can result from requiring a password as well as a user name and fingerprint scan. If this information is encrypted in transit, the system can be very resistant to spoofing or replay attack.

Multifactor authentication is better still. Consider how strong authentication can be with a USB device that must be plugged into the system, a PIN, and a fingerprint scan. Except for having to place one's finger on a pad and plug the USB into the system, this authentication method is no less convenient than that using normal passwords. Recall, though, that strong authentication by itself is not sufficient to guarantee the ID of the user. An authenticated session can still be hijacked if it is not encrypted.

16.6 Implementing Security Defenses

Just as there are myriad threats to system and network security, there are many security solutions. The solutions range from improved user education, through technology, to writing better software. Most security professionals subscribe to the theory of **defense in depth**, which states that more layers of defense are better than fewer layers. Of course, this theory applies to any kind of security. Consider the security of a house without a door lock, with a door lock, and with a lock and an alarm. In this section, we look at the major methods, tools, and techniques that can be used to improve resistance to threats. Note that some security-improving techniques are more properly part of protection than security and are covered in Chapter 17.

16.6.1 Security Policy

The first step toward improving the security of any aspect of computing is to have a **security policy**. Policies vary widely but generally include a statement of what is being secured. For example, a policy might state that all outside-accessible applications must have a code review before being deployed, or that users should not share their passwords, or that all connection points between a company and the outside must have port scans run every six months. Without a policy in place, it is impossible for users and administrators to know what is permissible, what is required, and what is not allowed. The policy is a road map to security, and if a site is trying to move from less secure to more secure, it needs a map to know how to get there.

Once the security policy is in place, the people it affects should know it well. It should be their guide. The policy should also be a **living document** that is reviewed and updated periodically to ensure that it is still pertinent and still followed.

16.6.2 Vulnerability Assessment

How can we determine whether a security policy has been correctly implemented? The best way is to execute a vulnerability assessment. Such assessments can cover broad ground, from social engineering through risk assessment to port scans. **Risk assessment**, for example, attempts to value the assets of the entity in question (a program, a management team, a system, or a facility) and determine the odds that a security incident will affect the entity and

decrease its value. When the odds of suffering a loss and the amount of the potential loss are known, a value can be placed on trying to secure the entity.

The core activity of most vulnerability assessments is a **penetration test**, in which the entity is scanned for known vulnerabilities. Because this book is concerned with operating systems and the software that runs on them, we concentrate on those aspects of vulnerability assessment.

Vulnerability scans typically are done at times when computer use is relatively low, to minimize their impact. When appropriate, they are done on test systems rather than production systems, because they can induce unhappy behavior from the target systems or network devices.

A scan within an individual system can check a variety of aspects of the system:

- Short or easy-to-guess passwords
- Unauthorized privileged programs, such as setuid programs
- Unauthorized programs in system directories
- Unexpectedly long-running processes
- Improper directory protections on user and system directories
- Improper protections on system data files, such as the password file, device files, or the operating-system kernel itself
- Dangerous entries in the program search path (for example, the Trojan horse discussed in Section 16.2.1), such as the current directory and any easily-written directories such as `/tmp`
- Changes to system programs detected with checksum values
- Unexpected or hidden network daemons

Any problems found by a security scan can be either fixed automatically or reported to the managers of the system.

Networked computers are much more susceptible to security attacks than are standalone systems. Rather than attacks from a known set of access points, such as directly connected terminals, we face attacks from an unknown and large set of access points—a potentially severe security problem. To a lesser extent, systems connected to telephone lines via modems are also more exposed.

In fact, the U.S. government considers a system to be only as secure as its most far-reaching connection. For instance, a top-secret system may be accessed only from within a building also considered top-secret. The system loses its top-secret rating if any form of communication can occur outside that environment. Some government facilities take extreme security precautions. The connectors that plug a terminal into the secure computer are locked in a safe in the office when the terminal is not in use. A person must have proper ID to gain access to the building and her office, must know a physical lock combination, and must know authentication information for the computer itself to gain access to the computer—an example of multifactor authentication.

Unfortunately for system administrators and computer-security professionals, it is frequently impossible to lock a machine in a room and disallow

all remote access. For instance, the Internet currently connects billions of computers and devices and has become a mission-critical, indispensable resource for many companies and individuals. If you consider the Internet a club, then, as in any club with millions of members, there are many good members and some bad members. The bad members have many tools they can use to attempt to gain access to the interconnected computers.

Vulnerability scans can be applied to networks to address some of the problems with network security. The scans search a network for ports that respond to a request. If services are enabled that should not be, access to them can be blocked, or they can be disabled. The scans then determine the details of the application listening on that port and try to determine if it has any known vulnerabilities. Testing those vulnerabilities can determine if the system is misconfigured or lacks needed patches.

Finally, though, consider the use of port scanners in the hands of an attacker rather than someone trying to improve security. These tools could help attackers find vulnerabilities to attack. (Fortunately, it is possible to detect port scans through anomaly detection, as we discuss next.) It is a general challenge to security that the same tools can be used for good and for harm. In fact, some people advocate **security through obscurity**, stating that no tools should be written to test security, because such tools can be used to find (and exploit) security holes. Others believe that this approach to security is not a valid one, pointing out, for example, that attackers could write their own tools. It seems reasonable that security through obscurity be considered one of the layers of security only so long as it is not the only layer. For example, a company could publish its entire network configuration, but keeping that information secret makes it harder for intruders to know what to attack. Even here, though, a company assuming that such information will remain a secret has a false sense of security.

16.6.3 Intrusion Prevention

Securing systems and facilities is intimately linked to intrusion detection and prevention. **Intrusion prevention**, as its name suggests, strives to detect attempted or successful intrusions into computer systems and to initiate appropriate responses to the intrusions. Intrusion prevention encompasses a wide array of techniques that vary on a number of axes, including the following:

- The time at which detection occurs. Detection can occur in real time (while the intrusion is occurring) or after the fact.
- The types of inputs examined to detect intrusive activity. These may include user-shell commands, process system calls, and network packet headers or contents. Some forms of intrusion might be detected only by correlating information from several such sources.
- The range of response capabilities. Simple forms of response include alerting an administrator to the potential intrusion or somehow halting the potentially intrusive activity—for example, killing a process engaged in such activity. In a sophisticated form of response, a system might transparently divert an intruder's activity to a **honeypot**—a false resource exposed

to the attacker. The resource appears real to the attacker and enables the system to monitor and gain information about the attack.

These degrees of freedom in the design space for detecting intrusions have yielded a wide range of solutions, known as **intrusion-prevention systems** (**IPS**). IPSs act as self-modifying firewalls, passing traffic unless an intrusion is detected (at which point that traffic is blocked).

But just what constitutes an intrusion? Defining a suitable specification of intrusion turns out to be quite difficult, and thus automatic IPSs today typically settle for one of two less ambitious approaches. In the first, called **signature-based detection**, system input or network traffic is examined for specific behavior patterns (or **signatures**) known to indicate attacks. A simple example of signature-based detection is scanning network packets for the string "`/etc/passwd`" targeted for a UNIX system. Another example is virus-detection software, which scans binaries or network packets for known viruses.

The second approach, typically called **anomaly detection**, attempts through various techniques to detect anomalous behavior within computer systems. Of course, not all anomalous system activity indicates an intrusion, but the presumption is that intrusions often induce anomalous behavior. An example of anomaly detection is monitoring system calls of a daemon process to detect whether the system-call behavior deviates from normal patterns, possibly indicating that a buffer overflow has been exploited in the daemon to corrupt its behavior. Another example is monitoring shell commands to detect anomalous commands for a given user or detecting an anomalous login time for a user, either of which may indicate that an attacker has succeeded in gaining access to that user's account.

Signature-based detection and anomaly detection can be viewed as two sides of the same coin. Signature-based detection attempts to characterize dangerous behaviors and to detect when one of these behaviors occurs, whereas anomaly detection attempts to characterize normal (or nondangerous) behaviors and to detect when something other than these behaviors occurs.

These different approaches yield IPSs with very different properties, however. In particular, anomaly detection can find previously unknown methods of intrusion (so-called **zero-day attacks**). Signature-based detection, in contrast, will identify only known attacks that can be codified in a recognizable pattern. Thus, new attacks that were not contemplated when the signatures were generated will evade signature-based detection. This problem is well known to vendors of virus-detection software, who must release new signatures with great frequency as new viruses are detected manually.

Anomaly detection is not necessarily superior to signature-based detection, however. Indeed, a significant challenge for systems that attempt anomaly detection is to benchmark "normal" system behavior accurately. If the system has already been penetrated when it is benchmarked, then the intrusive activity may be included in the "normal" benchmark. Even if the system is benchmarked cleanly, without influence from intrusive behavior, the benchmark must give a fairly complete picture of normal behavior. Otherwise, the number of **false positives** (false alarms) or, worse, **false negatives** (missed intrusions) will be excessive.

To illustrate the impact of even a marginally high rate of false alarms, consider an installation consisting of a hundred UNIX workstations from which

security-relevant events are recorded for purposes of intrusion detection. A small installation such as this could easily generate a million audit records per day. Only one or two might be worthy of an administrator's investigation. If we suppose, optimistically, that each actual attack is reflected in ten audit records, we can roughly compute the rate of occurrence of audit records reflecting truly intrusive activity as follows:

$$\frac{2\frac{\text{intrusions}}{\text{day}} \cdot 10\frac{\text{records}}{\text{intrusion}}}{10^6\frac{\text{records}}{\text{day}}} = 0.00002.$$

Interpreting this as a "probability of occurrence of intrusive records," we denote it as $P(I)$; that is, event I is the occurrence of a record reflecting truly intrusive behavior. Since $P(I) = 0.00002$, we also know that $P(\neg I) = 1 - P(I) = 0.99998$. Now we let A denote the raising of an alarm by an IDS. An accurate IDS should maximize both $P(I|A)$ and $P(\neg I|\neg A)$—that is, the probabilities that an alarm indicates an intrusion and that no alarm indicates no intrusion. Focusing on $P(I|A)$ for the moment, we can compute it using **Bayes' theorem**:

$$P(I|A) = \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)}$$

$$= \frac{0.00002 \cdot P(A|I)}{0.00002 \cdot P(A|I) + 0.99998 \cdot P(A|\neg I)}$$

Now consider the impact of the false-alarm rate $P(A|\neg I)$ on $P(I|A)$. Even with a very good true-alarm rate of $P(A|I) = 0.8$, a seemingly good false-alarm rate of $P(A|\neg I) = 0.0001$ yields $P(I|A) \approx 0.14$. That is, fewer than one in every seven alarms indicates a real intrusion! In systems where a security administrator investigates each alarm, a high rate of false alarms—called a "Christmas tree effect"—is exceedingly wasteful and will quickly teach the administrator to ignore alarms.

This example illustrates a general principle for IPSs: for usability, they must offer an extremely low false-alarm rate. Achieving a sufficiently low false-alarm rate is an especially serious challenge for anomaly-detection systems, as mentioned, because of the difficulties of adequately benchmarking normal system behavior. However, research continues to improve anomaly-detection techniques. Intrusion-detection software is evolving to implement signatures, anomaly algorithms, and other algorithms and to combine the results to arrive at a more accurate anomaly-detection rate.

16.6.4 Virus Protection

As we have seen, viruses can and do wreak havoc on systems. Protection from viruses thus is an important security concern. Antivirus programs are often used to provide this protection. Some of these programs are effective against only particular known viruses. They work by searching all the programs on a system for the specific pattern of instructions known to make up the virus.

When they find a known pattern, they remove the instructions, **disinfecting** the program. Antivirus programs may have catalogs of thousands of viruses for which they search.

Both viruses and antivirus software continue to become more sophisticated. Some viruses modify themselves as they infect other software to avoid the basic pattern-match approach of antivirus programs. Antivirus programs in turn now look for families of patterns rather than a single pattern to identify a virus. In fact, some antivirus programs implement a variety of detection algorithms. They can decompress compressed viruses before checking for a signature. Some also look for process anomalies. A process opening an executable file for writing is suspicious, for example, unless it is a compiler. Another popular technique is to run a program in a **sandbox** (Section 17.11.3), which is a controlled or emulated section of the system. The antivirus software analyzes the behavior of the code in the sandbox before letting it run unmonitored. Some antivirus programs also put up a complete shield rather than just scanning files within a file system. They search boot sectors, memory, inbound and outbound e-mail, files as they are downloaded, files on removable devices or media, and so on.

The best protection against computer viruses is prevention, or the practice of **safe computing**. Purchasing unopened software from vendors and avoiding free or pirated copies from public sources or disk exchange offer the safest route to preventing infection. However, even new copies of legitimate software applications are not immune to virus infection: in a few cases, disgruntled employees of a software company have infected the master copies of software programs to do economic harm to the company. Likewise, hardware devices can come from the factory pre-infected for your convenience. For macro viruses, one defense is to exchange Microsoft Word documents in an alternative file format called **rich text format** (**RTF**). Unlike the native Word format, RTF does not include the capability to attach macros.

Another defense is to avoid opening any e-mail attachments from unknown users. Unfortunately, history has shown that e-mail vulnerabilities appear as fast as they are fixed. For example, in 2000, the ***love bug*** virus became very widespread by traveling in e-mail messages that pretended to be love notes sent by friends of the receivers. Once a receiver opened the attached Visual Basic script, the virus propagated by sending itself to the first addresses in the receiver's e-mail contact list. Fortunately, except for clogging e-mail systems and users' inboxes, it was relatively harmless. It did, however, effectively negate the defensive strategy of opening attachments only from people known to the receiver. A more effective defense method is to avoid opening any e-mail attachment that contains executable code. Some companies now enforce this as policy by removing all incoming attachments to e-mail messages.

Another safeguard, although it does not prevent infection, does permit early detection. A user must begin by completely reformatting the hard disk, especially the boot sector, which is often targeted for viral attack. Only secure software is uploaded, and a signature of each program is taken via a secure message-digest computation. The resulting file name and associated message-digest list must then be kept free from unauthorized access. Periodically, or each time a program is run, the operating system recomputes the signature

and compares it with the signature on the original list; any differences serve as a warning of possible infection. This technique can be combined with others. For example, a high-overhead antivirus scan, such as a sandbox, can be used; and if a program passes the test, a signature can be created for it. If the signatures match the next time the program is run, it does not need to be virus-scanned again.

16.6.5 Auditing, Accounting, and Logging

Auditing, accounting, and logging can decrease system performance, but they are useful in several areas, including security. Logging can be general or specific. All system-call executions can be logged for analysis of program behavior (or misbehavior). More typically, suspicious events are logged. Authentication failures and authorization failures can tell us quite a lot about break-in attempts.

Accounting is another potential tool in a security administrator's kit. It can be used to find performance changes, which in turn can reveal security problems. One of the early UNIX computer break-ins was detected by Cliff Stoll when he was examining accounting logs and spotted an anomaly.

16.6.6 Firewalling to Protect Systems and Networks

We turn next to the question of how a trusted computer can be connected safely to an untrustworthy network. One solution is the use of a firewall to separate trusted and untrusted systems. A **firewal** is a computer, appliance, process, or router that sits between the trusted and the untrusted. A network firewall limits network access between the multiple **security domains** and monitors and logs all connections. It can also limit connections based on source or destination address, source or destination port, or direction of the connection. For instance, web servers use HTTP to communicate with web browsers. A firewall therefore may allow only HTTP to pass from all hosts outside the firewall to the web server within the firewall. The first worm, the Morris Internet worm, used the `finger` protocol to break into computers, so `finger` would not be allowed to pass, for example.

In fact, a network firewall can separate a network into multiple domains. A common implementation has the Internet as the untrusted domain; a semitrusted and semisecure network, called the **demilitarized zone** (**DMZ**), as another domain; and a company's computers as a third domain (Figure 16.10). Connections are allowed from the Internet to the DMZ computers and from the company computers to the Internet but are not allowed from the Internet or DMZ computers to the company computers. Optionally, controlled communications may be allowed between the DMZ and one company computer or more. For instance, a web server on the DMZ may need to query a database server on the corporate network. With a firewall, however, access is contained, and any DMZ systems that are broken into still are unable to access the company computers.

Of course, a firewall itself must be secure and attack-proof. Otherwise, its ability to secure connections can be compromised. Furthermore, firewalls do not prevent attacks that **tunnel**, or travel within protocols or connections

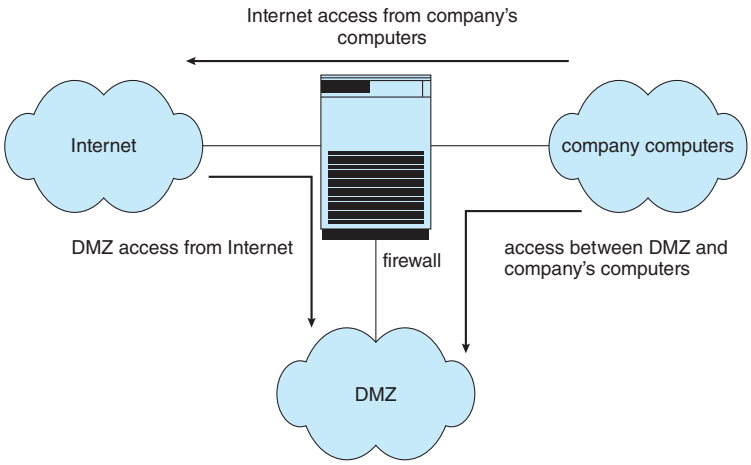

Figure 16.10 Domain separation via firewall.

that the firewall allows. A buffer-overflow attack to a web server will not be stopped by the firewall, for example, because the HTTP connection is allowed; it is the contents of the HTTP connection that house the attack. Likewise, denial-of-service attacks can affect firewalls as much as any other machines. Another vulnerability of firewalls is spoofing, in which an unauthorized host pretends to be an authorized host by meeting some authorization criterion. For example, if a firewall rule allows a connection from a host and identifies that host by its IP address, then another host could send packets using that same address and be allowed through the firewall.

In addition to the most common network firewalls, there are other, newer kinds of firewalls, each with its pros and cons. A **personal firewall** is a software layer either included with the operating system or added as an application. Rather than limiting communication between security domains, it limits communication to (and possibly from) a given host. A user could add a personal firewall to her PC so that a Trojan horse would be denied access to the network to which the PC is connected, for example. An **application proxy firewall** understands the protocols that applications speak across the network. For example, SMTP is used for mail transfer. An application proxy accepts a connection just as an SMTP server would and then initiates a connection to the original destination SMTP server. It can monitor the traffic as it forwards the message, watching for and disabling illegal commands, attempts to exploit bugs, and so on. Some firewalls are designed for one specific protocol. An **XML firewall**, for example, has the specific purpose of analyzing XML traffic and blocking disallowed or malformed XML. **System-call firewalls** sit between applications and the kernel, monitoring system-call execution. For example, in Solaris 10, the "least privilege" feature implements a list of more than fifty system calls that processes may or may not be allowed to make. A process that does not need to spawn other processes can have that ability taken away, for instance.