This chapter describes the types of security attacks that may take place in an internet. Security violations, such as the virus and the worm, are explained. The chapter concludes with a discussion of cookies, and how they can be both beneficial and harmful.

TYPES OF SECURITY PROBLEMS

What types of security problems should an organization protect against? Most security issues are associated with catchy names, such as a virus, a worm, and so on. This part of the book provides a review of these problems.

Denial of Service: Attacks and Counter-attacks

Several of the security violations explained in this chapter lead to a denial of service to the users of the attacked resource. The resource may be disabled by rogue code, or it may be simply saturated by an excessive

---

1Hereafter, the term Internet (upper case I) refers to the public Internet. The term internet (lower case i) refers to a private network.
workload as the attacker sends spurious traffic into the resource. This problem is called a denial of service attack. The service denial is to the legitimate users of the resource.

The counter-attack is to take preventive measures against the attack. In so doing, the system denies service to the attacker, for example, checking addresses in the incoming packets and discarding suspicious source addresses. This is called a denial of service counter-attack, and the service denial is to the attacker.

VIRUS

Typically, a virus is a piece of code that copies itself into a program, and executes when the program runs. It then may infect other programs. The infection may not occur immediately. It might not manifest itself until it is triggered by some kind of an event; as examples, a date, the detection of an event, such as a person's name being removed from a database, etc. A virus may also modify other programs. A graphical representation of the operation of a virus is shown in Figure 2–1.

The damage of a virus may only be irritating, such as the execution of a lot of superfluous code that degrades a system's performance. But a virus can do considerable damage. Indeed, some people define a virus as a program that causes the loss or contamination of data, or a program.

Figure 2–1  View of a virus operation
A virus may be difficult to detect or find. The virus may even get rid of itself at some point. That is, it may execute and then eradicate itself.

**WORM**

A worm is sometimes confused with a virus (see Figure 2–2). They have some similarities; the worm is code. However, it is an independent program that does not modify other programs, but reproduces itself over and over again until it slows down or shuts down a computer system or a network. One reason a worm is called a worm is because of two PARC researchers, John Schoch and Jon Hupp, who described a worm as code that existed in a machine. The worm segment on the machine can join or leave the computation.² A segment was likened to the segment of a worm—able to stay alive on its own.

CLOGGING OR FLOODING

Clogging or flooding is a form of worm. It entails sending a very large amount of bogus traffic to a node, such as a server or a router. The receiving node becomes clogged and is unable to service legitimate users, because of the excessive workload on the computing resources. Obviously, this fits the description of a denial of service attack, described earlier.

TROJAN HORSE

A Trojan horse is also a piece of code, and a worm or virus may be classified as a Trojan horse. It is so-named because it hides itself (inside another program) like the old story of the Greek soldiers. They hid inside a large hollow horse that was pulled by unwary Troy citizens into the city of Troy. Later, once inside the fortress of Troy, the soldiers came out of the horse and opened the city's gates to let in the rest of the soldiers.

In Figure 2–3, a piece of code is hidden inside a login program. A legitimate user logs on to a system in which the Trojan horse is hidden in the login program. The user's login IDs are intercepted by the Trojan horse and made available to the Trojan horse "soldier." Thereafter, the user's logon is compromised, and the Trojan horse interloper can use the ID to access the user's resources. It is possible that the Trojan horse may not be found because, after finding desired information, it exits the system and leaves no trace of itself.

BOMB

The bomb is yet another security-compromise instrument. It may take the form of a Trojan horse, and may do the harm of a virus or a worm. Its signature is that it is actuated by either a time trigger or a
logic trigger. The time trigger, introduced earlier, activates the bomb. One date-triggered bomb comes to mind; after a date is passed, the bomb prevents a program from executing further. The Y2K situation is an example of an inadvertent bomb, now diffused with the passing of midnight, December 31 to the year 2000.

The logic trigger is based on the bomb examining an event captured in the legitimate software’s normal execution. Once again, an example is the deletion of a record from a database. For example, an employee is dismissed and the employee record is removed from the personnel file. It so happens that this former employee was the programmer for the personnel system. So, the disgruntled programmer disables the system the programmer created. The trigger is a software routine that, upon detecting the absence of the rogue programmer’s record, initiates actions to damage the system.

TRAP DOOR

A trap door is a mechanism to get into a system. It occurs due to faulty security measures, allowing Mallory to penetrate a system. However, this “door” may be programmed in the code by the code’s programmer. I have found them useful in my past work because they allowed me to access a software system that had become a production system, but still needed my intervention on special occasions. In some applications, once the system is in production, its access may be restricted.

In an earlier part of my career, I was a partner in a communications consulting firm. One of our partners wrote the code for our accounting and billing system. Later, as this company devolved, bombs were inserted into the software. Thereafter, in subsequent entrepreneurial activities, I have either written critical code myself or established a means to make certain the code will not place me in jeopardy. The cliché, “Trust but verify,” is an aphorism that is applicable to arms control as well as information systems security.

In one of my programming jobs many years ago, I was the lead programmer for a Federal Reserve Board money supply simulation program. It was used for the Federal Reserve Open Market Committee (FOMC) operations. Obviously, this system and its database was very secret, since it was used as an information source to determine the creation or destruction of the US money supply. Since deadlines were critical and the FOMC often asked for some “tweaking” of the model, my trap door was an effective and productive tool to get into an otherwise secure system. Note to the Fed: I removed the trap door when I left the project.
As a good security practice, trap doors are removed from the code once it is debugged and given to the customer. If the trap doors are maintained (which is not uncommon), their entry should be very difficult. One approach is that the calculated or inadvertent entry into the trap door requires the entrant to go through another more discerning trap door, say one with encryption requirements. This operation is not as easy as it seems, since the code may be taken out of the hands of the original designer.

SALAMI

Another security violation merits our attention here. It is called the salami, and I ran across it during my programming days with the banking industry. It involves the small alteration of numbers in a file (a small piece of an eventual large salami). For example, the rounding-up or rounding-down of decimal places in a bank account, or the small, incremental shaving of a number in an inventory system to distort the goods in the inventory.

REPLAY VIOLATIONS

A replay violation is an active attack on a resource, in that it entails capturing data, perhaps modifying it, and resending it. An example of a replay attack is applying a transaction to a database more than one time, when it is supposed to a one-time application. The transaction, say to a financial accrual account, may or may not be altered, but the effect is to distort the accuracy of the accrual data, to the benefit of replay perpetrator, and to the detriment of the accrual account victim. Replay can be used in conjunction with the salami attack.

COOKIES

This part of the chapter describes cookies, and how they can create privacy problems. Parts of this general discussion are sourced from [SCHO99],5 and I recommend you read Mayer-Schönberger’s full paper.

A cookie is a piece of information sent by a Web server to store on a Web browser so it can later be read back from that browser. This practice saves time and reduces overhead by allowing the browser to store specific information about a session between a server and a client.

But this feature also allows a system that uses cookies to store information about the cookie’s user, and that is the crux of the debate today about cookies. Cookies are restricted as to what they can do in a user’s computer, so they are not considered dangerous from the standpoint of corrupting code or data. Of course, this statement is correct now; it may not be tomorrow.

The Web is built on a very simple, but powerful premise. All material on the Web is formatted in a uniform format called HTML (Hypertext Markup Language), and all information requests and responses conform to a standard protocol, the Hypertext Transfer Protocol (HTTP). This idea has become so popular and pervasive that other technologies are adapting the use of HTML and HTTP in order to be able to “gracefully” integrate with the Web. HTTP is used to transfer information between Web sites and clients, and emerging technologies such as voice over IP (VoIP), make use of HTML and HTTP.

Cookies are generated by a Web server and stored in the user’s computer, ready for future access. Cookies are embedded in the HTML/HTTP information flowing back and forth between the user’s computer and the servers.

In most cases, not only does the storage of personal information into a cookie go unnoticed, so does access to it. Web servers automatically gain access to relevant cookies whenever the user establishes a connection to them, usually in the form of Web requests.

Cookies operate with a two-stage process. First, the cookie is stored in the user’s computer. For example, with customizable Web search engines like My Yahoo!, a user selects categories of interest from the Web page. The Web server then creates a specific cookie, which is a tagged string of text containing the user’s preferences, and it transmits this cookie to the user’s computer. The user’s Web browser, if so-configured, receives the cookie and stores it in a special file called a cookie list. Once again, these operations occur without any notification to the user, or user consent. As a result, personal information (in this case, the user’s category preferences) is formatted by the Web server, transmitted, and saved by the user’s computer.

During the second stage, the cookie is automatically transferred from the user’s machine to a Web server through HTTP, without the user’s knowledge. Whenever a user directs the Web browser to display a
certain Web page from the server, the browser will, without the user’s knowledge, transmit the cookie containing personal information to the Web server.

For example, when you enter your password and user ID, say to access your account at a stock brokerage, a cookie stores your preferences for your browsing of the site, say which report you access at the site. Another example is the ability to customize what you see at a site. Perhaps you don’t want to be bothered with the weather news, and so forth. Cookies can be used to filter what you see or do not see. A cookie can also be stored for a long time, such that if you log on to a site after a long absence, that site might still have information about you (say credit information), thus reducing your hassle of inputting information again.

All these operations are mutually beneficial to the clients and their browsers. However, there is a downside to all this information collection. The way cookies operate, an HTTP cookie can be used to track where an Internet user browses, considered by some (count me in) as an invasion of privacy. Of course, any intelligent protocol analysis package (a sniffer) can track your activities through IP addresses, and domain names, but tracking your Web activities is much easier to do with cookies.

[COOK00] provides the following guidance of getting rid of cookies:

If you want to disallow cookies you can do so with version 3.0 or greater of Netscape. Go to the Options Menu. Select the Network Preferences Menu Item. From the window that appears select Protocols. Locate the section Show an Alert Before. Check the box labeled Accepting a Cookie. From now on you will get an Alert box telling you that a server is trying to set a cookie at your browser. It will tell you what the cookie value is and how long it will last before it is deleted.

Box 2–1 provides guidance on setting up cookies in a browser, and you can perform reverse operations to forbid their use.

APPLETS AND SANDBOXES

During the past few years, Java has become a popular programming language used on many Internet applications. It is the language of choice for writing small pieces of code, called applets. The applets are designed to be downline loaded to a user computer from another machine, such as

---

6[COOK00]. CookieCentral has some excellent material on its Web site about cookies. Go to www.cookiecentral.com.
OTHER PROBLEMS

Box 2–1 Enabling Cookies (do the opposite to disable them)

To enable cookies in **Netscape Navigator 3.x**:
– Click on the “Options” menu and select Network/Preferences.
– Uncheck the box that says, “Accepting Cookies.”

To enable cookies in **Netscape Navigator 4.x**:
– Click on the “Edit” menu and select Preferences/Advanced.
– Click on the radio button that says, “Accept all cookies.”
– Uncheck the box that says, “Warn me before accepting a cookie” to disable the warning, if you choose.

To enable cookies in **Internet Explorer 3.x**:
– Click on the “View” menu and select Options/Advanced.
– Uncheck the box that says, “Warn me before selecting a cookie,” if you choose.
– Click “Apply” and then “OK.”

To enable cookies in **Internet Explorer 4.x**:
– Click on the “View” and select Options/Advanced.
– Select “Accept all cookies.”
– Click “Apply” and then “OK.”

To enable cookies in **Internet Explorer 5.x**:
– Click on “Tools” and select Internet Options/Security.
– Click on “Custom Level.”
– Select “Enable” from both “Cookies” entries.
– Click “OK” and then “Apply.”

a server. The operation provides opportunities for security breaches; a tamperer could potentially design the applet to interfere with the user’s computer system, doing damage to files and programs.

To combat against this situation, Sun Microsystems has developed software called the Java Virtual Machine. Any applet that is to run on a machine must rely on this software. When the applet is downloaded, it initially is not allowed to access the computer’s vital resources, such as hard drives, device drivers, etc. This part of the operation is known as the Java sandbox, something like a child that is placed in a sandbox: safe, and at the same time guarded from pillaging the surroundings outside the sandbox.

The applet is allowed to move from the sandbox and access the computer’s resources, but only if the virtual machine verifies that the applet came from a known and trusted party.

The Java Virtual Machine is embedded in the user’s browser software, and allows the applet to reside in RAM, but limits its access to interfaces.
The virtual machine performs two major functions: (a) first, it checks the code of the applet to make sure that the code is proper (legitimate), (b) second, it examines a Digital Signature that is attached to the applet, which will reveal the source of the program, and if a third party has changed the applet.

If these checks are passed, the applet is allowed to access the user computer’s interfaces (for example, the disk drives). The virtual machine can determine how much access the applet is to be given as well; perhaps only part of a disk can be accessed.

If the checks are not passed, the applet may still execute, depending on what it is designed to do. But it must remain in the sandbox.

OTHER PROBLEMS

There are other forms of security problems, and many are variations of the systems described here. Many commercial software applications have measures to protect themselves against these intruders. But others do not. To be safe, it is a good idea to review the features of any code that is placed on a system critical to the enterprise. If you do not have a security mechanism, and you are on the Internet, it is only a matter of time until your system is penetrated, perhaps with unfortunate consequences.

SUMMARY

The intent of security attacks in a computer network is the denial of service to the users of a resource, such as software, data, or even a machine. Most attacks take the form of a virus or worm, with variations such as the Trojan horse. Cookies are making the news, because of their potential to compromise a user’s privacy, but they are not intrusive into a user’s automated resources.