So that is our utopia: log on once and you are good to go. What bursts this bubble? Mainly interoperability issues. For SSO to actually work, every platform, application, and resource needs to accept the same type of credentials, in the same format, and interpret their meanings the same. When Steve logs on to his Windows workstation and gets authenticated by a mixed-mode Windows domain controller, it must authenticate him to the resources he needs to access on the Apple MacBook, the Linux server running NIS, the PrinterLogic print server, and the Windows computer in a trusted domain that has the plotter connected to it. A nice idea, until reality hits.

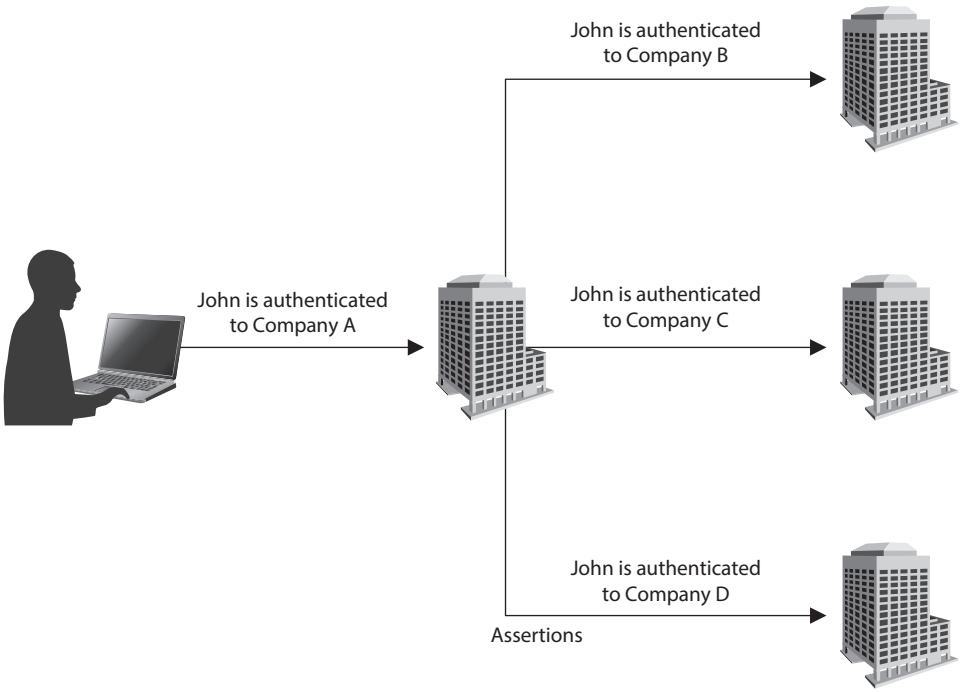
There is also a security issue to consider in an SSO environment. Once an individual is in, he is in. If an attacker is able to uncover one credential set, he has access to every resource within the environment that the compromised account has access to. This is certainly true, but one of the goals is that if a user only has to remember one password, and not ten, then a more robust password policy can be enforced. If the user has just one password to remember, then it can be more complicated and secure because he does not have nine other ones to remember also.

Federated Identity Management

The world continually gets smaller as technology brings people and companies closer together. Many times, when we are interacting with just one website, we are actually interacting with several different companies—we just don't know it. The reason we don't know it is because these companies are sharing our identity and authentication information behind the scenes. This is not done for nefarious purposes necessarily, but to make our lives easier and to allow merchants to sell their goods without much effort on our part.

For example, a person wants to book an airline flight and a hotel room. If the airline company and hotel company use a federated identity management (FIM) system, this means they have set up a trust relationship between the two companies and share customer identification and, potentially, authentication information. So when you book a flight on United Airlines, the website asks if you want to also book a hotel room. If you click Yes, you could then be brought to the Marriott website, which provides information on the closest hotel to the airport you're flying into. Now, to book a room you don't have to log in again. You logged in on the United website, and that website sent your information over to the Marriott website, all of which happened transparently to you.

A *federated identity* is a portable identity, and its associated entitlements, that can be used across business boundaries. It allows a user to be authenticated across multiple IT systems and enterprises. Identity federation is based upon linking a user's otherwise distinct identities at two or more locations without the need to synchronize or consolidate directory information. Federated identity offers businesses and consumers a more convenient way of accessing distributed resources and is a key component of e-commerce.



Web portal functions are parts of a website that act as a point of access to information. A portal presents information from diverse sources in a unified manner. It can offer various services, as in e-mail, news updates, stock prices, data access, price lookups, access to databases, and entertainment. Web portals provide a way for organizations to present one consistent interface with one "look and feel" and various functionality types. For example, you log into your company web portal and it provides access to many different systems and their functionalities, but it seems as though you are only interacting with one system because the interface is "clean" and organized. Portals combine web services (web-based functions) from several different entities and present them in one central website.

A web portal is made up of *portlets*, which are pluggable user-interface software components that present information from other systems. A portlet is an interactive application that provides a specific type of web service functionality (e-mail, news feed, weather updates, forums, etc.). A portal is made up of individual portlets to provide a plethora of services through one interface. It is a way of centrally providing a set of web services. Users can configure their view to the portal by enabling or disabling these various portlet functions.

Since each of these portlets can be provided by different entities, how user authentication information is handled must be tightly controlled, and there must be a

high level of trust between these different entities. A college, for example, might have one web portal available to students, parents, faculty members, and the public. The public should only be able to view and access a small subset of available portlets and not have access to more powerful web services (such as e-mail and database access). Students could be able to log in and gain access to their grades, assignments, and a student forum. Faculty members can gain access to all of these web services, including the school's e-mail service and access to the central database, which contains all of the students' information. If there is a software flaw or misconfiguration, it is possible that someone can gain access to something they are not supposed to.

Federated Identity with a Third-Party Service

It should not be surprising to consider that cloud service providers are also able to provide identification services. Identity as a Service (IDaaS) is a type of Software as a Service (SaaS) offering that is normally configured to provide SSO, FIM, and password management services. Though most IDaaS vendors are focused on cloud- and web-centric systems, it is also possible to leverage their products for FIM on legacy platforms within the enterprise network. Many organizations are transitioning to IDaaS providers for compliance reasons because this approach allows them to centralize access control and monitoring across the enterprise. This, in turn reduces risk and improves auditability, meaning there's a much lower chance of getting hit with a huge General Data Protection Regulation (GDPR) fine because some obscure part of the system didn't have proper access controls.

There are three basic approaches to architecting identity management services: on-premise, cloud-based, and a hybrid of both. The first approach, on-premise, is simple because all the systems and data are located within the enterprise. In the cloud-based model, on the other hand, most or all of the systems or data are hosted by an external party in the cloud. A hybrid FIM system includes both on-premise and cloud-based IdM components, each responsible for its environment but able to coordinate with each other. Regardless of the approach, it is important to ensure that all components play nice with each other. In the following sections we will explore some of the considerations that are common to the successful integration of these services.

Integration Issues

Integration of any set of different technologies or products is typically one of the most complex and risky phases of any deployment. In order to mitigate both the complexities and risks, it is necessary to carefully characterize each product or technology as well as the systems and networks into which they will be incorporated. Regardless of whether you ultimately use an on-premise or cloud-based (or hybrid) approach, you should carefully plan how you will address connectivity, trust, testing, and federation issues. As the old carpentry adage goes, "Measure twice and cut once."

Establishing Connectivity

A critical requirement is to ensure that the components are able to communicate with one another in a secure manner. The big difference between the in-house and outsourced models here is that in the former, the chokepoints are all internal to the organization's

network, while in the latter, they also exist in the public Internet. Clearing a path for this traffic typically means creating new rules for firewalls and IDS/IPS. These rules must be restrictive enough to allow the FIM traffic, but nothing else, to flow between the various nodes. Depending on the systems being used, ports, protocols, and user accounts may also need to be configured to enable bidirectional communication.

Establishing Trust

All traffic between nodes engaged in identity services must be encrypted. (To do otherwise would defeat the whole point of this effort.) From a practical perspective, this almost certainly means that PKI in general and certificate authorities (CAs) in particular will be needed. A potential issue here is that the CAs may not be trusted by default by all the nodes. This is especially true if the enterprise has implemented its own CA internally and is deploying an outsourced service. This is easy to plan ahead of time, but could lead to some big challenges if discovered during the actual rollout. Trust may also be needed between domains.

Incremental Testing

When dealing with complex systems, it is wise to assume that some important issue will not be covered in the plan. This is why it is important to incrementally test the integration of identity services instead of rolling out the entire system at once. Many organizations choose to roll out new services first to test accounts (i.e., not real users), then to one department or division that is used as the test case, and finally to the entire organization. For critical deployments (and one would assume that identity services would fall in this category), it is best to test as thoroughly as possible in a testbed or sandbox environment. Only then should the integration progress to real systems.

Legacy Systems

Unless your entire infrastructure is in the cloud, odds are that you have at least a handful of legacy systems that don't play nice with the FIM service or provider. To mitigate this risk, you should first ensure that you have an accurate asset inventory that clearly identifies any systems (or system dependencies) that will not integrate well. Then, you should get together with all stakeholders (e.g., business, IT, security, partners) to figure out which of these systems can be retired, replaced, or upgraded. The change management process we'll discuss in Chapter 20 is a great way to handle this. Finally, for any legacy systems that must remain as they are (and hence, not integrated into FIM), you want to minimize their authorized users and put additional controls in place to ensure they are monitored in an equivalent manner as the systems that fall under IdM.

On-Premise

An *on-premise* (or *on-premises*) FIM system is one in which all needed resources remain under your physical control. This usually means that you purchase or lease the necessary hardware, software, and licenses and then use your own team to build, integrate, and maintain the system. This kind of deployment, though rare, makes sense in cases where different organizations' networks are interconnected but not directly connected to the Internet, such as those of some critical infrastructure and military organizations. Though most

on-premise FIM solution providers offer installation, configuration, and support services, day-to-day operation and management of the system falls on your team. This requires them to have not only the needed expertise but also the time to devote to managing the system's life cycle.

Cloud

Arguably, the most cost-effective and secure way to implement FIM across an enterprise is to use a cloud-only solution. The economies of scale that IDaaS providers enjoy translate into cost savings for their customers. Even if you have the talent in your workforce to implement IdM on-premises, it would almost certainly be cheaper to outsource it to one of the many established vendors in this space. The visibility that an IDaaS provider has not only across your organization but also across the entire space of its customers allows it to detect and respond to threats faster and better than might otherwise be possible. This should be a dream come true, if only your entire infrastructure were cloud-based.

Hybrid

Most likely, your organization has a combination of cloud-based and on-premise systems. Some of the latter ones probably don't lend themselves to a cloud-based FIM solution, at least not without incurring exorbitant upgrade or integration costs. So, what should you do? You can implement a hybrid approach in which you have on-premise and cloud-based FIM platforms that are integrated with each other. One would be the primary and the other would be the secondary. As long as they are interoperable and properly configured, you get to have the best of both worlds. Most major IDaaS providers have solutions that support hybrid deployments.

Chapter Review

Identification, authentication, and authorization of users and systems are absolutely essential to cybersecurity. After all, how can we differentiate good and bad actors unless we know (at least) who the good ones are? This is why we spent so much time going over knowledge-based, biometric, and ownership-based authentication techniques and technologies. These, together with credential management products and practices, allow us to ensure we know who it is that our systems are interacting with.

The purpose of this chapter was to expose you to the multiple processes and technologies that make identity management possible, both at an individual level and at aggregate enterprise scales. This all sets the stage for the next chapter, in which we will delve into how to operationalize these concepts and build on them to ensure authorized parties (and no others) have access to the right assets (and no others).

Quick Review

- Identification describes a method by which a subject (user, program, or process) claims to have a specific identity (e.g., username, account number, or e-mail address).
- Authentication is the process by which a system verifies the identity of the subject, usually by requiring a piece of information that only the claimed identity should have.
- Credentials consist of an identification claim (e.g., username) and authentication information (e.g., password).
- Authorization is the determination of whether a subject has been given the necessary rights and privileges to carry out the requested actions.
- The three main types of factors used for authentication are something a person knows (e.g., password), something a person has (e.g., token), and something a person is (e.g., fingerprint), which can be combined with two additional factors: somewhere a person is (e.g., geolocation) and something a person does (e.g., keystroke behavior).
- Knowledge-based authentication uses information a person knows, such as a password, passphrase, or life experience.
- Salts are random values added to plaintext passwords prior to hashing to add more complexity and randomness.
- Cognitive passwords are fact- or opinion-based questions, typically based on life experiences, used to verify an individual's identity.
- A Type I biometric authentication error occurs when a legitimate individual is denied access; a Type II error occurs when an impostor is granted access.
- The crossover error rate (CER) of a biometric authentication system represents the point at which the false rejection rate (Type I errors) is equal to the false acceptance rate (Type II errors).
- Ownership-based authentication is based on something a person owns, such as a token device.
- A token device, or password generator, is usually a handheld device that has a display (and possibly a keypad), is synchronized in some manner with the authentication server, and displays to the user a one-time password (OTP).
- A synchronous token device requires the device and the authentication service to advance to the next OTP in sync with each other; an asynchronous token device employs a challenge/response scheme to authenticate the user.
- A memory card holds information but cannot process information; a smart card holds information and has the necessary hardware and software to actually process that information.

- Password managers or password vaults are a popular solution to remembering a myriad of complex passwords.
- Just-in-time (JIT) access is a provisioning methodology that elevates users to the necessary privileged access to perform a specific task.
- User provisioning refers to the creation, maintenance, and deactivation of user objects and attributes as they exist in one or more systems, directories, or applications, in response to business processes.
- An authoritative system of record (ASOR) is a hierarchical tree-like structure system that tracks subjects and their authorization chains.
- User provisioning refers to the creation, maintenance, and deactivation of user objects and attributes as they exist in one or more systems, directories, or applications, in response to business processes.
- A session is an agreement between two parties to communicate interactively.
- Auditing capabilities ensure users are accountable for their actions, verify that the security policies are enforced, and can be used as investigation tools.
- Deleting specific incriminating data within audit logs is called scrubbing.
- Identity management (IdM) is a broad term that encompasses the use of different products to identify, authenticate, and authorize users through automated means.
- Directory services map resource names to their corresponding network addresses, allowing discovery of and communication with devices, files, users, or any other asset.
- The most commonly implemented directory services, such as Microsoft Windows Active Directory (AD), implement the Lightweight Directory Access Protocol (LDAP).
- Single sign-on (SSO) systems allow users to authenticate once and be able to access all authorized resources, which reduces the amount of time users spend authenticating and enables administrators to streamline user accounts and better control access rights.
- A federated identity is a portable identity, and its associated entitlements, that allows a user to be authenticated across multiple IT systems and enterprises.
- Identity as a Service (IDaaS) is a type of Software as a Service (SaaS) offering that is normally configured to provide SSO, FIM, and password management services.
- There are three basic approaches to architecting identity management services: on-premise, cloud-based, and a hybrid of both.

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. Which of the following statements correctly describes biometric methods of authentication?
   - **A.** They are the least expensive and provide the most protection.
   - **B.** They are the most expensive and provide the least protection.
   - **C.** They are the least expensive and provide the least protection.
   - **D.** They are the most expensive and provide the most protection.
2. Which of the following statements correctly describes the use of passwords for authentication?
   - **A.** They are the least expensive and most secure.
   - **B.** They are the most expensive and least secure.
   - **C.** They are the least expensive and least secure.
   - **D.** They are the most expensive and most secure.
3. How is a challenge/response protocol utilized with token device implementations?
   - **A.** This type of protocol is not used; cryptography is used.
   - **B.** An authentication service generates a challenge, and the smart token generates a response based on the challenge.
   - **C.** The token challenges the user for a username and password.
   - **D.** The token challenges the user's password against a database of stored credentials.
4. The process of mutual authentication involves _____.
   - **A.** a user authenticating to a system and the system authenticating to the user
   - **B.** a user authenticating to two systems at the same time
   - **C.** a user authenticating to a server and then to a process
   - **D.** a user authenticating, receiving a ticket, and then authenticating to a service
5. What role does biometrics play in access control?
   - **A.** Authorization
   - **B.** Authenticity
   - **C.** Authentication
   - **D.** Accountability

6. Which of the following is the best description of directories that are used in identity management technology?

A. Most are hierarchical and follow the X.500 standard.

B. Most have a flat architecture and follow the X.400 standard.

C. Most have moved away from LDAP.

D. Most use RADIUS.

7. Which of the following is not part of user provisioning?

A. Creation and deactivation of user accounts

B. Business process implementation

C. Maintenance and deactivation of user objects and attributes

D. Delegating user administration

8. What is a technology that allows a user to remember just one password?

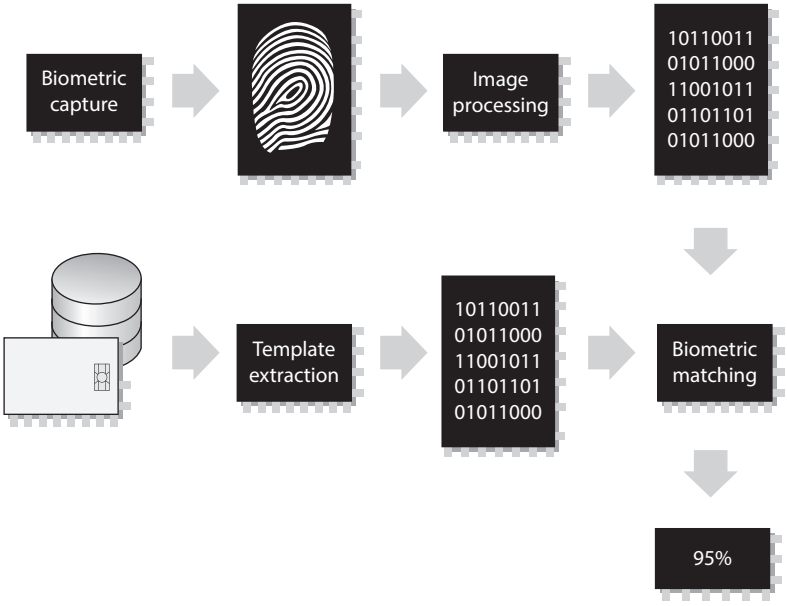
A. Password generation

B. Password dictionaries

C. Password rainbow tables

D. Password synchronization

9. This graphic covers which of the following?



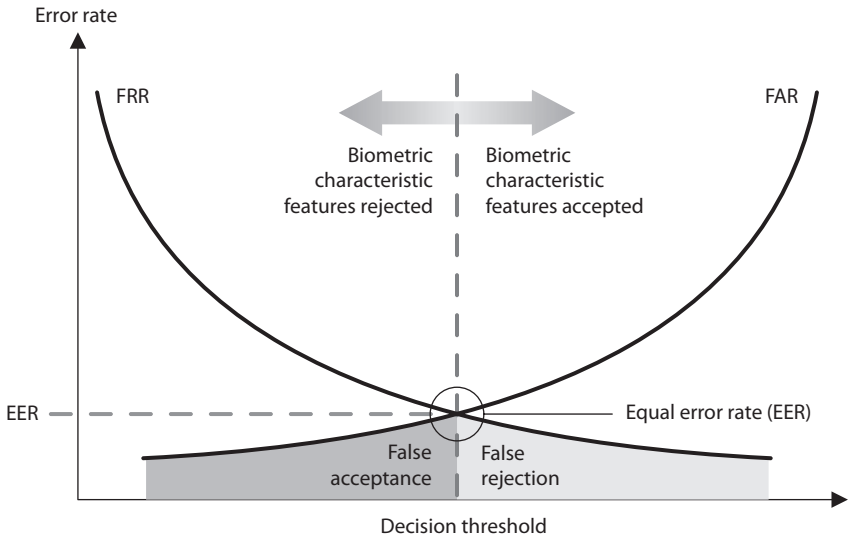
A. Crossover error rate

B. Identity verification

C. Authorization rates

D. Authentication error rates

10. The diagram shown here explains which of the following concepts?



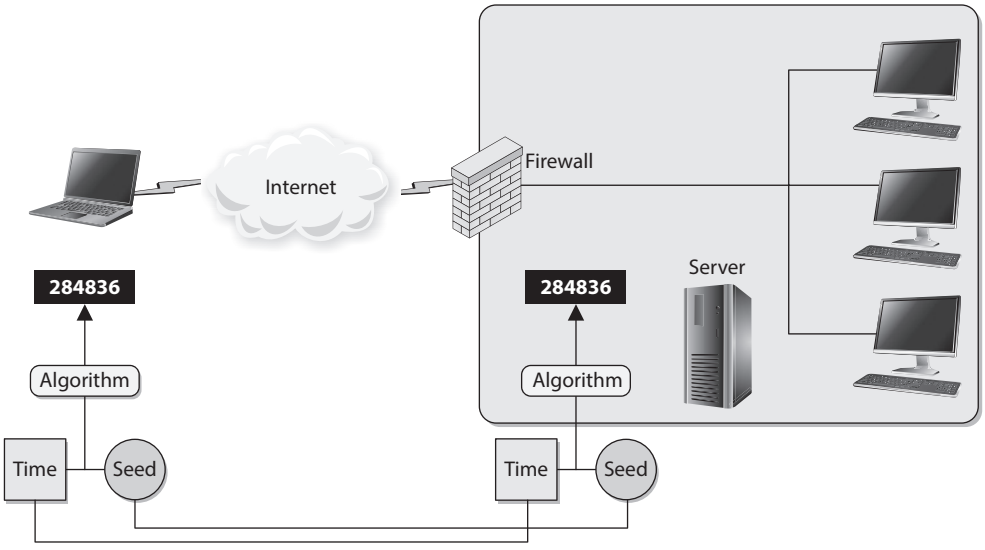
A. Crossover error rate.

B. Type III errors.

C. FAR equals FRR in systems that have a high crossover error rate.

D. Biometrics is a high acceptance technology.

11. The graphic shown here illustrates how which of the following works?



A. Rainbow tables

B. Dictionary attack

C. One-time password

D. Strong authentication

Answers

1. **D.** Compared with the other available authentication mechanisms, biometric methods provide the highest level of protection and are the most expensive.
2. **C.** Passwords provide the least amount of protection, but are the cheapest because they do not require extra readers (as with smart cards and memory cards), do not require devices (as do biometrics), and do not require a lot of overhead in processing (as in cryptography). Passwords are the most common type of authentication method used today.
3. **B.** An asynchronous token device is based on challenge/response mechanisms. The authentication service sends the user a challenge value, which the user enters into the token. The token encrypts or hashes this value, and the user uses this as her one-time password.
4. **A.** Mutual authentication means it is happening in both directions. Instead of just the user having to authenticate to the server, the server also must authenticate to the user.
5. **C.** Biometrics is a technology that validates an individual's identity by reading a physical attribute. In some cases, biometrics can be used for identification, but that was not listed as an answer choice.
6. **A.** Most organizations have some type of directory service that contains information pertaining to the organization's network resources and users. Most directories follow a hierarchical database format, based on the X.500 standard, and a type of protocol, as in Lightweight Directory Access Protocol (LDAP), that allows subjects and applications to interact with the directory. Applications can request information about a particular user by making an LDAP request to the directory, and users can request information about a specific resource by using a similar request.
7. **B.** User provisioning refers to the creation, maintenance, and deactivation of user objects and attributes as they exist in one or more systems, directories, or applications, in response to business processes. User provisioning software may include one or more of the following components: change propagation, self-service workflow, consolidated user administration, delegated user administration, and federated change control. User objects may represent employees, contractors, vendors, partners, customers, or other recipients of a service. Services may include e-mail, access to a database, access to a file server or mainframe, and so on.

8. **D.** Password synchronization technologies can allow a user to maintain just one password across multiple systems. The product synchronizes the password to other systems and applications, which happens transparently to the user.
9. **B.** These steps are taken to convert the biometric input for identity verification:
   - **i.** A software application identifies specific points of data as match points.
   - **ii.** An algorithm is used to process the match points and translate that information into a numeric value.
   - **iii.** Authentication is approved or denied when the database value is compared with the end user input entered into the scanner.
10. **A.** This rating is stated as a percentage and represents the point at which the false rejection rate equals the false acceptance rate. This rating is the most important measurement when determining a biometric system's accuracy.
    - **Type I error, false rejection rate (FRR)** Rejects authorized individual
    - **Type II error, false acceptance rate (FAR)** Accepts impostor
11. **C.** Different types of one-time passwords are used for authentication. This graphic illustrates a synchronous token device, which synchronizes with the authentication service by using time or a counter as the core piece of the authentication process.

This page intentionally left blank

CHAPTER 17

Managing Identities and Access

This chapter presents the following:

- Authorization mechanisms
- Implementing authentication systems
- Managing the identity and access provisioning life cycle
- Controlling physical and logical access

Locks keep out only the honest.

—Proverb

Identification and authentication of users and systems, which was the focus of the previous chapter, is only half of the access control battle. You may be able to establish that you are truly dealing with Ahmed, but what assets should he be allowed to access? It really depends on the sensitivity of the asset, Ahmed's role, and any applicable rules on how these assets are supposed be used. Access control can also depend on any number of other attributes of the user, the asset, and the relationship between the two. Finally, access control can be based on risk.

Once you decide what access control model is best for your organization, you still have to implement the right authentication and authorization mechanism. There are many choices, but in this chapter we'll focus on the technologies that you are likeliest to encounter in the real world (and on the CISSP exam). We'll talk about how to manage the user access life cycle, which is where a lot of organizations get in trouble by not changing authorizations as situations change. After we cover all these essentials, we'll see how it all fits together in the context of controlling access to physical and logical assets. Let's start by looking at authorization mechanisms.

Authorization Mechanisms

Authorization is the process of ensuring authenticated users have access to the resources they are authorized to use and don't have access to any other resources. This is preceded by authentication, of course, but unlike that process, which tends to be a one-time activity,

authorization controls every interaction of every user with every resource. It is an ongoing, all-seeing, access control mechanism.

An *access control mechanism* dictates how subjects access objects. It uses access control technologies and security mechanisms to enforce the rules and objectives of an *access control model*. As discussed in this section, there are six main types of access control models: discretionary, mandatory, role-based, rule-based, attribute-based, and risk-based. Each model type uses different methods to control how subjects access objects, and each has its own merits and limitations. The business and security goals of an organization, along with its culture and habits of conducting business, help prescribe what access control model it should use. Some organizations use one model exclusively, whereas others combine models to provide the necessary level of protection.

Regardless of which model or combination of models your organization uses, your security team needs a mechanism that consistently enforces the model and its rules. The *reference monitor* is an abstract machine that mediates all access subjects have to objects, both to ensure that the subjects have the necessary access rights and to protect the objects from unauthorized access and destructive modification. It is an access control concept, not an actual physical component, which is why it is normally referred to as the "reference monitor concept" or an "abstract machine." However the reference monitor is implemented, it must possess the following three properties to be effective:

- **Always invoked** To access an object, you have to go through the monitor first.
- **Tamper-resistant** It must ensure a threat actor cannot disable or modify it.
- **Verifiable** It must be capable of being thoroughly analyzed and tested to ensure that it works correctly all the time.

Let's explore the different approaches to implement and manage authorization mechanisms. The following sections explain the six different models and where they should be implemented.

Discretionary Access Control

If a user creates a file, he is the owner of that file. An identifier for this user is placed in the file header and/or in an access control matrix within the operating system. Ownership might also be granted to a specific individual. For example, a manager for a certain department might be made the owner of the files and resources within her department. A system that uses *discretionary access control (DAC)* enables the owner of the resource to specify which subjects can access specific resources. This model is called discretionary because the control of access is based on the discretion of the owner. Many times department managers or business unit managers are the owners of the data within their specific department. Being the owner, they can specify who should have access and who should not.

In a DAC model, access is restricted based on the authorization granted to the users. This means users are allowed to specify what type of access can occur to the objects they own. If an organization is using a DAC model, the network administrator can allow resource owners to control who has access to their files. The most common

Identity-Based Access Control

DAC systems grant or deny access based on the identity of the subject. The identity can be a user identity or a group membership. So, for example, a data owner can choose to allow Bob (user identity) and the Accounting group (group membership identity) to access his file. If Bob as a user is only granted Read access but he happens to be a member of the Accounting group, which has Change access, Bob would get the greater of the two: Change. The exception to this "greater access" rule is when No Access is set. In that case, it doesn't matter what other access levels a user may have gotten as an individual or through group membership, since that rule trumps all others.

implementation of DAC is through access control lists (ACLs), which are dictated and set by the owners and enforced by the operating system.

Most of the operating systems you may be used to dealing with (e.g., Windows, Linux, and macOS systems and most flavors of Unix) are based on DAC models. When you look at the properties of a file or directory and see the choices that allow you to control which users can have access to this resource and to what degree, you are witnessing an instance of ACLs enforcing a DAC model.

DAC can be applied to both the directory tree structure and the files it contains. The Microsoft Windows world has access permissions of No Access, Read (r), Write (w), Execute (x), Delete (d), Change (c), and Full Control. The Read attribute lets you read the file but not make changes. The Change attribute allows you to read, write, execute, and delete the file but does not let you change the ACLs or the owner of the files. Obviously, the attribute of Full Control lets you make any changes to the file and its permissions and ownership.

Access Control Lists

Access control lists (ACLs) are lists of subjects that are authorized to access a specific object, and they define what level of authorization is granted. Authorization can be specific to an individual, group, or role. ACLs are used in several operating systems, applications, and router configurations.

ACLs map values from the access control matrix to the object. Whereas a capability corresponds to a row in the access control matrix, the ACL corresponds to a column of the matrix. The ACL for a notional File1 object is shown in Table 17-1.

Table 17-1 The ACL for a Notional File1 Object

User	File1
Diane	Full control
Katie	Read and execute
Chrissy	Read, write, and execute
John	Read and execute

Challenges When Using DAC

While DAC systems provide a lot of flexibility to the user and less administration for IT, it is also the Achilles' heel of operating systems. Malware can install itself and work under the security context of the user. For example, if a user opens an attachment that is infected with a virus, the code can install itself in the background without the user's being aware of this activity. This code basically inherits all the rights and permissions that the user has and can carry out all the activities the user can on the system. It can send copies of itself out to all the contacts listed in the user's e-mail client, install a back door, attack other systems, delete files on the hard drive, and more. The user is actually giving rights to the virus to carry out its dirty deeds, because the user has discretionary rights and is considered the owner of many objects on the system. This is particularly problematic in environments where users are assigned local administrator or root accounts, because once malware is installed, it can do anything on a system.

While we may want to give users some freedom to indicate who can access the files that they create and other resources on their systems that they are configured to be "owners" of, we really don't want them dictating all access decisions in environments with assets that need to be protected. We just don't trust them that much, and we shouldn't if you think back to the zero-trust principle. In most environments, user profiles are created and loaded on user workstations that indicate the level of control the user does and does not have. As a security administrator you might configure user profiles so that users cannot change the system's time, alter system configuration files, access a command prompt, or install unapproved applications. This type of access control is referred to as *nondiscretionary*, meaning that access decisions are not made at the discretion of the user. Nondiscretionary access controls are put into place by an authoritative entity (usually a security administrator) with the goal of protecting the organization's most critical assets.

Mandatory Access Control

In a *mandatory access control (MAC)* model, users do not have the discretion of determining who can access objects as in a DAC model. For security purposes, an operating system that is based on a MAC model greatly reduces the amount of rights, permissions, and functionality that a user has. In most systems based on the MAC model, a user cannot install software, change file permissions, add new users, and so on. The system can be used by the user for very focused and specific purposes, and that is it. These systems are usually very specialized and are in place to protect highly classified data. Most people have never interacted directly with a MAC-based system because they are mainly used by government-oriented agencies that maintain top-secret information.

However, MAC is used behind the scenes in some environments you may have encountered at some point. For example, the optional Linux kernel security module called AppArmor allows system administrators to implement MAC for certain kernel resources. There is also a version of Linux called SELinux, developed by the NSA, that implements a flexible MAC model for enhanced security.

The MAC model is based on a security label system. Users are given a security clearance (secret, top secret, confidential, and so on), and data is classified in the same way. The clearance and classification data is stored in the security labels, which are

bound to the specific subjects and objects. When the system makes a decision about fulfilling a request to access an object, it is based on the clearance of the subject, the classification of the object, and the security policy of the system. This means that even if a user has the right clearance to read a file, specific policies (e.g., requiring "need to know") could still prevent access to it. The rules for how subjects access objects are made by the organization's security policy, configured by the security administrator, enforced by the operating system, and supported by security technologies.

NOTE Traditional MAC systems are based upon multilevel security policies, which outline how data at different classification levels is to be protected. Multilevel security (MLS) systems allow data at different classification levels to be accessed and interacted with by users with different clearance levels simultaneously.

When the MAC model is being used, every subject and object must have a security label, also called a sensitivity label. This label contains the object's security classification and any categories that may apply to it. The classification indicates the sensitivity level, and the categories enforce need-to-know rules. Figure 17-1 illustrates the use of security labels.

The classifications follow a hierarchical structure, with one level being more trusted than another. However, the categories do not follow a hierarchical scheme, because they represent compartments of information within a system. The categories can correspond to departments (intelligence, operations, procurement), project codenames (Titan, Jack Voltaic, Threatcasting), or management levels, among others. In a military environment, the classifications could be top secret, secret, confidential, and unclassified. Each classification is more trusted than the one below it. A commercial organization might use confidential, proprietary, corporate, and sensitive. The definition of the classification is up to the organization and should make sense for the environment in which it is used.

The categories portion of the label enforces need-to-know rules. Just because someone has a top secret clearance does not mean she now has access to all top secret information.

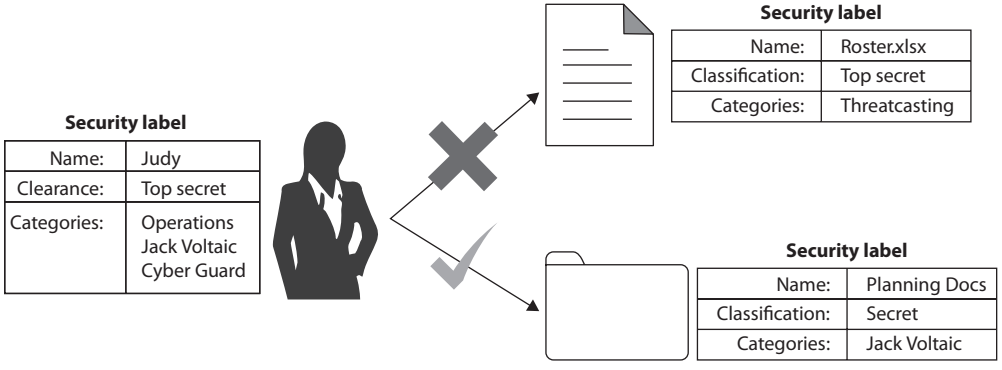


Figure 17-1 A security label is made up of a classification and categories.

She must also have a need to know. As shown in Figure 17-1, Judy is cleared top secret and has the codename Jack Voltaic as one of her categories. She can, therefore, access the folder with the planning documents for Jack Voltaic because her clearance is at least that of the object, and all the categories listed in the object match her own. Conversely, she cannot access the roster spreadsheet because, although her clearance is sufficient, she does not have a need to know that information. We know this last bit because whoever assigned the categories to Judy did not include Threatcasting among them.

EXAM TIP In MAC implementations, the system makes access decisions by comparing the subject's clearance and need-to-know level to the object's security label. In DAC implementations, the system compares the subject's identity to the ACL on the resource.

Software and hardware guards allow the exchange of data between trusted (high assurance) and less trusted (low assurance) systems and environments. For instance, if you were working on a MAC system (working in the dedicated security mode of secret) and you needed it to communicate to a MAC database (working in multilevel security mode, which goes up to top secret), the two systems would provide different levels of protection. If a system with lower assurance can directly communicate with a system of high assurance, then security vulnerabilities and compromises could be introduced.

A *software guard* is really just a front-end product that allows interconnectivity between systems working at different security levels. Different types of guards can be used to carry out filtering, processing requests, data blocking, and data sanitization. A *hardware guard* is a system with two network interface cards (NICs) connecting the two systems that need to communicate with one another. Guards can be used to connect different MAC systems working in different security modes, and they can be used to connect different networks working at different security levels. In many cases, the less trusted system can send messages to the more trusted system and can only receive acknowledgments back. This is common when e-mail messages need to go from less trusted systems to more trusted classified systems.

TIP The terms "security labels" and "sensitivity labels" can be used interchangeably.

Because MAC systems enforce strict access control, they also provide a wide range of security, particularly dealing with malware. Malware is the bane of DAC systems. Viruses, worms, and rootkits can be installed and run as applications on DAC systems. Since users that work within a MAC system cannot install software, the operating system does not allow any type of software, including malware, to be installed while the user is logged in. But while MAC systems might seem to be an answer to all our security prayers, they have very limited user functionality, require a lot of administrative overhead, are very expensive, and are not user friendly. DAC systems are general-purpose computers, while MAC systems serve a very specific purpose.

EXAM TIP Unlike DAC systems, MAC systems are considered nondiscretionary because users cannot make access decisions based on their own discretion (choice).

Role-Based Access Control

A *role-based access control (RBAC)* model uses a centrally administrated set of controls to determine how subjects and objects interact. The access control levels are based on the necessary operations and tasks a user needs to carry out to fulfill her responsibilities within an organization. This type of model lets access to resources be based on the role the user holds within the organization. The more traditional access control administration is based on just the DAC model, where access control is specified at the object level with ACLs. This approach is more complex because the administrator must translate an organizational authorization policy into permission when configuring ACLs. As the number of objects and users grows within an environment, users are bound to be granted unnecessary access to some objects, thus violating the least-privilege rule and increasing the risk to the organization. The RBAC approach simplifies access control administration by allowing permissions to be managed in terms of user job roles.

In an RBAC model, a role is defined in terms of the operations and tasks the role will carry out, whereas a DAC model outlines which subjects can access what objects based upon the individual user identity. Let's say we need a research and development analyst role. We develop this role not only to allow an individual to have access to all product and testing data but also, and more importantly, to outline the tasks and operations that the role can carry out on this data. When the analyst role makes a request to access the new testing results on the file server, in the background the operating system reviews the role's access levels before allowing this operation to take place.

NOTE Introducing roles also introduces the difference between rights being assigned explicitly and implicitly. If rights and permissions are assigned explicitly, they are assigned directly to a specific individual. If they are assigned implicitly, they are assigned to a role or group and the user inherits those attributes.

An RBAC model is the best system for an organization that has high employee turnover. If John, who is mapped to the Contractor role, leaves the organization, then Chrissy, his replacement, can be easily mapped to this role. That way, the administrator does not need to continually change the ACLs on the individual objects. He only needs to create a role (Contractor), assign permissions to this role, and map the new user to this role. Optionally, he can define roles that inherit access from other roles higher up in a hierarchy. These features are covered by two components of RBAC: core and hierarchical.

Core RBAC

There is a core component that is integrated into every RBAC implementation because it is the foundation of the model. Users, roles, permissions, operations, and sessions are defined and mapped according to the security policy. The core RBAC

- Has a many-to-many relationship among individual users and privileges
- Uses a session as a mapping between a user and a subset of assigned roles
- Accommodates traditional but robust group-based access control

Many users can belong to many groups with various privileges outlined for each group. When the user logs in (this is a session), the various roles and groups this user has been assigned are available to the user at one time. If you are a member of the Accounting role, RD group, and Administrative role, when you log on, all of the permissions assigned to these various groups are available to you.

This model provides robust options because it can include other components when making access decisions, instead of just basing the decision on a credential set. The RBAC system can be configured to also include time of day, location of role, day of the week, and so on. This means other information, not just the user ID and credential, is used for access decisions.

Hierarchical RBAC

This component allows the administrator to set up an organizational RBAC model that maps to the organizational structures and functional delineations required in a specific environment. This is very useful since organizations are already set up in a personnel hierarchical structure. In most cases, the higher you are in the chain of command, the more access you most likely have. Hierarchical RBAC has the following features:

- Uses role relations in defining user membership and privilege inheritance. For example, the Nurse role can access a certain set of files, and the Lab Technician role can access another set of files. The Doctor role inherits the permissions and access rights of these two roles and has more elevated rights already assigned to the Doctor role. So hierarchical RBAC is an accumulation of rights and permissions of other roles.
- Reflects organizational structures and functional delineations.
- Supports two types of hierarchies:
  - **Limited hierarchies** Only one level of hierarchy is allowed (Role 1 inherits from Role 2 and no other role)
  - **General hierarchies** Allows for many levels of hierarchies (Role 1 inherits Role 2's and Role 3's permissions)

Hierarchies are a natural means of structuring roles to reflect an organization's lines of authority and responsibility. Role hierarchies define an inheritance relation among roles. Different separations of duties are provided through RBAC:

- **Static separation of duty (SSD) relations** Deters fraud by constraining the combination of privileges (e.g., the user cannot be a member of both the Cashier and Accounts Receivable roles).
- **Dynamic separation of duty (DSD) relations** Deters fraud by constraining the combination of privileges that can be activated in any session (e.g., the user cannot be in both the Cashier and Cashier Supervisor roles at the same time, but the user can be a member of both). This one warrants a bit more explanation. Suppose José is a member of both the Cashier and Cashier Supervisor roles. If he logs in as a Cashier, the Supervisor role is unavailable to him during that session. If he logs in as Cashier Supervisor, the Cashier role is unavailable to him during that session.
- Role-based access control can be managed in the following ways:
  - **Non-RBAC** Users are mapped directly to applications and no roles are used.
  - **Limited RBAC** Users are mapped to multiple roles and mapped directly to other types of applications that do not have role-based access functionality.
  - **Hybrid RBAC** Users are mapped to multiapplication roles with only selected rights assigned to those roles.
  - **Full RBAC** Users are mapped to enterprise roles.

RBAC, MAC, and DAC

A lot of confusion exists regarding whether RBAC is a type of DAC model or a type of MAC model. Different sources claim different things, but in fact RBAC is a model in its own right. In the 1960s and 1970s, the U.S. military and NSA did a lot of research on the MAC model. DAC, which also sprang to life in the 1960s and 1970s, has its roots in the academic and commercial research laboratories. The RBAC model, which started gaining popularity in the 1990s, can be used in combination with MAC and DAC systems. For the most up-to-date information on the RBAC model, go to https://csrc.nist.gov/projects/role-based-access-control, which has documents that describe an RBAC standard and independent model, with the goal of clearing up this continual confusion.

In reality, operating systems can be created to use one, two, or all three of these models in some form, but just because they can be used together does not mean that they are not their own individual models with their own strict access control rules.

Rule-Based Access Control

Rule-based access control uses specific rules that indicate what can and cannot happen between a subject and an object. This access control model is built on top of traditional RBAC and is thus commonly called RB-RBAC to disambiguate the otherwise overloaded RBAC acronym. It is based on the simple concept of "if this, then that" (IFTTT) programming rules, which can be used to provide finer-grained access control to resources. Before a subject can access an object in a certain circumstance, the subject must meet a set of predefined rules. This can be simple and straightforward, as in, "If the subject's ID matches the unique ID value in the provided digital certificate, then the subject can gain access." Or there could be a set of complex rules that must be met before a subject can access an object. For example, "If the subject is accessing the object on a weekday between 8 A.M. and 5 P.M., and if the subject is accessing the object while physically in the office, and if the subject is in the procurement role, then the subject can access the object."

Rule-based access allows a developer to define specific and detailed situations in which a subject can or cannot access an object and what that subject can do once access is granted. Traditionally, rule-based access control has been used in MAC systems as an enforcement mechanism of the complex rules of access that MAC systems provide. Today, rule-based access is used in other types of systems and applications as well. Many routers and firewalls use rules to determine which types of packets are allowed into a network and which are rejected. Rule-based access control is a type of compulsory control, because the administrator sets the rules and the users cannot modify these controls.

Attribute-Based Access Control

Attribute-based access control (ABAC) uses attributes of any part of a system to define allowable access. These attributes can belong to subjects, objects, actions, or contexts. Here are some possible attributes we could use to describe our ABAC policies:

- **Subjects** Clearance, position title, department, years with the organization, training certification on a specific platform, member of a project team, location
- **Objects** Classification, files pertaining to a particular project, human resources (HR) records, location, security system component
- **Actions** Review, approve, comment, archive, configure, restart
- **Context** Time of day, project status (open/closed), fiscal year, ongoing audit

As you can see, ABAC provides the most granularity of any of the access control models. It would be possible, for example, to define and enforce a policy that allows only directors to comment on (but not edit) files pertaining to a project that is currently being audited. This specificity is a two-edged sword, since it can lead to an excessive number of policies that could interact with each other in ways that are difficult to predict.

Risk-Based Access Control

The access control models we've discussed so far all require that we decide exactly what is and is not allowed ahead of time. Whether these decisions involve users, security labels, roles, rules, or attributes, we codify them in our systems and, barring the occasional update, the policies are pretty static. But what if we were to make access control decisions dynamically based on the conditions surrounding the subjects' requests?

Risk-based access control (in case the term RBAC wasn't already ambiguous) estimates the risk associated with a particular request in real time and, if it doesn't exceed a given threshold, grants the subject access to the requested resource. It is an attempt to more closely align risk management and access control while striving to share objects as freely as possible. For example, suppose David works for a technology manufacturer that is about to release a super-secret new product that will revolutionize the world. If the details of this product are leaked before the announcement, it will negatively impact revenues and the return on investment of the marketing campaigns. Obviously, the product's specification sheet will be very sensitive until the announcement. Should David be granted access it?

Risk-based access control would look at it from the perspective of risk, which is the likelihood of an event multiplied by its impact. Suppose that the event about which we are concerned is a leak of the product details ahead of the official announcement. The impact is straightforward, so the real question is how likely it is that David's request will lead to a leak. That depends on several factors, such as his role (is he involved in the rollout?), trustworthiness (is he suspected of leaking anything before?), context (what is he doing that requires access to the specification sheet?), and possibly many others. The system would gather the necessary information, estimate the risk, compare it to the maximum tolerable threshold, and then make a decision.

Figure 17-2 illustrates the main components of risk-based access control. The risk factors are generally divided into categories like user context, resource sensitivity, action severity, and risk history. We've already touched on the first three of these in our example, but we also may want to learn from previous decisions. What is the risk history of similar requests? If the organization doesn't have a culture of secrecy and has experienced leaks

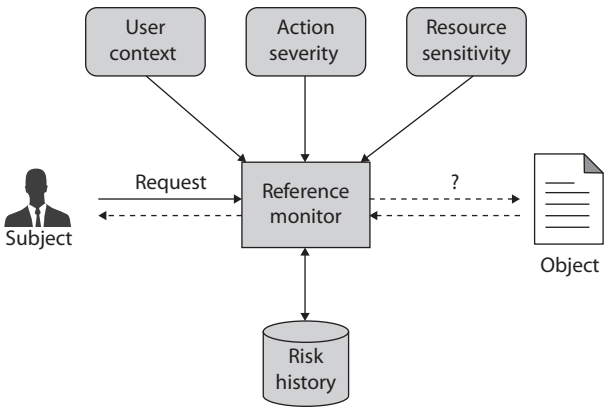


Figure 17-2 Components of a risk-based access control model

Access Control Models
The main characteristics of the six different access control models are important to understand.

- **DAC** Data owners decide who has access to resources, and ACLs are used to enforce these access decisions.
- **MAC** Operating systems enforce the system's security policy through the use of security labels.
- **RBAC** Access decisions are based on each subject's role and/or functional position.
- **RB-RBAC** Adds on to RBAC by imposing rules that further restrict access decisions.
- **ABAC** Access decisions are based on attributes of any component of or action on the system.
- **Risk BAC** Estimates the risk associated with a particular request in real time.

in the past, that drives risk way up. If a particular subject has a history of making bad decisions, that likewise points toward denying access. Regardless of how the decision is arrived at, there is an element of monitoring the user activities to add to this history so we can improve the accuracy of our risk estimates over time.

Implementing Authentication and Authorization Systems

Now that you know the theory and principles behind authorization mechanisms, let's turn our attention to how they are integrated with the authentication systems discussed in Chapter 16. Together, authentication and authorization are at the heart of cybersecurity. The following sections give you some technical details on the most common technologies with which you should be familiar. First, however, we have to talk a bit about markup languages, which, as you'll see shortly, play an important role in authentication and authorization.

Access Control and Markup Languages

If you can remember when *Hypertext Markup Language (HTML)* was *all* we had to make a static web page, you might be considered "old" in terms of the technology world; HTML came out in the early 1990s. HTML evolved from Standard Generalized Markup Language (SGML), which evolved from the Generalized Markup Language (GML). We still use HTML, so it is certainly not dead and gone; the industry has just improved upon the markup languages available for use to meet today's needs.

A *markup language* is a way to structure text and data sets, and it dictates how these will be viewed and used. When you adjust margins and other formatting capabilities in a word processor, you are marking up the text in the word processor's markup language. If you develop a web page, you are using some type of markup language. You can control how it looks and some of the actual functionality the page provides. The use of a standard markup language also allows for interoperability. If you develop a web page and follow basic markup language standards, the page will basically look and act the same no matter what web server is serving up the web page or what browser the viewer is using to interact with it.

As the Internet grew in size and the World Wide Web (WWW) expanded in functionality, and as more users and organizations came to depend upon websites and web-based communication, the basic and elementary functions provided by HTML were not enough. And instead of every website having its own proprietary markup language to meet its specific functionality requirements, the industry had to have a way for functionality needs to be met and still provide interoperability for all web server and web browser interaction. This is the reason that *Extensible Markup Language (XML)* was developed. XML is a universal and foundational standard that provides a structure for other independent markup languages to be built from and still allow for interoperability. Markup languages with various functionalities were built from XML, and while each language provides its own individual functionality, if they all follow the core rules of XML, then they are interoperable and can be used across different web-based applications and platforms.

As an analogy, let's look at the English language. Samir is a biology scientist, Trudy is an accountant, and Val is a network administrator. They all speak English, so they have a common set of communication rules, which allow them to talk with each other, but each has their own "spin-off" language that builds upon and uses the English language as its core. Samir uses words like "mitochondrial amino acid genetic strains" and "DNA polymerase." Trudy uses words such as "accrual accounting" and "acquisition indigestion." Val uses terms such as "multiprotocol label switching" and "subkey creation." Each profession has its own "language" to meet its own needs, but each is based off the same core language—English. In the world of the WWW, various websites need to provide different types of functionality through the use of their own language types but still need a way to communicate with each other and their users in a consistent manner, which is why they are based upon the same core language structure (XML). There are hundreds of markup languages based upon XML, but we are going to focus on the ones that are used for identity management and access control purposes.

The *Service Provisioning Markup Language (SPML)* allows for the exchange of provisioning data between applications, which could reside in one organization or many; allows for the automation of user management (account creation, amendments, revocation) and access entitlement configuration related to electronically published services across multiple provisioning systems; and allows for the integration and interoperation of service provisioning requests across various platforms.

When an organization hires a new employee, that employee usually needs access to a wide range of systems, servers, and applications. Setting up new accounts on every system, properly configuring access rights, and then maintaining those accounts throughout their lifetimes is time-consuming, laborious, and error-prone. What if the organization has 20,000 employees and thousands of network resources that each employee needs

various access rights to? This opens the door for confusion, mistakes, vulnerabilities, and a lack of standardization.

SPML allows for all these accounts to be set up and managed simultaneously across the various systems and applications. SPML is made up of three main entities: the Requesting Authority (RA), which is the entity that is making the request to set up a new account or make changes to an existing account; the Provisioning Service Provider (PSP), which is the software that responds to the account requests; and the Provisioning Service Target (PST), which is the entity that carries out the provisioning activities on the requested system.

So when a new employee is hired, there is a request to set up the necessary user accounts and access privileges on several different systems and applications across the enterprise. This request originates in a piece of software carrying out the functionality of the RA. The RA creates SPML messages, which provide the requirements of the new account, and sends them to a piece of software that is carrying out the functionality of the PSP. This piece of software reviews the requests and compares them to the organization's approved account creation criteria. If these requests are allowed, the PSP sends new SPML messages to the end systems (PST) that the user actually needs to access. Software on the PST sets up the requested accounts and configures the necessary access rights. If this same employee is fired three months later, the same process is followed and all necessary user accounts are deleted. This allows for consistent account management in complex environments. These steps are illustrated in Figure 17-3.

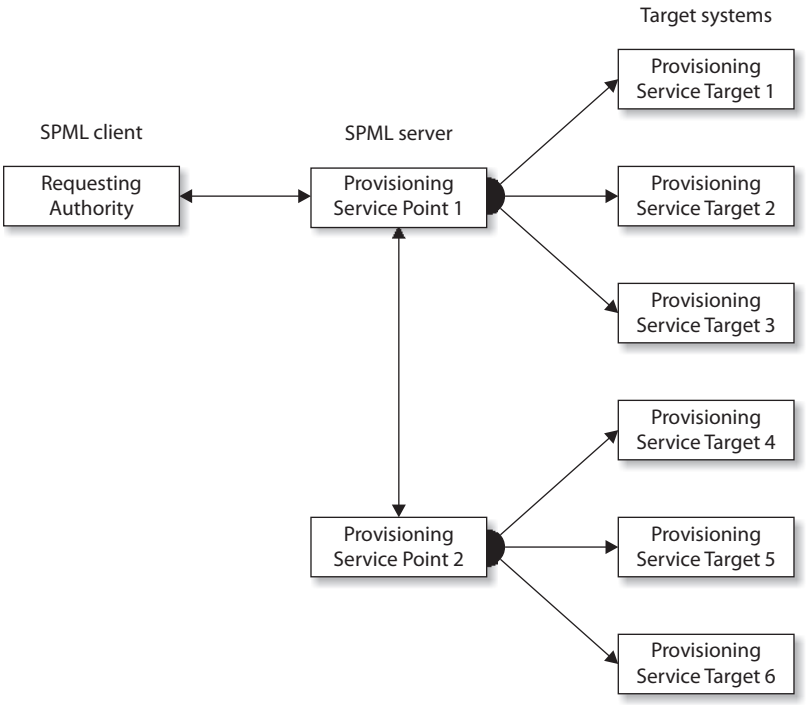


Figure 17-3 SPML provisioning steps

When there is a need to allow a user to log in one time and gain access to different and separate web-based applications, the actual authentication data has to be shared between the systems maintaining those web applications securely and in a standardized manner. This is the role that the *Security Assertion Markup Language (SAML)* plays. It is an XML standard that allows the exchange of authentication and authorization data to be shared between security domains. Suppose your organization, Acme Corp., uses Gmail as its corporate e-mail platform. You would want to ensure that you maintain control over user access credentials so that you could enforce password policies and, for example, prevent access to the e-mail account of an employee who just got fired. You could set up a relationship with Google that would allow you to do just this using SAML. Whenever one of your organization's users attempted to access their corporate Gmail account, Gmail would redirect their request to Acme's single sign-on (SSO) service, which would authenticate the user and relay (through the user) a SAML response. Figure 17-4 depicts this process, though its multiple steps are largely transparent to the user.

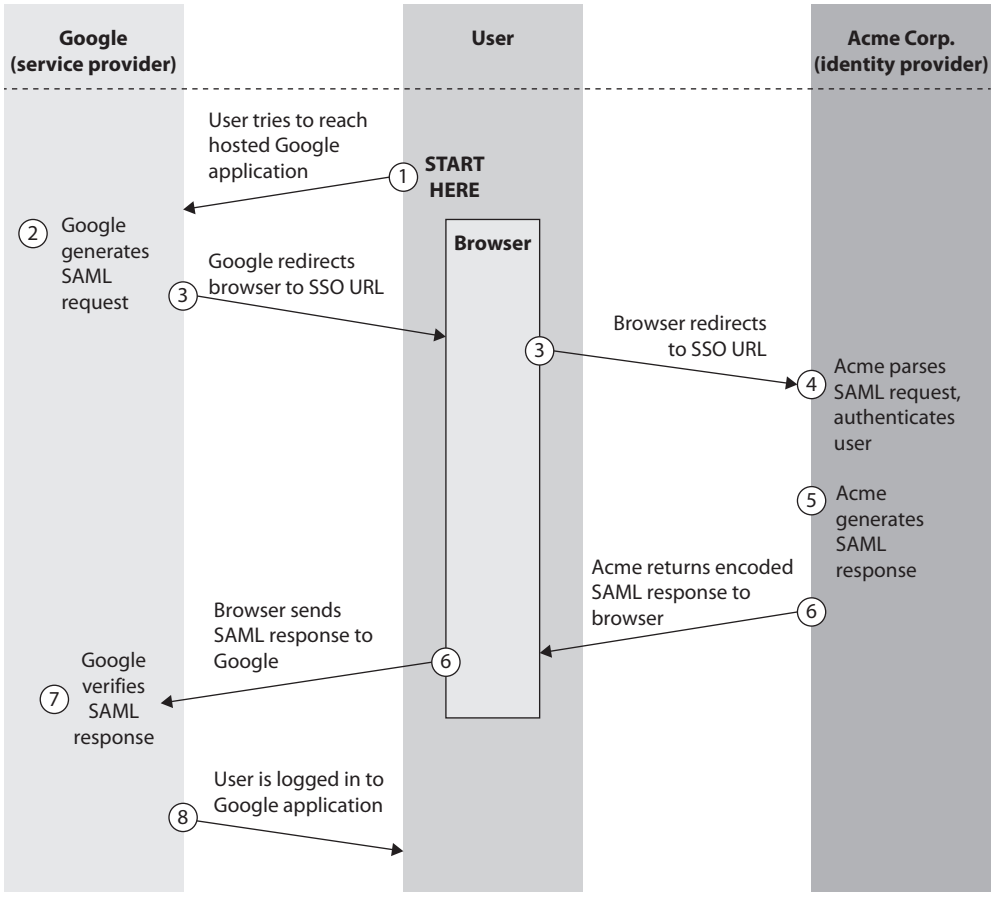


Figure 17-4 SAML authentication

SAML provides the authentication pieces to federated identity management systems to allow business-to-business (B2B) and business-to-consumer (B2C) transactions. In our previous example, the user is considered the *principal*, Acme Corporation is the *identity provider*, and Gmail is the *service provider.*

This is not the only way that the SAML language can be used. The digital world has evolved to being able to provide extensive services and functionality to users through web-based machine-to-machine communication standards. As we discussed in Chapter 13, *web services* is a collection of technologies and standards that allow services (weather updates, stock tickers, e-mail, customer resource management, etc.) to be provided on distributed systems and be "served up" in one place.

Transmission of SAML data can take place over different protocol types, but a common one is *Simple Object Access Protocol (SOAP)*. As you may recall from Chapter 13, SOAP is a specification that outlines how information pertaining to web services is exchanged in a structured manner. It provides the basic messaging framework, which allows users to request a service and, in exchange, the service is made available to that user. Let's say you need to interact with your company's CRM system, which is hosted and maintained by the vendor—for example, Salesforce.com. You would log into your company's portal and double-click a link for Salesforce. Your company's portal would take this request and your authentication data and package it up in an SAML format and encapsulate that data into a SOAP message. This message would be transmitted over an HTTP connection to the Salesforce vendor site, and once you were authenticated, you would be provided with a screen that shows you the company's customer database. The SAML, SOAP, and HTTP relationship is illustrated in Figure 17-5.

The use of web services in this manner also allows for organizations to provide *service-oriented architecture (SOA)* environments. An SOA is a way to provide independent

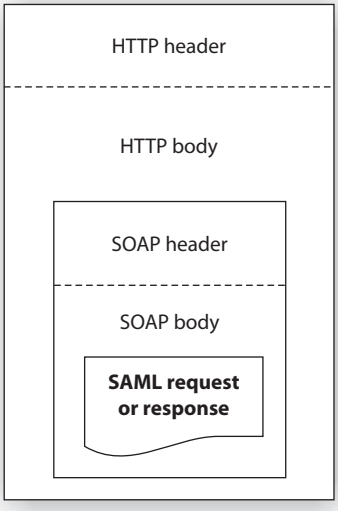


Figure 17-5 SAML material embedded within an HTTP message

services residing on different systems in different business domains in one consistent manner. For example, if your organization has a web portal that allows you to access the organization's CRM, an employee directory, and a help-desk ticketing application, this is most likely being provided through an SOA. The CRM system may be within the marketing department, the employee directory may be within the HR department, and the ticketing system may be within the IT department, but you can interact with all of them through one interface. SAML is a way to send your authentication information to each system, and SOAP allows this type of information to be presented and processed in a unified manner.

The last XML-based standard we will look at is *Extensible Access Control Markup Language (XACML).* XACML is used to express security policies and access rights to assets provided through web services and other enterprise applications. SAML is just a way to send around your authentication information, as in a password, key, or digital certificate, in a standard format. SAML does not tell the receiving system how to interpret and use this authentication data. Two systems have to be configured to use the same type of authentication data. If you log into System A and provide a password and try to access System B, which only uses digital certificates for authentication purposes, your password is not going to give you access to System B's service. So both systems have to be configured to use passwords. But just because your password is sent to System B does not mean you have complete access to all of System B's functionality. System B has access policies that dictate the operations that specific subjects can carry out on its resources. The access policies can be developed in the XACML format and enforced by System B's software.

XACML is both an access control policy language and a processing model that allows for policies to be interpreted and enforced in a standard manner. When your password is sent to System B, there is a rules engine on that system that interprets and enforces the XACML access control policies. If the access control policies are created in the XACML format, they can be installed on both System A and System B to allow for consistent security to be enforced and managed.

XACML uses a Subject element (requesting entity), a Resource element (requested entity), and an Action element (types of access). So if you request access to your company's CRM, you are the Subject, the CRM application is the Resource, and your access parameters are outlined in the Action element.

NOTE Who develops and keeps track of all of these standardized languages? The *Organization for the Advancement of Structured Information Standards (OASIS)*. This organization develops and maintains the standards for how various aspects of web-based communication are built and maintained.

Web services, SOA environments, and the implementation of these different XML-based markup languages vary in nature because they allow for extensive flexibility. Because so much of the world's communication takes place through web-based processes, it is increasingly important for security professionals to understand these issues and technologies.

OAuth

OAuth is an open standard for authorization (not authentication) to third parties. The general idea is that this lets you authorize a website to use something that you control at a different website. For instance, if you have a LinkedIn account, the system might ask you to let it have access to your Google contacts in order to find your friends who already have accounts in LinkedIn. If you agree, you next see a pop-up from Google asking whether you want to authorize LinkedIn to manage your contacts. If you agree to this, LinkedIn gains access to all your contacts until you rescind this authorization. With OAuth a user allows a website to access a third party. The latest version of OAuth, which is version 2.0, is defined in Request for Comments (RFC) 6749. It defines four roles as described here:

- **Client** A process that requests access to a protected resource. It is worth noting that this term describes the relationship of an entity with a resource provider in a client/server architecture. This means the "client" could actually be a web service (e.g., LinkedIn) that makes requests from another web service (e.g., Google).
- **Resource server** The server that controls the resource that the client is trying to access.
- **Authorization server** The system that keeps track of which clients are allowed to use which resources and issues access tokens to those clients.
- **Resource owner** Whoever owns a protected resource and is able to grant permissions for others to use it. These permissions are usually granted through a consent dialog box. The resource owner is typically an end user, but could be an application or service.

Figure 17-6 shows a resource owner granting an OAuth client access to protected resources in a resource server. This could be a user who wants to tweet directly from

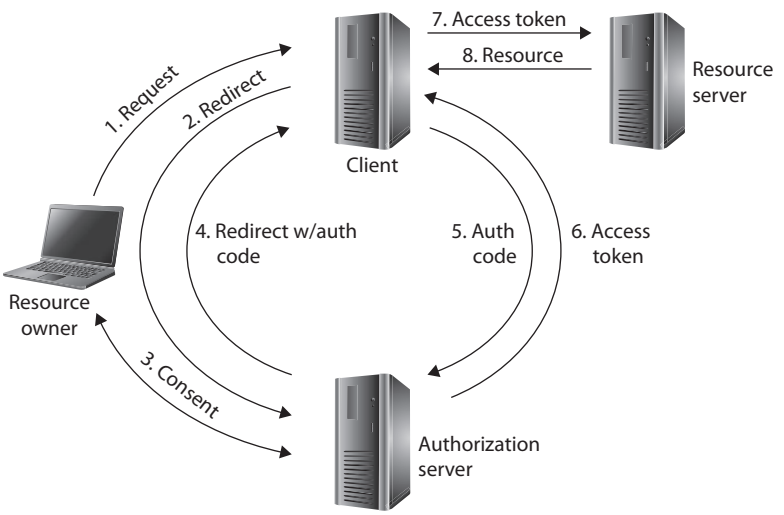


Figure 17-6 OAuth authorization steps

a LinkedIn page, for example. The resource owner sends a request to the client, which is redirected to an authorization server. This server negotiates consent with the resource owner and then redirects an HTTPS-secured message back to the client, including in the message an authorization code. The client next contacts the authorization server directly with the authorization code and receives in return an access token for the protected resource. Thereafter, as long as the token has not expired or the authorization is not rescinded by the resource owner, the client is able to present the token to the resource server and access the resource. Note that it is possible (and indeed fairly common) for the resource server and authorization server to reside on the same computing node.

Although OAuth is an authorization framework, it relies on some sort of authentication mechanism to verify the identity of the resource owner whenever permissions are changed on a protected resource. This authentication is outside the scope of the OAuth standard, but can be implicitly used, as described in the following section.

OpenID Connect

OpenID Connect (OIDC) is a simple authentication layer built on top of the OAuth 2.0 protocol. It allows transparent authentication and authorization of client resource requests, as shown in Figure 17-7. Most frequently, OIDC is used to allow a web application (relying party) to not only authenticate an end user using a third-party identity provider (IdP) but also get information about that user from that IdP. When end users attempt to log into the web service, they see a login prompt from the IdP (e.g., Google) and, after correctly authenticating, are asked for consent to share information (e.g., name, e-mail address) with the web service. The information shared can be arbitrary as long as it is configured at the IdP, the relying party explicitly requests it, and the end user consents to it being shared.

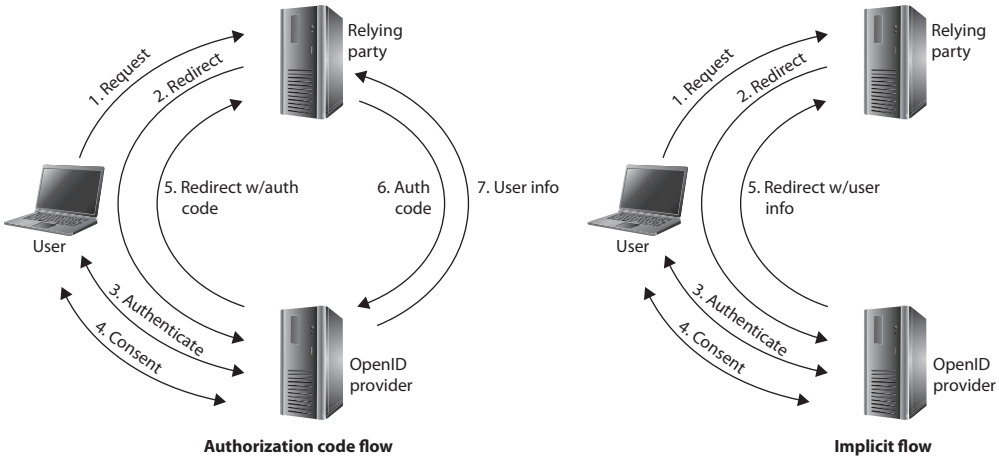


Figure 17-7 Two common OpenID Connect process flows

OIDC supports three flows:

- **Authorization code flow** The relying party is provided an authorization code (or token) and must use it to directly request the ID token containing user information from the IdP.
- **Implicit flow** The relying party receives the ID token containing user information with the redirect response from the IdP after user authentication and consent. The token is passed through the user's browser, potentially exposing it to tampering.
- **Hybrid flow** Essentially a combination of the previous two flows.

Figure 17-7 illustrates the first two flows, which are the most common ones in use. In the *authorization code flow*, the user requests a protected resource on the relying party's server, which triggers a redirect to the OpenID provider for authentication. The OpenID provider authenticates the user and then requests user consent to sharing specific kinds of information (e.g., e-mail, phone, profile, address), which are called *scope values*. The OpenID provider then redirects the user's browser back to the relying party and includes an authorization code. The relying party then presents this code to the OpenID provider and requests the user information, which is delivered to it in an ID token.

The *implicit flow* is similar to the authorization code flow, but the relying party includes the requested scope values in the authentication redirect to the OpenID provider. After the user is authenticated and consents to sharing the information, the OpenID provider includes the ID token with the user's information in the redirect back to the relying party.

The authorization code flow is preferred because it is more secure. The client app on the relying party obtains the ID token directly from the IdP, which means the user is unable to tamper with it. It also allows the OpenID provider to authenticate the client app that is requesting the user's information. This flow requires that the client app have a server backend. If the client app is browser-based (e.g., JavaScript) and doesn't have a server backend, then the implicit flow can be used. It is considered less secure because the ID token with the user information is given to the user's browser, where it could be compromised or manipulated.

Kerberos

The previous access control technologies were focused on service-oriented architectures and web services. Alas, not every system fits those architectures. We still need authentication and authorization when we log into our work computers and in many other scenarios. Enter Kerberos.

Kerberos is the name of a three-headed dog that guards the entrance to the underworld in Greek mythology. This is a great name for a security technology that provides authentication functionality, with the purpose of protecting an organization's assets. Kerberos is an authentication protocol and was designed in the mid-1980s as part of MIT's Project Athena. It works in a client/server model and is based on symmetric key cryptography. The protocol has been used for years in Unix systems and is the default authentication method for Windows operating systems. In addition, Apple macOS, Oracle Solaris, and Red Hat Enterprise Linux all use Kerberos authentication. Commercial products supporting Kerberos are fairly common, so this one might be a keeper.

Kerberos is an example of an SSO system for distributed environments, and it has become a de facto standard for heterogeneous networks. Kerberos incorporates a wide range of security capabilities, which gives organizations much more flexibility and scalability when they need to provide an encompassing security architecture. It has four elements necessary for enterprise access control: scalability, transparency, reliability, and security. However, this open architecture also invites interoperability issues. When vendors have a lot of freedom to customize a protocol, it usually means no two vendors will customize it in the same fashion. This creates interoperability and incompatibility issues.

Kerberos uses symmetric key cryptography and provides end-to-end security. Although it allows the use of passwords for authentication, it was designed specifically to eliminate the need to transmit passwords over the network. Most Kerberos implementations work with shared secret keys.

Main Components in Kerberos

The *Key Distribution Center (KDC)* is the most important component within a Kerberos environment. The KDC holds all users' and services' secret keys. It provides an authentication service, as well as key distribution functionality. The clients and services trust the integrity of the KDC, and this trust is the foundation of Kerberos security.

The KDC provides security services to *principals*, which can be users, applications, or network services. The KDC must have an account for, and share a secret key with, each principal. For users, a password is transformed into a secret key value. The secret key can be used to send sensitive data back and forth between the principal and the KDC, and is used for user authentication purposes.

A *ticket* is generated by the *ticket granting service (TGS)* on the KDC and given to a principal when that principal, let's say a user, needs to authenticate to another principal, let's say a print server. The ticket enables one principal to authenticate to another principal. If Emily needs to use the print server, she must prove to the print server she is who she claims to be and that she is authorized to use the printing service. So Emily requests a ticket from the TGS. The TGS gives Emily the ticket, and in turn, Emily passes this ticket on to the print server. If the print server approves this ticket, Emily is allowed to use the print service.

A KDC provides security services for a set of principals. This set is called a *realm* in Kerberos. The KDC is the trusted authentication server for all users, applications, and services within a realm. One KDC can be responsible for one realm or several realms. Realms enable an administrator to logically group resources and users.

So far, we know that principals (users, applications, and services) require the KDC's services to authenticate to each other; that the KDC has a database filled with information about every principal within its realm; that the KDC holds and delivers cryptographic keys and tickets; and that tickets are used for principals to authenticate to each other. So how does this process work?

The Kerberos Authentication Process

The user and the KDC share a secret key, while the service and the KDC share a different secret key. The user and the requested service do not share a symmetric key in the beginning. The user trusts the KDC because they share a secret key. They can encrypt

and decrypt data they pass between each other, and thus have a protected communication path. Once the user authenticates to the service, they, too, will share a symmetric key (session key) that is used for authentication purposes.

Here are the exact steps:

1. Emily comes in to work and enters her username and password into her workstation at 8:00 A.M. The Kerberos software on Emily's computer sends the username to the authentication service (AS) on the KDC, which in turn sends Emily a ticket granting ticket (TGT) that is encrypted with the TGS's secret key.
2. If Emily has entered her correct password, the TGT is decrypted and Emily gains access to her local workstation desktop.
3. When Emily needs to send a print job to the print server, her system sends the TGT to the TGS, which runs on the KDC, and a request to access the print server. (The TGT allows Emily to prove she has been authenticated and allows her to request access to the print server.)
4. The TGS creates and sends a second ticket to Emily, which she will use to authenticate to the print server. This second ticket contains two instances of the same session key, one encrypted with Emily's secret key and the other encrypted with the print server's secret key. The second ticket also contains an *authenticator*, which contains identification information on Emily, her system's IP address, sequence number, and a timestamp.
5. Emily's system receives the second ticket, decrypts and extracts the embedded session key, adds a second authenticator set of identification information to the ticket, and sends the ticket on to the print server.
6. The print server receives the ticket, decrypts and extracts the session key, and decrypts and extracts the two authenticators in the ticket. If the print server can decrypt and extract the session key, it knows the KDC created the ticket, because only the KDC has the secret key used to encrypt the session key. If the authenticator information that the KDC and the user put into the ticket matches, then the print server knows it received the ticket from the correct principal.
7. Once this is completed, it means Emily has been properly authenticated to the print server and the server prints her document.

This is an extremely simplistic overview of what is going on in any Kerberos exchange, but it gives you an idea of the dance taking place behind the scenes whenever you interact with any network service in an environment that uses Kerberos. Figure 17-8 provides a simplistic view of this process.

The authentication service is the part of the KDC that authenticates a principal, and the TGS is the part of the KDC that makes the tickets and hands them out to the principals. TGTs are used so the user does not have to enter his password each time he needs to communicate with another principal. After the user enters his password, it is temporarily stored on his system, and any time the user needs to communicate with another principal, he just reuses the TGT.

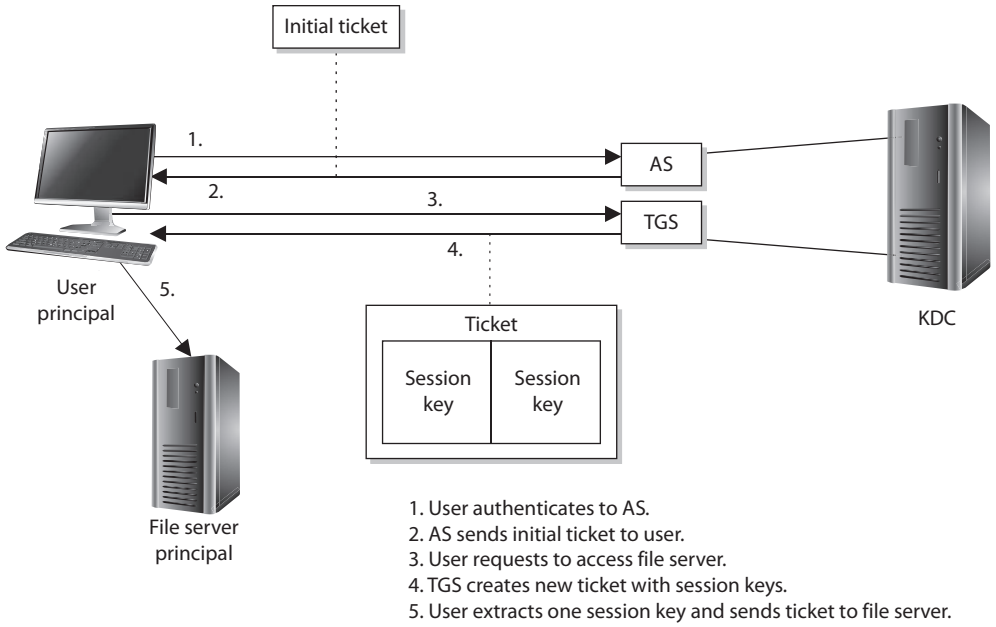


Figure 17-8 The user must receive a ticket from the KDC before being able to use the requested resource.

EXAM TIP Be sure you understand that a session key is different from a secret key. A secret key is shared between the KDC and a principal and is static in nature. A session key is shared between two principals and is generated when needed and is destroyed after the session is completed.

If a Kerberos implementation is configured to use an *authenticator*, the user sends to the print server her identification information and a timestamp and sequence number encrypted with the session key they share. The print server decrypts this information and compares it with the identification data the KDC sent to it about this requesting user. If the data is the same, the print server allows the user to send print jobs. The timestamp is used to help fight against replay attacks. The print server compares the sent timestamp with its own internal time, which helps determine if the ticket has been sniffed and copied by an attacker and then submitted at a later time in hopes of impersonating the legitimate user and gaining unauthorized access. The print server checks the sequence number to make sure that this ticket has not been submitted previously. This is another countermeasure to protect against replay attacks.

NOTE A replay attack is when an attacker captures and resubmits data (commonly a credential) with the goal of gaining unauthorized access to an asset.

The primary reason to use Kerberos is that the principals do not trust each other enough to communicate directly. In our example, the print server will not print anyone's print job without that entity authenticating itself. So none of the principals trust each other directly; they only trust the KDC. The KDC creates tickets to vouch for the individual principals when they need to communicate. Suppose Rodrigo needs to communicate directly with you, but you do not trust him enough to listen and accept what he is saying. If he first gives you a ticket from something you do trust (KDC), this basically says, "Look, the KDC says I am a trustworthy person. The KDC asked me to give this ticket to you to prove it." Once that happens, *then* you will communicate directly with Rodrigo.

The same type of trust model is used in PKI environments. In a PKI environment, users do not trust each other directly, but they all trust the certificate authority (CA). The CA vouches for the individuals' identities by using digital certificates, the same as the KDC vouches for the individuals' identities by using tickets.

So why are we talking about Kerberos? Because it is one example of an SSO technology. The user enters a user ID and password one time and one time only. The tickets have time limits on them that administrators can configure. Many times, the lifetime of a TGT is eight to ten hours, so when the user comes in the next day, he has to present his credentials again.

NOTE Kerberos is an open protocol, meaning that vendors can manipulate it to work properly within their products and environments. The industry has different "flavors" of Kerberos, since various vendors require different functionality.

Weaknesses of Kerberos

The following are some of the potential weaknesses of Kerberos:

- The KDC can be a single point of failure. If the KDC goes down, no one can access needed resources. Redundancy is necessary for the KDC.
- The KDC must be able to handle the number of requests it receives in a timely manner. It must be scalable.
- Secret keys are temporarily stored on the users' workstations, which means it is possible for an intruder to obtain these cryptographic keys.
- Session keys are decrypted and reside on the users' workstations, either in a cache or in a key table. Again, an intruder might capture these keys.
- Kerberos is vulnerable to password guessing. The KDC does not know if a dictionary attack is taking place.
- Network traffic is not protected by Kerberos if encryption is not enabled.
- If the keys are too short, they can be vulnerable to brute-force attacks.
- Kerberos needs all client and server clocks to be synchronized.

Kerberos and Password-Guessing Attacks

Just because an environment uses Kerberos does not mean the systems are vulnerable to password-guessing attacks. The operating system itself will (should) provide the protection of tracking failed login attempts. The Kerberos protocol does not have this type of functionality, so another component must be in place to counter these types of attacks. No need to start ripping Kerberos out of your network environment after reading this section; your operating system provides the protection mechanism for this type of attack.

Kerberos must be transparent (work in the background without the user needing to understand it), scalable (work in large, heterogeneous environments), reliable (use distributed server architecture to ensure there is no single point of failure), and secure (provide authentication and confidentiality).

Remote Access Control Technologies

The following sections present some examples of centralized remote access control technologies. Each of these authentication protocols is referred to as an AAA protocol, which stands for authentication, authorization, and auditing. (Some resources have the last A stand for accounting, but it is the same functionality—just a different name.)

Depending upon the protocol, there are different ways to authenticate a user in this client/server architecture. The traditional authentication protocol is the Challenge Handshake Authentication Protocol (CHAP), but many systems are now using Extensible Authentication Protocol (EAP). We discussed each of these authentication protocols at length in Chapter 15.

RADIUS

Remote Authentication Dial-In User Service (RADIUS) is a network protocol that provides client/server authentication and authorization and audits remote users. A network may have access servers, DSL, ISDN, or a T1 line dedicated for remote users to communicate through. The access server requests the remote user's logon credentials and passes them back to a RADIUS server, which houses the usernames and password values. The remote user is a client to the access server, and the access server is a client to the RADIUS server.

Most ISPs today use RADIUS to authenticate customers before they are allowed access to the Internet. The access server and customer's software negotiate through a handshake procedure and agree upon an authentication protocol (CHAP or EAP). The customer provides to the access server a username and password. This communication takes place over a Point-to-Point Protocol (PPP) connection. The access server and RADIUS server communicate over the RADIUS protocol. Once the authentication is completed properly, the customer's system is given an IP address and connection parameters and is allowed access to the Internet. The access server notifies the RADIUS server when the session starts and stops for billing purposes.

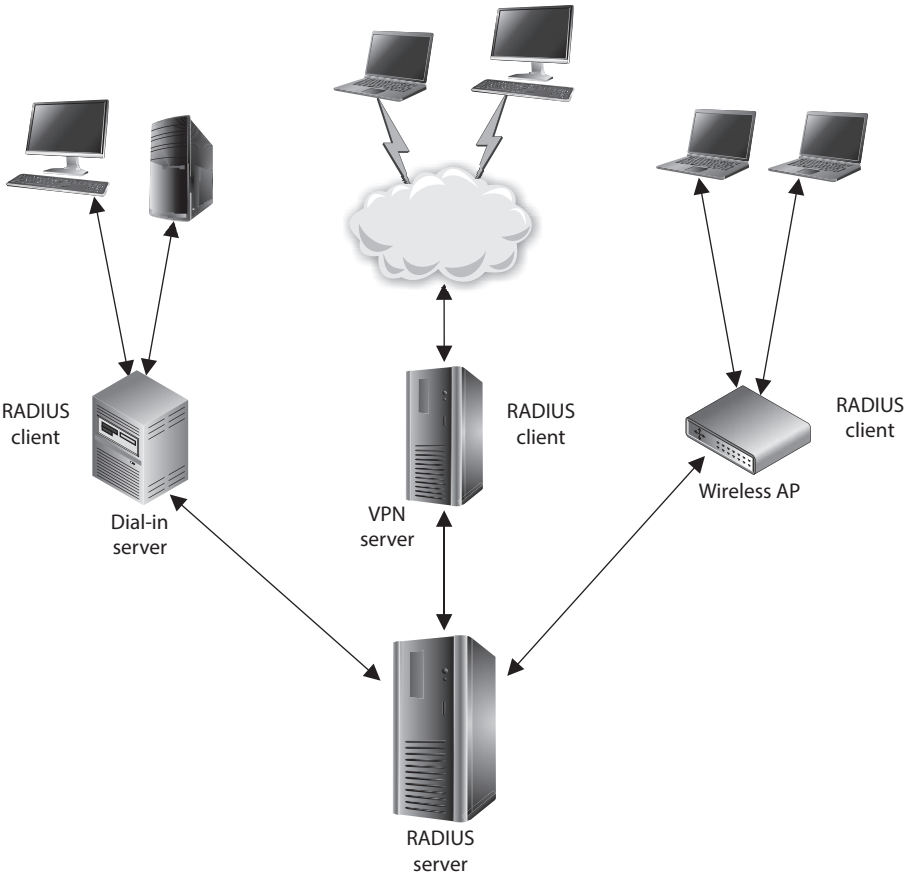


Figure 17-9 Environments can implement different RADIUS infrastructures.

RADIUS was developed by Livingston Enterprises for its network access server product series, but was then published as a set of standards (RFC 2865 and RFC 2866). This means it is an open protocol that any vendor can use and manipulate so that it works within its individual products. Because RADIUS is an open protocol, it can be used in different types of implementations. The format of configurations and user credentials can be held in LDAP servers, various databases, or text files. Figure 17-9 shows some examples of possible RADIUS implementations.

TACACS

Terminal Access Controller Access Control System (TACACS) has a very funny name. Not funny ha-ha, but funny "huh?" TACACS has been through three generations: TACACS, Extended TACACS (XTACACS), and TACACS+. TACACS combines its authentication and authorization processes; XTACACS separates authentication, authorization, and auditing processes; and TACACS+ is XTACACS with extended two-factor user authentication.

TACACS uses fixed passwords for authentication, while TACACS+ allows users to employ dynamic (one-time) passwords, which provides more protection. Although TACACS+ is now an open standard, both it and XTACACS started off as Cisco-proprietary protocols that were inspired by, but are not compatible with, TACACS.

NOTE TACACS+ is really not a new generation of TACACS and XTACACS; it is a distinct protocol that provides similar functionality and shares the same naming scheme. Because it is a totally different protocol, it is not backward-compatible with TACACS or XTACACS.

TACACS+ provides basically the same functionality as RADIUS with a few differences in some of its characteristics. First, TACACS+ uses TCP as its transport protocol, while RADIUS uses UDP. "So what?" you may be thinking. Well, any software that is developed to use UDP as its transport protocol has to be "fatter" with intelligent code that looks out for the items that UDP will not catch. Since UDP is a connectionless protocol, it will not detect or correct transmission errors. So RADIUS must have the necessary code to detect packet corruption, long timeouts, or dropped packets. Since the developers of TACACS+ chose to use TCP, the TACACS+ software does not need to have the extra code to look for and deal with these transmission problems. TCP is a connection-oriented protocol, and that is its job and responsibility.

RADIUS encrypts the user's password only as it is being transmitted from the RADIUS client to the RADIUS server. Other information, as in the username, accounting, and authorized services, is passed in cleartext. This is an open invitation for attackers to capture session information for replay attacks. Vendors who integrate RADIUS into their products need to understand these weaknesses and integrate other security mechanisms to protect against these types of attacks. TACACS+ encrypts all of this data between the client and server and thus does not have the vulnerabilities inherent in the RADIUS protocol.

The RADIUS protocol combines the authentication and authorization functionality. TACACS+ uses a true AAA architecture, which separates the authentication, authorization, and accounting functionalities. This gives a network administrator more flexibility in how remote users are authenticated. For example, if Tomika is a network administrator and has been assigned the task of setting up remote access for users, she must decide between RADIUS and TACACS+. If the current environment already authenticates all of the local users through a domain controller using Kerberos, then Tomika can configure the remote users to be authenticated in this same manner, as shown in Figure 17-10. Instead of having to maintain a remote access server database of remote user credentials and a database within Active Directory for local users, Tomika can just configure and maintain one database. The separation of authentication, authorization, and accounting functionality provides this capability. TACACS+ also enables the network administrator to define more granular user profiles, which can control the actual commands users can carry out.

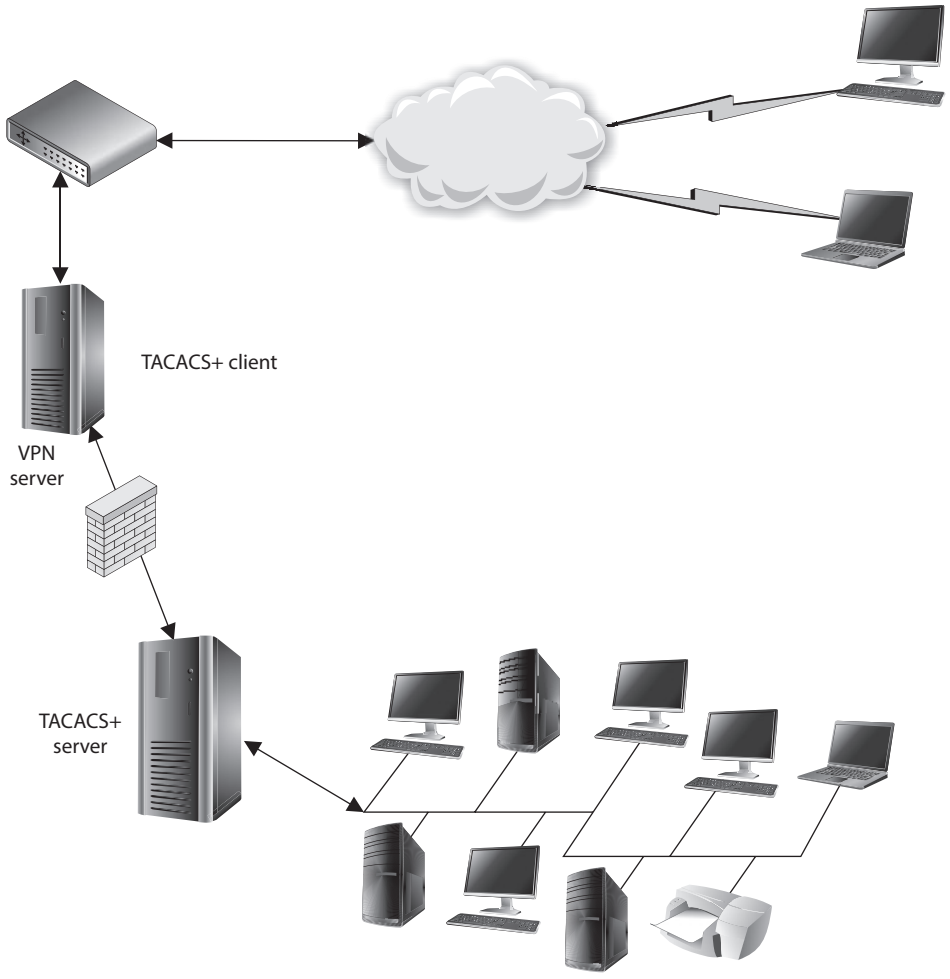


Figure 17-10 TACACS+ works in a client/server model.

Remember that RADIUS and TACACS+ are both protocols, and protocols are just agreed-upon ways of communication. When a RADIUS client communicates with a RADIUS server, it does so through the RADIUS protocol, which is really just a set of defined fields that will accept certain values. These fields are referred to as *attribute-value pairs (AVPs)*. As an analogy, suppose Ivan sends you a piece of paper that has several different boxes drawn on it. Each box has a headline associated with it: first name, last name, hair color, shoe size. You fill in these boxes with your values and send it back to Ivan. This is basically how protocols work; the sending system just fills in the boxes (fields) with the necessary information for the receiving system to extract and process.

	RADIUS	TACACS+
Packet delivery	UDP	TCP
Packet encryption	Encrypts only the password from the RADIUS client to the server.	Encrypts all traffic between the client and server.
AAA support	Combines authentication and authorization services.	Uses the AAA architecture, separating authentication, authorization, and auditing.
Multiprotocol support	Works over PPP connections.	Supports other protocols, such as AppleTalk, NetBIOS, and IPX.
Responses	Uses single-challenge response when authenticating a user, which is used for all AAA activities.	Uses multiple-challenge response for each of the AAA processes. Each AAA activity must be authenticated.

Table 17-2 Specific Differences Between These Two AAA Protocols

Since TACACS+ allows for more granular control on what users can and cannot do, TACACS+ has more AVPs, which allows the network administrator to define ACLs, filters, user privileges, and much more. Table 17-2 points out the differences between RADIUS and TACACS+.

So, RADIUS is the appropriate protocol when simplistic username/password authentication can take place and users only need an Accept or Deny for obtaining access, as in ISPs. TACACS+ is the better choice for environments that require more sophisticated authentication steps and tighter control over more complex authorization activities, as in corporate networks.

Diameter

Diameter is a protocol that has been developed to build upon the functionality of RADIUS and overcome many of its limitations. The creators of this protocol decided to call it Diameter as a play on the term RADIUS—as in *the diameter is twice the radius.*

Diameter is another AAA protocol that provides the same type of functionality as RADIUS and TACACS+ but also provides more flexibility and capabilities to meet the demands of today's complex and diverse networks. Today, we want our wireless devices and smartphones to be able to authenticate themselves to our networks, and we use roaming protocols, Mobile IP, Ethernet over PPP, Voice over IP (VoIP), and other crazy stuff that the traditional AAA protocols cannot keep up with. So the smart people came up with a new AAA protocol, Diameter, that can deal with these issues and many more.

Mobile IP

This technology allows a user to move from one network to another and still use the same IP address. It is an improvement upon the IP protocol because it allows a user to have a *home IP address*, associated with his home network, and a *care-of address*. The care-of address changes as the user moves from one network to the other. All traffic that is addressed to his home IP address is forwarded to his care-of address.

The Diameter protocol consists of two portions. The first is the base protocol, which provides the secure communication among Diameter entities, feature discovery, and version negotiation. The second is the extensions, which are built on top of the base protocol to allow various technologies to use Diameter for authentication.

Up until the conception of Diameter, the Internet Engineering Task Force (IETF) had individual working groups who defined how VoIP, Fax over IP (FoIP), Mobile IP, and remote authentication protocols work. Defining and implementing them individually in any network can easily result in too much confusion and interoperability. It requires customers to roll out and configure several different policy servers and increases the cost with each new added service. Diameter provides a base protocol, which defines header formats, security options, commands, and AVPs. This base protocol allows for extensions to tie in other services, such as VoIP, FoIP, Mobile IP, wireless, and cell phone authentication. So Diameter can be used as an AAA protocol for all of these different uses.

As an analogy, consider a scenario in which ten people all need to get to the same hospital, which is where they all work. They all have different jobs (doctor, lab technician, nurse, janitor, and so on), but they all need to end up at the same location. So, they can either all take their own cars and their own routes to the hospital, which takes up more hospital parking space and requires the gate guard to authenticate each car, or they can take a bus. The bus is the common element (base protocol) to get the individuals (different services) to the same location (networked environment). Diameter provides the common AAA and security framework that different services can work within.

RADIUS and TACACS+ are client/server protocols, which means the server portion cannot send unsolicited commands to the client portion. The server portion can only speak when spoken to. Diameter is a peer-based protocol that allows either end to initiate communication. This functionality allows the Diameter server to send a message to the access server to request the user to provide another authentication credential if she is attempting to access a secure resource.

Diameter is not directly backward-compatible with RADIUS but provides an upgrade path. Diameter uses TCP and AVPs and provides proxy server support. It has better error detection and correction functionality than RADIUS, as well as better failover properties, and thus provides better network resilience.

Diameter has the functionality and ability to provide the AAA functionality for other protocols and services because it has a large AVP set. RADIUS has 2^8 (256) AVPs, while Diameter has 2^{32} (a whole bunch). Recall from earlier in the chapter that AVPs are like boxes drawn on a piece of paper that outline how two entities can communicate back and forth. So, having more AVPs allows for more functionality and services to exist and communicate between systems.

Diameter provides the AAA functionality, as listed next.

Authentication:

- CHAP and EAP
- End-to-end protection of authentication information
- Replay attack protection

Authorization:

- Redirects, secure proxies, relays, and brokers
- State reconciliation
- Unsolicited disconnect
- Reauthorization on demand

Accounting:

- Reporting, roaming operations (ROAMOPS) accounting, event monitoring

Managing the Identity and Access Provisioning Life Cycle

Once an organization develops access control policies and determines the appropriate mechanisms, techniques, and technologies, it needs to implement procedures to ensure that identity and access are deliberately and systematically being issued to (and taken away from) users and systems. Many of us have either heard of or experienced the dismay of discovering that the credentials for someone who was fired months or years ago are still active in a domain controller. Some of us have even had to deal with that account having been used long after the individual left.

Identity and access have a life cycle, as illustrated in Figure 17-11. It begins with provisioning of an account, which we've already touched on in Chapter 16 in the context of registration and proofing of identities. Identities spend most of their lives being used for access control, which, as discussed in this chapter and the previous one, entails identification, authentication, and authorization of accounts. Changes invariably occur in organizations, and these changes impact identity and access control. For example, an employee gets promoted and her authorizations change. When changes occur, we want to ensure that our access control configurations remain up to date and effective. At some point, we need to

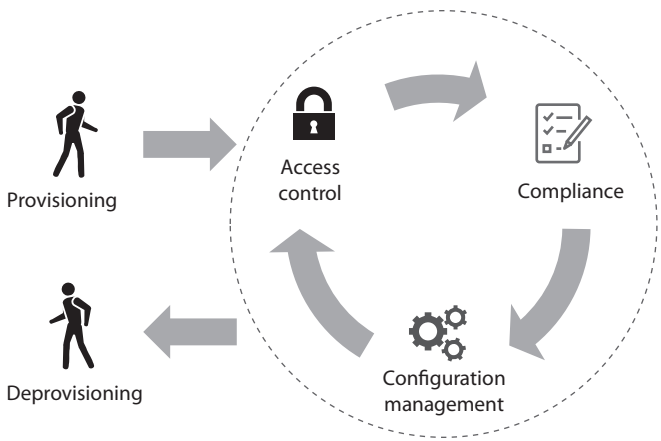


Figure 17-11 The identity and access management life cycle

ensure that we are in compliance with all applicable policies and regulations, so we also have to periodically review all the identities in the organization and their accesses. If all checks out, we let them continue to be used for access control. Inevitably, however, accounts need to be deprovisioned, which pops them out of the life-cycle model.

Provisioning

As introduced in Chapter 5, *provisioning* is the set of all activities required to provide one or more new information services to a user or group of users ("new" meaning previously not available to that user or group). In terms of identification and access management, this pertains to the creation of user objects or accounts and the assignment of permissions to those accounts. Sometimes, the term provisioning is used to describe the whole life cycle, but our focus in this section is on the first phase only, which is generating the account.

Normally, provisioning happens when a new user or system is added to an organization. For people, this is part of onboarding, which we discussed in Chapter 1. It is important to have an established process for ensuring that digital identities are issued only to the right folks. This usually involves reviews and approvals from HR staff, the individual's supervisor, and the IT department. The process is crucial for the rest of the life cycle because it answers the important question "Why did we provision this account?" The answer determines whether the account remains active or is deprovisioned at some point in the future. Keep in mind that some accounts are used for long periods of time, and rationales for provisioning accounts that were obvious in the beginning may be forgotten with the passage of time and with staffing changes.

Identity and access provisioning also pertains to system accounts, which are usually associated with services and automated agents and oftentimes require privileged access. A challenge here is that most organizations have a lot of system accounts that are largely invisible on a day-to-day basis. Just as for user identity and access, the trick here is to document what accounts were created, where they were created, and why they were created.

Access Control

We've already covered much of what happens in this phase of the life cycle, but it bears highlighting the fact that this is where most of the risk resides. Most security incidents are enabled by compromised authentication (e.g., user passwords are guessed, allowing adversaries to impersonate them) or authorization abuses (e.g., data to which users have legitimate access is used inappropriately or leaked). This is why it is essential to continuously monitor access controls, detect suspicious or malicious events, and generate alerts automatically. One of the most effective ways to do this is through the use of user and entity behavior analytics (UEBA), which we'll discuss in detail in Chapter 21.

Compliance

Monitoring access and detecting interesting events early is essential to ensure compliance both with internal policies and procedures and with any applicable external regulations. Keep in mind that the intent of these policies and regulations is to ensure information

systems security, so being able to attest that we are doing things "by the book" is good all around. It gives us personal peace of mind, protects our systems, and ensures our organizations don't get hit with hefty fines. Note that rights and permission reviews have been incorporated into many regulatory-induced processes. For example, as part of the Sarbanes-Oxley Act (SOX) regulations (introduced in Chapter 1), managers have to review their employees' permissions to data on an annual basis.

Compliance can be summed up in three major components:

- Written standards must exist that assign responsibilities to individuals. This could range from acceptable use policies to national laws like SOX.
- There must be a program by which applicable aspects of the security program are compared against the standard. In terms of identity and access management, this program is centered on user and system access reviews.
- Any discrepancies between the standard and reality must be addressed and resolved in a systemic way that ensures the issues are not likely to resurface next month.

We've already covered the first part (policies, regulations, and such) in Chapter 3, so let's look at user and system access reviews (in turn) and resolving discrepancies.

User Access Review

One day, every user account will be deprovisioned because no one works (or lives) forever. It may also be necessary to change permissions (such as when a user changes roles) or temporarily disable accounts (such as when a user goes on an extended leave of absence or is under some sort of adverse administrative action). The list of conditions under which an account is disabled or deprovisioned will vary by organization, but we all need to have such a list. We need a process by which we periodically (or upon certain conditions) check every user account against that list. The purpose of these user access reviews is to ensure we don't have excessive authorizations or active accounts that are no longer needed.

Ideally, we conduct reviews periodically (say, every six months) for all accounts (or, at least, a sampling of them). A review should also be triggered by certain administrative actions. A best practice is to integrate these reviews into the HR procedures because HR typically is involved in administrative actions anyway. Though it is obvious that user accounts should be disabled or deprovisioned when an employee is terminated, other situations are not as clear-cut, such as the following, and require a deliberate review by the individual's supervisor and/or the IT department:

- Promotion
- Transfer
- Extended vacation or sabbatical
- Hospitalization
- Long-term disability (with an expected future return)
- Investigation for possible wrongdoing
- Unexpected disappearance

System Account Access Review

As with user access, we should conduct system account access reviews both periodically and when certain conditions are met. Though HR would not be involved in these reviews, the principle is the same: every system account eventually needs to be changed, disabled, or deprovisioned. What makes reviewing systems accounts a bit trickier is that it is easy to forget they exist in the first place. It is not unusual for a service to require multiple accounts with which nobody interacts directly and yet are absolutely critical for the system. What's worse, sometimes software updates can remove the need for legacy system accounts that are not deprovisioned as part of the updating process and remain in place as a potential vulnerability. A systematic approach to system account access review is your best way to avoid ending up with unneeded, potentially privileged accounts.

Although these two terms are synonymous to most IT and security professionals, the CISSP CBK distinguishes between a system account and a service account. Technically, a *system account* is created by the operating system for use by a particular process, not by a human. Most OSs have a "system" context in which they run privileged operations. A *service account* is a system account for a process that runs as a service (i.e., it listens for and responds to requests from other processes).

Resolving Discrepancies

So, you just completed an account access review (user, service, and/or system) and have found some discrepancies. What do you do? A possible approach would be to change, disable, or deprovision the accounts and go on your merry way. This solves the immediate problem but may result in your having to repeat this process after every review if the cause for the discrepancies is systemic. That is why you are much better off treating the discrepancies as symptoms of a problem, not the end problems themselves.

A good approach is to list all the deficiencies and, for each one, answer the four guiding questions listed here. For each, we offer an example answer for illustration purposes.

- *What happened?* You came across an account used by a service that was removed during a software upgrade months ago. The service is gone, but its account remains.
- *Why did it happen?* Because your team didn't fully understand the impact of the software update and didn't check the state of the system after it was applied.
- *What can we infer from it about discrepancies in other places and/or the future?* There may be other similarly orphaned service accounts in the environment, so we should audit them all.
- *How do we best correct this across the board?* We should create a process by which all services are characterized prior to being updated, the effects of the update are determined ahead of time, and any changes to access controls are implemented during the update process. This should all be included in our configuration management program going forward.

The example shouldn't be surprising, since many of us have come across this (or a very similar) situation. We find an account that is still enabled even though the staff member left the organization a year ago and we rush to fix it without thinking about how this

happened in the first place. Fixing a broken process is way more effective than fixing a single discrepancy that was caused by it, and there is no more important process in the identity and access life cycle than configuration management.

Configuration Management

Configuration management is a broad subject that we'll discuss in detail in Chapter 20. However, with regard to the identity and access management life cycle, it really boils down to having a firm grasp (and control) of all the subjects and objects in our environment and of the manner in which they interrelate. If you think of the identity and access management (IAM) configuration of your systems in this way, you'll see that there are three drivers of change to your systems' configurations: users change, objects change, and authorizations change. The first of these (users change) is covered here in the provisioning and deprovisioning sections. We already gave an example of the second (objects change) in our previous example of a system account access review earlier in this chapter. Let's turn our attention in the sections that follow to what is, perhaps, the thorniest of the three: authorizations change.

Role Definitions

We tend to be very thorough in assigning the right authorizations to our staff members when we onboard them and provision their accounts. Usually, we follow lots of processes and ensure that various forms are signed by the correct people. We tend to not be as thorough, however, when somebody's role in the organization changes. This could be a promotion, a transfer to a new role, or a merger (or split) of roles. It is just as important to be thorough in these cases to ensure we maintain proper access controls.

A good way to do this is for the HR, IT, security, and business teams to jointly develop a role matrix that specifies who should have access to what and to periodically review and update the matrix as needed. Then, whenever there is a personnel action, the HR team notifies the IT staff of it. IT, in turn, updates the person's authorizations and everyone is happy.

More commonly, however, staff members are often assigned more and more access rights and permissions over time as their roles change and they move from one department to another. This is commonly referred to as *authorization creep*. It can be a large risk for an organization because it leads to too many users having too much privileged access to the organization's assets. In the past, it has usually been easier for network administrators to give more access than less, because then the user would not come back and require more work to be done on her profile. It is also difficult to know the exact access levels different individuals require unless something like a role matrix exists. Enforcing least privilege on user accounts should be an ongoing job, which means it should be part of a formal configuration management process.

Privilege Escalation

If we apply the principle of least privilege effectively, then everyone should be running with minimal access to get their day-to-day tasks done. Even system administrators should have to temporarily elevate their privileges to perform sensitive tasks. In Linux environments, this is normally done using the `sudo` command, which temporarily

changes the authorizations of a user, typically to the root user. The Windows graphical user interface gives you the option to "run as administrator," which is similar to `sudo` but not as powerful or flexible. If you want that level of power, you need to drop to the Windows command line and use the `runas` command.

Privilege escalation, necessary as it might be, should be minimized in any environment. That means you want the least number of folks with admin privileges running around. It also means that you want to put in place policies regarding when and why account privileges should be escalated. Lastly, you want to log any such escalation so the use of newly escalated privileges use can be audited. We get into managing privileged accounts in Chapter 20, but these principles are worth keeping in mind (and repeating a few times).

Managed Service Accounts

Closely related to privilege escalation is the concept of service accounts that need to be accessible to certain administrators. Service accounts can be tricky because they are typically privileged but are not normally used by a human user. It is (sadly) all too common to see administrators install and configure a service, create an account for it to run under, and forget about the account. Passwords for these accounts are frequently exempt from password policy enforcement (e.g., complexity, expiration), so you can have some well-intentioned admins set an easy-to-remember password (because it could be disastrous if they ever forgot it) that never expired on a privileged account that few knew existed in the first place. Not good.

Microsoft Windows includes a feature that helps solve this challenge. *Managed service accounts (MSAs)* are Active Directory (AD) domain accounts that are used by services and provide automatic password management. MSAs can be used by multiple users and systems without any of them having to know the password. The way MSAs work is that AD creates policy-compliant passwords for these accounts and regularly changes them (every 30 days by default). Systems and users that are authorized to use these accounts need only be authenticated themselves on a domain controller and be included in that MSA's ACL. Think of it as an extension to SSO where you get to be authenticated as yourself or as any MSA that you are authorized to access.

Deprovisioning

As we already said, sooner or later every account should get deprovisioned. For users, this is usually part of the termination procedures we covered in Chapter 1. For system accounts, this could happen because we got rid of a system or because some configuration change rendered an account unnecessary. Whatever the type of account or the reason for getting rid of it, it is important to document the change so we no longer track that account for reviewing purposes.

A potential challenge with deprovisioning accounts is that it could leave orphaned resources. Suppose that Jonathan is the owner of project files on a shared folder to which no one else has access. If he leaves the company and has his account deprovisioned, that shared folder would be left on the server, but nobody (except administrators) could do anything with it. Apart from being wasteful, this situation could hinder the business if those files were important later on. When deprovisioning an account, therefore, it is important to transfer ownership of its resources to someone else.

Controlling Physical and Logical Access

We've focused our discussion so far on logical access control to our information, applications, and systems. The mechanisms we've covered can be implemented at various layers of a network and individual systems. Some controls are core components of operating systems or embedded into applications and devices, and some security controls require third-party add-on packages. Although different controls provide different functionality, they should all work together to keep the bad guys out and the good guys in and to provide the necessary quality of protection.

We also need to consider physical access controls to protect the devices on which these assets run as well as the facilities in which they reside. No organization wants people to be able to walk into their buildings arbitrarily, sit down at an employee's workstation, and be able to access any assets. While the mechanisms we've discussed in this chapter are mostly not applicable to physical security, most of the access control models (with the possible exception of risk-based access control) are.

Information Access Control

Controlling access to information assets is particularly tricky. Sure, we can implement the mechanisms described in the preceding sections to ensure only authorized subjects can read, write, or modify information in digital files. However, information can exist in a multitude of other places. Consider a fancy briefing room with a glass wall or doors leading into it. Inside, one of our staff members is giving a presentation on a product we'll soon be releasing that will turn our company's fortunes around. The slides are being projected, but the presenter provided hard copies for all attendees so they could take notes on them. Consider all the media on which our sensitive product information can exist:

- **Secondary storage** The slides exist in a file somewhere in the presenter's computer or in some network share (though this is probably the easiest medium to secure using the mechanisms discussed in this chapter).
- **Screen** Anyone walking by the room can see (or take a photo of) the slides being presented.
- **Handouts** If the attendees don't protect the hard copies of the slides or dispose of them improperly (say, by dumping them in a trash or recycling bin), the custodial staff (or anyone up for some dumpster diving) could get the information.
- **Voices** Not only can the presenter's remarks be overheard by unauthorized individuals but so can conversations that attendees have with each other in the break area.

The point is that our information access controls need to consider the variety of media and contexts in which sensitive information can exist. This is why it is important to consider both logical and physical controls for it. We've covered the logical side in depth here (though you may want to review Chapter 6 on data security); for a similar coverage of physical security, we refer you to Chapter 10's discussion of site and facility controls.

System and Application Access Control

Systems are a lot more closely aligned with the logical controls we described in this chapter, compared to the information assets we just covered. We kind of lumped all software together when we covered access controls. There is, however, a subtlety between systems and applications that bears pointing out, particularly with regard to the CISSP exam. Technically, a system is a type of software that provides services to other software. Web, e-mail, and authentication services are all examples of systems. An application is a type of software that interacts directly with a human user. A web browser, e-mail client, and even the authentication box that pops up to request your credentials are all examples of applications, as are stand-alone products like word processors and spreadsheets.

Access Control to Devices

All information that is stored in electronic media and all software exist within hardware devices. Whether it is the smartphone in your pocket, the laptop on your desk, or the server in the data center (or the cloud), you have to concern yourself with who can physically access them just as much as you worry about who can logically do so. After all, if an attacker can physically touch a device, then he can own it. We make the threat actors' jobs significantly more difficult by controlling physical access to our devices and assets. We can install and configure physical controls on each computer, such as install locks on the cover so the internal parts cannot be stolen, remove the USB and optical drives to prevent copying of confidential information, and implement a protection device that reduces the electrical emissions to thwart attempts to gather information through airwaves.

Speaking of electrical signals, different types of cabling can be used to carry information throughout a network. As a review of some of the cabling issues from Chapter 14, recall that some cable types have sheaths that protect the data from being affected by the electrical interference of other devices that emit electrical signals. Some types of cable have protection material around each individual wire to ensure there is no crosstalk between the different wires. Choosing the right kind of cable can help protect the devices they connect from accidental or environmental interferences. There is also, of course, the issue of deliberate tapping of these cables. If an adversary cannot get to a device but can tap its network cable, we could have issues unless all traffic is end-to-end encrypted (which it almost never is). Recall that distribution facilities (where one end of these cables terminates) need security controls and that some types of cables (UTP) are easier to tap than others (fiber optic).

Facilities Access Control

We discussed facility security in Chapter 10, but it is worthwhile to review and perhaps extend our conversation around access control. An example of facilities access controls is having a security guard verify individuals' identities prior to allowing them to enter a facility. How might the guard make that decision? In a classified facility, the guard may check that the clearance of the individual meets or exceeds that of the facility she is trying to enter and that she has a need to be there (perhaps from an access roster). This would be a simplified implementation of a MAC model. If the facility had different floors for

different departments, another approach would be to check the person's department and grant her access to that floor only, which would be a form of RBAC. We could refine that access control by putting some rules around it, such as access is only granted during working hours, unless the person is an executive or a manager. We would then be using an RB-RBAC model.

These examples are simply meant to illustrate that physical access controls, just like their logical counterparts, should be deliberately designed and implemented. To this end, the models we discussed at the beginning of this chapter are very helpful. They can then inform the specific controls we implement. In the sections that follow, we take a closer look at some of the major considerations when thinking of facilities access control.

Perimeter Security

Perimeter security is concerned with controlling physical access to facilities. How it is implemented depends upon the organization and the security requirements of that environment. One environment may require employees to be authorized by a security guard by showing a security badge that contains a picture identification before being allowed to enter a section. Another environment may require no authentication process and let anyone and everyone into different sections. Perimeter security can also encompass closed-circuit TVs that scan the parking lots and waiting areas, fences surrounding a building, the lighting of walkways and parking areas, motion detectors, sensors, alarms, and the location and visual appearance of a building. These are examples of perimeter security mechanisms that provide physical access control by providing protection for individuals, facilities, and the components within facilities.

Work Area Separation

Some environments might dictate that only particular individuals can access certain areas of the facility. For example, research companies might not want office personnel to be able to enter laboratories, so that they can't disrupt or taint experiments or access test data. Most network administrators allow only network staff in the server rooms and wiring closets to reduce the possibilities of errors or sabotage attempts. In financial institutions, only certain employees can enter the vaults or other restricted areas. These examples of work area separation are physical controls used to support access control and the overall security policy of the company.

Control Zone

An organization's facility should be split up into zones that are based on the sensitivity of the activity that takes place per zone. The front lobby could be considered a public area, the product development area could be considered top secret, and the executive offices could be considered secret. It does not matter what classifications are used, but it should be understood that some areas are more sensitive than others, which will require different access controls based on the needed protection level. The same is true of the organization's network. It should be segmented, and access controls should be chosen for each zone based on the criticality of devices and the sensitivity of data being processed.

Chapter Review

This is one of the more important chapters in the book for a variety of reasons. First, access control is central to security. The models and mechanisms discussed in this chapter are security controls you should know really well and be able to implement in your own organization. Also, the CISSP exam has been known to include lots of questions covering the topics discussed in this chapter, particularly the access control models.

We chose to start this chapter with these models because they set the foundations for the discussion. You may think that they are too theoretical to be useful in your daily job, but you might be surprised how often we've seen them crop up in the real world. They also inform the mechanisms we discussed in more detail, like OAuth, OpenID Connect, and Kerberos. While these technologies are focused on logical access control, we wrapped up the chapter with a section on how physical and logical controls need to work together to protect our organizations.

Quick Review

- An access control mechanism dictates how subjects access objects.
- The reference monitor is an abstract machine that mediates all access subjects have to objects, both to ensure that the subjects have the necessary access rights and to protect the objects from unauthorized access and destructive modification.
- There are six main access control models: discretionary, mandatory, role-based, rule-based, attribute-based, and risk-based.
- Discretionary access control (DAC) enables data owners to dictate what subjects have access to the files and resources they own.
- Access control lists are bound to objects and indicate what subjects can use them.
- The mandatory access control (MAC) model uses a security label system. Users have clearances, and resources have security labels that contain data classifications. MAC systems compare these two attributes to determine access control capabilities.
- The terms "security labels" and "sensitivity labels" can be used interchangeably.
- Role-based access control (RBAC) is based on the user's role and responsibilities (tasks) within the company.
- Rule-based RBAC (RB-RBAC) builds on RBAC by adding "if this, then that" (IFTTT) rules that further restrict access.
- Attribute-based access control (ABAC) is based on attributes of any component of the system. It is the most granular of the access control models.
- Risk-based access control estimates the risk associated with a particular request in real time and, if it doesn't exceed a given threshold, grants the subject access to the requested resource.
- Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form to allow for interoperability between various web-based technologies.

- The Service Provisioning Markup Language (SPML) allows for the automation of user management (account creation, amendments, revocation) and access entitlement configuration related to electronically published services across multiple provisioning systems.
- The Security Assertion Markup Language (SAML) allows for the exchange of authentication and authorization data to be shared between security domains.
- Extensible Access Control Markup Language (XACML), which is both a declarative access control policy language implemented in XML and a processing model, describes how to interpret security policies.
- OAuth is an open standard that allows a user to grant authority to some web resource, like a contacts database, to a third party.
- OpenID Connect is an authentication layer built on the OAuth 2.0 protocol that allows transparent authentication and authorization of client resource requests.
- Kerberos is a client/server authentication protocol based on symmetric key cryptography that can provide single sign-on (SSO) for distributed environments.
- The Key Distribution Center (KDC) is the most important component within a Kerberos environment because it holds all users' and services' secret keys, provides an authentication service, and securely distributes keys.
- Kerberos users receive a ticket granting ticket (TGT), which allows them to request access to resources through the ticket granting service (TGS), which in turn generates a new ticket with the session keys.
- The following are weaknesses of Kerberos: the KDC is a single point of failure; it is susceptible to password guessing; session and secret keys are locally stored; KDC needs to always be available; and management of secret keys is required.
- Some examples of remote access control technologies are RADIUS, TACACS+, and Diameter.
- The identity and access provisioning life cycle consists of provisioning, access control, compliance, configuration management, and deprovisioning.
- A system account is created by the operating system for use by a particular process, not by a human. A service account is a system account for a process that runs as a service (i.e., it listens for and responds to requests from other processes).
- Authorization creep takes place when a user gains too much access rights and permissions over time.
- Managed service accounts (MSAs) are Active Directory domain accounts that are used by services and provide automatic password management.

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. Which access control method is considered user directed?
   - **A.** Nondiscretionary
   - **B.** Mandatory
   - **C.** Identity-based
   - **D.** Discretionary
2. Which item is not part of a Kerberos authentication implementation?
   - **A.** Message authentication code
   - **B.** Ticket granting service
   - **C.** Authentication service
   - **D.** Users, applications, and services
3. If a company has a high turnover rate, which access control structure is best?
   - **A.** Role-based
   - **B.** Decentralized
   - **C.** Rule-based
   - **D.** Discretionary
4. In discretionary access control security, who has delegation authority to grant access to data?
   - **A.** User
   - **B.** Security officer
   - **C.** Security policy
   - **D.** Owner
5. Who or what determines if an organization is going to operate under a discretionary, mandatory, or nondiscretionary access control model?
   - **A.** Administrator
   - **B.** Security policy
   - **C.** Culture
   - **D.** Security levels
6. Which of the following best describes what role-based access control offers organizations in terms of reducing administrative burdens?
   - **A.** It allows entities closer to the resources to make decisions about who can and cannot access resources.
   - **B.** It provides a centralized approach for access control, which frees up department managers.

C. User membership in roles can be easily revoked and new ones established as job assignments dictate.

D. It enforces enterprise-wide security policies, standards, and guidelines.

Use the following scenario to answer Questions 7–9. Tanya is working with the company's internal software development team. Before a user of an application can access files located on the company's centralized server, the user must present a valid one-time password, which is generated through a challenge/response mechanism. The company needs to tighten access control for these files and reduce the number of users who can access each file. The company is looking to Tanya and her team for solutions to better protect the data that has been classified and deemed critical to the company's missions. Tanya has also been asked to implement a single sign-on technology for all internal users, but she does not have the budget to implement a public key infrastructure.

7. Which of the following best describes what is currently in place?

A. Capability-based access system

B. Synchronous tokens that generate one-time passwords

C. RADIUS

D. Kerberos

8. Which of the following is one of the easiest and best solutions Tanya can consider for proper data protection?

A. Implementation of mandatory access control

B. Implementation of access control lists

C. Implementation of digital signatures

D. Implementation of multilevel security

9. Which of the following is the best single sign-on technology for this situation?

A. PKI

B. Kerberos

C. RADIUS

D. TACACS+

Use the following scenario to answer Questions 10–12. Harry is overseeing a team that has to integrate various business services provided by different company departments into one web portal for both internal employees and external partners. His company has a diverse and heterogeneous environment with different types of systems providing customer relationship management, inventory control, e-mail, and help-desk ticketing

capabilities. His team needs to allow different users access to these different services in a secure manner.

10. Which of the following best describes the type of environment Harry's team needs to set up?
    - **A.** RADIUS
    - **B.** Service-oriented architecture
    - **C.** Public key infrastructure
    - **D.** Web services
11. Which of the following best describes the types of languages and/or protocols that Harry needs to ensure are implemented?
    - **A.** Security Assertion Markup Language, Extensible Access Control Markup Language, Service Provisioning Markup Language
    - **B.** Service Provisioning Markup Language, Simple Object Access Protocol, Extensible Access Control Markup Language
    - **C.** Extensible Access Control Markup Language, Security Assertion Markup Language, Simple Object Access Protocol
    - **D.** Service Provisioning Markup Language, Security Association Markup Language
12. The company's partners need to integrate compatible authentication functionality into their web portals to allow for interoperability across the different company boundaries. Which of the following will deal with this issue?
    - **A.** Service Provisioning Markup Language
    - **B.** Simple Object Access Protocol
    - **C.** Extensible Access Control Markup Language
    - **D.** Security Assertion Markup Language

Answers

1. **D.** The discretionary access control (DAC) model allows users, or data owners, the discretion of letting other users access their resources. DAC is implemented by ACLs, which the data owner can configure.
2. **A.** Message authentication code (MAC) is a cryptographic function and is not a key component of Kerberos. Kerberos is made up of a Key Distribution Center (KDC), a realm of principals (users, applications, services), an authentication service, tickets, and a ticket granting service.
3. **A.** A role-based structure is easier on the administrator because she only has to create one role, assign all of the necessary rights and permissions to that role, and plug a user into that role when needed. Otherwise, she would need to assign and extract permissions and rights on all systems as each individual joined the company and left the company.

4. **D.** Although user might seem to be the correct choice, only the data owner can decide who can access the resources she owns. She may or may not be a user. A user is not necessarily the owner of the resource. Only the actual owner of the resource can dictate what subjects can actually access the resource.
5. **B.** The security policy sets the tone for the whole security program. It dictates the level of risk that management and the company are willing to accept. This in turn dictates the type of controls and mechanisms to put in place to ensure this level of risk is not exceeded.
6. **C.** With role-based access control, an administrator does not need to revoke and reassign permissions to individual users as they change jobs. Instead, the administrator assigns permissions and rights to a role, and users are plugged into those roles.
7. **A.** A capability-based access control system means that the subject (user) has to present something, which outlines what it can access. The item can be a ticket, token, or key. A capability is tied to the subject for access control purposes. A synchronous token is not being used, because the scenario specifically states that a challenge\response mechanism is being used, which indicates an asynchronous token.
8. **B.** Systems that provide mandatory access control (MAC) and multilevel security are very specialized, require extensive administration, are expensive, and reduce user functionality. Implementing these types of systems is not the easiest approach out of the list. Since there is no budget for a PKI, digital signatures cannot be used because they require a PKI. In most environments, access control lists (ACLs) are in place and can be modified to provide tighter access control. ACLs are bound to objects and outline what operations specific subjects can carry out on them.
9. **B.** The scenario specifies that PKI cannot be used, so the first option is not correct. Kerberos is based upon symmetric cryptography; thus, it does not need a PKI. RADIUS and TACACS+ are remote centralized access control protocols.
10. **B.** A service-oriented architecture (SOA) will allow Harry's team to create a centralized web portal and offer the various services needed by internal and external entities.
11. **C.** The most appropriate languages and protocols for the purpose laid out in the scenario are Extensible Access Control Markup Language, Security Assertion Markup Language, and Simple Object Access Protocol. Harry's group is not necessarily overseeing account provisioning, so the Service Provisioning Markup Language is not necessary, and there is no language called "Security Association Markup Language."
12. **D.** Security Assertion Markup Language allows the exchange of authentication and authorization data to be shared between security domains. It is one of the most commonly used approaches to allow for single sign-on capabilities within a web-based environment.

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

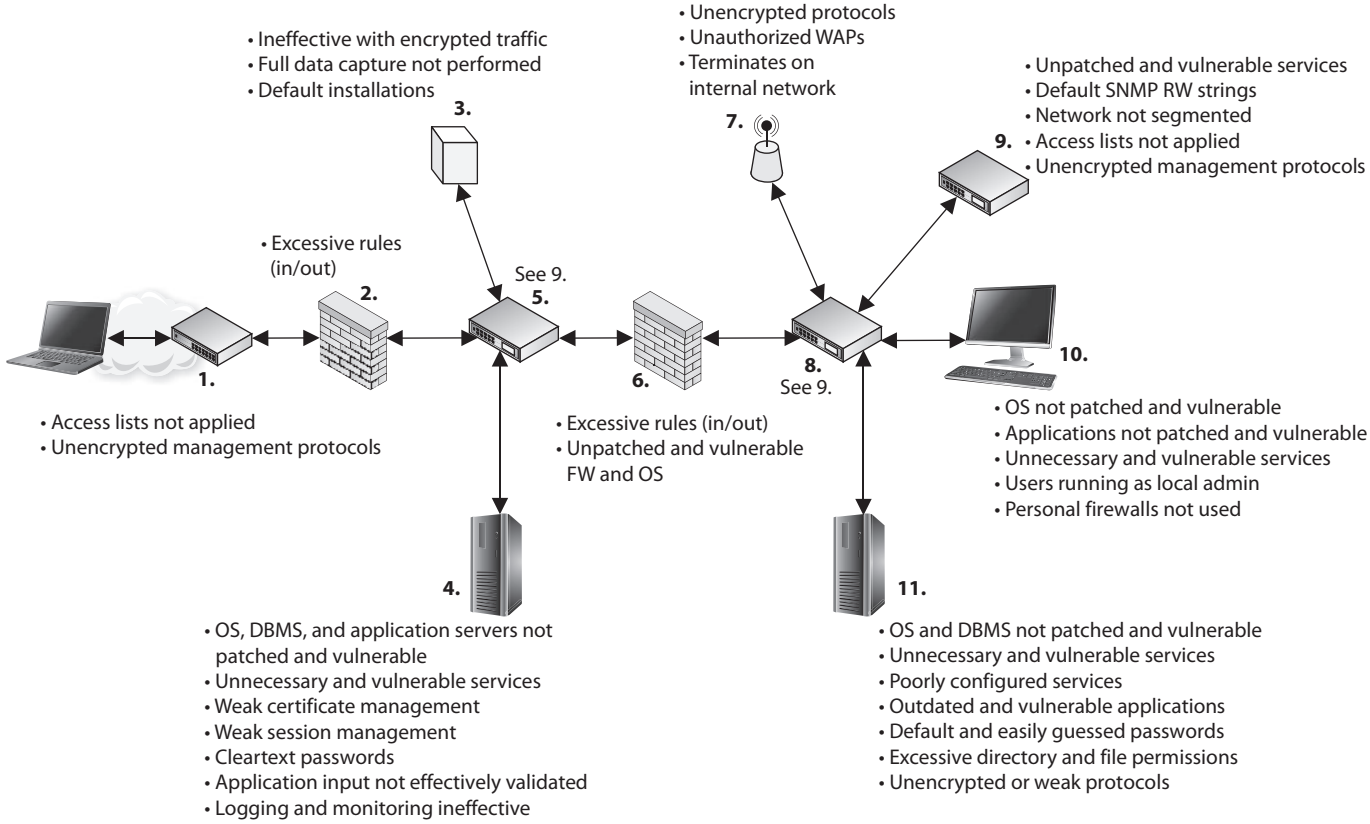


Figure 18-1 Vulnerabilities in heterogeneous networks

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.

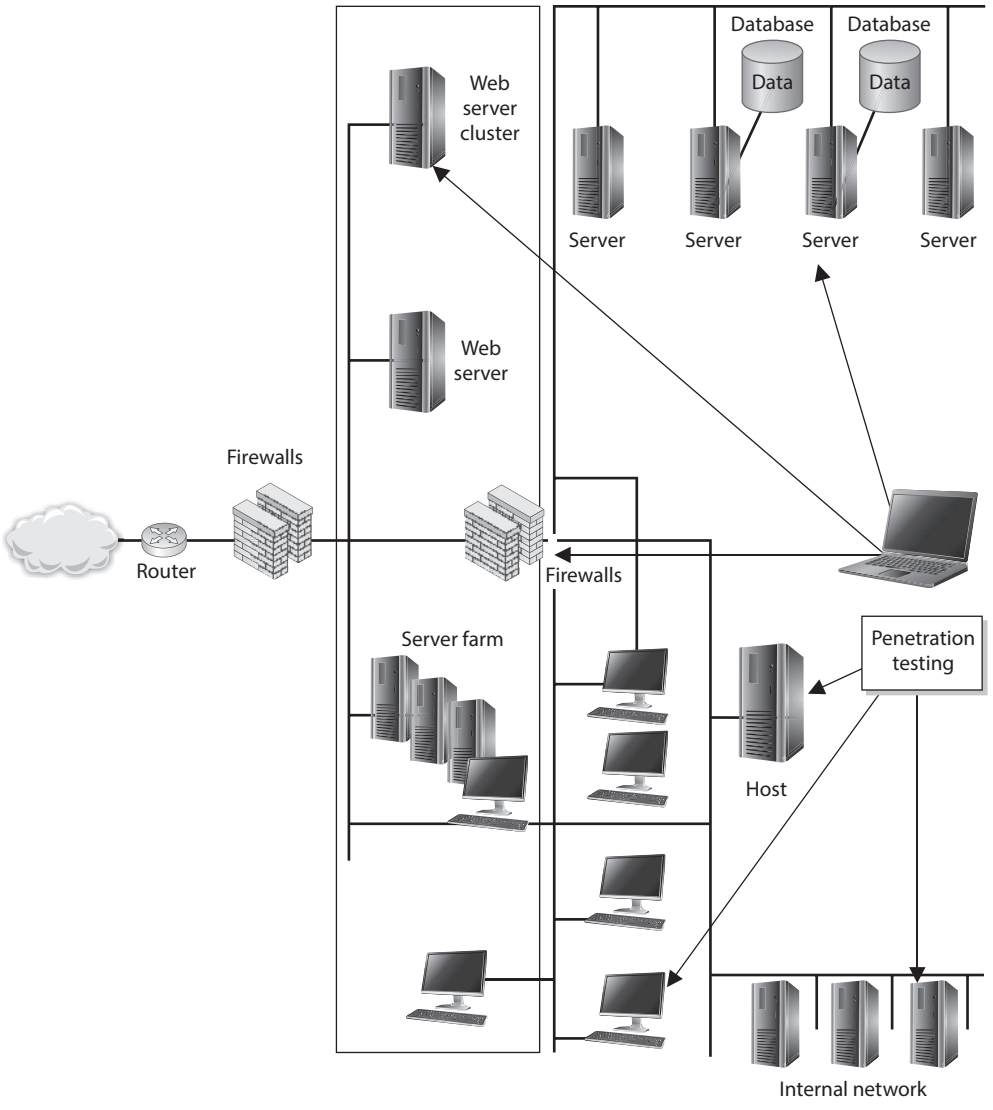


Figure 18-2 Penetration testing is used to prove an attacker can actually compromise systems.

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.

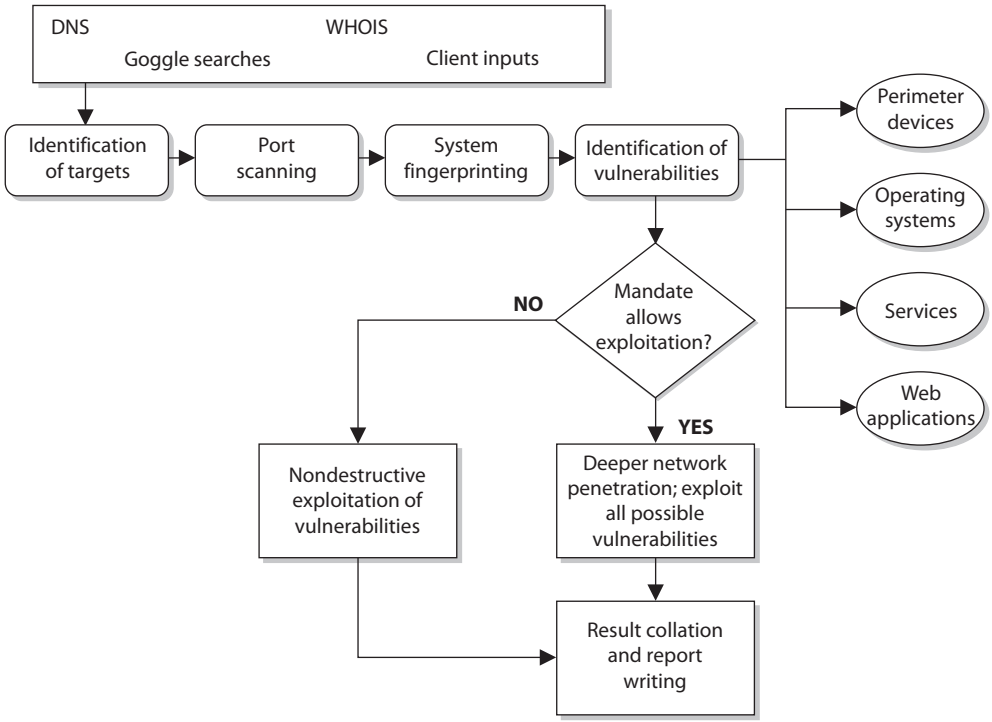
Security professionals should obtain an authorization letter that includes the extent of the testing authorized, and this letter or memo should be available to members of the team during the testing activity. This type of letter is commonly referred to as a "Get Out of Jail Free Card." Contact information 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



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

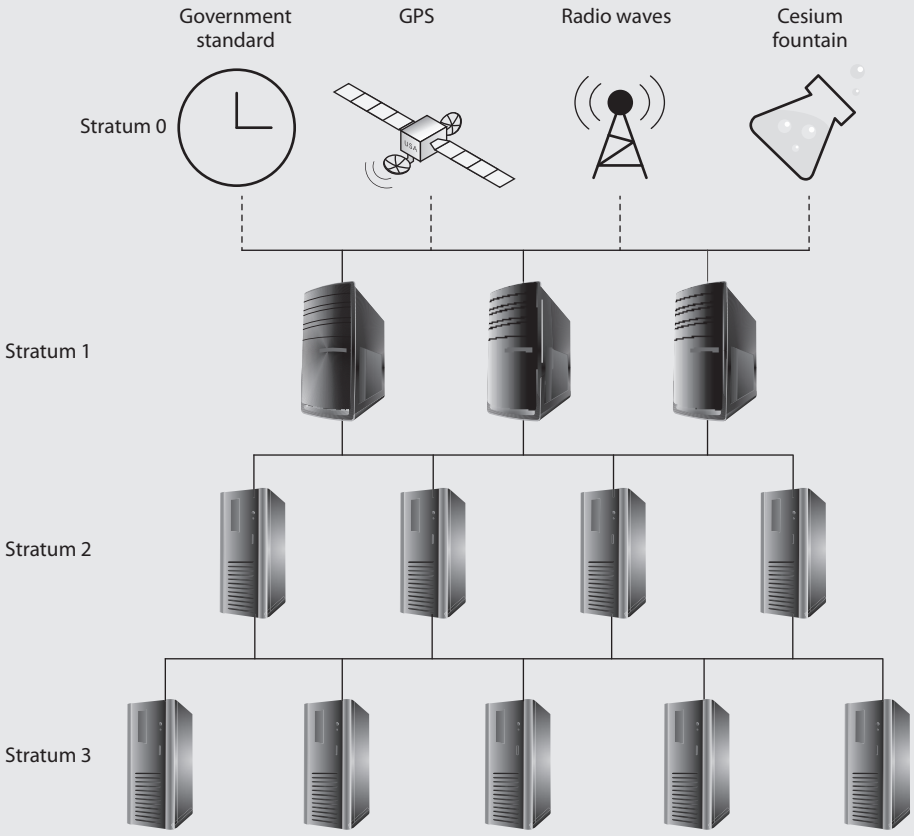
Network Time Protocol

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.

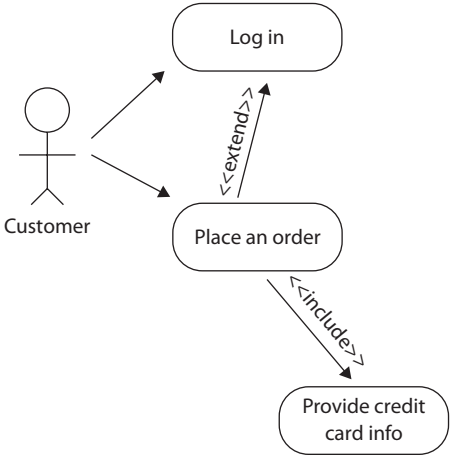


Figure 18-3 UML use case diagram

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.

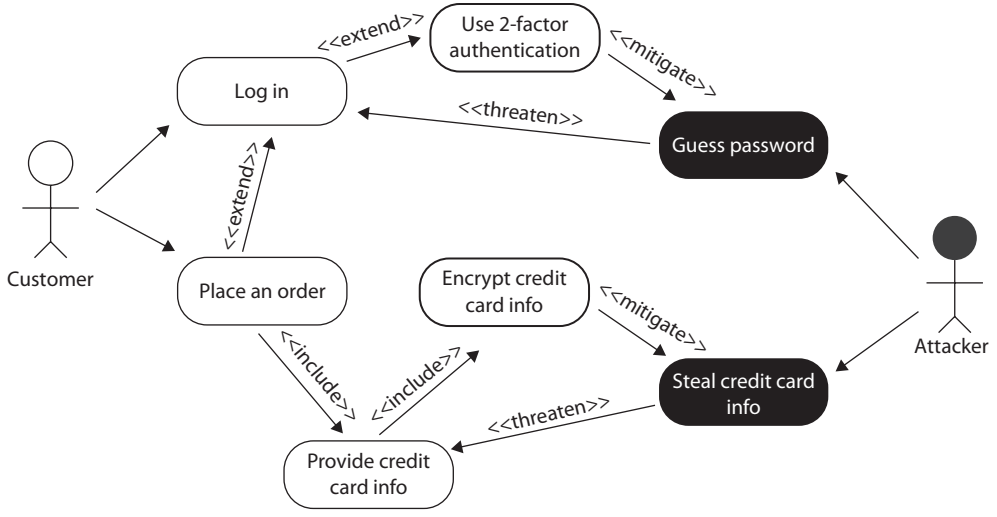


Figure 18-4 UML misuse case diagram

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