn't update the guest OSs. Also, don't forget that you need to keep the hypervisor updated as well.

When using virtualized systems, it's important to protect the stability of the host. This usually means avoiding using the host for any purpose other than hosting the virtualized elements, especially in a server-focused deployment. If host availability is compromised, the availability and stability of the virtual systems are also compromised.

Additionally, organizations should maintain backups of their virtual assets. Many virtualization tools include built-in tools to create full backups of virtual systems and create periodic snapshots, allowing relatively easy point-in-time restores.

Virtualized systems should be security tested. The virtualized OSs can be tested in the same manner as hardware installed OSs, such as with vulnerability assessment and penetration testing.

VM sprawl occurs when an organization deploys numerous virtual machines without an overarching IT management or

security plan in place. Although VMs are easy to create and clone, they have the same licensing and security management requirements as a metal-installed OS. Uncontrolled VM creation can quickly lead to a situation where manual oversight cannot keep up with system demand. To prevent or avoid VM sprawl, a policy for developing and deploying VMs must be established and enforced. This should include establishing a library of initial or foundation VM images that are to be used to develop and deploy new services. In some instances, VM sprawl relates to the use of lower-powered equipment that results in poorly performing VMs. VM sprawl is a virtual variation of server sprawl and could allow for virtual shadow IT.

Server Sprawl and Shadow IT

Server sprawl or system sprawl is the situation where numerous underutilized servers are operating in your organization's server room. These servers are taking up space, consuming electricity, and placing demands on other resources, but their provided workload or productivity does not justify their presence. This can occur if an organization purchases cheap lower-end hardware in bulk instead of selecting optimal equipment for specific use cases.

Somewhat related to server sprawl is shadow IT. *Shadow IT* is a term used to describe the IT components (physical or virtual) deployed by a department without the knowledge or permission of senior management or the IT group. The existence of shadow IT is often due to complex bureaucracy that makes the acquisition of needed equipment overly difficult and time-consuming. Other terms that might be used to refer to shadow IT include embedded IT, feral IT, stealth IT, hidden IT, secret IT, and client IT.

Shadow IT usually does not follow company security policy, and it might not be kept current and updated with patches. Shadow IT often lacks proper documentation, is not under consistent oversight and control, and may not be reliable or fault-tolerant. Shadow IT greatly increases the risk of disclosure of sensitive, confidential, proprietary, and personal information to unauthorized insiders and outsiders. Shadow IT can be composed of physical devices, virtual machines, or cloud services.

VM escaping occurs when software within a guest OS is able to breach the isolation-protection provided by the hypervisor to violate the container of other guest OSs or to infiltrate a host OS. Several VM escape vulnerabilities have been discovered in a variety of hypervisors. Fortunately, the vendors have been fast to release patches. For example, Virtualized Environment Neglected Operations Manipulation (VENOM) (CVE-2015-3456) was able to

breach numerous VM products that employed a compromised open source virtual floppy disk driver to allow malicious code to jump between VMs and even access the host.

VM escaping can be a serious problem, but steps can be implemented to minimize the risk. First, keep highly sensitive systems and data on separate physical machines. An organization should already be concerned about over-consolidation resulting in a single point of failure; running numerous hardware servers so that each supports a handful of guest OSs helps with this risk. Keeping enough physical servers on hand to maintain physical isolation between highly sensitive guest OSs will further protect against a VM escape exploit. Second, keep all hypervisor software current with vendor-released patches. Third, monitor attack, exposure, and abuse indexes for new threats to your environment.

To search for, locate, or research vulnerabilities, exploits, and attacks (whether related to virtualization or not), use exploit-db.com, cve.org, and nvd.nist.gov.

Containerization

Containerization is the next stage in the evolution of the virtualization trend for both internally hosted systems and cloud providers and services. A virtual machine–based system uses a hypervisor installed onto the bare metal of the host server and then operates a full guest OS within each virtual machine, and each virtual machine often supports only a single primary application. This is a resource-wasteful design and reveals its origins as separate physical machines.

Containerization or *OS-virtualization* is based on the concept of eliminating the duplication of OS elements in a virtual machine. Instead, each application is placed into a container that includes only the actual resources needed to support the enclosed application, and the common or shared OS elements are then part

of the container engine. Some deployments (such as Docker and Kubernetes) eliminate the hypervisor altogether and use a collection of common binaries and libraries for the containers to call upon when needed. Containerization is able to provide 10 to 100 times more application density per physical server than that provided by traditional hypervisor virtualization solutions.

Application cells or *application containers* (Figure 9.4) are used to virtualize software so that they can be ported to almost any OS.

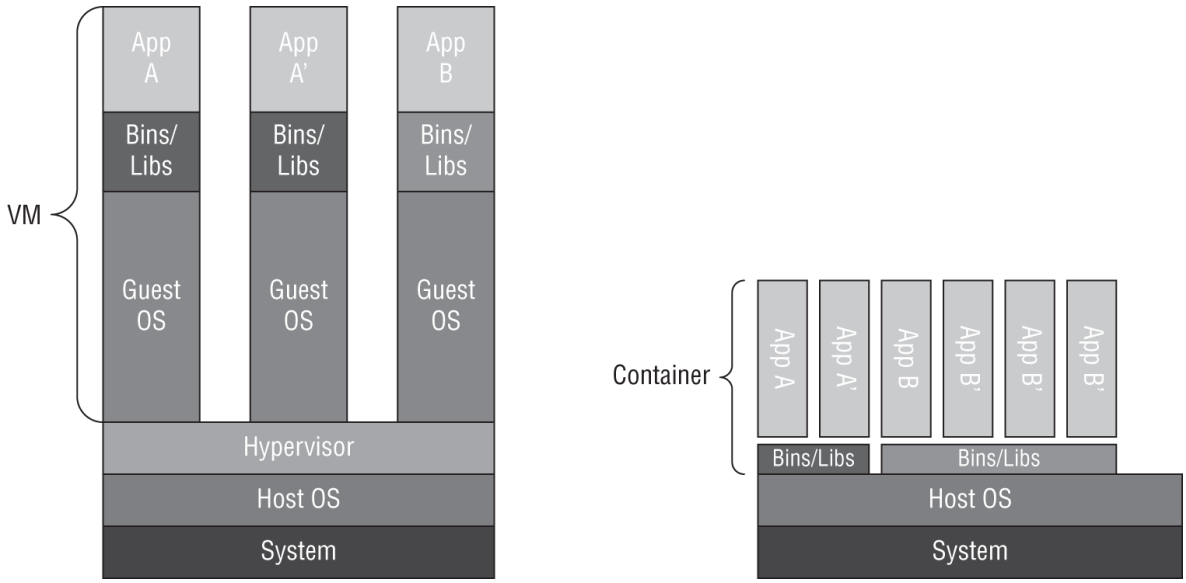


FIGURE 9.4 Application containers versus a hypervisor

There are many different technological solutions that are grouped into the concept of containerization. Some containerization solutions allow for multiple concurrent applications within a single container, whereas others are limited to one per container. Many containerization solutions allow for customization of how much interaction applications in separate containers are allowed.

Mobile Devices

A *mobile device* is anything with a battery (unless you also want to include things that are field powered, solar powered, etc., so generally anything that does not need a power cord to operate). However, we mostly discuss issues related to smartphones, tablets, or portable computers (i.e., notebooks and laptops). It may be tempting to only consider smartphones in relation to

mobile device questions, but you should also consider the question in regard to a laptop computer, a tablet, and maybe even a smart watch or fitness tracker. These other perspectives may assist you in answering the question correctly.

Some mobile devices have less than typical default or even available security features because they often run stripped-down OSs or custom mobile OSs without the long history of security improvements found in popular PC OSs. Many of the “features” of a mobile device can be the focus of attacks, compromises, and intrusions. Extra care and attention need to be paid to any mobile device's security for both personal and business/work use.

Smartphones and other mobile devices present an ever-increasing security risk as they become more and more capable of interacting with the Internet as well as corporate networks. These devices have internal memory and may support removable memory cards that can hold a significant amount of data. Additionally, many devices include applications that allow users to read and manipulate different types of files and documents. When personally owned devices are allowed to enter and leave a secured facility without limitation, oversight, or control, the potential for harm is significant.

Malicious insiders can bring in malicious code from outside on various storage devices, including mobile phones, audio players, digital cameras, memory cards, optical discs, and Universal Serial Bus (USB) drives. These same storage devices can be used to leak or steal internal confidential and private data to disclose it to the outside world. Malicious insiders can execute malicious code, visit dangerous websites, or intentionally perform harmful activities.

Mobile devices often contain sensitive data such as contacts, text messages, email, scheduling information, and possibly notes and documents. Any mobile device with a camera feature can take photographs of sensitive information or locations. The loss or theft of a mobile device could mean the compromise of personal and/or corporate secrets.

Additionally, mobile devices aren't immune to eavesdropping. With the right type of sophisticated equipment, most mobile phone conversations can be tapped into—not to mention the fact that anyone nearby can hear you talking. Employees should be coached to be discreet about what they discuss over mobile phones in public spaces.

Android and iOS

Two of the most widely used device OSs are Android and iOS.

Android

Android is a mobile device OS based on Linux, which was acquired by Google in 2005. The Android source code is made open source through the Apache license, but most Android devices also include proprietary software. Although it's mostly intended for use on phones and tablets, Android is being used on a wide range of devices, including televisions, game consoles, digital cameras, microwaves, watches, e-readers, cordless phones, and ski goggles.

The use of Android in phones and tablets allows for a wide range of user customization: you can install Google Play Store apps as well as apps from unknown external sources (such as the Amazon Appstore), and many devices support the replacement of the default version of Android with a customized or alternate version. However, when Android is used on other devices, it can be implemented as something closer to a static system.

Whether static or not, Android has numerous security vulnerabilities. These include exposure to malicious apps, running scripts from malicious websites, and allowing insecure data transmissions.

Improvements are made to Android security as new updates are released. Users can adjust numerous configuration settings to reduce vulnerabilities and risks. Also, users may be able to install apps that add additional security features to the platform.

Security-Enhanced Android (SEAndroid) is a security improvement for Android. SEAndroid is a framework to integrate elements of Security-Enhanced Linux into Android devices. These improvements include adding support for mandatory access control (MAC) and middleware mandatory

access control (MMAC), reducing privilege daemon vulnerabilities, sandboxing and isolating apps, blocking app privilege escalation, enabling app privilege adjustments both during installation and at runtime, and defining a centralized security policy that can be scrutinized.

iOS

iOS is the mobile device OS from Apple that is standard on the *iPhone*. iOS isn't licensed for use on any non-Apple hardware. Thus, Apple is in full control of the features and capabilities of iOS. However, iOS is not really an example of a static environment, because users can install apps from the Apple App Store (although it can be argued that iOS is a static OS).

Mobile Device Security Features

A wide range of security features may be available on mobile devices, such as portable computers, tablets, and smartphones. Not all mobile devices have good security features. Be sure to consider the security options of a new device before you make a purchase decision. But even if security features are available, they're of no value unless they're enabled and properly configured. A security benefit is gained only when the security function is in force. Be sure to check that all desired security features are operating as expected on any device allowed to connect to the organization's network or enter the organization's facility.

The following sections discuss various examples of on-device security features that are often present on or available for mobile devices.

Mobile Device Management

Administrators register employee devices with a mobile device management system. *Mobile device management (MDM)* is a software solution to the challenging task of managing the myriad mobile devices that employees use to access company resources.

The MDM system monitors and manages mobile devices and ensures that they are kept up-to-date. The goals of MDM are to improve security, provide monitoring, enable remote management, and support troubleshooting. Many MDM solutions support a wide range of devices and can operate across many service providers. You can use MDM to push or remove apps, manage data, and enforce configuration settings both over the air (across a carrier network) and over Wi-Fi connections. MDM can be used to manage company-owned devices as well as personally owned devices.

Unified endpoint management (UEM) is a type of software tool that provides a single management platform to control mobile, PC, IoT, wearables, ICS, and other devices. UEM is intended to replace MDM and enterprise mobility management (EMM) products by combining the features of numerous products into one solution.

Device Authentication

Authentication on or to a mobile device is often fairly simple, especially for mobile phones and tablets. This is known as *device authentication*. However, a swipe or pattern access shouldn't be considered true authentication. Whenever possible, use a password, provide a personal identification number (PIN), offer your eyeball or face for recognition, scan your fingerprint, provide a USB key, or use a proximity device such as a near-field communication (NFC) or radio-frequency identification (RFID) ring or tile. These means of device authentication are much more difficult for a thief to bypass if properly implemented. It's also prudent to combine device authentication with device encryption to block access to stored information via a connection cable.

Retina-, iris-, face-, and fingerprint-based authentication are all examples of biometrics. See Chapter 13, “Managing Identity and Authentication,” for the full discussion of biometrics or the “something you are” authentication factor.

A strong password would be a great idea on a phone or other mobile device if locking the phone provided true security. But most mobile devices aren't that secure, so even with a strong password, the device may still be accessible over Bluetooth, wireless, or a USB cable. If a specific mobile device blocked access to the device when the system lock was enabled, this would be a worthwhile feature to set to trigger automatically after a period of inactivity or manual initialization (often related to screen lock). This benefit is usually obtained when you enable both a device password and storage encryption.

When accessing an online website, service, or cloud offering from a mobile device, a form of multifactor authentication (MFA) may be implemented by combining your user credentials with context-aware authentication. Context-aware authentication evaluates the origin and context of a user's attempt to access a system. If the user originates from a known trusted system, such as a system inside the company facility or the same personal mobile device, then a low-risk context is present and a modest level of authentication is mandated for gaining access. If the context and origin of the user is from an unknown device and/or external/unknown location, the context is high risk. The authentication system will then demand that the user traverse a more complex multifactor authentication gauntlet to gain access. Context-aware authentication is thus an adaptive authentication that may be able to reduce the burden of authentication during low-risk scenarios but thwart impersonation attempts during high-risk scenarios.

Full-Device Encryption

Some mobile devices, including portable computers, tablets, and mobile phones, may offer *full-device encryption (FDE)*. Many mobile devices are either pre-encrypted or can be encrypted by the user/owner. Once a mobile device is encrypted, the user's data is protected whenever the screen is locked, which causes the physical data port on the device to be deactivated. This prevents unauthorized access to data on the device through a physical cable connection as long as the screen remains locked.

If most or all of the storage media of a device can be encrypted, this is usually a worthwhile feature to enable. However, encryption isn't a guarantee of protection for data, especially if the device is stolen while unlocked or if the system itself has a known backdoor attack vulnerability.

Communication Protection

Voice encryption may be possible on mobile devices when Voice over Internet Protocol (VoIP) services are used. VoIP service between computer-like devices is more likely to offer an encryption option than VoIP connections to a traditional landline phone or typical mobile phone. When a voice conversation is encrypted, attempting to listen in on the communications becomes almost worthless because the contents of the conversation are undecipherable.

This concept of communication protection should be applied to any type of transmission, whether video, text, or data. There are numerous apps that provide encrypted communications, many using standard and well-respected cryptography solutions, such as the Signal protocol (see Chapters 6 and 7 for more on encryption).

Remote Wiping

Remote wipe or *remote sanitization* is to be performed if a device is lost or stolen. A remote wipe lets you delete all data and possibly even configuration settings from a device remotely. The wipe process can be triggered over mobile phone service or sometimes over any Internet connection (such as Wi-Fi). However, a remote wipe isn't a guarantee of data security. The wiping trigger signal might not be received by the device. Thieves may be smart enough to prevent connections that would trigger the wipe function while they dump out the data. This could be accomplished by removing the subscriber identity module (SIM) card, deactivating Wi-Fi, and/or placing the device in a Faraday cage.

Additionally, a remote wipe is mostly a deletion operation and resetting the device back to factory conditions. The use of an undelete or data recovery utility can often recover data on a wiped device. To ensure that a remote wipe destroys data beyond recovery, the device should be encrypted (aka full-device encryption [FDE]). Thus, the undelete operation would only be recovering encrypted data, which the attacker should be unable to decipher.

Screen Locks

A *screen lock* is designed to prevent someone from casually picking up and being able to use your phone or mobile device. However, most screen locks can be unlocked by swiping across the screen or drawing a pattern. Neither of these is truly a secure operation. These easy-bypass options may be the default on the device but should be changed to something more secure and resistive of unauthorized access, such as a PIN, password, or biometric.

Screen locks may have workarounds on some devices, such as accessing the phone application through the emergency calling feature. And a screen lock doesn't necessarily protect the device if a malicious actor connects to it over Bluetooth, wireless, or a USB cable.

Screen locks are often triggered after a timeout period of nonuse. Most devices can be configured to auto-trigger a password-protected screen lock if the system is left idle for a few minutes. Similarly, many tablets and mobile phones can be set to trigger a screen lock and dim or turn off the display after a set time period, such as 30 seconds. The lockout feature ensures that if you leave your device unattended or it's lost or stolen, it will be difficult for anyone else to be able to access your data or applications. To unlock the device, you must enter valid credentials.

Device Lockout

Lockout on a mobile device is similar to account lockout on a company workstation. When a user fails to provide their credentials after repeated attempts, the account or device is deactivated (i.e., locked out) for a period of time or until an administrator clears the lockout flag.

Mobile devices may offer a *device lockout* feature, but it's only effective if a screen lock has been configured. Otherwise, a simple screen swipe to access the device doesn't provide sufficient security, because an authentication process doesn't occur. Some devices trigger ever longer delays between access attempts as a greater number of authentication failures occur. Some devices

allow for a set number of attempts (such as three) before triggering a lockout that lasts minutes or hours. Other devices trigger a persistent lockout and require the use of a different account or master password/code to regain access to the device. Some devices may even have a maximum number of logon attempts (such as 10), before securely wiping all data on the device and resetting back to factory settings. Be sure to know the exact nature of a device's lockout mechanism before attempting to guess credentials; otherwise you might inadvertently trigger a security wipe.

GPS and Location Services

The *Global Positioning System (GPS)* is a satellite-based geographical location service. Many mobile devices include a GPS chip to support and benefit from localized services, such as navigation, so it's possible to track those devices. The GPS chip itself is usually just a receiver of signals from orbiting GPS satellites. However, applications on the mobile device can record the GPS location of the device and then report it to an online service. You can use GPS tracking to monitor your own movements, track the movements of others (such as minors or delivery personnel), or track down a stolen device. But for GPS tracking to work, the mobile device must have Internet or wireless phone service over which to communicate its location information. Apps are able to provide location-based services as well as reveal the location of the device (and thus its user/owner) to third parties (sometimes without consent). This risk needs to be evaluated in regard to the organizational security policy and relative location-based risks.

Geolocation data is commonly used in navigation tools, authentication services, and many location-based services, such as offering discounts or coupons to nearby retail stores.

Location-based authorization policies for controlling access can be used to grant or deny resource access based on where the subject is located. This might be based on whether the network connection is local wired, local wireless, or remote. Location-based policies can also grant or deny access based on MAC

address, IP address, OS version, patch level, and/or subnet in addition to logical or geographical location, which is a feature of both network access control (NAC) (see Chapter 11) and context-aware authentication. Location-based policies should be used only in addition to standard authentication processes, not as a replacement for them.

Geotagging is the ability of a mobile device to include details about its location in any media created by the device, such as photos, videos, and social media posts. Mobile devices with location services enable embedding geographical location in the form of latitude and longitude as well as date/time information into media created on the device. This allows an adversary (or angry ex) to view photos from social networking or similar sites and determine exactly when and where a photo was taken.

Geotagging can be used for nefarious purposes, such as determining when a person normally performs routine activities. Once a geotagged photo has been uploaded to the Internet, a potential cyber stalker may have access to more information than the uploader intended. This is prime material for security-awareness briefs for end users.

Other Location Services

The most commonly discussed location service of a mobile device is that of GPS. However, it is important to recognize that there are at numerous location determination services or capabilities in many mobile devices. These include *wireless positioning system (WiPS) or Wi-Fi positioning system (WFPS)*, cellular/mobile service tower triangulation, Bluetooth location services, and environmental sensors.

WiPS uses the known location of wireless access points/base stations to determine a mobile device's location. WiPS is often used as a supplement to GPS when sufficient satellite signals are unavailable, such as when underground, inside buildings, or near tall structures.

Due to U.S. 911 regulations (which established E911), mobile devices can be located using mobile service tower triangulation. However, E911 location tracking is not as accurate as GPS.

iBeacon is a technology developed by Apple to track devices based on their Bluetooth device address and signal properties. Though originally designed to track people inside Apple stores, it is now used in many other contexts by a wide range of organizations to track devices via Bluetooth and their related user/owner.

Environmental sensors on many mobile devices include accelerometers, compasses, thermometers, altimeters (altitude sensors), and barometric pressure sensors. With this vast range of sensing data, if the initial location of a device is known or can be approximated, then its location at any future point in time can be determined if continuous sensor data is recorded.

It is also possible for a device to be located through its camera and microphones, but so far, this method is not as reliable as the others. This final concept measures light levels, intensity, and color to potentially determine if a device is outside or

inside and if it is located near a window, which, based on the time of day, may be able to determine the general area (such as a city) based on light levels caused by the sun's position in the sky. This can then be combined with monitoring of background noise via the microphone to further refine the location. But this requires extensive knowledge of regional sounds, a massive dataset of noise from across the globe, or access to real-time microphone networks or sensors.

Geofencing is the designation of a specific geographical area that is then used to implement features or trigger settings automatically on mobile devices. A geofence can be defined by GPS coordinates, WiPS, or the presence of or lack of a specific wireless signal. A device can be configured to enable or deactivate features based on a geofenced area, such as an onboard camera or the Wi-Fi capability.

Content Management

Content management is the control over mobile devices and their access to content hosted on company systems as well as the control of access to company data stored on mobile devices. Typically, an *MCM (mobile content management) system* is used to control company resources and the means by which they are accessed or used on mobile devices. An MCM can take into account a device's capabilities, storage availability, screen size, bandwidth limitations, memory (RAM), and processor capabilities when rendering or sending data to mobile devices.

The goal of a *content management system (CMS)* for mobile devices is to maximize performance and work benefits while reducing complexity, confusion, and inconvenience. An MCM may also be tied to an MDM to ensure secure use of company data.

A content filter, which may block access to resources, data, or services based on IP address, domain name, protocol, or keyword, is often implemented as a firewall service rather than an on-device mechanism. Therefore, content filtering is usually enforced by the network through which communication occurs.

Application Control

Application control or *application management* is a device-management solution that limits which applications can be installed onto a device. It can also be used to force specific applications to be installed or to enforce the settings of certain applications to support a security baseline or maintain other forms of compliance. Using application control can often reduce exposure to malicious applications by limiting the user's ability to install apps that come from unknown sources or that offer non-work-related features. This mechanism is often implemented by an MDM. Without application control, users could theoretically install malicious code, run data stealing software, operate apps that reveal location data, or not install business-necessary applications.

Application allow listing (also known as whitelisting) is a security option that prohibits unauthorized software from being able to execute. Allow listing is also known as *deny by default* or *implicit deny*. In application security, allow listing prevents any and all software, including malware, from executing unless it's on the preapproved exception list: the allow list. This is a significant departure from the typical device-security stance, which is to allow by default and deny by exception (also known as deny listing or block listing, also known as blacklisting). Deny listing allows anything and everything, both benign and malicious, to execute by default, unless it is added to the deny list, which prevents execution from that point forward.

Due to the growth of malware, an application allow listing approach is one of the few options remaining that shows real promise in protecting devices and data. However, no security solution is perfect, including allow listing. All known allow listing solutions can be circumvented with kernel-level vulnerabilities and application configuration issues.

Mobile application management (MAM) is similar to an MDM but focuses only on app management rather than managing the entire mobile device.

Push Notifications

Push notification services are able to send information to your device rather than having the device (or its apps) pull information from an online resource. Push notifications are useful in being notified about a concern immediately, but they can also be a nuisance if they are advertising or spam. Many apps and services can be configured to use push and/or pull notifications. Push notifications are mostly a distraction, but it is possible to perform social engineering attacks via these messages as well as distribute malicious code or links to abusive sites and services.

Push notifications are also a concern in browsers for both mobile devices and PCs. Another issue is that malicious or pernicious notifications may capture a user in a push locker. If the user denies agreement to a push prompt, it may redirect them to a subdomain where another push notification is displayed. If they deny again, then they are redirected again to yet another subdomain, to then see another push notification. This can be repeated indefinitely. Until your browser and/or host-based intrusion detection system (HIDS) can detect and respond to push lockers, the only response is to close/terminate the browser and not return to the same URL.

Third-Party Application Stores

The primary, default, and authorized application (aka app) stores of Apple App Store and Google Play Store are the official sources for apps for use on the typical or standard Apple and Android smartphone or device, respectively. *Third-party app stores* often have less rigorous security rules regarding hosting an app. On Android devices, simply enabling a single feature to install apps from unknown sources allows the use of third-party app stores (as well as sideloading; see the section “Sideloading,” later in this chapter). For Android devices, the Amazon Appstore is an example of a third-party app store. For Apple iOS devices, currently you are limited to the official Apple App Store unless you jailbreak or root the device (which is not usually a security recommendation).

When a mobile device is being managed by an organization, especially when using an MDM/UEM/MAM, most third-party sources of apps will be blocked. Such third-party app sources represent a significant increase in risk of data leakage or malware intrusion to an organizational network.

Storage Segmentation

Storage segmentation is used to artificially compartmentalize various types or values of data on a storage medium. On a mobile device, storage segmentation may be used to isolate the device's OS and preinstalled apps from user-installed apps and user data. Some MDMs/UEMs further impose storage segmentation to separate company data and apps from user data and apps. This allows for ownership and rights over user data to be retained by the user, while granting ownership and rights over business data (such as remote wiping) to the organization, even on devices owned by the employee.

With or without storage segmentation, risk can be reduced by minimizing the storage of nonessential data, sensitive data, and personal data (i.e., PII and PHI) on a device. So, even if a device is lost or stolen, the loss potential is kept to a minimum if there is little to no valuable data on the system for an adversary to gain access to.

Asset Tracking

Asset tracking is the management process used to maintain oversight over an inventory, such as deployed mobile devices. An asset-tracking system can be passive or active. Passive systems rely on the asset itself to check in with the management service on a regular basis, or the device is detected as being present in the office each time the employee arrives at work. An active system uses polling or pushing technology to send out queries to devices to elicit a response.

You can use asset tracking to verify that a device is still in the possession of the assigned authorized user. Some asset-tracking solutions can locate missing or stolen devices.

Some asset-tracking solutions expand beyond hardware inventory management and can oversee the installed apps, app usage, stored data, and data access on a device. You can use this type of monitoring to verify compliance with security guidelines or to check for exposure of confidential information to unauthorized entities.

Removable Storage

Many mobile devices support removable storage. Some devices support microSD cards, which can be used to expand available storage on a mobile device. However, most mobile phones require the removal of a back plate and sometimes removal of the battery to add or remove a storage card. Larger mobile phones, tablets, and laptop computers may support an easily accessible card slot on the side of the device.

Many mobile devices also support external USB storage devices, such as flash drives and external hard drives. Some may require a special on-the-go (OTG) cable. *USB On-The-Go (OTG)* is a specification that allows a mobile device with a USB port to act as a host and use other standard peripheral USB equipment, such as storage devices, mice, keyboards, and digital cameras. USB OTG is a feature that can be deactivated via MDM/UEM if it is perceived as a risk vector for mobile devices used within an organization.

In addition, mobile storage devices may provide Bluetooth- or Wi-Fi-based access to stored data through an onboard wireless interface.

Organizations need to consider whether the use of removable storage on portable and mobile devices is a convenient benefit or a significant risk vector. If the former, proper access limitations and use training are necessary. If the latter, then a prohibition of removable storage can be implemented via MDM/UEM.

Deactivating Unused Features

Although enabling security features is essential for them to have any beneficial effect, it's just as important to remove apps and

deactivate features that aren't essential to business tasks or common personal use. The wider the range of enabled features and installed apps, the greater the chance that an exploitation or software flaw will cause harm to the device and/or the data it contains. Following common security practices, such as hardening, reduces the attack surface of mobile devices.

Rooting or Jailbreaking

Rooting or *jailbreaking* (the special term for rooting Apple devices) is the action of breaking the digital rights management (DRM) security on the bootloader of a mobile device to be able to operate the device with root or full system privileges. Most mobile devices are locked in such a way as to restrict end-user activity to that of a limited user. But a root user can manipulate the OS, enable or deactivate hardware features, and install software applications that are not available to the limited user. Rooting may enable a user to change the core OS or operate apps that are unavailable in the standard app stores. However, this is not without its risks. Operating in rooted status also reduces security, since any executable also launches with full root privileges. Many forms of malicious code cannot gain footing on normal mode devices but can easily take root (pun intended) when the user has rooted or jailbroken their device.

Generally, an organization should prohibit the use of rooted devices on the company network or even access to company resources whenever possible.

It is legal to root a device if you fully own the device, if you are in a one- or two-year contract with a hardware fee, or if you are in a lease-to-own contract and you do not fully own the device until that contract is fulfilled. Legal root does not require a manufacturer, vendor, or telco to honor any warranty. In most cases, any form of system tampering, including rooting, voids your warranty. Rooting may also void your support contract or replacement contract. Rooting is actively suppressed by the telcos, many carriers, and some product vendors, Apple being the main example. A rooted device might be prohibited to operate over a telco network, access resources, download apps, or receive

future updates. Thus, though it is often legal to root a device, there are numerous consequences to consider prior to altering a mobile device in that manner.

"Bricking" refers to rendering a device completely nonfunctional or as useless as a brick, typically through a malfunction or intentional action. When a device is "bricked," it loses its normal functionality and becomes inoperable, often requiring significant efforts to restore its original state. This term is commonly used in the context of electronic devices such as smartphones, tablets, routers, or other hardware that may become unusable due to software errors, firmware corruption, or unauthorized modifications. "Bricking" has become less of a concern with modern devices, which often include a nonreplaceable recovery ROM in addition to its standard flashable firmware (i.e., updatable firmware).

Sideloading

Sideloading is the activity of installing an app on a device by bringing the installer file to the device through some form of file transfer or USB storage method. Most organizations should prohibit user sideloading, because it may be a means to bypass security restrictions imposed by an app store, application allow listing, or the MDM/UEM/MAM. An MDM/UEM/MAM-enforced configuration can require that all apps be digitally signed; this would eliminate sideloading and likely jailbreaking as well.

Custom Firmware

Mobile devices come preinstalled with a vendor- or telco-provided firmware or core OS. If a device is rooted or jailbroken, it can allow the user to install alternate custom firmware in place of the default firmware. Custom firmware may remove bloatware, add or remove features, and streamline the OS to optimize performance. You can find online discussion forums and communities, such as xda-developers.com and howardforums.com, that specialize in custom firmware for Apple and Android devices.

An organization should not allow users to operate mobile devices that have custom firmware unless that firmware is preapproved

by the organization.

Carrier Unlocking

Most mobile devices purchased directly from a telco are carrier locked. This means you are unable to use the device on any other telco network until the carrier lock is removed or carrier unlocked. Once you fully own a device, the telco should freely *carrier unlock* the phone, but you will have to ask for it specifically because they don't do so automatically. If you have an account in good standing and are traveling to another country with compatible telco service, you may be able to get a telco to carrier unlock your phone for your trip so that you can use another SIM card (or eSIM) for local telco services.

Having a device carrier unlocked is not the same as rooting. Carrier unlocked status only allows the switching of telco services (which is technically possible only if your device uses the same radio frequencies as the telco). A carrier unlocked device should not represent any additional risk to an organization; thus, there is likely no need for a prohibition of carrier unlocked devices on company networks.

Firmware Over-the-Air (OTA) Updates

Firmware over-the-air (OTA) updates are firmware updates that are downloaded from the telco or vendor over-the-air (via a data connection either provided by the carrier or via Wi-Fi). Generally, as a mobile device owner, you should install new firmware OTA updates onto a device once they become available. However, some updates may alter the device configuration or interfere with MDM/UEM restrictions. You should attempt to test new updates before allowing managed devices to receive them. You may have to establish a waiting period so that the MDM/UEM vendor can update their management product to properly oversee the deployment and configuration of the new firmware update. An organization's standard patch management, configuration management, and change management policies should be applied to mobile devices.

Credential Management

The storage of credentials in a central location is referred to as credential management. Given the wide range of Internet sites and services, each with its own particular logon requirements, using unique names and passwords can be a burden. *Credential management* solutions offer a means to securely store a plethora of credential sets. Often, these tools employ a master credential set (multifactor being preferred) to unlock the dataset when needed. Some credential-management options can even provide auto-login options for apps and websites.

A *password vault* is another term for a credential manager. These are often software solutions, sometimes hardware-based, sometimes local only, and sometimes using cloud storage. They are used to generate and store credentials for sites, services, devices, and whatever other secrets you want to keep private. The vault itself is encrypted and must be unlocked to regain access to the stored items. Most password vaults use Password-Based Key Derivation Function 2 (PBKDF2) or Bcrypt (see Chapter 7) to convert the vault's master password into a reasonably strong encryption key.

Text Messaging

Short Message Service (SMS), Multimedia Messaging Service (MMS), and Rich Communication Services (RCS) are all useful communication systems, but they also serve as an attack vector (such as smishing and SPIM, discussed in Chapter 2, “Personnel Security and Risk Management Concepts”). These testing and messaging services are primarily operated and supported by the telco providers. Texting can be used as an authentication factor known as SMS-based 2FA. SMS-based 2FA is better than single-factor password-only authentication, but it is not recommended if any other second-factor option is available. See Chapter 13 for more on SMS-based 2FA.

Many non-telco/non-carrier texting and messaging services are supported via apps on mobile devices. It is important to keep any messaging service app updated and restrict its use to nonsensitive content.

Mobile Device Deployment Policies

A number of deployment models are available for allowing and/or providing mobile devices for employees to use while at work and to perform work tasks when away from the office. A *mobile device deployment policy* must address the wide range of security concerns regarding the use of a PED in relation to the organization's IT infrastructure and business tasks.

Users need to understand the benefits, restrictions, and consequences of using mobile devices at work and for work. Reading and signing off on the BYOD, CYOD, COPE, COMS/COBO, etc., policy along with attending an overview or training program may be sufficient to accomplish reasonable awareness. These topics are covered in the next sections.

An alternative to allowing personal or business-provided mobile devices to interact with company resources directly would be to implement a VDI or VMI solution (see section “Software-Defined Everything” earlier this chapter).

Bring Your Own Device (BYOD)

Bring your own device (BYOD) is a policy that allows employees to bring their own personal mobile devices to work and may allow them to use those devices to connect to business resources and/or the Internet through the company network. Although BYOD may improve employee morale and job satisfaction, it increases security risk to the organization. If the BYOD policy is open-ended, any device may be allowed to connect to the company network. Not all mobile devices have sufficient security features, and thus such a policy allows noncompliant devices onto the production network. Employees likely retain full control over their devices, allowing them to disable or bypass security features imposed by the organization.

This is likely the least secure option for the organization since company data and applications will be on the personal mobile

device, it exposes the organization's network to malicious code from the PEDs, and the devices will have the widest range of variation and security capabilities (or more likely the lack of security capabilities). Additionally, this option potentially exposes the worker's PII on the device to the organization.

Choose Your Own Device (CYOD)

The concept of *choose your own device (CYOD)* provides users with a list of approved devices from which to select the device to implement. A CYOD policy can be implemented so that employees purchase their own devices from the approved list (a BYOD variant).

This option attempts to keep the expense of devices the responsibility of workers rather than the organization, but it often results in much more complex and challenging situations. For example, how will it handle a situation wherein a worker has already spent considerable money on a device that is not on the preapproved list? Will they be given money to purchase an approved device? What about the person who paid for an approved device—will the company reimburse them because they already paid for someone else's device? What about the person who decides they don't want to use a mobile device for work activities—will they be paid the funds anyway, allowing them to treat it as a paycheck bonus?

Also, this option has the same security issues as COPE: the potential for malware transfer and the comingling of business and personal data on the same device.

Corporate-Owned, Personally Enabled (COPE)

Corporate-owned, personally enabled (COPE) is a mobile policy where the organization purchases devices and provides them to employees. Each user is then able to customize the device and use it for both work activities and personal activities. COPE allows the organization to select exactly which devices are to be allowed on the organizational network—specifically only those devices that can be configured into compliance with the security policy.

This option reduces the mobile devices to those preselected by the organization and that have the minimum security capabilities mandated by company security policy. However, this option still has the risk of exposing company data through user error, exposes the organization to malware via the device, and puts worker PII at risk of being accessed by the organization.

Corporate-Owned Mobile Strategy (COMS)

A *corporate-owned mobile strategy (COMS)* or *corporate-owned, business-only (COBO)* strategy is when the company purchases the mobile devices that can support security compliance with the security policy. These devices are to be used exclusively for company purposes, and users should not perform any personal tasks on the devices. This often requires workers to carry a second device for personal use.

This is the best option for both the organization as well as the individual worker. The option maintains clear separation between work activities and personal activities, since the device is for work use exclusively. This option protects company resources from personal activity risks, and it protects personal data from unauthorized or unethical organizational access. Yes, it is a hassle to carry a second device for personal activities, but that inconvenience is well worth the security benefits for both parties.

Mobile Device Deployment Policy Details

No matter which mobile device deployment policy you select and implement, your policy needs to address the many device security features listed earlier in this section. You can ensure this by defining required features and how they are to be configured for company security policy compliance. The mobile device deployment policy must also address several other concerns that are operational, legal, and logistic-based as well. These are discussed in the following sections.

Data Ownership

When a personal device is used for business tasks, commingling of personal data and business data is likely to occur. Some

devices can support storage segmentation, but not all devices can provide data-type isolation. Establishing data ownership can be complicated. For example, if a device is lost or stolen, the company may wish to trigger a remote wipe, clearing the device of all valuable information. However, the employee will often be resistant to this, especially if there is any hope that the device will be found or returned. A wipe may remove all business and personal data, which may be a significant loss to the individual—especially if the device is recovered, because then the wipe would seem to have been an overreaction. Clear policies about data ownership should be established. Some MDM/UEM solutions can provide data isolation/segmentation and support business data sanitization without affecting personal data.

The mobile device deployment policy regarding data ownership should address backups for mobile devices. Business data and personal data should be protected by a backup solution—either a single solution for all data on the device or separate solutions for each type or class of data. Backups reduce the risk of data loss in the event of a remote-wipe event as well as device failure or damage.

Support Ownership

When an employee's mobile device experiences a failure, a fault, or damage, who is responsible for the device's repair, replacement, or technical support? The mobile device deployment policy should define what support will be provided by the company and what support is left to the individual and, if relevant, their service provider.

Patch and Update Management

The mobile device deployment policy should define the means and mechanisms of secure patch management and update management for a personally owned mobile device. Is the user responsible for installing updates? Should the user install all available updates? Should the organization test updates prior to on-device installation? Are updates to be handled over the air (via service provider) or over Wi-Fi? Are there versions of the mobile

OS that cannot be used? What patch or update level is required? These issues should be addressed both for the primary OS of the device and for all apps installed on the device.

Security Product Management

The mobile device deployment policy should dictate whether antivirus, antimalware, antispyware scanners, firewalls, HIDS, or other security tools are to be installed on mobile devices. The policy should indicate which products/apps are recommended for use, as well as the settings for those solutions.

Forensics

The mobile device deployment policy should address forensics and investigations related to mobile devices. Users need to be aware that in the event of a security violation or a criminal activity, their devices might be involved. An investigation would mandate gathering evidence from those devices. Some processes of evidence gathering can be destructive, and some legal investigations require the confiscation of devices. An owner of a personal device may refuse access to the contents of their device, even when that content is, in theory, the property of the organization. A company-owned device could have a secondary account, a master password, or a remote management tool preinstalled that would grant the organization the ability to access the device's contents without the user's consent.

In all legal matters, including mobile device forensics and privacy, consult your own attorney(s) for the best course of action and policy contents.

Privacy

The mobile device deployment policy should address privacy and monitoring. When a personal device is used for business tasks, the user often loses some or all of the privacy they enjoyed prior to using their mobile device at work. Workers may need to agree

to be tracked and monitored on their mobile device, even when not on company property and outside work hours. A personal device in use under BYOD or CYOD should be considered by the individual to be quasi-company property.

A primary way for a worker to protect their privacy in regard to a mobile device is to not use a single device for both work and personal activities.

Architecture/Infrastructure Considerations

When implementing mobile device deployment policies, organizations should evaluate their network and security design, architecture, and infrastructure. If every worker brings in a personal device, the number of endpoint devices on the network may double. This requires planning to handle IP assignments, communications isolation, data-priority management, and increased intrusion detection system (IDS)/intrusion prevention system (IPS) monitoring load, as well as increased bandwidth consumption, both internally and across any Internet link. Most mobile devices are wireless enabled, so this will likely require a more robust wireless network and dealing with Wi-Fi congestion and interference. Your mobile device deployment policy must be considered in light of the additional infrastructure costs it will trigger.

Legal Concerns

Company attorneys should evaluate the legal concerns of mobile devices. Using personal devices in the execution of business tasks probably means an increased burden of liability and risk of data leakage. Mobile devices may make employees happy, but they might not be a worthwhile or cost-effective endeavor for your organization if they significantly increase risk and legal liability.

Acceptable Use Policy

The mobile device deployment policy should either reference the company's acceptable use policy (AUP) or include a mobile device–specific version focusing on unique issues. The use of personal mobile devices at work is accompanied by an increased

risk of information disclosure, distraction, and access to inappropriate content. Workers should remain mindful that the primary goal when at work is to accomplish productivity tasks.

Onboard Camera/Video

The mobile device deployment policy needs to address mobile devices with onboard cameras. Some environments disallow cameras of any type. This would require that mobile devices be without a camera. If cameras are allowed, a description of when they may and may not be used should be clearly documented and explained to workers. A mobile device can act as a storage device, provide an alternate wireless connection pathway to an outside provider or service, and may be used to collect images and videos that disclose confidential information or equipment.

If geofencing is available, it may be possible to use MDM/UEM to implement a location-specific hardware-deactivate profile to turn off the camera (or other components) while the device is on company premises but return the feature to operational status once the device leaves the geofenced area.

Recording Microphone

Most mobile devices with a speaker also have a microphone. The microphone can be used to record audio, noise, and voices nearby. Many mobile devices also support external microphones connected by a USB adapter, Bluetooth, or a 1/8″ stereo jack. If microphone recording is deemed a security risk, this feature should be deactivated using an MDM/UEM or deny presence of mobile devices in sensitive areas or meetings.

Tethering and Hotspots

Tethering is the activity of sharing the cellular network data connection of a mobile device with other devices. This is also known as a *hotspot*. This effectively allows the mobile device to act as a portable wireless access point (WAP). The sharing of data connections can take place over Wi-Fi, Bluetooth, or USB cable. Some service providers include tethering in their service plans,

whereas others charge an additional fee and some block tethering completely.

Tethering may represent a risk to the organization. It is a means for a user to grant Internet access to devices that are otherwise network-isolated, and it can be used as a means to bypass the company's filtering, blocking, and monitoring of Internet use. Thus, tethering should be blocked while a mobile device is within a company facility.

Hotspot devices that operate as portable WAPs are available and can be used to create a Wi-Fi network linked to a telco's or carrier's data network. Hotspot devices should be barred from use in most organizations because they provide a direct link to the Internet without a company's security restrictions being enforced.

Contactless Payment Methods

A number of mobile device–based payment systems, called *contactless payment methods*, do not require direct physical contact between the mobile device and the point-of-sale (PoS) device. Some are based on NFC, others on RFID, some on SMS, and still others on optical camera–based solutions, such as scanning *Quick Response (QR) codes*. Mobile payments are convenient for the shopper but might not always be a secure mechanism. Users should only employ mobile payment solutions that require a per-transaction confirmation or that require the device to be unlocked and an app launched to perform a transaction. Without these precautions, it may be possible to clone your device's contactless payment signals and perform transaction abuse.

Your organization is unlikely to see any additional risk based on mobile payment solutions. However, use caution when implementing them on company-owned equipment or when they are linked to your company's financial accounts.

SIM Cloning

Subscriber identity module (SIM) cards are used to associate a device with a subscriber's identity and service at a mobile or wireless telco. SIMs can be easily swapped between devices and cloned to abuse a victim's telco services. If a SIM card is cloned, then the cloned SIMs may be able to connect other devices to the telecommunications services and link the use back to the account of the original owner. Physical control must be maintained on mobile devices and an account or service lock established on mobile services with the telco carrier.

Essential Security Protection Mechanisms

The need for security mechanisms within an OS comes down to one simple fact: Software should not be trusted. Third-party software is inherently untrustworthy, no matter who or where it comes from. This is not to say that all software is evil. Instead, this is a protection stance—because all third-party software is written by someone other than the OS creator, that software might cause problems. Thus, treating all non-OS software as potentially damaging allows the OS to prevent many disastrous occurrences through the use of software management protection mechanisms. The OS must employ protection mechanisms to keep the computing environment stable and to keep processes isolated from one another. Without these efforts, the security of data could never be reliable or even possible. This is effectively applying the principle of zero trust (see Chapter 8).

Computer system designers should adhere to a number of common protection mechanisms when designing secure systems. These principles are specific instances of the more general security rules that govern safe computing practices. Designing security into a system during the earliest stages of development will help ensure that the overall security architecture has the best chance for success and reliability.

Process Isolation

Process isolation requires that the OS provide separate memory spaces for each process's instructions and data. It also requires that the OS enforce those boundaries, preventing one process from reading or writing data that belongs to another process. There are two major advantages to using this technique:

- It prevents unauthorized data access.
- It protects the integrity of processes.

Without such controls, a poorly designed process could go haywire and write data to memory spaces allocated to other processes, causing the entire system to become unstable rather than affecting only the execution of the errant process. In a more malicious vein, processes could attempt (and perhaps even succeed at) reading or writing to memory spaces outside their scope, intruding on or attacking other processes.

Many modern OSs address the need for process isolation by implementing virtual machines on a per-user or per-process basis. A virtual machine presents a user or process with a processing environment—including memory, address space, and other key system resources and services—that allows that user or process to behave as though they have sole, exclusive access to the entire computer. This allows each user or process to operate independently without requiring it to take cognizance of other users or processes that might be active simultaneously on the same machine. As part of the mediated access to the system that the OS provides, it maps virtual resources and access in lower-privileged layers/rings, and system calls are used to request access to the corresponding real resources. This not only makes things easier for programmers, but also protects individual users and processes from one another.

Hardware Segmentation

Hardware segmentation is similar to process isolation in purpose—it prevents access to information that belongs to a different

process/security level. The main difference is that hardware segmentation enforces these requirements through the use of physical hardware controls rather than the logical process isolation controls imposed by an OS. Such implementations are rare, and they are generally restricted to national security implementations, where the extra cost and complexity is offset by the sensitivity of the information involved and the risks inherent in unauthorized access or disclosure.

Root of Trust

The *root of trust (RoT)* is a foundational concept in cybersecurity and cryptographic systems. It represents the starting point or anchor of a security chain, providing a secure and trustworthy foundation for various security functions. The root of trust is critical for establishing and verifying the integrity, authenticity, and confidentiality of digital information within a system.

A *trust anchor* is a specific entity or component within a system that is inherently trusted. It serves as a reference point for establishing trust in other entities or components within the system. The trust anchor is typically a well-protected and tamper-resistant element, and trust in the overall system is derived from the trustworthiness of the trust anchor.

A hardware-based RoT refers to the implementation of the root of trust using dedicated hardware components. This often involves the integration of secure hardware modules, such as Trusted Platform Modules (TPMs) or hardware security modules (HSMs), which are designed to provide a secure and isolated environment for cryptographic operations and key management. Hardware-based RoTs enhance security by isolating critical security functions from the general-purpose computing environment, making them more resistant to various forms of attacks. These concepts are crucial for building secure systems and ensuring the integrity and trustworthiness of digital interactions.

System Security Policy

Just as security policy guides the day-to-day security operations, processes, and procedures in organizations, they have an important role to play when designing and implementing systems. This is equally true whether a system is entirely hardware-based, entirely software-based, or a combination of both. In this case, the role of a *system security policy* is to inform and guide the design, development, implementation, testing, and maintenance of a particular system. Thus, this kind of security policy tightly targets a single implementation effort. (Although it may be adapted from other, similar efforts, it should reflect the target as accurately and completely as possible.)

For system developers, a system security policy is best encountered in the form of a document that defines a set of rules, practices, and procedures that describe how the system should manage, protect, and distribute sensitive information. Security policies that prevent information flow from higher security levels to lower security levels are called multilevel security policies. As a system is developed, the security policy should be designed, built, implemented, and tested as it relates to all applicable system components or elements, including any or all of the following: physical hardware components, firmware, software, and how the organization interacts with and uses the system. The overall point is that security must be considered for the entire life of the project. When security is applied only at the end, it typically fails.

Common Security Architecture Flaws and Issues

No security architecture is totally secure. Every computer system has weaknesses and vulnerabilities. The goal of security models and architectures is to address as many known weaknesses as possible. Due to this fact, corrective actions must be taken to resolve security issues. The following sections present some common security issues that affect computer systems in relation to vulnerabilities of security architectures. You should understand each of the issues and how they can degrade the

overall security of your system. Some issues and flaws overlap one another and are used in creative ways to attack systems. Although the following discussion covers the most common flaws, the list is not exhaustive. Attackers are very clever.

Many attacks and exploits are covered elsewhere that are also relevant to this chapter's content, such as denial of service (DoS) (Chapter 17), buffer overflow (Chapter 21), malware (Chapter 21), escalation of privilege (Chapter 21), and maintenance hooks/backdoors (Chapter 21). We covered numerous malicious issues earlier in this chapter, such as emanation eavesdropping, the cold boot attack against memory, phlashing, mobile-code-based client-side attacks, exploitation of local Internet caches, and VM escaping. Several additional adversarial threats are included here, such covert channels, design/coding flaws, rootkits, and incremental attacks.

Covert Channels

A *covert channel* is a method that is used to pass information over a path that is not normally used for communication. Because the path is not normally used for communication, it may not be protected by the system's normal security controls. Using a covert channel provides a means to violate, bypass, or circumvent a security policy undetected. Covert channels are one of the important examples of vulnerabilities of security architectures.

As you might imagine, a covert channel is the opposite of an overt channel. An *overt channel* is a known, expected, authorized, designed, monitored, and controlled method of communication. Therefore, a covert channel is an unknown, unexpected, unauthorized, not designed (at least not by the original system designers), unmonitored, and uncontrolled method of data transfer.

There are two basic types of covert channels:

Covert Timing Channel A *covert timing channel* conveys information by altering the performance of a system component or modifying a resource's timing in a predictable

manner. Using a covert timing channel is generally a method to secretly transfer data and is very difficult to detect.

Covert Storage Channel A *covert storage channel* conveys information by writing data to a common storage area where another process can read it. When assessing the security of software, be diligent for any process that writes to any area of memory that another process can read.

Examples of covert timing channels include the following:

- Blinking a light visible outside the building so that if a reading is taken every two seconds when the light is on count it as a 1 and when the light is off count it as a 0. With an external camera linked to a recording system, a slow transmission of binary data can occur.
- Using a microphone to listen to the noise occurring in an area or related to a computer system. Then modify a case fan to spin faster (for a 1) or slower (for a 0) to force a change in the noise generated every 10 seconds.
- Monitoring utilization levels of an Internet connection when an insider is artificially padding or restricting traffic every 30 seconds. When traffic is above 80 percent utilization, record a 1; when below 40 percent utilization, record a 0.

Here are examples of covert storage channels; notice that they all involve placing data in a location that is either unseen by the OS or ignored by the OS:

- Writing data into unallocated or unpartitioned space, which may be accomplished using a hex editor
- Writing data directly into a bad sector of an HDD or a bad block on an SSD
- Writing data into the unused space at the end of a cluster, an area known as slack space
- Writing data directly into sectors or clusters without proper registration with the directory system, file container, or

header

Both types of covert channels rely on the use of communication techniques to exchange information with otherwise unauthorized subjects. Detecting such abuse can be difficult because the covert channel is outside the normal data transfer environment and security oversight. The best defense is to implement detailed and thorough auditing of all user and application activities and analyze log files for any covert channel activity, which may be anomalous behavior or may elicit known malicious activities via heuristics or pattern matching.

Attacks Based on Design or Coding Flaws

Certain attacks may result from poor design techniques, questionable implementation practices and procedures, or poor or inadequate testing. Some attacks may result from deliberate design decisions when special points of entry, built into code to circumvent access controls, login, or other security checks often added to code while under development, are not removed when that code is put into production. For what we hope are obvious reasons, such points of egress are properly called maintenance hooks or backdoors because they avoid security measures by design. Extensive testing and code review are required to uncover such covert means of access, which are easy to remove during the final phases of development but can be incredibly difficult to detect during the testing and maintenance phases.

Poor coding practices and lack of security consideration are common sources or causes of vulnerabilities in system architectures that can be attributed to failures in design, implementation, prerelease code cleanup, or out-and-out coding mistakes. Although such flaws are avoidable, finding and fixing them requires rigorous security-conscious design from the beginning of a development project and extra time and effort spent in testing and analysis. This helps to explain the often lamentable state of software security, but it does not excuse it! Although functionality testing is commonplace for commercial code and applications, separate testing for security issues has

been gaining attention and credibility only in the past few years, courtesy of widely publicized virus and worm attacks, SQL injection attacks, cross-site scripting attacks, and occasional defacements of or disruptions to widely used public sites online. Check out the OWASP Top 10 Web Application Security Risks report at owasp.org/www-project-top-ten. Most of these coding concerns are addressed in Chapter 20, "Software Development Security," and Chapter 21.

Humans will never write completely secure (flawless) code. Any program that does not gracefully handle any exception is in danger of exiting in an unstable state. It is possible to cleverly crash a program after it has increased its security level to carry out a normal task. If an attacker is successful in crashing the program at the right time, they can attain a higher security level and cause damage to the confidentiality, integrity, and availability of your system. These are just a few of the myriad ways that code can be compromised.

Perfect security might be impossible, but you can definitely take many strong measures to secure your code better. Source code analysis tools implemented throughout the development cycle will minimize the number of flaws in the production release, and the flaws identified prior to production release will cost much less to mitigate. All programs that are executed directly or indirectly must be fully tested to comply with your security model. Make sure you have the latest version of any software installed, and be aware of any known security vulnerabilities. Because each security model, and each security policy, is different, you must ensure that the software you execute does not exceed the authority you allow. Writing secure code is difficult, but it's certainly possible. Make sure all programs you use are designed to address security concerns. The concepts of code review and testing are covered in Chapter 15, "Security Assessment and Testing."

Rootkits

A *rootkit* is malware that embeds itself deep within an OS. The term is a derivative of the concept of rooting and a utility kit of

hacking tools. Rooting is gaining total or full control over a system.

A rootkit can manipulate information seen by the OS and displayed to users. A rootkit may replace the OS kernel, shim itself under the kernel, replace device drivers, or infiltrate application libraries so that whatever information it feeds to or hides from the OS, the OS thinks it is normal and acceptable. This allows a rootkit to hide itself from detection, prevent its files from being viewed by file management tools, and prevent its active processes from being viewed by task management or process management tools. Thus, a rootkit is a type of invisibility shield used to hide itself and other malicious tools.

Several rootkit-detection tools are available, some of which are able to remove known rootkits. However, once you suspect a rootkit is on a system, the only truly secure response is to reconstitute or replace the entire computer. Reconstitution involves performing a thorough storage sanitization operation on all storage devices on that system, reinstalling the OS and all applications from trusted original sources, and then restoring files from trusted rootkit-free backups. Obviously, the best protection against rootkits is defense (i.e., don't get infected in the first place) rather than response.

There are often no noticeable symptoms or indicators of compromise related to a rootkit infection. In the moments after the initial rootkit installation, there might be some system sluggishness and unresponsiveness as the rootkit installs itself, but otherwise, it will actively mask any symptoms. In some rootkit infections, the malware's initial infector, dropper, or installer will perform privilege escalation.

A means to potentially detect the presence of a rootkit is to notice when system files, such as device drivers and dynamic-link libraries (DLLs), have a file size and/or hash value change. File hash tracking can be performed manually by an administrator or automatically by HIDSs and system monitoring security tools.

Incremental Attacks

Some forms of attack occur in slow, gradual increments rather than through obvious or recognizable attempts to compromise system security or integrity. Two such forms of *incremental attack* are data diddling and the salami attack.

Data diddling occurs when an attacker gains access to a system and makes small or random changes to data during processing, input, or transaction rather than obviously altering file contents or damaging or deleting entire files. Such changes can be difficult to detect unless files and data are protected by encryption or unless some kind of integrity check (such as a checksum or message digest) is routinely performed and applied each time a file is read or written. Encrypted filesystems, file-level encryption techniques, or some form of file monitoring (which includes integrity checks performed by file integrity monitoring [FIM] tools) usually offer adequate guarantees that no data diddling is underway. Data diddling is often considered an attack performed more often by insiders rather than outsiders (external intruders).

The *salami attack* is more mythical by all published reports. The name of the attack refers to a systematic whittling at assets in accounts or other records with financial value, where very small amounts are deducted from balances regularly and routinely. Metaphorically, the attack may be explained as stealing a very thin slice from a salami each time it's put on the slicing machine when a paying customer is accessing it. Most security experts concede that salami attacks are possible, especially when organizational insiders could be involved. Organizations can prevent or eliminate such an attack only by proper separation of duties and proper control over code. Setting financial transaction monitors to track very small transfers of funds or other items of value should help to detect such activity; regular employee notification of the practice should help to discourage attempts at such attacks.

If you want an entertaining method of learning about the salami attack or the salami technique, view the movies *Office Space* and *Superman III*. You can also read the article from *Wired* about an attack of this nature from 2008: www.wired.com/2008/05/man-allegedly-b.

Summary

Shared responsibility is the security design principle indicating that organizations do not operate in isolation. It is because we participate in shared responsibility that we must research, implement, and manage engineering processes using secure design principles.

Designing secure computing systems begins with an investigation of hardware, software, and firmware and how those pieces fit into the security puzzle. It's important to understand the principles of common computer and network organizations, architectures, and designs; the difference between address space and memory space; and machine types or system variations.

Additionally, a security professional must have a good grasp of operating modes (user, supervisor, privileged), storage types (primary, secondary, real, virtual, volatile, nonvolatile, random, sequential), and common protection mechanisms (such as process isolation and hardware segmentation).

System function, purpose, and design work *toward* establishing and supporting security or *against* it. Client-based systems should be concerned about running code from unknown sources as well as protecting local caches. Server-based systems need to manage data flow and optimize operations using large-scale parallel data systems, grid computing, or peer-to-peer solutions when appropriate. Additional concerns relate to industrial control systems, the Internet of Things, microservices, and infrastructure as code.

Virtualization technology is used to host one or more OSs within the memory of a single host computer. Virtual software, virtual networking, software-defined everything, containerization, serverless architecture, and other related advancements often dictate the need for virtualization security management.

Static environments, embedded systems, cyber-physical systems, HPC systems, edge computing devices, and fog computing devices, mobile devices, and other limited or single-purpose computing environments need security management.

No matter how sophisticated a security architecture is, flaws exist that attackers can exploit. Some flaws are introduced by programmers, whereas others are architectural design issues.

Study Essentials

Understand shared responsibility. The security design principle indicates that organizations do not operate in isolation. It is because we participate in shared responsibility that we must research, implement, and manage engineering processes using secure design principles.

Understand the concept of protection rings. From a security standpoint, protection rings organize code and components in an OS into concentric rings. The deeper inside the circle you go, the higher the privilege level associated with the code that occupies a specific ring.

Describe the different types of memory used by a computer. ROM is nonvolatile and can't be written to by the end user. Data can be written to PROM chips only once. EPROM/UVEPROM chips may be erased with ultraviolet light. EEPROM chips may be erased with electrical current. RAM chips are volatile and lose their contents when the computer is powered off.

Know the security issues surrounding memory components. Some security issues surround memory components: the fact that data may remain on the chip after

power is removed and the control of access to memory in a multiuser system.

Know the concepts of memory addressing. Means of memory addressing include register addressing, immediate addressing, direct addressing, indirect addressing, and base+offset addressing.

Describe the different characteristics of storage devices used by computers. Primary storage is the same as memory. Secondary storage consists of magnetic, flash, and optical media that must be first read into primary memory before the CPU can use the data. Random access storage devices can be read at any point, whereas sequential access devices require scanning through all the data physically stored before the desired location.

Know the security issues surrounding secondary storage devices. Three main security issues surround secondary storage devices: removable media can be used to steal data, access controls and encryption must be applied to protect data, and data can remain on the media even after file deletion or media formatting.

Know about emanation security. Many electrical devices emanate electrical signals or radiation that can be intercepted by unauthorized individuals. These signals may contain confidential, sensitive, or private data. TEMPEST/EMSEC countermeasures to Van Eck phreaking (i.e., eavesdropping) include Faraday cages, white noise, control zones, and shielding.

Understand security risks that input and output devices can pose. Input/output devices can be subject to eavesdropping and tapping, are subject to shoulder surfing, are used to smuggle data out of an organization, or are used to create unauthorized, insecure points of entry into an organization's systems and networks. Be prepared to recognize and mitigate such vulnerabilities.

Be aware of JavaScript concerns. JavaScript is the most widely used scripting language in the world and is embedded into HTML documents. Whenever you allow code from an unknown

and thus untrusted source to execute on your system, you are putting your system at risk of compromise.

Know about large-scale parallel data systems. Systems designed to perform numerous calculations simultaneously include SMP, AMP, and MPP. Grid computing is a form of parallel distributed processing that loosely groups a significant number of processing nodes to work toward a specific processing goal. Peer-to-peer (P2P) technologies are networking and distributed application solutions that share tasks and workloads among peers.

Be able to define OT/ICS. An industrial control system (ICS) or an operational technology (OT) is a form of computer-management device that controls industrial processes and machines. ICS examples include distributed control systems (DCSs), programmable logic controllers (PLCs), and supervisory control and data acquisition (SCADA).

Be aware of distributed systems. A distributed system or a distributed computing environment (DCE) is a collection of individual systems that work together to support a resource or provide a service. The primary security concern is the interconnectedness of the components.

Understand data sovereignty. Data sovereignty is the concept that, once information has been converted into a binary form and stored as digital files, it is subject to the laws of the country within which the storage device resides.

Be able to define IoT. The Internet of Things (IoT) is a class of devices that are Internet-connected to provide automation, remote control, or AI processing to appliances or devices. The security issues related to IoT often relate to access and encryption.

Understand microservices. A microservice is simply one element, feature, capability, business logic, or function of a web application that can be called upon or used by other web applications. It is the conversion or transformation of a capability of one web application into a microservice that can be called upon

by numerous other web applications. It allows large complex solutions to be broken into smaller self-contained functions.

Be able to define IaC. Infrastructure as code (IaC) is a change in how hardware management is perceived and handled. Instead of seeing hardware configuration as a manual, direct hands-on, one-on-one administration hassle, it is viewed as just another collection of elements to be managed in the same way that software and code are managed under DevSecOps (development, security, and operations).

Understand hypervisors. The hypervisor, also known as the virtual machine monitor/manager (VMM), is the component of virtualization that creates, manages, and operates virtual machines.

Understand virtual software. A virtual application or virtual software is a software product deployed in such a way that it is fooled into believing it is interacting with a full host OS. A virtual (or virtualized) application has been packaged or encapsulated so that it can execute but operate without full access to the host OS. A virtual application is isolated from the host OS so that it cannot make any direct or permanent changes to the host OS.

Know virtual networking. A virtualized network or network virtualization is the combination of hardware and software networking components into a single integrated entity. The resulting solution allows for software control over all network functions: management, traffic shaping, address assignment, and so on.

Know about SDx. Software-defined everything (SDx) refers to a trend of replacing hardware with software using virtualization. SDx includes virtualization, virtualized software, virtual networking, containerization, serverless architecture, infrastructure as code, SDN, VSAN, software-defined storage (SDS), VDI, VMI, SDV, and software-defined data center (SDDC).

Know about VDI and VMI. Virtual desktop infrastructure (VDI) is a means to reduce the security risk and performance requirements of end devices by hosting desktop/workstation OS virtual machines on central servers that are remotely accessed by

users. Virtual mobile infrastructure (VMI) is where the OS of a mobile device is virtualized on a central server.

Be aware of SDV. Software-defined visibility (SDV) is a framework to automate the processes of network monitoring and response. The goal is to enable the analysis of every packet and make deep intelligence-based decisions on forwarding, dropping, or otherwise responding to threats.

Know some of the security issues of virtualization. Virtualization doesn't lessen the security management requirements of an OS. Thus, patch management is still essential. It's important to protect the stability of the host. Organizations should maintain backups of their virtual assets. Virtualized systems should be security tested. VM sprawl occurs when an organization deploys numerous virtual machines without an overarching IT management or security plan in place.

Understand containerization. Containerization or OS virtualization is based on the concept of eliminating the duplication of OS elements in a virtual machine. Each application is placed into a container that includes only the actual resources needed to support the enclosed application, and the common or shared OS elements are then part of the hypervisor.

Understand embedded systems. An embedded system is typically designed around a limited set of specific functions in relation to the larger product to which it is attached.

Be aware of microcontrollers. A microcontroller is similar to but less complex than a system on a chip (SoC). A microcontroller may be a component of an SoC. A microcontroller is a small computer consisting of a CPU (with one or more cores), memory, various input/output capabilities, RAM, and often nonvolatile storage in the form of flash or ROM/PROM/EEPROM. Examples include Raspberry Pi, Arduino, and FPGA.

Understand embedded systems and static environment security concerns. Static environments, embedded systems, cyber-physical systems, HPC systems, edge computing devices, fog computing devices, mobile devices, and other limited or

single-purpose computing environments need security management. These techniques may include network segmentation, security layers, application firewalls, manual updates, firmware version control, and control redundancy and diversity.

Know about HPC systems. High-performance computing (HPC) systems are computing platforms designed to perform complex calculations or data manipulations at extremely high speeds. Supercomputers and MPP solutions are common examples of HPC systems.

Be aware of RTOS. A real-time operating system (RTOS) is designed to process or handle data as it arrives on the system with minimal latency or delay. An RTOS is usually stored on read-only memory (ROM) and is designed to operate in a hard real-time or soft real-time condition.

Understand edge computing. Edge computing is a philosophy of network design where data and the compute resources are located as close as possible to optimize bandwidth use while minimizing latency. In edge computing, the intelligence and processing are contained within each device. Thus, rather than having to send data off to a master processing entity, each device can process its own data locally.

Know about fog computing. Fog computing is another example of advanced computation architectures, which is also often used as an element in an IIoT deployment. Fog computing relies on sensors, IoT devices, or even edge computing devices to collect data, and then transfer it back to a central location for processing. Thus, intelligence and processing is centralized.

Understand mobile device security. Personal electronic device (PED) security features can often be managed using a mobile device management (MDM) or unified endpoint management (UEM) solution. These include device authentication, full-device encryption, communication protection, remote wiping, screen locks, device lockout, GPS and location services management, content management, application control, push notification management, third-party application

store control, storage segmentation, asset tracking, removable storage, deactivating unused features, rooting/jailbreaking, sideloading, custom firmware, carrier unlocking, firmware OTA updates, credential management, and text messaging security.

Understand mobile device deployment policies. A number of deployment models are available for allowing and/or providing mobile devices for employees to use while at work and to perform work tasks when away from the office. Examples include BYOD, CYOD, COPE, and COMS/COBO. You should also consider VDI and VMI options.

Understand process isolation. Process isolation requires that the OS provide separate memory spaces for each process's instructions and data. It also requires that the OS enforce those boundaries, preventing one process from reading or writing data that belongs to another process.

Be aware of hardware segmentation. Hardware segmentation is similar to process isolation in purpose—it prevents the access of information that belongs to a different process/security level. The main difference is that hardware segmentation enforces these requirements through the use of physical hardware controls rather than the logical process isolation controls imposed by an OS.

Understand the need for system security policy. The role of a system security policy is to inform and guide the design, development, implementation, testing, and maintenance of a particular system. Thus, this kind of security policy tightly targets a single implementation effort.

Be able to explain what covert channels are. A covert channel is a method that is used to pass information over a path that is not normally used for communication. Using a covert channel provides a means to violate, bypass, or circumvent a security policy undetected. Basic types are timing and storage.

Know about vulnerabilities due to design and coding flaws. Certain attacks may result from poor design techniques, questionable implementation practices and procedures, or poor or inadequate testing. Some attacks may result from deliberate

design decisions when special points of entry, built into code to circumvent access controls, login, or other security checks often added to code while under development, are not removed when that code is put into production. Poor coding practices and lack of security consideration are common sources or causes of vulnerabilities of system architectures that can be attributed to failures in design, implementation, prerelease code cleanup, or out-and-out coding mistakes.

Written Lab

1. Name three types of ICSs and describe what they do or how they are used.
2. Name the three pairs of aspects or features used to describe storage.
3. Name some vulnerabilities found in distributed architectures.
4. There are numerous server-based technologies that both increase computation and resource access capabilities and also introduce new risks to be managed. Name at least 10 examples of these technologies (over 20 were included in this chapter).
5. In relation to mobile devices, list seven of the potential on-device security features, list the four main deployment models, and list seven of the issues that should be addressed on a mobile device deployment policy.

Review Questions

1. While designing the security for the organization, you realize the importance of not only balancing the objectives of the organization against security goals but also focusing on the shared responsibility of security. Which of the following is considered an element of shared responsibility? (Choose all that apply.)