This page intentionally left blank

PART VI

Security Assessment and Testing

This page intentionally left blank

CHAPTER 18

Security Assessments

This chapter presents the following:

- Test, assessment, and audit strategies
- Testing technical security controls
- Conducting or facilitating security audits

Trust, but verify.

—Russian proverb

You can hire the best people, develop sound policies and procedures, and deploy world-class technology in an effort to secure your information systems, but if you do not regularly assess the effectiveness of these measures, your organization will not be secure for long. Unfortunately, thousands of well-intentioned organizations have learned the truth of this statement the hard way, realizing only after a security breach has occurred that the state-of-the-art controls they put into place initially have become less effective over time. So, unless your organization is continuously assessing and improving its security posture, that posture will become ineffective over time.

This chapter covers some of the most important elements of security assessments and testing. It is divided into three sections. We start by discussing assessment, test, and audit strategies, particularly how to design and assess them. From there, we get into the nitty-gritty of various common forms of testing with which you should be familiar. The third and final section discusses the various kinds of formal security audits and how you can conduct or facilitate them.

Test, Assessment, and Audit Strategies

Let's start by establishing some helpful definitions in the context of information systems security. A *test* is a procedure that records some set of properties or behaviors in a system being tested and compares them against predetermined standards. If you install a new device on your network, you might want to test its attack surface by running a network scanner against it, recording the open ports, and then comparing them against the appropriate security standards used in your organization. An *assessment* is a series of planned tests that are somehow related to each other. For example, we could conduct a vulnerability assessment against a new software system to determine how secure it is.

This assessment would include some specific (and hopefully relevant) vulnerability tests together with static and dynamic analysis of its software. An *audit* is a systematic assessment of significant importance to the organization that determines whether the system or process being audited satisfies some external standards. By "external" we mean that the organization being audited did not author the standards all by itself.

EXAM TIP You don't have to memorize these definitions. They are presented simply to give you an idea of the different scopes. Many security professionals use the terms almost interchangeably.

Each of these three types of system evaluations plays an important role in ensuring the security of our organizations. Our job as cybersecurity leaders is to integrate them into holistic strategies that, when properly executed, give us a complete and accurate picture of our security posture. It all starts with the risk management concepts we discussed in Chapter 2. Remember that risks determine which security controls we use, which are the focus of any tests, assessments, or audits we perform. So, a good security assessment strategy verifies that we are sufficiently protected against the risks we're tracking.

The security assessment strategy guides the development of standard testing and assessment procedures. This standardization is important because it ensures these activities are done in a consistent, repeatable, and cost-effective manner. We'll cover testing procedures later in this chapter, so let's take a look at how to design and validate a security assessment.

Designing an Assessment

The first step in designing anything is to figure out what it is that we are trying to accomplish. Are we getting ready for an external audit? Did we suffer a security incident because a control was not properly implemented? The answers to these sample questions point to significantly different types of assessment. In the first case, the assessment would be a very broad effort to verify an external standard with which we are supposed to be compliant. In the second case, the assessment would be focused on a specific control, so it would be much narrower in scope. Let's elaborate on the second case as we develop a notional assessment plan.

Suppose the security incident happened because someone clicked a link on a phishing e-mail and downloaded malware that the endpoint detection and response (EDR) solution was able to block. The EDR solution worked fine, but we are now concerned about our e-mail security controls. The objective of our assessment, therefore, is to determine the effectiveness of our e-mail defenses.

Once we have the objective identified, we can determine the necessary scope to accomplish it. When we talk about the scope of an assessment, we really mean which specific controls we will test. In our example, we would probably want to look at our e-mail security gateway, but we should also look at our staff members' security awareness. Next, we have to decide how many e-mail messages and how many users we need to assess to have confidence in our results.

The scope, in turn, informs the methods we use. We'll cover some of the testing techniques shortly, but it's important to note that it's not just *what* we do but *how* we do it that matters. We should develop standardized methodologies so that different tests are consistent and comparable. Otherwise, we won't necessarily know whether our posture is deteriorating from one assessment to the next.

Another reason to standardize the methodologies is to ensure we take into account business and operational impacts. For example, if we decide to test the e-mail security gateway using a penetration test, then there is a chance that we will interfere with e-mail service availability. This could present operational risks that we need to mitigate. Furthermore, on the off-chance that we break something during our assessment, we need to have a contingency plan that allows for the quick restoration of all services and capabilities.

A key decision is whether the assessment will be performed by an internal team or by a third party. If you don't have the in-house expertise, then this decision may very well already have been made for you. But even if your team has this expertise, you may still choose to bring in external auditors for any of a variety of reasons. For example, there may be a regulatory requirement that an external party test your systems; or you may want to benchmark your own internal assets against an external team; or perhaps your own team of testers is not large enough to cover all the auditing requirements and thus you want to bring in outside help.

Finally, once the assessment plan is complete, we need to get it approved for execution. This doesn't just involve our direct bosses but should also involve any stakeholders that might be affected (especially if something goes wrong and services are interrupted). The approval is not just for the plan itself, but also for any needed resources (e.g., funding for an external assessor) as well as for the scheduling. There is nothing worse than to schedule a security assessment that we later find out coincides with a big event like end-of-month accounting and reporting.

Validating an Assessment

Once the tests are over and the interpretation and prioritization are done, management will have in its hands a compilation of many of the ways the organization could be successfully attacked. This is the input to the next cycle in the remediation strategy. Every organization has only so much money, time, and personnel to commit to defending its network, and thus can mitigate only so much of the total risk. After balancing the risks and risk appetite of the organization and the costs of possible mitigations and the value gained from each, management must direct the system and security administrators as to where to spend those limited resources. An oversight program is required to ensure that the mitigations work as expected and that the estimated cost of each mitigation action is closely tracked by the actual cost of implementation. Any time the cost rises significantly or the value is found to be far below what was expected, the process should be briefly paused and reevaluated. It may be that a risk-versus-cost option initially considered less desirable now makes more sense than continuing with the chosen path.

Finally, when all is well and the mitigations are underway, everyone can breathe easier…except the security engineer who has the task of monitoring vulnerability announcements and discussion mailing lists, as well as the early warning services offered by some vendors. To put it another way, the risk environment keeps changing. Between tests, monitoring may make the organization aware of newly discovered vulnerabilities that would be found the next time the test is run but that are too high risk to allow to wait that long. And so another, smaller cycle of mitigation decisions and actions must be taken, and then it is time to run the tests again.

Table 18-1 provides an example of a testing schedule that each operations and security department should develop and carry out.

Test Type	**Frequency**	**Benefits**
Network scanning	Continuously to quarterly	• Enumerates the network structure and determines the set of active hosts and associated software • Identifies unauthorized hosts connected to a network • Identifies open ports • Identifies unauthorized services
Log reviews	Daily for critical systems	• Validates that the system is operating according to policy
Password cracking	Continuously to same frequency as expiration policy	• Verifies the policy is effective in producing passwords that are difficult to break • Verifies that users select passwords compliant with the organization's security policy
Vulnerability scanning	Quarterly or bimonthly (more often for high-risk systems), or whenever the vulnerability database is updated	• Enumerates the network structure and determines the set of active hosts and associated software • Identifies a target set of computers to focus vulnerability analysis • Identifies potential vulnerabilities on the target set • Validates operating systems and major applications are up to date with security patches and software versions
Penetration testing	Annually	• Determines how vulnerable an organization's network is to penetration and the level of damage that can be incurred • Tests the IT staff's response to perceived security incidents and their knowledge and implementation of the organization's security policy and the system's security requirements
Integrity checkers	Monthly and in case of a suspicious event	• Detects unauthorized file modifications

Table 18-1 Example Testing Schedules for Each Operations and Security Department

Testing Technical Controls

A *technical control* is a security control implemented through the use of an IT asset. This asset is usually, but not always, some sort of software or hardware that is configured in a particular way. When we test our technical controls, we are verifying their ability to mitigate the risks that we identified in our risk management process (see Chapter 2 for a detailed discussion). This linkage between controls and the risks they are meant to mitigate is important because we need to understand the context in which specific controls were implemented.

Once we understand what a technical control was intended to accomplish, we are able to select the proper means of testing whether it is being effective. We may be better off testing third-party software for vulnerabilities than attempting a code review. As security professionals, we must be familiar, and ideally experienced, with the most common approaches to testing technical controls so that we are able to select the right one for the job at hand.

Vulnerability Testing

Vulnerability testing, whether manual, automated, or—preferably—a combination of both, requires staff and/or consultants with a deep security background and the highest level of trustworthiness. Even the best automated vulnerability scanning tool will produce output that can be misinterpreted as crying wolf (false positive) when there is only a small puppy in the room, or alert you to something that is indeed a vulnerability but that either does not matter to your environment or is adequately compensated for elsewhere. There may also be two individual vulnerabilities that exist, which by themselves are not very important but when put together are critical. And, of course, false negatives will also crop up, such as an obscure element of a single vulnerability that matters greatly to your environment but is not called out by the tool.

NOTE Before carrying out vulnerability testing, a written agreement from management is required! This protects the tester against prosecution for doing his job and ensures there are no misunderstandings by providing in writing what the tester should—and should not—do.

The goals of the assessment are to

- Evaluate the true security posture of an environment (don't cry wolf, as discussed earlier).
- Identify as many vulnerabilities as possible, with honest evaluations and prioritizations of each.
- Test how systems react to certain circumstances and attacks, to learn not only what the known vulnerabilities are (such as this version of the database, that version of the operating system, or a user ID with no password set) but also how the unique elements of the environment might be abused (SQL injection attacks, buffer overflows, and process design flaws that facilitate social engineering).

- Before the scope of the test is decided and agreed upon, the tester must explain the testing ramifications. Vulnerable systems could be knocked offline by some of the tests, and production could be negatively affected by the loads the tests place on the systems.

Management must understand that results from the test are just a "snapshot in time." As the environment changes, new vulnerabilities can arise. Management should also understand that various types of assessments are possible, each one able to expose different kinds of vulnerabilities in the environment, and each one limited in the completeness of results it can offer:

- *Personnel testing* includes reviewing employee tasks and thus identifying vulnerabilities in the standard practices and procedures that employees are instructed to follow, demonstrating social engineering attacks and the value of training users to detect and resist such attacks, and reviewing employee policies and procedures to ensure those security risks that cannot be reduced through physical and logical controls are met with the final control category: administrative.
- *Physical testing* includes reviewing facility and perimeter protection mechanisms. For instance, do the doors actually close automatically, and does an alarm sound if a door is held open too long? Are the interior protection mechanisms of server rooms, wiring closets, sensitive systems, and assets appropriate? (For example, is the badge reader working, and does it really limit access to only authorized personnel?) Is dumpster diving a threat? (In other words, is sensitive information being discarded without proper destruction?) And what about protection mechanisms for manmade, natural, or technical threats? Is there a fire suppression system? Does it work, and is it safe for the people and the equipment in the building? Are sensitive electronics kept above raised floors so they survive a minor flood? And so on.
- *System and network testing* are perhaps what most people think of when discussing information security vulnerability testing. For efficiency, an automated scanning product identifies known system vulnerabilities, and some may (if management has signed off on the performance impact and the risk of disruption) attempt to exploit vulnerabilities.

Because a security assessment is a point-in-time snapshot of the state of an environment, assessments should be performed regularly. Lower-priority, better-protected, and less-at-risk parts of the environment may be scanned once or twice a year. High-priority, more vulnerable targets, such as e-commerce web server complexes and the middleware just behind them, should be scanned nearly continuously.

To the degree automated tools are used, more than one tool—or a different tool on consecutive tests—should be used. No single tool knows or finds every known vulnerability. The vendors of different scanning tools update their tools' vulnerability databases at different rates, and may add particular vulnerabilities in different orders. Always update the vulnerability database of each tool just before the tool is used. Similarly, from time to time different experts should run the test and/or interpret the results. No single expert always sees everything there is to be seen in the results.

Most networks consist of many heterogeneous devices, each of which will likely have its own set of potential vulnerabilities, as shown in Figure 18-1. The potential issues we would seek in, say, the perimeter router ("1." in Figure 18-1) are very different than those in a wireless access point (WAP) ("7." in Figure 18-1) or a back-end database management server (DBMS) ("11." in Figure 18-1). Vulnerabilities in each of these devices, in turn, will depend on the specific hardware, software, and configurations in use. Even if you were able to find an individual or tool who had expert knowledge on the myriad of devices and device-specific security issues, that person or tool would come with its own inherent biases. It is best to leverage team/tool heterogeneity in order to improve the odds of covering blind spots.

Other Vulnerability Types

As noted earlier, vulnerability scans find the potential vulnerabilities. Penetration testing is required to identify those vulnerabilities that can actually be exploited in the environment and cause damage.

Commonly exploited vulnerabilities include the following:

- **Kernel flaws** These are problems that occur below the level of the user interface, deep inside the operating system. Any flaw in the kernel that can be reached by an attacker, if exploitable, gives the attacker the most powerful level of control over the system.

  Countermeasure: Ensure that security patches to operating systems—after sufficient testing—are promptly deployed in the environment to keep the window of vulnerability as small as possible.
- **Buffer overflows** Poor programming practices, or sometimes bugs in libraries, allow more input than the program has allocated space to store it. This overwrites data or program memory after the end of the allocated buffer, and sometimes allows the attacker to inject program code and then cause the processor to execute it. This gives the attacker the same level of access as that held by the program that was attacked. If the program was run as an administrative user or by the system itself, this can mean complete access to the system.

  Countermeasure: Good programming practices and developer education, automated source code scanners, enhanced programming libraries, and strongly typed languages that disallow buffer overflows are all ways of reducing this extremely common vulnerability.
- **Symbolic links** Though the attacker may be properly blocked from seeing or changing the content of sensitive system files and data, if a program follows a symbolic link (a stub file that redirects the access to another place) and the attacker can compromise the symbolic link, then the attacker may be able to gain unauthorized access. (Symbolic links are used in Unix and Linux systems.) This may allow the attacker to damage important data and/or gain privileged access to the system. A historical example of this was to use a symbolic link to cause a

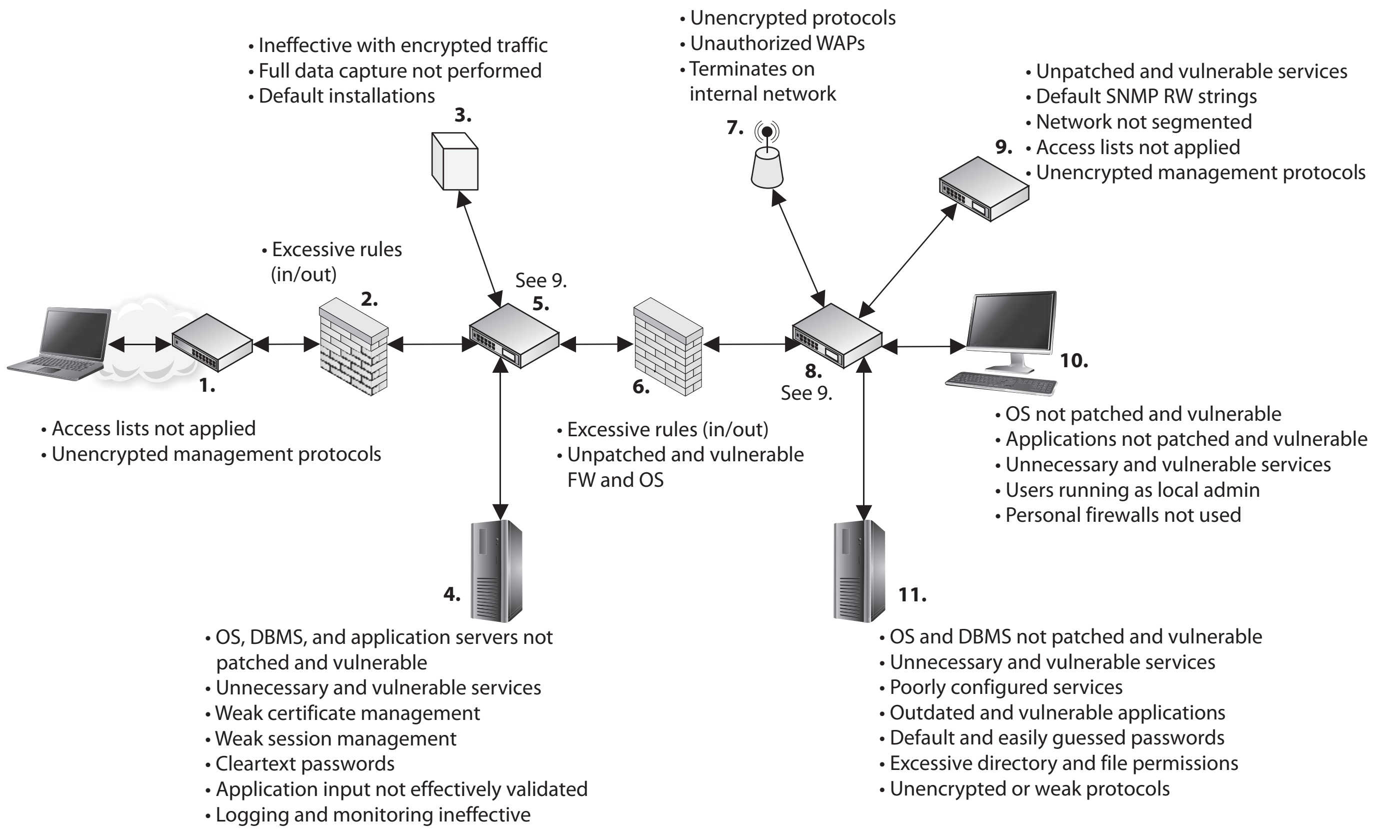

1.
• Access lists not applied
• Unencrypted management protocols
2.
• Excessive rules (in/out)
3.
• Ineffective with encrypted traffic
• Full data capture not performed
• Default installations
4.
• OS, DBMS, and application servers not patched and vulnerable
• Unnecessary and vulnerable services
• Weak certificate management
• Weak session management
• Cleartext passwords
• Application input not effectively validated
• Logging and monitoring ineffective
5.
See 9.
6.
• Excessive rules (in/out)
• Unpatched and vulnerable FW and OS
7.
• Unencrypted protocols
• Unauthorized WAPs
• Terminates on internal network
8.
See 9.
9.
• Unpatched and vulnerable services
• Default SNMP RW strings
• Network not segmented
• Access lists not applied
• Unencrypted management protocols
10.
• OS not patched and vulnerable
• Applications not patched and vulnerable
• Unnecessary and vulnerable services
• Users running as local admin
• Personal firewalls not used
11.
• OS and DBMS not patched and vulnerable
• Unnecessary and vulnerable services
• Poorly configured services
• Outdated and vulnerable applications
• Default and easily guessed passwords
• Excessive directory and file permissions
• Unencrypted or weak protocols

Figure 18-1 Vulnerabilities in heterogeneous networks

program to delete a password database, or replace a line in the password database with characters that, in essence, created a password-less root-equivalent account.

Countermeasure: Programs, and especially scripts, must be written to ensure that the full path to the file cannot be circumvented.

- **File descriptor attacks** File descriptors are numbers many operating systems use to represent open files in a process. Certain file descriptor numbers are universal, meaning the same thing to all programs. If a program makes unsafe use of a file descriptor, an attacker may be able to cause unexpected input to be provided to the program, or cause output to go to an unexpected place with the privileges of the executing program.

  Countermeasure: Good programming practices and developer education, automated source code scanners, and application security testing are all ways of reducing this type of vulnerability.

- **Race conditions** Race conditions exist when the design of a program puts it in a vulnerable condition before ensuring that those vulnerable conditions are mitigated. Examples include opening temporary files without first ensuring the files cannot be read or written to by unauthorized users or processes, and running in privileged mode or instantiating dynamic link library (DLL) functions without first verifying that the DLL path is secure. Either of these may allow an attacker to cause the program (with its elevated privileges) to read or write unexpected data or to perform unauthorized commands. An example of a race condition is a time-of-check to time-of-use (TOC/TOU) attack, discussed in Chapter 2.

  Countermeasure: Good programming practices and developer education, automated source code scanners, and application security testing are all ways of reducing this type of vulnerability.

- **File and directory permissions** Many of the previously described attacks rely on inappropriate file or directory permissions—that is, an error in the access control of some part of the system, on which a more secure part of the system depends. Also, if a system administrator makes a mistake that results in decreasing the security of the permissions on a critical file, such as making a password database accessible to regular users, an attacker can take advantage of this to add an unauthorized user to the password database or an untrusted directory to the DLL search path.

  Countermeasure: File integrity checkers, which should also check expected file and directory permissions, can detect such problems in a timely fashion, hopefully before an attacker notices and exploits them.

Many, many types of vulnerabilities exist, and we have covered some, but certainly not all, here in this book. The previous list includes only a few specific vulnerabilities you should be aware of for exam purposes.

Vulnerability Scanning Recap

Vulnerability scanners provide the following capabilities:

- The identification of active hosts on the network
- The identification of active and vulnerable services (ports) on hosts
- The identification of operating systems
- The identification of vulnerabilities associated with discovered operating systems and applications
- The identification of misconfigured settings
- Test for compliance with host applications' usage/security policies
- The establishment of a foundation for penetration testing

Penetration Testing

Penetration testing (also known as *pen testing*) is the process of simulating attacks on a network and its systems at the request of the owner, senior management. Penetration testing uses a set of procedures and tools designed to test and possibly bypass the security controls of a system. Its goal is to measure an organization's level of resistance to an attack and to uncover any exploitable weaknesses within the environment. Organizations need to determine the effectiveness of their security measures and not just trust the promises of the security vendors. Good computer security is based on reality, not on some lofty goals of how things are supposed to work.

A penetration test emulates the same methods attackers would use. Attackers can be clever, creative, and resourceful in their techniques, so penetration test attacks should align with the newest hacking techniques along with strong foundational testing methods. The test should look at each and every computer in the environment, as shown in Figure 18-2, because an attacker will not necessarily scan one or two computers only and call it a day.

The type of penetration test that should be used depends on the organization, its security objectives, and the management's goals. Some organizations perform periodic penetration tests on themselves using different types of tools. Other organizations ask a third party to perform the vulnerability and penetration tests to provide a more objective view.

Penetration tests can evaluate web servers, Domain Name System (DNS) servers, router configurations, workstation vulnerabilities, access to sensitive information, open ports, and available services' properties that a real attacker might use to compromise the organization's overall security. Some tests can be quite intrusive and disruptive. The timeframe for the tests should be agreed upon so productivity is not affected and personnel can bring systems back online if necessary.

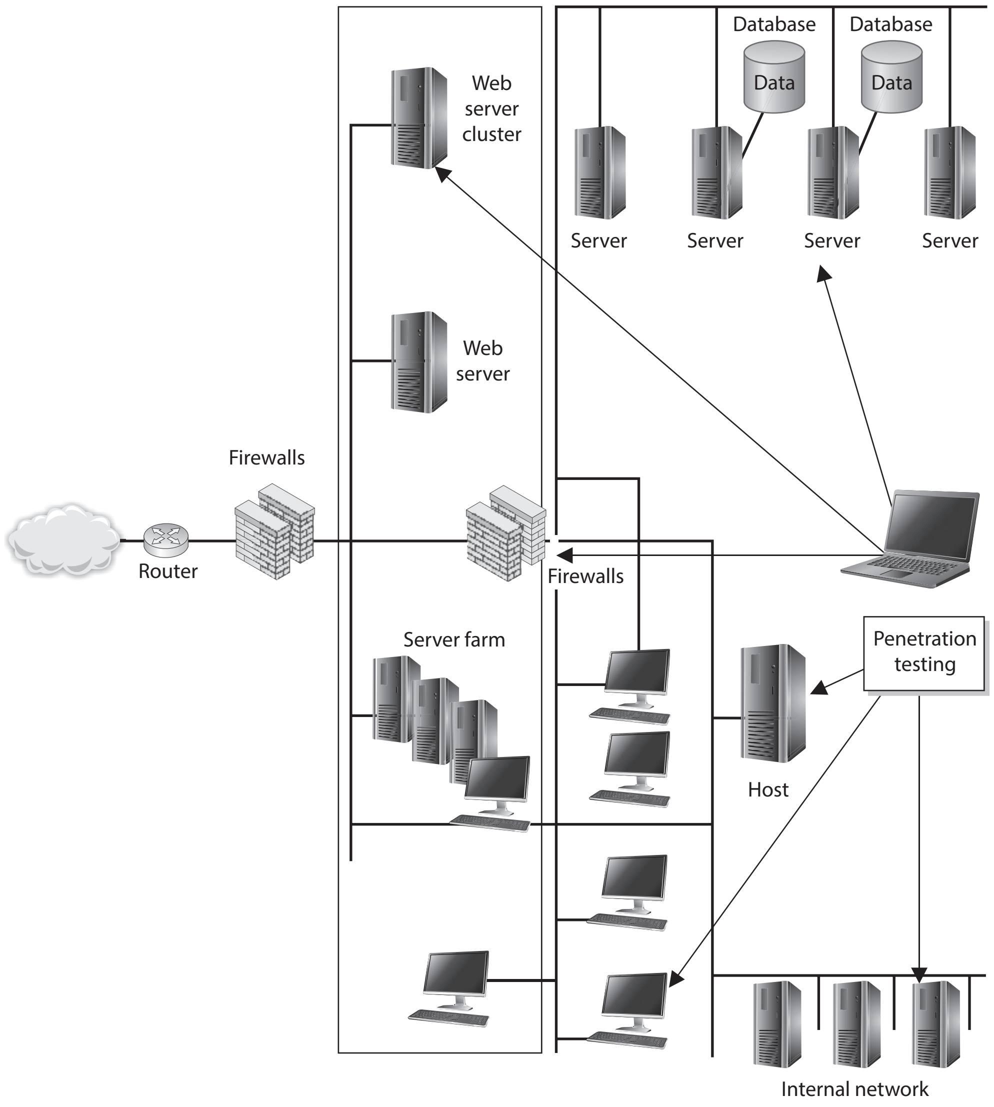


Figure 18-2 Penetration testing is used to prove an attacker can actually compromise systems.

NOTE Penetration tests are not necessarily restricted to information technology, but may include physical security as well as personnel security. Ultimately, the purpose is to compromise one or more controls, which could be technical, physical, or administrative.

The result of a penetration test is a report given to management that describes the vulnerabilities identified and the severity of those vulnerabilities, along with descriptions of how they were exploited by the testers. The report should also include suggestions on how to deal with the vulnerabilities properly. From there, it is up to management to determine how to address the vulnerabilities and what countermeasures to implement.

It is critical that senior management be aware of any risks involved in performing a penetration test before it gives the authorization for one. In rare instances, a system or application may be taken down inadvertently using the tools and techniques employed during the test. As expected, the goal of penetration testing is to identify vulnerabilities, estimate the true protection the security mechanisms within the environment are providing, and see how suspicious activity is reported—but accidents can and do happen.

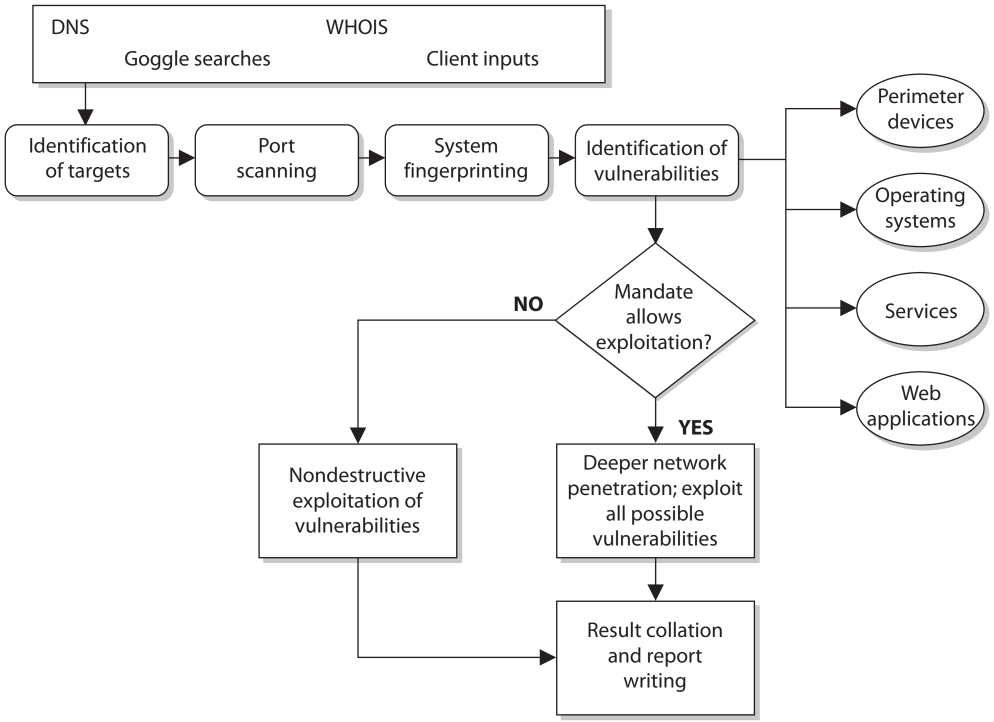
Security professionals should obtain an authorization letter that includes the extent of the testing authorized, and this letter or memo should be available to members of the team during the testing activity. This type of letter is commonly referred to as a "Get Out of Jail Free Card." Contact information for key personnel should also be available, along with a call tree in the event something does not go as planned and a system must be recovered.

NOTE A "Get Out of Jail Free Card" is a document you can present to someone who thinks you are up to something malicious, when in fact you are carrying out an approved test. More than that, it's also the legal agreement you have between you and your customer that protects you from liability, and prosecution.

When performing a penetration test, the team goes through a five-step process:

1. **Discovery** Footprinting and gathering information about the target
2. **Enumeration** Performing port scans and resource identification methods
3. **Vulnerability mapping** Identifying vulnerabilities in identified systems and resources
4. **Exploitation** Attempting to gain unauthorized access by exploiting vulnerabilities

5. **Report to management** Delivering to management documentation of test findings along with suggested countermeasures


DNS
WHOIS
Goggle searches
Client inputs
Identification of targets
Port scanning
System fingerprinting
Identification of vulnerabilities
Perimeter devices
Operating systems
Services
Web applications
Mandate allows exploitation?
NO
YES
Nondestructive exploitation of vulnerabilities
Deeper network penetration; exploit all possible vulnerabilities
Result collation and report writing

The penetration testing team can have varying degrees of knowledge about the penetration target before the tests are actually carried out:

- **Zero knowledge** The team does not have any knowledge of the target and must start from ground zero.
- **Partial knowledge** The team has some information about the target.
- **Full knowledge** The team has intimate knowledge of the target.

Security testing of an environment may take several forms, in the sense of the degree of knowledge the tester is permitted to have up front about the environment, and also the degree of knowledge the environment is permitted to have up front about the tester.

Tests can be conducted externally (from a remote location) or internally (meaning the tester is within the network). Combining both can help better understand the full scope of threats from either domain (internal and external).

Penetration tests may be blind, double-blind, or targeted. A *blind test* is one in which the testers only have publicly available data to work with (which is also known as zero knowledge or black box testing). The network security staff is aware that this type of test will take place, and will be on the lookout for pen tester activities. Part of the planning

Vulnerability and Penetration Testing: What Color Is Your Box?

Vulnerability testing and penetration testing come in boxes of at least three colors: black, white, and gray. The color, of course, is metaphorical, but security professionals need to be aware of the three types. None is clearly superior to the others in all situations, so it is up to us to choose the right approach for our purposes.

- *Black box testing* treats the system being tested as completely opaque. This means that the tester has no *a priori* knowledge of the internal design or features of the system. All knowledge will come to the tester only through the assessment itself. This approach simulates an external attacker best and may yield insights into information leaks that can give an adversary better information on attack vectors. The disadvantage of black box testing is that it probably won't cover all of the internal controls since some of them are unlikely to be discovered in the course of the audit. Another issue is that, with no knowledge of the innards of the system, the test team may inadvertently target a subsystem that is critical to daily operations.
- *White box testing* affords the pen tester complete knowledge of the inner workings of the system even before the first scan is performed. This approach allows the test team to target specific internal controls and features and should yield a more complete assessment of the system. The downside is that white box testing may not be representative of the behaviors of an external attacker, though it may be a more accurate depiction of an insider threat.
- *Gray box testing* meets somewhere between the other two approaches. Some, but not all, information on the internal workings is provided to the test team. This helps guide their tactics toward areas we want to have thoroughly tested, while also allowing for a degree of realism in terms of discovering other features of the system. This approach mitigates the issues with both white and black box testing.

for this type of test involves determining what actions, if any, the defenders are allowed to take. Stopping every detected attack will slow down the pen testing team and may not show the depths they could've reached without forewarning to the staff.

A *double-blind test* (stealth assessment) is also a blind test to the assessors, as mentioned previously, but in this case the network security staff is not notified. This enables the test to evaluate the network's security level and the staff's responses, log monitoring, and escalation processes, and is a more realistic demonstration of the likely success or failure of an attack.

Targeted tests can involve external consultants and internal staff carrying out focused tests on specific areas of interest. For example, before a new application is rolled out, the team might test it for vulnerabilities before installing it into production. Another example is to focus specifically on systems that carry out e-commerce transactions and not the other daily activities of the organization.

Vulnerability Test vs. Penetration Test

A vulnerability assessment identifies a wide range of vulnerabilities in the environment. This is commonly carried out through a scanning tool. The idea is to identify any vulnerabilities that *potentially* could be used to compromise the security of our systems. By contrast, in a penetration test, the security professional exploits one or more vulnerabilities to prove to the customer (or your boss) that a hacker can *actually* gain access to the organization's resources.

It is important that the team start off with only basic user-level access to properly simulate different attacks. The team needs to utilize a variety of different tools and attack methods and look at all possible vulnerabilities because actual attackers only need to find one vulnerability that the defenders missed.

Red Teaming

While penetration testing is intended to uncover as many exploitable vulnerabilities as possible in the allotted time, it doesn't really emulate threat actor behaviors all that well. Adversaries almost always have very specific objectives when they attack, so just because your organization passed its pen test doesn't mean there isn't a creative way for adversaries to get in. *Red teaming* is the practice of emulating a specific threat actor (or type of threat actor) with a particular set of objectives. Whereas pen testing answers the question "How many ways can I get in?" red teaming answers the question "How can I get in and accomplish this objective?"

Red teaming more closely mimics the operational planning and attack processes of advanced threat actors that have the time, means, and motive to defeat even advanced defenses and remain undetected for long periods of time. A red team operation begins by determining the adversary to be emulated and a set of objectives. The red team then conducts reconnaissance (typically with a bit of insider help) to understand how the systems work and locate the team's objectives. The next step is to draw up a plan on how to accomplish the objectives while remaining undetected. In the more elaborate cases, the red team may create a replica of the target environment inside a cyber range in which to perform mission rehearsals and ensure the team's actions will not trigger any alerts. Finally, the red team launches the attack on the actual system and tries to reach its objectives.

As you can imagine, red teaming as described here is very costly and beyond the reach of all but the most well-resourced organizations. Most often, what you'll see is a hybrid approach that is more focused than pen testing but less intense than red teaming. We all have to do what we can with the resources we have. This is why many organizations (even very small ones) establish an internal red team that periodically comes together to think like an adversary would about some aspect of the business. It could be a new or critical information system, or a business process, or even a marketing campaign. Their job is to ask the question "If we were adversaries, how would we exploit this aspect of the business?"

EXAM TIP Don't worry about differentiating penetration testing and red teaming during the exam. If the term "red team" shows up on your test, it will most likely describe the group of people who conduct both penetration tests and red teaming.

Breach Attack Simulations

One of the problems with both pen testing and red teaming is that they amount to a point-in-time snapshot of the organizational defenses. Just because your organization did very well against the penetration testers last week doesn't necessarily mean it would do well against a threat actor today. There are many reasons for this, but the two most important ones are that things change and the test (no matter how long it takes) is never 100 percent thorough. What would be helpful as a complement to human testing would be automated testing that could happen periodically or even continually.

Breach and attack simulations (BAS) are automated systems that launch simulated attacks against a target environment and then generate reports on their findings. They are meant to be realistic but not cause any adverse effect to the target systems. For example, a ransomware simulation might use "defanged" malware that looks and propagates just like the real thing but, when successful, will only encrypt a sample file on a target host as a proof of concept. Its signature should be picked up by your network detection and response (NDR) or your endpoint detection and response (EDR) solutions. Its communications with a command-and-control (C2) system via the Internet will follow the same processes that the real thing would. In other words, each simulation is very realistic and meant to test your ability to detect and respond to it.

BAS is typically offered as a Software as a Service (SaaS) solution. All the tools, automation, and reporting take place in the provider's cloud. BAS agents can also be deployed in the target environment for better coverage under an assumed breach scenario. This covers the cases in which the adversary breached your environment using a zero-day exploit or some other mechanism that successfully evaded your defenses. In that case, you're trying to determine how well your defense in depth is working.

Log Reviews

A *log review* is the examination of system log files to detect security events or to verify the effectiveness of security controls. Log reviews actually start way before the first event is examined by a security specialist. In order for event logs to provide meaningful information, they must capture a very specific but potentially large amount of information that is grounded on both industry best practices and the organization's risk management process. There is no one-size-fits-all set of event types that will help you assess your security posture. Instead, you need to constantly tune your systems in response to the ever-changing threat landscape.

Another critical element when setting up effective log reviews for an organization is to ensure that time is standardized across all networked devices. If an incident affects three devices and their internal clocks are off by even a few seconds, then it will be significantly more difficult to determine the sequence of events and understand the overall flow of the attack. Although it is possible to normalize differing timestamps, it is an extra step that adds complexity to an already challenging process of understanding an adversary's behavior on our networks. Standardizing and synchronizing time is not a difficult thing to do. The Network Time Protocol (NTP) version 4, described in RFC 5905, is the industry standard for synchronizing computer clocks between networked devices.

Now that you have carefully defined the events you want to track and ensured all timestamps are synchronized across your network, you still need to determine where the events will be stored. By default, most log files are stored locally on the corresponding device. The challenge with this approach is that it makes it more difficult to correlate events across devices to a given incident. Additionally, it makes it easier for attackers to alter the log files of whatever devices they compromise. By centralizing the location of all log files across the organization, you address both issues and also make it easier to archive the logs for long-term retention.

Efficient archiving is important because the size of these logs will likely be significant. In fact, unless your organization is extremely small, you will likely have to deal with thousands (or perhaps even millions) of events each day. Most of these are mundane and probably irrelevant, but we usually don't know which events are important and which

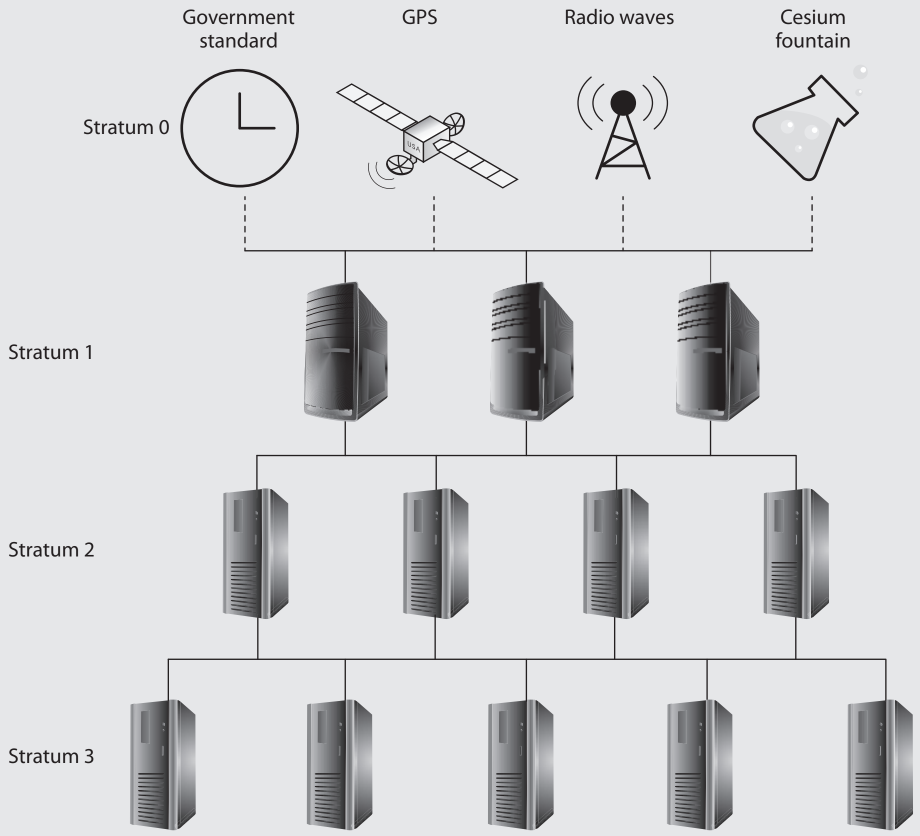
Network Time Protocol

The Network Time Protocol (NTP) is one of the oldest protocols used on the Internet and is still in widespread use today. It was originally developed in the 1980s in part to solve the problem of synchronizing trans-Atlantic network communications. Its current version, 4, still leverages statistical analysis of round-trip delays between a client and one or more time servers. The time itself is sent in a UDP datagram that carries a 64-bit timestamp on port 123.

Despite its client/server architecture, NTP employs a hierarchy of time sources organized into strata, with stratum 0 being the most authoritative. A network device on a lower stratum acts as a client to a server on a higher stratum, but could itself be a server to a node further downstream from it. Furthermore, nodes on the same stratum can and often do communicate with each other to improve the accuracy of their times.

(*Continued*)

Stratum 0 consists of highly accurate time sources such as atomic clocks, global positioning system (GPS) clocks, or radio clocks. Stratum 1 consists of primary time sources, typically network appliances with highly accurate internal clocks that are connected directly to a stratum 0 source. Stratum 2 is where you would normally see your network servers, such as your local NTP servers and your domain controllers. Stratum 3 can be thought of as other servers and the client computers on your network, although the NTP standard does not define this stratum as such. Instead, the standard allows for a hierarchy of up to 16 strata wherein the only requirement is that each strata gets its time from the higher one and serves time to the lower strata if it has any.



aren't until we've done some analysis. In many investigations, the seemingly unimportant events of days, weeks, or even months ago turn out to be the keys to understanding a security incident. So while retaining as much as possible is necessary, we need a way to quickly separate the wheat from the chaff.

Preventing Log Tampering

Log files are often among the first artifacts that attackers will use to attempt to hide their actions. Knowing this, it is up to us as security professionals to do what we can to make it infeasible, or at least very difficult, for attackers to successfully tamper with our log files. The following are the top five steps we can take to raise the bar for the bad folks:

- **Remote logging** When attackers compromise a device, they often gain sufficient privileges to modify or erase the log files on that device. Putting the log files on a separate box requires the attackers to target that box too, which at the very least buys you some time to notice the intrusion.
- **Simplex communication** Some high-security environments use one-way (or simplex) communications between the reporting devices and the central log repository. This is easily accomplished by severing the "receive" pairs on an Ethernet cable. The term *data diode* is sometimes used to refer to this approach to physically ensuring a one-way path.
- **Replication** It is never a good idea to keep a single copy of such an important resource as the consolidated log entries. By making multiple copies and keeping them in different locations, you make it harder for attackers to alter the log files, particularly if at least one of the locations is not accessible from the network (e.g., a removable device).
- **Write-once media** If one of the locations to which you back up your log files can be written to only once, you make it impossible for attackers to tamper with that copy of the data. Of course, they can still try to physically steal the media, but now you force them to move into the physical domain, which many attackers (particularly ones overseas) will not do.
- **Cryptographic hash chaining** A powerful technique for ensuring events that are modified or deleted are easily noticed is to use cryptographic hash chaining. In this technique, each event is appended the cryptographic hash (e.g., SHA-256) of the preceding event. This creates a chain that can attest to the completeness and the integrity of every event in it.

Fortunately, many solutions, both commercial and free, now exist for analyzing and managing log files and other important event artifacts. *Security information and event management (SIEM)* systems enable the centralization, correlation, analysis, and retention of event data in order to generate automated alerts. Typically, a SIEM provides a dashboard interface that highlights possible security incidents. It is then up to the security specialists to investigate each alert and determine if further action is required. The challenge, of course, is ensuring that the number of false positives is kept fairly low and that the number of false negatives is kept even lower.

Synthetic Transactions

Many of our information systems operate on the basis of transactions. A user (typically a person) initiates a transaction that could be anything from a request for a given web page to a wire transfer of half a million dollars to an account in Switzerland. This transaction is processed by any number of other servers and results in whatever action the requestor wanted. This is considered a real transaction. Now suppose that a transaction is not generated by a person but by a script. This is considered a *synthetic transaction*.

The usefulness of synthetic transactions is that they allow us to systematically test the behavior and performance of critical services. Perhaps the simplest example is a scenario in which you want to ensure that your home page is up and running. Rather than waiting for an angry customer to send you an e-mail saying that your home page is unreachable, or spending a good chunk of your day visiting the page on your browser, you could write a script that periodically visits your home page and ensures that a certain string is returned. This script could then alert you as soon as the page is down or unreachable, allowing you to investigate before you would've otherwise noticed it. This could be an early indicator that your web server was hacked or that you are under a distributed denial-of-service (DDoS) attack.

Synthetic transactions can do more than simply tell you whether a service is up or down. They can measure performance parameters such as response time, which could alert you to network congestion or server overutilization. They can also help you test new services by mimicking typical end-user behaviors to ensure the system works as it ought to. Finally, these transactions can be written to behave as malicious users by, for example, attempting a cross-site scripting (XSS) attack and ensuring your controls are effective. This is an effective way of testing software from the outside.

Real User Monitoring vs. Synthetic Transactions

Real user monitoring (RUM) is a passive way to monitor the interactions of real users with a web application or system. It uses agents to capture metrics such as delay, jitter, and errors from the user's perspective. RUM differs from synthetic transactions in that it uses real people instead of scripted commands. While RUM more accurately captures the actual user experience, it tends to produce noisy data (e.g., incomplete transactions due to users changing their minds or losing mobile connectivity) and thus may require more back-end analysis. It also lacks the elements of predictability and regularity, which could mean that a problem won't be detected during low utilization periods.

Synthetic transactions, on the other hand, are very predictable and can be very regular, because their behaviors are scripted. They can also detect rare occurrences more reliably than waiting for a user to actually trigger that behavior. Synthetic transactions also have the advantage of not having to wait for a user to become dissatisfied or encounter a problem, which makes them a more proactive approach.

It is important to note that RUM and synthetic transactions are different ways of achieving the same goal. Neither approach is the better one in all cases, so it is common to see both employed contemporaneously.

Code Reviews

So far, all the security testing we have discussed looks at the behaviors of our systems. This means that we are only assessing the externally visible features without visibility into the inner workings of the system. If you want to test your own software system from the inside, you could use a *code review*, a systematic examination of the instructions that comprise a piece of software, performed by someone other than the author of that code. This approach is a hallmark of mature software development processes. In fact, in many organizations, developers are not allowed to push out their software modules until someone else has signed off on them after doing a code review. Think of this as proofreading an important document before you send it to an important person. If you try to proofread it yourself, you will probably not catch all those embarrassing typos and grammatical errors as easily as someone else could who is checking it for you.

Code reviews go way beyond checking for typos, though that is certainly one element of it. It all starts with a set of coding standards developed by the organization that wrote the software. This could be an internal team, an outsourced developer, or a commercial vendor. Obviously, code reviews of commercial off-the-shelf (COTS) software are extremely rare unless the software is open source or you happen to be a major government agency. Still, each development shop will have a style guide or a set of documented coding standards that covers everything from how to indent the code to when and how to use existing code libraries. So a preliminary step to the code review is to ensure the author followed the team's style guide or standards. In addition to helping the maintainability of the software, this step gives the code reviewer a preview of the magnitude of the work ahead; a sloppy coder will probably have a lot of other, harder-to-find defects in his code.

After checking the structure and format of the code, the reviewer looks for uncalled or unneeded functions or procedures. These lead to "code bloat," which makes it harder to maintain and secure the application. For this same reason, the reviewer looks for modules that are excessively complex and should be restructured or split into multiple routines. Finally, in terms of reducing complexity, the reviewer looks for blocks of repeated code that could be refactored. Even better, these could be pulled out and turned into external reusable components such as library functions.

An extreme example of unnecessary (and dangerous) procedures are the code stubs and test routines that developers often include in their developmental software. There have been too many cases in which developers left test code (sometimes including hard-coded credentials) in final versions of software. Once adversaries discover this condition, exploiting the software and bypassing security controls is trivial. This problem is insidious, because developers sometimes comment out the code for final testing, just in case the tests fail and they have to come back and rework it. They may make a mental note to revisit the file and delete this dangerous code, but then forget to do so. While commented code is unavailable to an attacker after a program is compiled (unless they have access to the source code), the same is not true of the scripts that are often found in distributed applications.

Defensive programming is a best practice that all software development operations should adopt. In a nutshell, it means that as you develop or review the code, you are constantly looking for opportunities for things to go badly. Perhaps the best example of defensive programming is the practice of treating all inputs, whether they come from a keyboard, a file,

A Code Review Process

1. Identify the code to be reviewed (usually a specific function or file).
2. The team leader organizes the inspection and makes sure everyone has access to the correct version of the source code, along with all supporting artifacts.
3. Everyone on the team prepares for inspection by reading through the code and making notes.
4. A designated team member collates all the obvious errors offline (not in a meeting) so they don't have to be discussed during the inspection meeting (which would be a waste of time).
5. If everyone agrees the code is ready for inspection, then the meeting goes ahead.
6. The team leader displays the code (with line numbers) via an overhead projector so everyone can read through it. Everyone discusses bugs, design issues, and anything else that comes up about the code. A scribe (not the author of the code) writes everything down.
7. At the end of the meeting, everyone agrees on a "disposition" for the code:
   - Passed: Code is good to go
   - Passed with rework: Code is good so long as small changes are fixed
   - Reinspect: Fix problems and have another inspection
8. After the meeting, the author fixes any mistakes and checks in the new version.
9. If the disposition of the code in step 7 was passed with rework, the team leader checks off the bugs that the scribe wrote down and makes sure they're all fixed.
10. If the disposition of the code in step 7 was reinspect, the team leader goes back to step 2 and starts over again.

or the network, as untrusted until proven otherwise. This user input validation can be a bit trickier than it sounds, because you must understand the context surrounding the input. Are you expecting a numerical value? If so, what is the acceptable range for that value? Can this range change over time? These and many other questions need to be answered before we can decide whether the inputs are valid. Keep in mind that many of the oft-exploited vulnerabilities we see have a lack of input validation as their root cause.

Code Testing

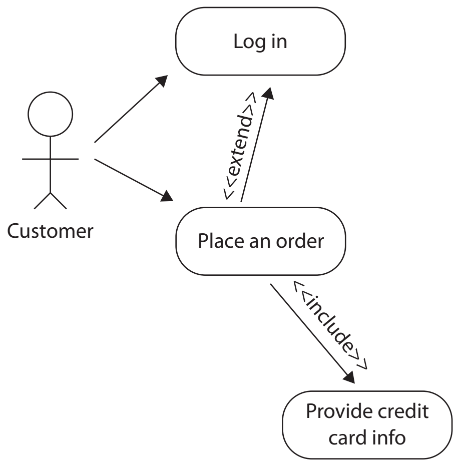
We will discuss in Chapters 24 and 25 the multiple types of tests to which we must subject our code as part of the software development process. However, once the code comes out of development and before we put it into a production environment, we must ensure that it meets our security policies. Does it encrypt all data in transit? Is it possible to bypass authentication or authorization controls? Does it store sensitive data in unencrypted temporary files? Does it reach out to any undocumented external resources (e.g., for library updates)? The list goes on, but the point is that security personnel are incentivized differently than software developers. The programmer gets paid to implement features in software, while the security practitioner gets paid to keep systems secure.

Most mature organizations have an established process to certify that software systems are secure enough to be installed and operated on their networks. There is typically a follow-on to that process, which is when a senior manager (hopefully after reading the results of the certification) authorizes (or accredits) the system.

Misuse Case Testing

Use cases are structured scenarios that are commonly used to describe required functionality in an information system. Think of them as stories in which an external actor (e.g., a user) wants to accomplish a given goal on the system. The use case describes the sequence of interactions between the actor and the system that result in the desired outcome. Use cases are textual but are often summarized and graphically depicted using a Unified Modeling Language (UML) use case diagram such as the one shown in Figure 18-3. This figure illustrates a very simple view of a system in which a customer places online orders. According to the UML, actors such as our user are depicted using stick figures, and the actors' use cases are depicted as verb phrases inside ovals. Use cases can be related to one another in a variety of ways, which we call *associations*. The most common ways in which use cases are associated are by including another use case (that is, the included use case is always executed when the preceding one is) or by extending a use case (meaning that the second use case may or may not be executed depending on a decision point in the main use case). In Figure 18-3, our customer attempts to place an order and may be prompted to log in if she hasn't already done so, but she will always be asked to provide her credit card information.

Figure 18-3
UML use case diagram



While use cases are very helpful in analyzing requirements for the normal or expected behavior of a system, they are not particularly useful for assessing its security. That is what misuse cases do for us. A *misuse case* is a use case that includes threat actors and the tasks they want to perform on the system. Threat actors are normally depicted as stick figures with shaded heads and their actions (or misuse cases) are depicted as shaded ovals, as shown in Figure 18-4. As you can see, the attacker in this scenario is interested in guessing passwords and stealing credit card information.

Misuse cases introduce new associations to our UML diagram. The threat actor's misuse cases are meant to threaten a specific portion or legitimate use case of our system. You will typically see shaded ovals connected to unshaded ones with an arrow labeled <<threaten>> to denote this relationship. On the other hand, system developers and security personnel can implement controls that mitigate these misuses. These create new unshaded ovals connected to shaded ones with arrows labeled <<mitigate>>.

The idea behind misuse case testing is to ensure we have effectively addressed each of the risks we identified and decided to mitigate during our risk management process and that are applicable to the system under consideration. This doesn't mean that misuse case testing needs to include all the possible threats to our system, but it should include the ones we decided to address. This process forces system developers and integrators to incorporate the products of our risk management process into the early stages of any system development effort. It also makes it easier to quickly step through a complex system and ensure that effective security controls are in the right places without having to get deep into the source code, which is what we describe next.

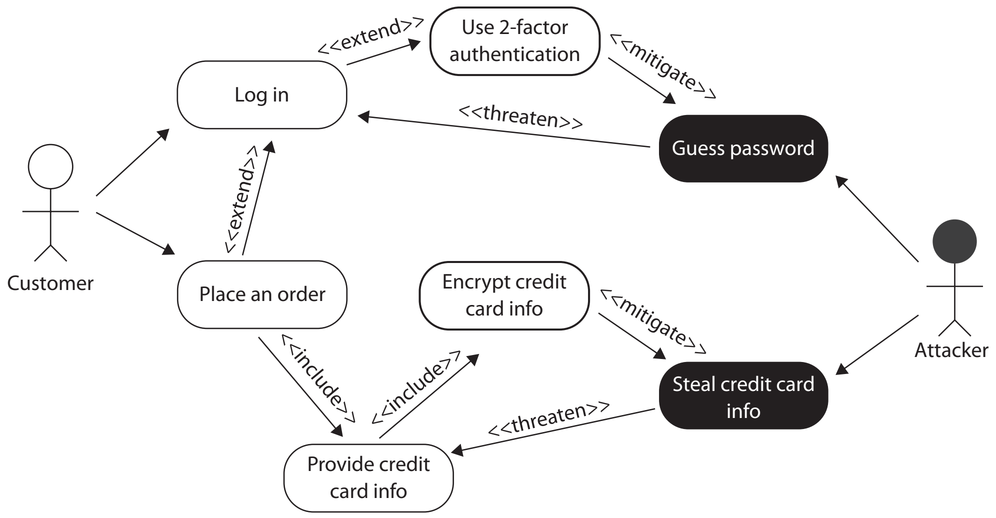


Figure 18-4 UML misuse case diagram

Test Coverage

Test coverage is a measure of how much of a system is examined by a specific test (or group of tests), which is typically expressed as a percentage. For example, if you are developing a software system with 1,000 lines of code and your suite of unit tests executes 800 of those, then you would have 80 percent test coverage. Why wouldn't we just go for 100 percent? Because it likely would be too expensive for the benefit we would get from it. We normally only see this full coverage being required in safety-critical systems like those used in aviation and medical devices.

Test coverage also applies to things other than software. Suppose you have 100 security controls in your organization. Testing all of them in one assessment or audit may be too disruptive or expensive (or both), so you schedule smaller evaluations throughout the year. Each quarter, for instance, you run an assessment with tests for one quarter of the controls. In this situation, your quarterly test coverage is 25 percent but your annual coverage is 100 percent.

Interface Testing

When we think of interfaces, we usually envision a graphical user interface (GUI) for an application. While GUIs are one kind of interface, there are others that are potentially more important. At its essence, an interface is an exchange point for data between systems and/or users. You can see this in your computer's network interface card (NIC), which is the exchange point for data between your computer (a system) and the local area network (another system). Another example of an interface is an application programming interface (API), a set of points at which a software system (e.g., the application) exchanges information with another software system (e.g., the libraries).

Interface testing is the systematic evaluation of a given set of these exchange points. This assessment should include both known good exchanges and known bad exchanges in order to ensure the system behaves correctly at both ends of the spectrum. The real rub is in finding test cases that are somewhere in between. In software testing, these are called *boundary conditions* because they lie at the boundary that separates the good from the bad. For example, if a given packet should contain a payload of no more than 1024 bytes, how would the system behave when presented with 1024 bytes plus one bit (or byte) of data? What about exactly 1024 bytes? What about 1024 bytes minus one bit (or byte) of data? As you can see, the idea is to flirt with the line that separates the good from the bad and see what happens when we get really close to it.

There are many other test cases we could consider, but the most important lesson here is that the primary task of interface testing is to dream up all the test cases ahead of time, document them, and then insert them into a repeatable and (hopefully) automated test engine. This way you can ensure that as the system evolves, a specific interface is always tested against the right set of test cases. We will talk more about software testing in Chapter 24, but for now you should remember that interface testing is a special case of something called *integration testing*, which is the assessment of how different parts of a system interact with each other.

Compliance Checks

We have framed our discussion of security tests in terms of evaluating the effectiveness of our technical controls. That really should be our first concern. However, many of us work in organizations that must comply with some set of regulations. If you are not in a (formally) regulated sector, you may still have regulations or standards with which you voluntarily comply. Finally, if you have an information security management system (ISMS) in place like we discussed in Chapter 1 (and you really should), you want to ensure the controls that are listed in there are actually working. In any of these three cases you can use the testing techniques we've discussed to demonstrate compliance.

Compliance checks are point-in-time verifications that specific security controls are implemented and performing as expected. For example, your organization may process payment card transactions for its customers and, as such, be required by PCI DSS to run annual penetration tests and quarterly vulnerability scans by an approved vendor. Your compliance checks would be the reports for each of these tests. Or, you may have an SPF record check as one of the e-mail security controls in your ISMS. (Recall from Chapter 13 that the Sender Policy Framework mitigates the risk of forged e-mail sources.) You could perform a log review to perform and document a compliance check on that control. If your e-mail server performed SPF checks on all e-mail messages in a selected portion of its logs, then you know the control is in place and performing as expected.

Conducting Security Audits

While compliance checks are point-in-time tests, audits tend to cover a longer period of time. The question is not "Is this control in place and working?" but rather "Has this control been in place for the last year?" Audits, of course, can leverage compliance checks. In the SPF record check example at the end of the previous section, the auditor would simply look through the compliance checks and, if they were performed and documented regularly, use those as evidence of compliance during an audit.

As simple as it sounds, establishing a clear set of goals is probably the most important step of planning a security audit. Since we usually can't test everything, we have to focus our efforts on whatever it is that we are most concerned about. An audit could be driven by regulatory or compliance requirements, by a significant change to the architecture of the information system, or by new developments in the threat facing the organization. There are many other possible scenarios, but these examples are illustrative of the vastly different objectives for our assessments.

Once our goals are established, we need to define the scope of the assessment:

- Which subnets and systems are we going to test?
- Are we going to look at user artifacts, such as passwords, files, and log entries, or at user behaviors, such as their response to social engineering attempts?
- Which information will we assess for confidentiality, integrity, and availability?
- What are the privacy implications of our audit?
- How will we evaluate our processes, and to what extent?

If our goals are clearly laid out, answering these questions should be a lot easier.

The scope of the audit should be determined in coordination with business unit managers. All too often security professionals focus on IT and forget about the business cases. In fact, business managers should be included early in the audit planning process and should remain engaged throughout the event. This not only helps bridge the gap between the two camps but also helps identify potential areas of risk to the organization brought on by the audit itself. Just imagine what would happen if your assessment interfered with a critical but nonobvious business process and ended up costing the organization a huge amount of money. (We call that an RGE, or résumé-generating event.)

Having decided who will actually conduct our audit, we are now in a position to plan the event. The plan is important for a variety of reasons:

- We must ensure that we are able to address whatever risks we may be introducing into the business processes. Without a plan, these risks are unknown and not easily mitigated.
- Documenting the plan ensures that we meet each of our audit goals. Audit teams sometimes attempt to follow their own scripted plan, which may or may not address all of the organization's goals for a specific audit.
- Documenting the plan will help us remember the items that were *not* in the scope of the assessment. Recall that we already acknowledged that we can't possibly test everything, so this specifies the things we did not test.
- The plan ensures that the audit process is repeatable. Like any good science experiment, we should be able to reproduce the results by repeating the process. This is particularly important because we may encounter unexpected results worth further investigation.

Information System Security Audit Process

1. **Determine the goals**, because everything else hinges on this.
2. **Involve the right business unit leaders** to ensure the needs of the business are identified and addressed.
3. **Determine the scope**, because not everything can be tested.
4. **Choose the audit team**, which may consist of internal or external personnel, depending on the goals, scope, budget, and available expertise.
5. **Plan the audit** to ensure all goals are met on time and on budget.
6. **Conduct the audit** while sticking to the plan and documenting any deviations therefrom.
7. **Document the results**, because the wealth of information generated is both valuable and volatile.
8. **Communicate the results** to the right leaders to achieve and sustain a strong security posture.

Having developed a detailed plan for the audit, we are finally in a position to get to the fun stuff. No matter how much time and effort we put into planning, inevitably we will find tasks we have to add, delete, change, or modify. Though we clearly want to minimize the number of these changes, they are really a part of the process that we just have to accept. The catch is that we must consciously decide to accept them, and then we absolutely must document them.

NOTE In certain cases, such as regulatory compliance, the parameters of the audit may be dictated and performed by an external team of auditors. This means that the role of the organization is mostly limited to preparing for the audit by ensuring all required resources are available to the audit team.

The documentation we start during the planning process must continue all the way through to the results. In all but the most trivial assessments, we are likely to generate reams of data and information. This information is invaluable in that it captures a snapshot in time of our security posture. If nothing else, it serves to benchmark the effectiveness of our controls so that we can compare audits and determine trends. Typically, however, this detailed documentation allows the security staff to drill into unexpected or unexplainable results and do some root cause analysis. If you capture all the information, it will be easier to produce reports for target audiences without concern that you may have deleted (or failed to document) any important data points.

Ultimately, the desired end state of any audit is to effectively communicate the results to the target audiences. The manner in which we communicate results to executives is very different from the manner in which we communicate results to the IT team members. This gets back to the point made earlier about capturing and documenting both the plan and the details and products of its execution. It is always easier to distill information from a large data set than to justify a conclusion when the facts live only in our brains. Many a security audit has been ultimately unsuccessful because the team has not been able to communicate effectively with the key stakeholders.

Internal Audits

In a perfect world, every organization would have an internal team capable of performing whatever audits were needed. Alas, we live in a far-from-perfect world in which even some of the best-resourced organizations lack this capability. But if your organization does have such a team on hand, its ability to implement continuous improvement of your organization's security posture offers some tremendous advantages.

One of the benefits of using your own personnel to do an audit is that they are familiar with the inner workings of your organization. This familiarity allows them to get right to work and not have to spend too much time getting oriented to the cyber terrain. Some may say that this insider knowledge gives them an unrealistic advantage because few adversaries could know as much about the systems as those who operate and defend them. It is probably more accurate to state that advanced adversaries can often approach the level of knowledge about an organization that an internal audit team would have. In any case, if the purpose of the audit is to leave no stone unturned and test the weakest, most obscure parts of an information system, then an internal team will likely get closer to that goal than any other.

Using internal assets also allows the organization to be more agile in its assessment efforts. Since the team is always available, all that the leadership would need to do is to reprioritize their tests to adapt to changing needs. For example, suppose a business unit is scheduled to be audited yearly, but the latest assessment's results from a month ago were abysmal and represent increased risk to the organization. The security management could easily reschedule other tests to conduct a follow-up audit three months later. This agility comes at no additional cost to the organization, which typically would not be true if engaging a third-party team.

The downsides of using an internal team include the fact that they likely have limited exposure to other approaches to both securing and exploiting information systems. Unless the team has some recent hires with prior experience, the team will probably have a lot of depth in the techniques they know, but not a lot of breadth, since they will have developed mostly the skills needed to test only their own organization.

A less obvious disadvantage of using internal auditors is the potential for conflicts of interest to exist. If the auditors believe that their bosses or coworkers may be adversely affected by a negative report or even by the documented presence of flaws, the auditors may be reluctant to accurately report their findings. The culture of the organization is probably the most influential factor in this potential conflict. If the climate is one of openness and trust, then the auditors are less likely to perceive any risk to their higher-ups or coworkers regardless of their findings. Conversely, in very rigid bureaucratic organizations with low tolerance for failures, the potential for conflicts of interest will likely be higher.

Another aspect of the conflict-of-interest issue is that the team members or their bosses may have an agenda to pursue with the audit. If they are intent on securing better

Conducting Internal Audits

Here are some best practices to get the most bang out of internal audits that you conduct:

- **Mark your calendars** Nothing takes the wind out of your audit's sails quicker than not having all key personnel and resources available. Book them early.
- **Prepare the auditors** Rehearse the process with the auditors so everyone is on the same sheet of music. Ensure everyone knows the relevant policies and procedures.
- **Document everything** Consider having note-takers follow the auditors around documenting everything they do and observe.
- **Make the report easy to read** Keep in mind that you will have at least two audiences: managers and technical personnel. Make the report easy to read for both.

funding, they may be tempted to overstate or even fabricate security flaws. Similarly, if they believe that another department needs to be taught a lesson (perhaps to get them to improve their willingness to "play nice" with the security team), the results could deliberately or subconsciously be less than objective. Politics and team dynamics clearly should be considered when deciding whether to use internal audit teams.

External Audits

When organizations come together to work in an integrated manner, special care must be taken to ensure that each party promises to provide the necessary level of protection, liability, and responsibility, which should be clearly defined in the contracts each party signs. Auditing and testing should be performed to ensure that each party is indeed holding up its side of the bargain. An *external audit* (sometimes called a second-party audit) is one conducted by (or on behalf of) a business partner.

External audits are tied to contracts. In today's business and threat environments, it is becoming commonplace for contracts to have security provisions. For example, a contract for disposing of computers may require the service provider to run background checks on all its employees, to store the computers in secure places until they are wiped, to overwrite all storage devices with alternating 1's and 0's at least three times, and to agree to being audited for any or all of these terms. Once the contract is in place, the client organization could demand access to people, places, and information to verify that the security provisions are being met by the contractor.

To understand why external audits are important, you don't have to go any further than the Target data breach of 2013. That incident was possible because Target was doing

Conducting and Facilitating External Audits

It would be pretty unusual for you to conduct an external audit on a contractor. Instead, you would normally ask the contractor to perform an internal audit (scoped in accordance with the contract) or bring in a third-party auditor (described in the next section). Regardless, here are some tips to consider whether you are on the giving or receiving end of the deal:

- **Learn the contract** An external audit, by definition, is scoped to include only the contractual obligations of an organization. Be sure the audit doesn't get out of control.
- **Schedule in- and out-briefs** Schedule an in-brief to occur right before the audit starts to bring all stakeholders together. Schedule an out-brief to occur immediately after the audit is complete to give the audited organization a chance to address any misconceptions or errors.
- **Travel in pairs** Ensure the organization being audited has someone accompanying each team of auditors. This will make things go smoother and help avoid misunderstandings.
- **Keep it friendly** The whole goal of this process is to engender trust.

business with Fazio Mechanical Services, who provided Target with heating, ventilation, and air conditioning (HVAC) services. The security postures of both organizations were vastly different, so the attackers targeted the weaker link: Fazio. Admittedly, Target made some costly mistakes that got it into that mess, but had its IT security personnel understood the information system security management practices of its partner, they may have been able to avoid the breach. How could they have learned of Fazio's weaknesses? By auditing them.

Third-Party Audits

Sometimes, you have no choice but to bring in a third party to audit your information systems' security. This is most often the case when you need to demonstrate compliance with some government regulation or industry standard. Even if you do have a choice, bringing in external auditors has advantages over using an internal team. For starters, the external auditors probably have seen and tested many information systems in different organizations. This means that they will almost certainly bring to your organization knowledge that it wouldn't otherwise be able to acquire. Even if you have some internal auditors with prior experience, they are unlikely to approach the breadth of experience that contractors who regularly test a variety of organizations will bring to the table.

Another advantage of third-party auditors is that they are unaware of the internal dynamics and politics of the target organization. This means that they have no favorites or agendas other than the challenge of finding flaws. This objectivity may give them an edge in testing, particularly if the alternative would've been to use internal personnel who played a role in implementing the controls in the first place and thus may overlook or subconsciously impede the search for defects in those controls.

The obvious disadvantage of hiring an external team is cost. Price tags in the tens of thousands of dollars are not uncommon, even on the low end of the scale. If nothing else, this probably means that you won't be able to use external auditors frequently (if at all). Even at the high end of the pay scale, it is not uncommon to find testers who rely almost exclusively on high-end scanners that do all the work (and thinking) for them. It is truly unfortunate when an organization spends a significant amount of money only to find out the tester simply plugs his laptop into the network, runs a scanner, and prints a report.

Even if you find an affordable and competent team to test your information systems, you still have to deal with the added resources required to orient them to the organization and supervise their work. Even with signed nondisclosure agreements (NDAs), most companies don't give free rein to their external auditors without some level of supervision. In addition, the lack of knowledge of the inner workings of the organization typically translates into the auditors taking a longer time to get oriented and be able to perform the test.

NOTE Signing a nondisclosure agreement is almost always a prerequisite before a third-party team is permitted to audit an organization's systems.

Facilitating Third-Party Audits

Your organization will typically pay for the third party to audit you, but if you're doing the audit for compliance or contractual reasons, the auditor won't be working for you. The job of a third-party auditor is to certify (using their own reputation) that you are meeting whatever standards are in scope. Regardless, the following are useful tips:

- **Know the requirements** Go through the audit requirements line by line to ensure you know exactly what the third-party auditor will be looking at. Call the auditor if you have any questions.
- **Pre-audit** Conduct your own internal audit using the same list of requirements to minimize the number of surprises.
- **Lock in schedules** Ensure the right staff will be available when the auditors show up, even if there's only a small chance they'll be needed.
- **Get organized** The audit team will likely need access to a large and diverse set of resources, so make sure you have them all assembled in one place and organized.
- **Keep the boss informed** A third-party audit, by definition, is an important event for the organization, and we all know that bad news doesn't get better with time. Be sure to keep the senior managers informed, especially of any potential deficient areas.

While there is no clear winner between using internal auditors and third-party auditors, sometimes the latter is the only choice where regulatory requirements such as the Sarbanes-Oxley Act force an organization to outsource the test. These are called *compliance audits* and must be performed by external parties.

EXAM TIP You will most likely see questions on external audits in the context of supply-chain risk management. Third-party audits will show up in questions dealing with compliance issues.

Chapter Review

As security professionals, evaluating the security posture of our organizations is an iterative and continuous process. This chapter discussed a variety of techniques that are helpful in determining how well you are mitigating risks with your technical and administrative controls. Whether you are doing your own audits or validating the audit plans provided by a third party, you should now know what to look for and how to evaluate proposals. Along the way, this chapter also covered some specific threats and opportunities that should play a role in your assessment plan. It is important to keep in mind that everything covered in this chapter is grounded in the risk management discussed in Chapter 2. If you do not keep in mind the specific threats and risks with which your organization is concerned, then it is very difficult to properly address them.

Quick Review

- An audit is a systematic assessment of the security controls of an information system.
- Setting a clear set of goals is probably the most important step of planning a security audit.
- A vulnerability test is an examination of a system for the purpose of identifying, defining, and ranking its vulnerabilities.
- Penetration testing is the process of simulating attacks on a network and its systems at the request of the owner.
- Red teaming is the practice of emulating a specific threat actor (or type of threat actor) with a particular set of objectives.
- Black box testing treats the system being tested as completely opaque.
- White box testing affords the auditor complete knowledge of the inner workings of the system even before the first scan is performed.
- Gray box testing gives the auditor some, but not all, information about the internal workings of the system.
- A blind test is one in which the assessors only have publicly available data to work with and the network security staff is aware that the testing will occur.
- A double-blind test (stealth assessment) is a blind test in which the network security staff is not notified that testing will occur.
- Breach and attack simulations (BAS) are automated systems that launch simulated attacks against a target environment and then generate reports on their findings.
- A log review is the examination of system log files to detect security events or to verify the effectiveness of security controls.
- Synthetic transactions are scripted events that mimic the behaviors of real users and allow security professionals to systematically test the performance of critical services.
- A code review is a systematic examination of the instructions that comprise a piece of software, performed by someone other than the author of that code.
- A misuse case is a use case that includes threat actors and the tasks they want to perform on the system.
- Test coverage is a measure of how much of a system is examined by a specific test (or group of tests).

- Interface testing is the systematic evaluation of a given set of exchange points for data between systems and/or users.
- Compliance checks are point-in-time verifications that specific security controls are implemented and performing as expected.
- Internal audits benefit from the auditors' familiarity with the systems, but may be hindered by a lack of exposure to how others attack and defend systems.
- External audits happen when organizations have a contract in place that includes security provisions. The contracting party can demand to audit the contractor to ensure those provisions are being met.
- Third-party audits typically bring a much broader background of experience that can provide fresh insights, but can be expensive.

Questions

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

1. Internal audits are the preferred approach when which of the following is true?

A. The organization lacks the organic expertise to conduct them.

B. Regulatory requirements dictate the use of a third-party auditor.

C. The budget for security testing is limited or nonexistent.

D. There is concern over the spillage of proprietary or confidential information.

2. All of the following are steps in the security audit process *except*

A. Document the results.

B. Convene a management review.

C. Involve the right business unit leaders.

D. Determine the scope.

3. Which of the following is an advantage of using third-party auditors?

A. They may have knowledge that an organization wouldn't otherwise be able to leverage.

B. Their cost.

C. The requirement for NDAs and supervision.

D. Their use of automated scanners and reports.

4. Choose the term that describes an audit performed to demonstrate that an organization is complying with its contractual obligations to another organization.
   - **A.** Internal audit
   - **B.** Third-party audit
   - **C.** External audit
   - **D.** Compliance audit
5. Which of the following is true of a vulnerability assessment?
   - **A.** The aim is to identify as many vulnerabilities as possible.
   - **B.** It is not concerned with the effects of the assessment on other systems.
   - **C.** It is a predictive test aimed at assessing the future performance of a system.
   - **D.** Ideally it is fully automated, with no human involvement.
6. An assessment whose goal is to assess the susceptibility of an organization to social engineering attacks is best classified as
   - **A.** Physical testing
   - **B.** Personnel testing
   - **C.** Vulnerability testing
   - **D.** Network testing
7. Which of the following is an assessment that affords the auditor detailed knowledge of the system's architecture before conducting the test?
   - **A.** White box testing
   - **B.** Gray box testing
   - **C.** Black box testing
   - **D.** Zero knowledge testing
8. Vulnerability scans normally involve all the following *except*
   - **A.** The identification of active hosts on the network
   - **B.** The identification of malware on all hosts
   - **C.** The identification of misconfigured settings
   - **D.** The identification of operating systems
9. Security event logs can best be protected from tampering by which of the following?
   - **A.** Encrypting the contents using asymmetric key encryption
   - **B.** Ensuring every user has administrative rights on their own workstations
   - **C.** Using remote logging over simplex communications media
   - **D.** Storing the event logs on DVD-RW

10. Synthetic transactions are best described as
    - **A.** Real user monitoring (RUM)
    - **B.** Transactions that fall outside the normal purpose of a system
    - **C.** Transactions that are synthesized from multiple users' interactions with the system
    - **D.** A way to test the behavior and performance of critical services
11. Suppose you want to study the actions an adversary may attempt against your system and test the effectiveness of the controls you have emplaced to mitigate the associated risks. Which of the following approaches would best allow you to accomplish this goal?
    - **A.** Misuse case testing
    - **B.** Use case testing
    - **C.** Real user monitoring (RUM)
    - **D.** Fuzzing
12. Code reviews include all of the following *except*
    - **A.** Ensuring the code conforms to applicable coding standards
    - **B.** Discussing bugs, design issues, and anything else that comes up about the code
    - **C.** Agreeing on a "disposition" for the code
    - **D.** Fuzzing the code
13. Interface testing could involve which of the following?
    - **A.** The application programming interface (API)
    - **B.** The graphical user interface (GUI)
    - **C.** Both of the above
    - **D.** None of the above

Answers

1. **C.** Third-party auditors are almost always fairly expensive, so if the organization's budget does not support their use, it may be necessary to use internal assets to conduct the audit.
2. **B.** The management review is not a part of any audit. Instead, this review typically uses the results of one or more audits in order to make strategic decisions.
3. **A.** Because they perform audits in multiple other organizations, and since their knowledge is constantly refreshed, third-party auditors almost always have knowledge and insights that would otherwise be unavailable to the organization.

4. **C.** External audits are used to ensure that contractors are meeting their contractual obligations, so that is the best answer. A compliance audit would apply to regulatory or industry standards and would almost certainly be a third-party audit, which makes answer D a poor fit in most cases.
5. **A.** One of the principal goals of a vulnerability assessment is to identify as many security flaws as possible within a given system, while being careful not to disrupt other systems.
6. **B.** Social engineering is focused on people, so personnel testing is the best answer.
7. **A.** White box testing gives the tester detailed information about the internal workings of the system under study. Gray box testing provides *some* information, so it is not the best answer to this question.
8. **B.** Vulnerability testing does not normally include scanning hosts for malware. Instead, it focuses on finding flaws that malware could potentially exploit.
9. **C.** Using a remote logging host raises the bar for attackers because if they are able to compromise one host, they would have to compromise the remote logger in order to tamper with the logs. The use of a simplex channel further hinders the attackers.
10. **D.** Synthetic transactions are those that simulate the behavior of real users, but are not the result of real user interactions with the system. They allow an organization to ensure that services are behaving properly without having to rely on user complaints to detect problems.
11. **A.** Misuse case testing allows us to document both an adversary's desired actions on a system and the controls that are meant to thwart that adversary. It is similar to developing use cases, but with a malicious user's actions in mind instead of those of legitimate users.
12. **D.** Fuzzing is a technique for detecting flaws in the code by bombarding it with massive amounts of random data. This is not part of a code review, which focuses on analyzing the source code, not its response to random data.
13. **C.** Interface testing covers the exchange points within different components of the system. The API is the exchange point between the system and the libraries it leverages, while the GUI is the exchange point between the system and the users. Testing either of these would constitute an interface test.

This page intentionally left blank

CHAPTER 19

Measuring Security

This chapter presents the following:

- Security metrics
- Security process data
- Reporting
- Management review and approval

One accurate measurement is worth a thousand expert opinions.

—Grace Hopper

The reason we conduct security assessments is that we want to answer specific questions. Is the firewall blocking dangerous traffic? How many systems are vulnerable? Can we detect (and block) phishing attacks? (The list goes on.) These questions all are important but also are very tactical. The typical board of directors or C-suite won't understand or care about them. What they do care about is how the organization is generating value for its stakeholders. Part of our job as cybersecurity leaders is to turn tactical observations into strategic insights. To do this, we need to be able to measure the right parts of our information security management system (ISMS) over time, analyze the results, and present them in an actionable manner to other leaders in our organizations. This is what this chapter is all about.

Quantifying Security

How can you tell whether you are moving toward or away from your destination? In the physical world, we use all sorts of environmental cues such as road signs and landmarks. Oftentimes, we can also use visual cues to assess the likely risk in our travels. For instance, if a sign on a hiking trail is loose and can pivot around its pole, then we know that there is a chance that the direction in which it points is not the right one. If a landmark is a river crossing and the waters are much higher than normal, we know we run the risk of being swept downstream. But when it comes to our security posture, how can we tell whether we're making progress and whether we're taking risks?

Attempting to run an ISMS without adequate metrics is perhaps more dangerous than not managing security at all. The reason is that, like following misplaced trail signs, using the wrong metrics can lead the organization down the wrong path and result in worse outcomes than would be seen if all is left to chance. Fortunately, the International Organization for Standardization (ISO) has published an industry standard for developing and using metrics that measure the effectiveness of a security program. ISO/IEC 27004, *Information security management — Monitoring, measurement, analysis and evaluation*, outlines a process by which to measure the performance of security controls and processes. Keep in mind that a key purpose of this standard is to support continuous improvement in an organization's security posture.

At this point, it is helpful to define a handful of terms:

- **Factor** An attribute of a system (such as an ISMS) that has a value that can change over time. Examples of factors are the number of alerts generated by an intrusion detection system (IDS) or the number of events investigated by the incident response (IR) team.
- **Measurement** A quantitative observation of a factor at a particular point in time. In other words, this is raw data. Two examples of measurements would be 356 IDS alerts in the last 24 hours and 42 verified events investigated by the IR team in the month of January.
- **Baseline** A value for a factor that provides a point of reference or denotes that some condition is met by achieving some threshold. For example, a baseline could be the historic trend in the number of IDS alerts over the past 12 months (a reference line) or a goal that the IR team will investigate 100 events or less in any given month (a threshold value).
- **Metric** A derived value that is generated by comparing multiple measurements against each other or against a baseline. Metrics are, by their very nature, comparative. Building upon the previous examples, an effective metric could be the ratio of verified incidents to IDS alerts during a 30-day period.
- **Indicator** A particularly important metric that describes a key element of the effectiveness of a system (such as an ISMS). In other words, indicators are meaningful to business leaders. If one of management's goals is to minimize the number of high-severity incidents, then an indicator could be the ratio of such incidents declared during a reporting period compared to an established baseline.

To put this all together visually, Figure 19-1 shows the relationship between these terms. Suppose you have a bunch of processes that make up your ISMS but there are two factors that are particularly important for you to track. Let's call them Factor A and Factor B, which could be anything, but for this hypothetical scenario, assume that they are network detection and response (NDR) system alerts and incidents declared. You intend to periodically take measurements for these factors, and the very first time you do so you get measurements A1 (say, 42 alerts) and B1 (7 incidents). You compare these measurements to each other and get your first metric A1:B1, which is the ratio of alerts generated to incidents declared (in this case, six). This metric might tell you how well tuned your NDR solution is; the higher the metric, the higher the number of false positives you're having to track down and, therefore, the less efficient your time is spent.

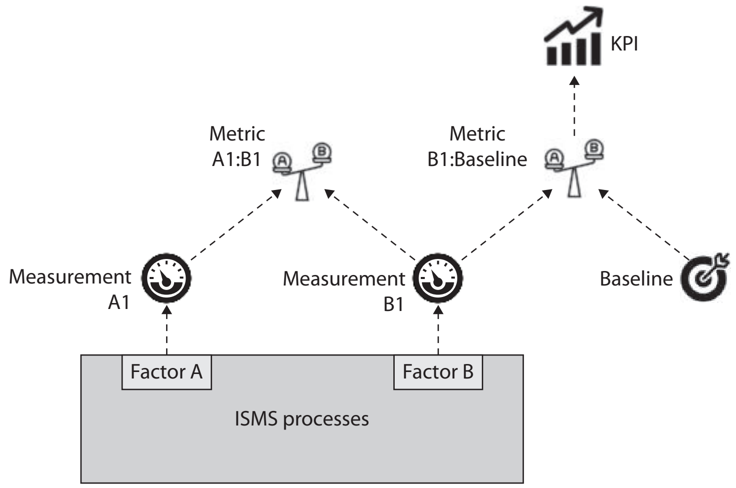


Figure 19-1 Relationship between factors, measurements, baselines, metrics, and KPIs

The second metric is a bit different. It is the comparison of B1 (the measurement of incidents) to some baseline you've established. Assume that management has determined that anything over five incidents per reporting period is a bad thing and might show that either your risk assessment is off (i.e., the threat is higher than you thought) or your ISMS is not doing well (i.e., your controls are not working as you need them to). This metric is particularly important to tracking your strategic business goals and thus becomes a key performance indicator (KPI). KPIs are discussed a bit later in this section.

Security Metrics

Security metrics sometimes get a bad reputation as tedious, burdensome, or even irrelevant. Many organizations pay lip service to them by latching on to the simple things that can easily be measured, such as the number of open tickets or how long it takes analysts to close each ticket. The problem with this approach is that it doesn't really generate valuable insights into how we are doing and how can we do better. In fact, if we measure the wrong things, we can end up doing more harm than good. For example, plenty of organizations rate their analysts based on how many tickets they close in a shift. This approach incentivizes throughput rather than careful analysis and oftentimes leads to missing evidence of ongoing attacks.

The best information security professionals use metrics to tell stories and, like any good storyteller, know their audience. What metrics we need and how we link them together depends on both the story we want to tell and the people to whom we'll be telling it. Board members typically like to hear about strategic threats and opportunities for the organization. Business managers may want to know how protected their business units are. Security operations leaders usually are most interested in how their teams are performing. In order to effectively engage each of these audiences, you need a different type of security metric.

Before we dive into the different types of metrics, it is important to identify what makes a good security metric in the first place. There are six characteristics of a good metric that you should consider:

- **Relevant** Does it align with your goals? Is it correlated with improved outcomes? The metric should be directly tied to your security goals and indirectly tied to your organization's business goals.
- **Quantifiable** Can it be objectively measured in a consistent manner? Could someone game it? A good metric should be relatively easy to measure with the tools at your disposal.
- **Actionable** Can you do something with it? Does it tell you what you must do to make things better? The best metrics inform your immediate actions, justify your requests, and directly lead to improved outcomes.
- **Robust** Will it be relevant in a year? Can you track it over many years? A good metric must allow you to track your situation over time to detect trends. It should capture information whose value endures.
- **Simple** Does it make intuitive sense? Will all stakeholders "get it?" Will they know what was measured and why it matters to them? If you can't explain the metric in a simple sentence, it probably is not a good one.
- **Comparative** Can it be evaluated against something else? Good metrics are the result of comparing measurements to each other or to some baseline or standard. This is why the best metrics are ratios, or percentages, or changes over time.

NOTE Another commonly used approach is to ensure metrics are SMART (specific, measurable, achievable, relevant, and time-bound).

Risk Metrics

Risk metrics capture the organizational risk and how it is changing over time. These are the metrics you're most likely to use if you are communicating with an executive audience, because these metrics are not technical. They are also forward-looking because they address things that may happen in the future. For these reasons, risk metrics are the best ones to support strategic analyses.

Depending on your organization's risk management program, you may already have risk metrics defined. Recall from Chapter 2 that quantitative risk management requires that you identify and capture risk metrics. Even if you use a qualitative approach, you probably have a good place from which to start. All you have to do is identify the variables that make your risks more or less likely to be realized and then start tracking those. Here are some examples of commonly used risk metrics:

- The percentage of change in your aggregated residual risks
- The percentage of change in your current worst-case risk
- The ratio of organizational security incidents to those reported by comparable organizations

Preparedness Metrics

There is another type of metric that indicates how prepared your organization is to deal with security incidents. These metrics are sometimes useful when dealing with executives, but they are more commonly used to monitor the security program as a whole. Preparedness metrics look at all your controls and how well they are being maintained. For example, are your policies and procedures being followed? Do your staff members know what they're supposed to (and not supposed to) do? What about your business partners?

Given the importance (and difficulty) of securing your organization's supply chain, your preparedness metrics should include the preparedness of organizations that are upstream from yours (in other words, your organization's suppliers of goods and services). Depending on your organization's relationship with them, you may have a lot of visibility into their security programs, which makes your assessment and measuring of them a lot easier. Even if they are fairly opaque and unwilling to give you enough information, you should develop a mechanism for rating their security, perhaps using an external or third-party audit, as discussed in Chapter 18.

Here are some examples of preparedness metrics used by many organizations:

- Monthly change in the mean time to patch a system
- Percentage of systems that are fully patched
- Percentage of staff that is up to date on security awareness training
- Ratio of privileged accounts to nonprivileged accounts
- Annual change in vendor security rating (i.e., how prepared is your organization's supply chain)

Performance Metrics

If risk metrics are fairly strategic and preparedness metrics are more operational, performance metrics are as tactical as they come. They measure how good your team and systems are at detecting, blocking, and responding to security incidents. In other words, performance metrics tell you how good you are at defeating your adversaries day in and day out.

If you've ever worked in or led a security operations center, performance metrics are the metrics you're probably most used to seeing. Some examples are listed here:

- Number of alerts analyzed this week/month compared to last week/month
- Number of security incidents declared this week/month compared to last week/month
- Percent change in mean time to detect (MTTD)
- Percent change in mean time to resolve (MTTR)

Key Performance and Risk Indicators

There is no shortage of security metrics in the industry, but not all are created equal. There are some that are important for tracking our processes and day-to-day operations. Other metrics, however, can tell us whether or not we are meeting strategic business goals.

These are the key performance indicators (KPIs) and key risk indicators (KRIs). KPIs measure how well things are going now, while KRIs measure how badly things could go in the future.

Key Performance Indicators

A *key performance indicator* is an indicator that is particularly significant in showing the performance of an ISMS compared to its stated goals. KPIs are carefully chosen from among a larger pool of indicators to show at a high level whether our ISMS is keeping pace with the threats to our organization or showing decreased effectiveness. KPIs should be easily understood by business and technical personnel alike and should be aligned with one or (better yet) multiple organizational goals.

Choosing KPIs really forces us to wrestle with the question "What is it that we're trying to accomplish here?" The process by which we choose KPIs is really driven by organizational goals. In an ideal case, the senior leadership sets (or perhaps approves) goals for the security of the organization. The ISMS team then gets to work on how to show whether we are moving toward or away from those goals. The process can be summarized as follows:

1. Choose the factors that can show the state of our security. In doing this, we want to strike a balance between the number of data sources and the resources required to capture all their data.
2. Define baselines for some or all of the factors under consideration. As we do this, it is helpful to consider which measurements will be compared to each other and which to some baseline. Keep in mind that a given baseline may apply to multiple factors' measurements.
3. Develop a plan for periodically capturing the values of these factors, and fix the sampling period. Ideally, we use automated means of gathering this data so as to ensure the periodicity and consistency of the process.
4. Analyze and interpret the data. While some analysis can (and probably should) be automated, there will be situations that require human involvement. In some cases, we'll be able to take the data at face value, while in others we will have to dig into it and get more information before reaching a conclusion about it.
5. Communicate the indicators to all stakeholders. In the end, we need to package the findings in a way that is understandable by a broad range of stakeholders. A common approach is to start with a nontechnical summary that is supported by increasingly detailed layers of supporting technical information. On the summary side of this continuum is where we select and put our KPIs.

This process is not universally accepted but represents some best security industry practices. At the end of the day, the KPIs are the product of distilling a large amount of information with the goal of answering one specific question: "Are we managing our information security well enough?" There is no such thing as perfect security, so what we are really trying to do is find the sweet spot where the performance of the ISMS is adequate and sustainable using an acceptable amount of resources. Clearly, this spot is a moving target given the ever-changing threat and risk landscape.

Key Risk Indicators

While KPIs tell us where we are today with regard to our goals, *key risk indicators* tell us where we are today in relation to our risk appetite. They measure how risky an activity is so that leadership can make informed decisions about that activity, all the while taking into account potential resource losses. Like KPIs, KRIs are selected for their impact on the decisions of the senior leaders in the organization. This means that KRIs often are not specific to one department or business function, but rather affect multiple aspects of the organization. KRIs have, by definition, a very high business impact.

When considering KRIs, it is useful to relate them to single loss expectancy (SLE) equations. Recall from Chapter 2 that SLE is the organization's potential monetary loss if a specific threat were to be realized. It is the product of the loss and the likelihood that the threat will occur. In other words, if we have a proprietary process for building widgets valued at $500,000 and we estimate a 5 percent chance of an attacker stealing and monetizing that process, then our SLE would be $25,000. Now, clearly, that 5 percent figure is affected by a variety of activities within the organization, such as IDS tuning, IR team proficiency, and end-user security awareness.

Over time, the likelihood of the threat being realized will change based on multiple activities going on within the organization. As this value changes, the risk changes too. A KRI would capture this and allow us to notice when we have crossed a threshold that makes our current activities too risky for our stated risk appetite. This trigger condition enables the organization to change its behavior to compensate for excessive risk. For instance, it could trigger an organizational stand-down for security awareness training.

In the end, the important thing to remember about KRIs is that they are designed to work much as coal mine canaries: they alert us when something bad is likely to happen so that we can change our behavior and defeat the threat.

EXAM TIP KPIs and KRIs are used to measure progress toward attainment of strategic business goals.

Security Process Data

Most of our metrics and indicators come from the security processes that make up our ISMS. There are other sources of measures, of course, but if we want to assess the effectiveness of our security controls, clearly we have to look at them first. To determine whether our controls are up to speed, we need to collect security process data from a variety of places. From how we manage our accounts to how we verify backups to the security awareness of our employees, administrative controls are probably more pervasive and less visible than our technical controls. It shouldn't be surprising that sophisticated threat actors often try to exploit administrative controls.

We covered a number of technical processes in the previous chapter. These included vulnerability assessments, various forms of attack simulations, and log reviews, to name a few. In the sections that follow, we look at some of the more administrative processes from which we can also collect data to help us determine our current posture and help us improve it over time. This is by no means an exhaustive list; it is simply a sampling that (ISC)^2 emphasizes in the CISSP exam objectives.

Account Management

A preferred technique of attackers is to become "normal" privileged users of the systems they compromise as soon as possible. They can accomplish this in at least three ways: compromise an existing privileged account, create a new privileged account, or elevate the privileges of a regular user account. The first approach can be mitigated through the use of strong authentication (e.g., strong passwords or, better yet, multifactor authentication) and by having administrators use privileged accounts only for specific tasks. The second and third approaches can be mitigated by paying close attention to the creation, modification, or misuse of user accounts. These controls all fall in the category of *account management*.

Adding Accounts

When new employees arrive, they should be led through a well-defined process that is aimed at ensuring not only that they understand their duties and responsibilities, but also that they are assigned the required organizational assets and that these are properly configured, protected, and accounted for. While the specifics of how this is accomplished vary from organization to organization, there are some specific administrative controls that should be universal.

First, all new users should be required to read through and acknowledge they understand (typically by signing) all policies that apply to them. At a minimum, every organization should have (and every user should sign) an acceptable use policy (AUP) that specifies what the organization considers acceptable use of the information systems that are made available to the employee. Using a workplace computer to view pornography, send hate e-mail, or hack other computers is almost always specifically forbidden in the AUP. On the other hand, many organizations allow their employees limited personal use, such as checking personal e-mail or surfing the Web during breaks. The AUP is a useful first line of defense, because it documents when each user was made aware of what is and is not acceptable use of computers (and other resources) at work. This makes it more difficult for a user to claim ignorance if they subsequently violate the AUP.

Testing that all employees are aware of the AUP and other applicable policies can be the first step in auditing user accounts. Since every user should have a signed AUP, for instance, all we need is to get a list of all users in the organization and then compare it to the files containing the signed documents. In many cases, all the documents a new employee signs are maintained by human resources (HR) and the computer accounts are maintained by IT. Cross-checking AUPs and user accounts can also verify that these two departments are communicating effectively.

The policies also should dictate the default expiration date of accounts, the password policy, and the information to which a user should have access. This last part becomes difficult because the information needs of individual users typically vary over time.

Modifying Accounts

Suppose a newly hired IT technician is initially assigned the task of managing backups for a set of servers. Over time, you realize this individual is best suited for internal user support, including adding new accounts, resetting passwords, and so forth. The privileges needed in each role are clearly different, so how should you handle this? Many organizations, unfortunately, resort to giving all privileges that a user may need. We have all been in, seen, or heard of organizations where every user is a local admin on his or her computer and every member of the IT department is a domain admin. This is an exceptionally dangerous practice, especially if they all use these elevated credentials by default. This is often referred to as *authorization creep*, which we discussed in Chapter 17.

Adding, removing, or modifying the permissions that a user has should be a carefully controlled and documented process. When are the new permissions effective? Why are they needed? Who authorized the change? Organizations that are mature in their security processes have a change control process in place to address user privileges. While many auditors focus on who has administrative privileges in the organization, there are many custom sets of permissions that approach the level of an admin account. It is important, then, to have and test processes by which elevated privileges are issued.

The Problem with Running as Root

It is undoubtedly easier to do all your work from one user account, especially if that account has all the privileges you could ever need. The catch, as you may well know, is that if your account is compromised, the malicious processes will run with whatever privileges the account has. If you run as root (or admin) all the time, you can be certain that if an attacker compromises your box, he instantly has the privileges to do whatever he needs or wants to do.

A better approach is to do as much of your daily work as you can using a restricted account and elevate to a privileged account only when you must. The way in which you do this varies by operating system:

- Windows operating systems allow you to right-click any program and select Run As to elevate your privileges. From the command prompt, you can use the command `runas /user:<AccountName>` to accomplish the same goal.
- In Linux operating systems, you can simply type `sudo<SomeCommand>` at the command line to run a program as the super (or root) user. Some Linux GUI desktop environments also offer the user the option of running with sudo (usually by checking a box) and prompting for a password.
- In macOS, you use `sudo` from the Terminal app just like you would do from a Linux terminal. However, if you want to run a GUI app with elevated privileges, you need to use `sudo open –a <AppName>` since there is no `gksudo` or `kdesudo` command.