3

AUTHORIZATION AND ACCESS CONTROLS

After you've received a party's claim of identity and established whether that claim is valid, as discussed in Chapter 2, you have to decide whether to allow the party access to your resources. You can achieve this with two main concepts: authorization and access control. *Authorization* is the process of determining exactly what an authenticated party can do. You typically implement authorization using *access controls*, which are the tools and systems you use to deny or allow access.

You can base access controls on physical attributes, sets of rules, lists of individuals or systems, or other, more complex factors. When it comes to logical resources, you'll probably find simple access controls implemented in everyday applications and operating systems and elaborate, multilevel

configurations in military or government environments. In this chapter, you'll learn about access controls in more detail and look at some ways of implementing them.

What Are Access Controls?

Although the term *access controls* may sound technical, like it belongs only in high-security computing facilities, we all deal with access controls daily.

- When you lock or unlock the doors of your house, you're using a form of physical access control, based on your keys. (Your keys are something you have, as discussed in Chapter 2; in this case, they function as methods of both authentication and authorization.)
- When you start your car, you're also likely to use a key. For some newer cars, your key may even include an extra layer of security with radio-frequency identification (RFID) tags, which are certificate-like identifiers stored on the key.
- Upon reaching your place of employment, you might use a badge (again, something you have) to enter the building.
- When you sit down in front of your computer at work and enter your password (something you know), you're authenticating yourself and using a logical access control system to access the resources for which you've been given permission.

Most of us regularly encounter multiple implementations like these while working, going to school, and performing the other activities that make up our day.

You'll probably want to use access controls to carry out four basic tasks: allowing access, denying access, limiting access, and revoking access. We can describe most access control issues or situations using these four actions.

Allowing access is giving a party access to a given resource. For example, you might want to give a user access to a file, or you may want to give an entire group of people access to all the files in a given directory. You might also allow someone physical access to a resource by giving your employees a key or badge to your facility.

Denying access is the opposite of granting access. When you deny access, you are preventing a given party from accessing the resource in question. You might deny access to a person attempting to log onto a machine based on the time of day, or you might block unauthorized individuals from entering the lobby of your building beyond business hours. Many access control systems are set to deny by default.

Limiting access is allowing only some degree of access to your resources. In a physical security scheme, you might have a master key that can open any door in the building, an intermediate key that can open only a few doors, and a low-level key that can open only one door. You might also implement

limited access when you're using applications that may be exposed to attack-prone environments, like web browsers used on the internet.

One way to limit access is by running sensitive applications in *sandboxes*, which are isolated environments containing a set of resources for a given purpose (Figure 3-1).

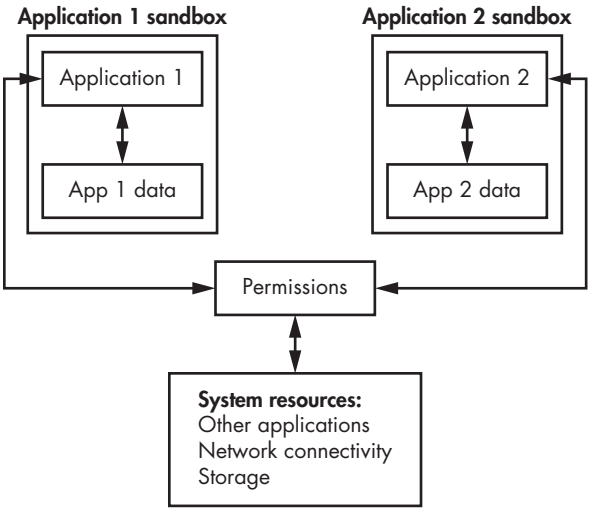

Figure 3-1: A sandbox is an isolated environment that protects a set of resources.

We use sandboxes to prevent their contents from accessing files, memory, and other system resources with which they shouldn't be interacting. Sandboxes can be useful for containing things that you can't trust, such as code from public websites. One example of a sandbox is the Java Virtual Machine (JVM) used to run programs written in the Java programming language. The JVM is specifically constructed to protect users against potentially malicious downloaded software.

Revoking access is taking access away from a party after you've granted it. Being able to revoke access is vital to the security of your system. If you were, for instance, to fire an employee, you'd want to revoke any accesses they might have, including access to their email account, your virtual private network (VPN), and your facility. When you're working with computer resources, it may be particularly important to be able to revoke access to a given resource quickly.

Implementing Access Controls

The two main methods of implementing access controls are with access control lists and capabilities. Both of these methods have strengths and weaknesses, as well as different ways of carrying out the four basic tasks we covered earlier.

Access Control Lists

Access control lists (ACLs), often pronounced "ackles," are lists containing information about what kind of access certain parties are allowed to have to a given system. We often see ACLs implemented as part of application software or operating systems and in the firmware of some hardware appliances, such as network infrastructure devices. We may even see ACL concepts extend into the physical world, through software systems that control physical resources, such as badge readers for door control systems. According to the ACL in Figure 3-2, Alice is allowed access to the resource, while Bob is specifically denied access.

Alice	Allow	✓
Bob	Deny	✗

Figure 3-2: A simple access control list

This may seem like a simple concept, but in larger implementations, ACLs can become quite complex. Organizations commonly use ACLs to control access in the file systems on which their operating systems run and to control the flow of traffic in the networks to which their systems are attached. You'll learn about these two types of ACLs in this chapter.

File System ACLs

The ACLs in most file systems will have three types of permissions (the authorizations that allow access to specific resources in a specific manner): *read*, which allows a user to access the contents of a file or directory; *write*, which allows a user to write to a file or directory; and *execute*, which allows a user to execute the contents of the file if that file contains either a program or a script capable of running on the system in question.

A file or directory may also have multiple ACLs attached to it. In UNIX-like operating systems, for instance, a given file might have separate access lists for specific users or groups. The system might give a certain individual user (like a specific developer) specific read, write, and execute permissions; a certain group of users (like the entire developer group) different read, write, and execute permissions; and any other authenticated users a third set of read, write, and execute permissions. On Linux-based operating systems, you can view these three sets of permissions by issuing the following command:

```
ls -la
```

Figure 3-3 shows these permissions displayed in the system.

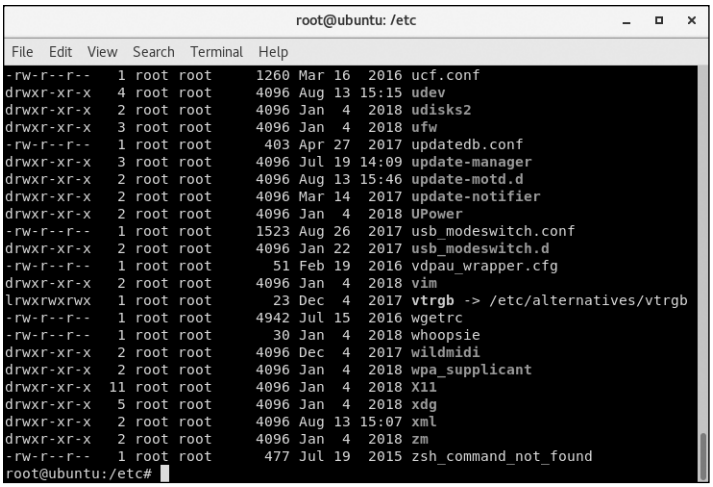

Figure 3-3: File permissions on a UNIX-like operating system

Each line in Figure 3-3 represents the permissions for an individual file. The permissions for the first file, *ucf.conf*, are displayed as follows:

```
- r w - r - - r - -
```

This may seem a bit cryptic. To interpret the permissions, it'll help to divide them into the following sections:

```
- | r w - | r - - | r - -
```

The first character generally represents the file type: `-` represents a regular file, and `d` represents a directory. The second segment represents the *user* who owns the file's permissions and is set to `r w -`, meaning that the user can read and write to the file but not execute it.

The third segment, the *group* permissions, is set to `r - -`, meaning that members of the group that was given ownership of the file can read it but not write or execute it. The last segment, *other*, is also set to `r - -`, meaning that anyone who is not the user who owns the file or in the group that owns the file can also read it but not write or execute it. In Linux, the user permissions apply to a single user only, and the group permissions apply to a single group.

By using sets of file permissions, you can control access to the operating systems and applications that use your file system. Most file systems use systems that are similar to the one described for assigning permissions.

Network ACLs

If you look at the variety of activities that take place on networks, both private and public, you'll notice ACLs regulating the activity. In network ACLs, you typically filter access based on identifiers used for network transactions,

such as Internet Protocol (IP) addresses, Media Access Control addresses, and ports. You can see such ACLs at work in network infrastructure such as routers, switches, and firewall devices, as well as in software firewalls, websites like Facebook and Google, email, and other forms of software.

Permissions in network ACLs tend to be binary in nature; rather than read, write, and execute, they generally either allow or deny some activity. Instead of users, network ACLs typically grant permissions to traffic. For example, when you set up the ACL, you use your chosen identifier or identifiers to dictate which traffic you're referring to and whether the traffic is allowed. It's best to rely on multiple identifiers to filter traffic, for reasons that will become clear shortly.

Media Access Control address filtering is one of the simplest forms of network-oriented ACLs. Media Access Control addresses are unique identifiers hard-coded into each network interface in a given system.

Unfortunately, the software settings in most operating systems can override a network interface's Media Access Control address. Changing this address is easy, so it's not a good choice for a unique identifier of a device on the network.

You could use *IP addresses* instead. Theoretically, an IP address is a unique address assigned to each device on any network that uses the Internet Protocol for communication. You can filter based on individual addresses or an entire range of IP addresses. For instance, you could allow the IP addresses 10.0.0.2 through 10.0.0.10 to pass traffic but deny any traffic from 10.0.0.11 and higher. Unfortunately, like Media Access Control addresses, you can falsify IP addresses, and they're not unique to a network interface. Additionally, IP addresses issued by internet service providers are subject to frequent change, so making IP addresses the sole basis for filtering is a shaky prospect at best.

BLACK HOLES

Some organizations, such as those that operate web servers, mail servers, and other services exposed to the internet, apply large-scale filtering to block out known attacks, spammers, and other undesirable traffic. Such filtering might include dropping traffic from individual IP addresses, ranges of IP addresses, or the entire IP spaces of large organizations, internet service providers, or even entire countries. This practice is commonly called *blackholing*, because from the user's perspective, any traffic sent to filtered destinations appears to have vanished into a black hole.

A third way of filtering traffic is by the *port* used to communicate over the network. The network port is a numerical designation for one side of a connection between two devices, and we use them to identify the application to which traffic should be routed. Many common services and applications

use specific ports. For instance, FTP uses ports 20 and 21 to transfer files, Internet Message Access Protocol (IMAP) uses port 143 for managing email, and Secure Shell (SSH) uses port 22 to manage remote connections to systems. There are many more examples, since there are 65,535 ports in all.

You can control the use of many applications over the network by allowing or denying traffic originating from or sent to any ports that you care to manage. However, like Media Access Control and IP addresses, the specific ports used for applications are conventions, not absolute rules. You can, with relative ease, change the ports that applications use to entirely different ones.

As you just saw, if you use any single attribute to construct a network ACL, you'll likely encounter a variety of issues. If you're using IP addresses, your attribute might not necessarily be unique. If you're using Media Access Control addresses, your attribute will be easy to alter, and if you use ports, you're banking on conventions rather than rules.

When you combine several attributes, you begin to arrive at a more secure technique. For example, it's common to use both an IP address and a port, a combination typically called a *socket*. Using sockets, you can allow or deny network traffic from one or more IP addresses with one or more applications on your network in a workable fashion.

You can also construct ACLs to filter based on a wide variety of other criteria. In some cases, you want to allow or deny traffic based on more specific information, such as the content of an individual packet or a related series of packets. Using such techniques, you could, for example, filter out traffic related to networks used to illegally share copyrighted material.

Weaknesses of ACL Systems

Systems that use ACLs to manage permissions are vulnerable to a type of attack called the *confused deputy problem*. This problem occurs when the software with access to a resource (the deputy) has a greater level of permission to access the resource than the user who is controlling the software. If you can trick the software into misusing its greater level of authority, you can potentially carry out an attack.[1]

Several attacks take practical advantage of the confused deputy problem. These often involve tricking the user into taking some action when they really think they are doing something else entirely. Many of these attacks are *client-side* attacks, which take advantage of weaknesses in applications running on the user's computer. These attacks might be code sent through the web browser and executed on the local machine, malformed PDF files, or images and videos with attack code embedded. In the past several years, software vendors have become increasingly aware of such attacks and have begun building defensive measures into their software, but new attacks appear on a regular basis. Two of the more common attacks that exploit the confused deputy problem are cross-site request forgery (CSRF) and clickjacking.

CSRF is an attack that misuses the authority of the browser on the user's computer. If the attacker knows of, or can guess, a website that has already authenticated the user—perhaps a common site such as Amazon.com—the

attacker can embed a link in a web page or HTML-based email, generally to an image hosted from a site controlled by the attacker. When the target's browser attempts to retrieve the image in the link, it also executes the additional commands the attacker has embedded in it, often in a fashion completely invisible to the target.

In the example in Figure 3-4, the attacker has embedded a request to transfer funds from an account at BankCo to the attacker's offshore account. As the BankCo server sees the request as coming from an authenticated and authorized user, it proceeds with the transfer. In this case, the confused deputy is the bank server.

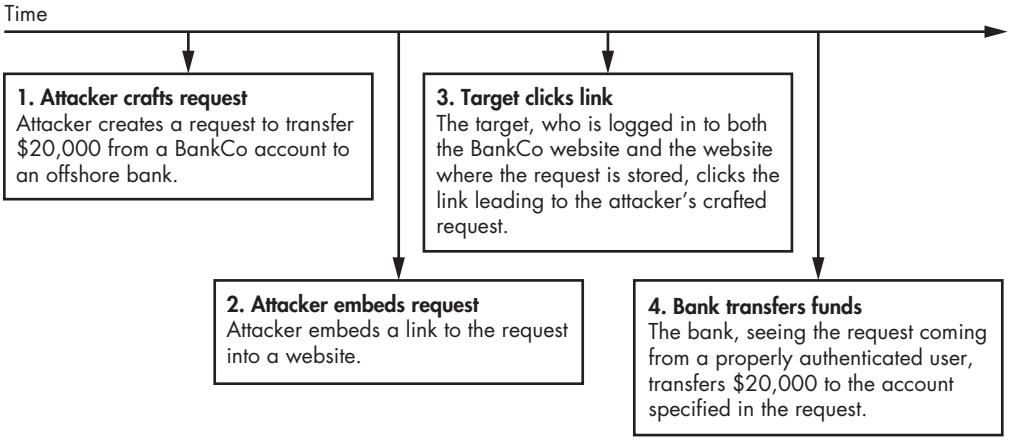

Figure 3-4: An example of a CSRF attack

Clickjacking, also known as *user interface redressing*, is a particularly sneaky and effective client-side attack that takes advantage of some of the page rendering features that are available in newer web browsers. To carry out a clickjacking attack, the attacker must legitimately control or have taken control of some portion of a website. The attacker constructs or modifies the site by placing an invisible layer over something the client would normally click. This causes the client to execute a command that's different than the one they think they're performing. You can use clickjacking to trick the client into making purchases, changing permissions in their applications or operating systems, or performing other unwanted activities.

Capabilities

Whereas ACLs define permissions based on a given resource, an identity, and a set of permissions, all generally held in a file of some sort, you can also define permissions based on a user's *token*, or key, otherwise known as a *capability*. Although the token isn't a physical object in most cases, you can think of it as the badge you might use to open the door of a building. The building has one door, and many people have a token that will open it, but

each person has a different level of access. One person might be able to access the building only during business hours on weekdays, while another person may have permission to enter the building at any time of day on any day of the week.

In capability-based systems, the right to access a resource is based entirely on possession of the token, rather than *who* possesses it. If you were to give your badge to someone else, he would be able to use it to access the building with whatever set of permissions you have. When it comes to logical assets, applications can share their token with other applications.

If you were to use capabilities instead of ACLs to manage permissions, you could protect against confused deputy attack. Neither of the attacks you learned about earlier, CSRF and clickjacking, would be possible, because the attacker wouldn't be able to misuse the authority of the user unless they had access to the user's token.

Access Control Models

An *access control model* is a way of determining who should be allowed access to what resources. There are quite a few different access control models out there. The most common ones, covered here, include discretionary access control, mandatory access control, rule-based access control, role-based access control, attribute-based access control, and multilevel access control.

Discretionary Access Control

In the *discretionary access control* (DAC) model, the owner of the resource determines who gets access to it and exactly what level of access they can have. You can see DAC implemented in most operating systems; if you decide to create a network share in a Microsoft operating system, for instance, you're in charge of people's access to it.

Mandatory Access Control

In the *mandatory access control* (MAC) model, the owner of the resource doesn't get to decide who gets to access it. Instead, a separate group or individual has the authority to set access to resources. You can often find MAC implemented in government organizations, where access to a given resource is largely dictated by the sensitivity label applied to it (secret or top secret, for example), by the level of sensitive information the individual is allowed to access (perhaps only secret), and by whether the individual actually has a need to access the resource (a concept called the *principle of least privilege*, discussed in the box).

THE PRINCIPLE OF LEAST PRIVILEGE

The principle of least privilege dictates that you should give a party only the bare minimum level of access it needs to perform its functionality. For example, someone working in an organization's sales department should not need access to data in the organization's internal human resources system to do their job. Violation of the principle of least privilege is at the heart of many of the security problems we face today.

One of the more common ways the principle of least privilege gets improperly implemented is in the permissions given to operating system user accounts. In Microsoft operating systems in particular, you'll often find that casual users, who are performing tasks such as creating documents in word processors and exchanging emails, are configured with administrative access, allowing them to carry out any task that the operating system allows.

Because of this, whenever the over-privileged user opens an email attachment containing malware or encounters a website that pushes attack code to the client computer, these attacks have free rein on the system. The attacker can simply turn off anti-malware tools, install any additional attack tools they care to, and proceed with completely compromising the system.

Rule-Based Access Control

Rule-based access control allows access according to a set of rules defined by the system administrator. If the rule is matched, access to the resource will be granted or denied accordingly.

A good example of rule-based access control is an ACL used by a router. You might see a rule specifying that traffic coming from source A to destination B on port C is allowed. Any other traffic between the two devices would be denied.

Role-Based Access Control

The *role-based access control* (RBAC) model allows access based on the role of the individual being granted access. For example, if you have an employee whose only role is to enter data into an application, RBAC would mandate that you allow the employee access to only that application.

If you have an employee with a more complex role—customer service for an online retailer, perhaps—the employee's role might require him to have access to information about customers' payment status and information, shipping status, previous orders, and returns. In this case, RBAC would grant him considerably more access. You can see RBAC implemented in many large-scale applications that are oriented around sales or customer service.

Attribute-Based Access Control

Attribute-based access control (ABAC) is based on the specific attributes of a person, resource, or environment. You can often find it implemented on infrastructure systems, such as those in network or telecommunications environments.

Subject attributes belong to an individual. We could choose any number of attributes, such as height in the classic "you must be this tall to ride" access control in amusement park rides. Another common example of subject attributes are *CAPTCHAs*, or "completely automated public Turing tests to tell humans and computers apart" (Figure 3-5).[2] CAPTCHAs control access based on whether the party on the other end can pass a test that is (in theory) too difficult for a machine to complete.

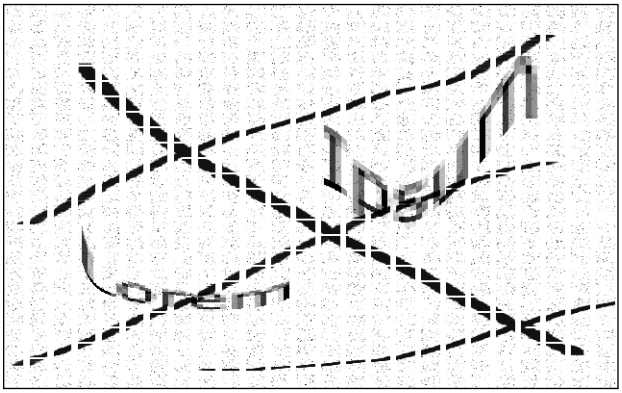

Figure 3-5: A CAPTCHA, designed to prove that the user is human

Resource attributes belong to a resource, such as an operating system or application. You'll often see access controlled by resource attributes, although usually this is for technical reasons rather than security reasons; some software runs only on a particular operating system, and some websites work only with certain browsers. You might apply this type of access control as a security measure by requiring someone to use specific software or protocols for communication.

You can use *environmental attributes* to enable access controls based on environmental conditions. People commonly use time to control access to physical and logical resources. Access controls on buildings often allow access only during business hours. Many VPN connections have time limits that force the user to reconnect every 24 hours to prevent users from keeping a connection running after their authorization for using it has been removed.

Multilevel Access Control

Multilevel access control models combine several of the access control models discussed in this section. They're used when the simpler access control models aren't considered robust enough to protect the information

to which you're controlling access. Military and government organizations, which handle data of a sensitive nature, often use multilevel access control models to control access to a variety of data, from nuclear secrets to protected health information. You'll learn about a few of these models now.

The Bell–LaPadula Model

The *Bell–LaPadula* model implements a combination of discretionary and mandatory access controls (DAC and MAC) and is primarily concerned with the confidentiality of the resource in question—in other words, making sure unauthorized people can't read it. Generally, in cases where you see these two models implemented together, MAC takes precedence over DAC, and DAC works within the accesses allowed by the MAC permissions.

For example, you might have a resource that is classified as secret and a user who has a secret level of clearance; under a mandatory access model, the user would have access to the resource. However, you might also have an additional layer of DAC under the MAC access so that if the resource owner has not given the user access, they would not be able to access it, despite the MAC permissions. In Bell–LaPadula, two security properties define how information can flow to and from the resource.[3]

The Simple Security Property The level of access granted to an individual must be at least as high as the classification of the resource in order for the individual to access it. In other words, an individual cannot read a resource classified at a higher level, but they can read resources at a lower level.

The * Property (or Star Property) Anyone accessing a resource can only write (or copy) its contents to another resource classified at the same level or higher.

You can summarize these properties as "no read up" and "no write down," respectively, as shown in Figure 3-6.

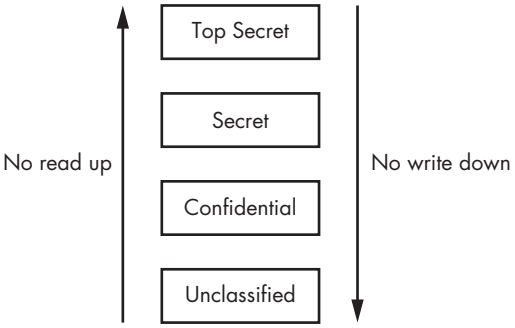

Figure 3-6: The Bell–LaPadula model

In short, this means that when you're handling classified information, you can't read any higher than your clearance level, and you can't write classified data down to any lower level.

The Biba Model

The *Biba* model of access control is primarily concerned with protecting the integrity of data, even at the expense of confidentiality. That means it's more important to keep people from altering the data than from viewing it. Biba has two security rules that are the exact opposite of those discussed in the Bell–LaPadula model.[4]

The Simple Integrity Axiom The level of access granted to an individual must be no lower than the classification of the resource. In other words, access to one level does not grant access to lower levels.

The * Integrity Axiom (or Star Integrity Axiom) Anyone accessing a resource can only write its contents to a resource classified at the same level or lower.

We can summarize these rules as "no read down" and "no write up," respectively, as shown in Figure 3-7. This means that assets that are of high integrity (meaning they shouldn't be altered) and assets that are of low integrity are kept strictly apart.

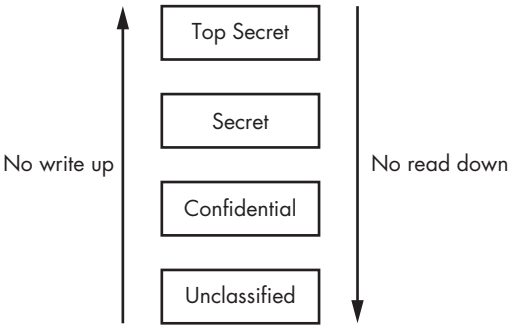

Figure 3-7: The Biba model

This may seem completely counterintuitive when it comes to protecting information. However, these principles protect integrity by ensuring that your resource can be written to only by those with a high level of access and that those with a high level of access do not access a resource with a lower classification. Consider an organization that performs both a low-integrity process that collects (potentially malicious) PDF uploads from users and a high-integrity process that scans document inputs from highly classified systems. In the Biba model, the upload process wouldn't be able to send data to the scanning process, so it wouldn't be able to corrupt the classified input; on top of this, the scanning process would be unable to access the low-level data, even if it was directed to.

The Brewer and Nash Model

The *Brewer and Nash* model, also known as the *Chinese Wall* model, is an access control model designed to prevent conflicts of interest. Brewer and Nash is

commonly used in industries that handle sensitive data, such as the financial, medical, or legal industries. This model considers three main resource classes.[5]

- *Objects*: Resources, such as files or information, pertaining to a single organization
- *Company groups*: All objects pertaining to an organization
- *Conflict classes*: All groups of objects concerning competing parties

A commercial law firm that represents companies in a certain industry might have files that pertain to various competing individuals and companies. Since an individual lawyer at the firm accesses files for different clients, the lawyer could potentially access confidential data that would generate a conflict of interest. In the Brewer and Nash model, the level of access to resources and case materials that the lawyer is allowed would dynamically change based on the materials previously accessed (Figure 3-8).

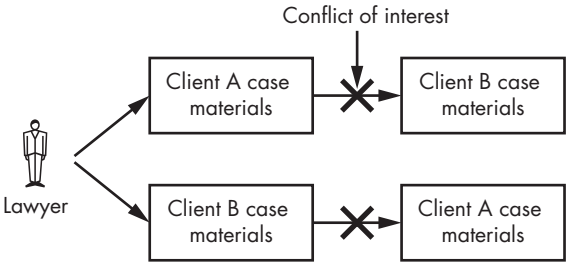

Figure 3-8: Brewer and Nash model

In this example, after the lawyer views Client A's case materials, the lawyer would no longer be able to access information pertaining to Client B or any other parties competing with the current client, resolving any conflicts of interest.

Physical Access Controls

So far you've seen logical examples to illustrate the access control concepts discussed in this chapter, but many of these methods apply to physical security, as well. Let's go over some examples of those now.

Physical access controls are often concerned with controlling the movement of individuals and vehicles. Access controls for individuals typically regulate their movement in and out of buildings or facilities, often using badges that open a facility's doors (something you have, from Chapter 2). Door control systems that make use of badges frequently use ACLs in the software that runs them to permit or deny access for certain doors and times of day.

One of the more common security issues with regulating people's access into buildings is *tailgating*, which occurs when you authenticate your physical access control measure, such as a badge, and another person follows directly

behind you without also being authenticated. Tailgating can cause a variety of issues, including creating an inaccurate representation of who is in the building in the case of emergencies.

We can attempt to solve tailgating in a variety of ways, including implementing a policy that forbids it, posting a guard in the area, or simply (but expensively) installing a physical access control solution that allows only one person to pass through at a time, such as a turnstile. All of these are reasonable solutions, but, depending on the environment in question, they may or may not be effective. You'll often find that a combination of several solutions works better than any single one.

A much more complex example of a physical access control is the security system in use at many airports. After the terrorist attacks of September 11, 2001, in the United States, the level of security at airports increased. Once you've entered the airport security system, you are required to present a boarding pass and identification (something you have, times two). You typically pass through several steps to ensure that you're not carrying any dangerous devices—a form of attribute-based access control. You then proceed to your gate and, once again, present your boarding pass before stepping on the airplane. Such processes may differ slightly depending on the country, but they're generally the same from an access control perspective.

Physical access control for vehicles often revolves around keeping said vehicles from moving through unauthorized areas, typically using various simple barriers, including Jersey barriers (Figure 3-9), bollards, one-way spike strips, and fences. You may also see more complex installations that include staffed or unstaffed rising barriers, automated gates or doors, and other similar controls.

Figure 3-9: A Jersey barrier

There are, of course, a huge number of other physical access controls and methods. Additionally, when referring to physical access control devices, or access controls in general, the line between an authentication device and an access control device often becomes rather blurry, or overlaps entirely. For example, a key for a physical lock could be considered identification,

authentication, and authorization, all the while being a component of a physical access control. Often these terms are used inaccurately or inappropriately, even within the security field, which does not help matters.

Summary

Authorization is a key step in the process of allowing parties to access resources—in other words, the identification, authentication, and authorization process. You implement authorization by using access controls. Typically, you use one of two access control methods: access control lists or capabilities. Although capabilities can provide safeguards against confused deputy attacks, they're not implemented as often as they should be.

When putting together an access control system, you use an access control model that outlines who should be given access to what resources. In our daily lives, we often encounter simpler access control models, such as discretionary access control, mandatory access control, role-based access control, and attribute-based access control. Environments that handle more sensitive data, such as those involved in the government, military, medical, or legal industry, typically use multilevel access control models, including Bell–LaPadula, Biba, and Brewer and Nash.

The next chapter will discuss auditing and accountability, which is how you keep track of the activities that have taken place after you've gone through the process of identification, authentication, and authorization.

Exercises

1. Discuss the difference between authorization and access control.
2. What does the Brewer and Nash model protect against?
3. Why does access control based on the Media Access Control address of the systems on our network not represent strong security?
4. Which should take place first, authorization or authentication?
5. What are the differences between the MAC and DAC models of access control?
6. The Bell–LaPadula and Biba multilevel access control models both have a primary security focus. Can these two models be used together?
7. If you have a file containing sensitive data on a Linux operating system, would setting the permissions to `rw-rw-rw-` cause a potential security issue? If so, which portions of the CIA triad might be affected?
8. Which access control model could you use to prevent users from logging into their accounts after business hours?
9. Explain how the confused deputy problem could allow users to carry out activities for which they are not authorized.
10. What are some of the differences between access control lists and capabilities?