6

Trojan Horses

You might have read the last chapter on backdoors and thought to yourself, "I'd never run a program named Netcat or VNC on my machine, so I'm safe!" Unfortunately, it isn't that easy. Attackers with any modest level of skill will disguise the nasty backdoors we covered in the last chapter or hide them inside of other programs. That's the whole idea of a Trojan horse, which we define as follows:

A Trojan horse is a program that appears to have some useful or benign purpose, but really masks some hidden malicious functionality.

As you might expect, Trojan horses are called *Trojans* for short, and the verb referring to the act of planting a Trojan horse is *to Trojanize* or even simply *to Trojan*. If you recall your ancient Greek history, you'll remember that the original Trojan horse allowed an army to sneak right through a highly fortified gate. Amazingly, the attacking army hid inside a giant wooden horse offered as a gift to the unsuspecting victims. It worked like a charm. In a similar fashion, today's Trojan horses try to sneak past computer security fortifications, such as firewalls, by employing like-minded trickery. By looking like normal, happy software, Trojan horse programs are used for the following goals:

- Duping a user or system administrator into installing the Trojan horse in the first place. In this case, the Trojan horse and the unsuspecting user become the entry vehicle for the malicious software on the system.
- Blending in with the "normal" programs running on a machine. The Trojan horse camouflages itself to appear to belong on the

system so users and administrators blithely continue their activity, unaware of the malicious code's presence.

Many people often incorrectly refer to any program that gives remote control of or a remote command shell on a victim machine as a Trojan horse. This notion is mistaken. I've seen people label the VNC and Netcat tools we covered in the last chapter as Trojan horses. However, although these tools can be used as backdoors, by themselves they are not Trojan horses. If a program merely gives remote access, it is just a backdoor, as we discussed in Chapter 5. On the other hand, if the attacker works to *disguise* these backdoor capabilities as some other benign program, then we are dealing with a true Trojan horse.

Attackers have devised a myriad of methods for hiding malicious capabilities inside their wares on your computer. These techniques include employing simple, yet highly effective naming games, using executable wrappers, attacking software distribution sites, manipulating source code, co-opting software installed on your system, and even disguising items using polymorphic coding techniques. As we discuss each of these elements throughout this chapter, remember the attackers' main goal: to disguise their malicious code so that users of the system and other programs running on the machine do not realize what the attacker is up to.

In this chapter, we'll discuss both widely used and cutting-edge techniques. Keep in mind, however, that attackers are a creative and devious lot. They use the concepts we'll cover, but tweak them in innumerable ways to achieve maximum subterfuge.

What's in a Name?

'Tis but thy name that is my enemy.

—William Shakespeare, *Romeo and Juliet,* 1595

At the very simplest level of Trojan horse techniques, an attacker might merely alter the name of malicious code on a system so that it appears to belong on that machine. By giving a backdoor program the same name of some other program you'd normally expect to be on your system, an attacker might be able to operate undetected. After all, only the lamest of attackers would run malicious code using the well-known name of that code, such as Netcat or VNC. Don't get me wrong, however. If a really dim-witted bad guy attacks my system and uses tech-
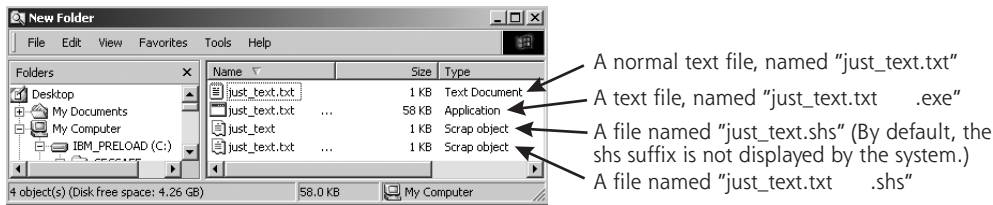
niques that I can easily spot, I'm all for it. That makes my job easier. I'm perfectly happy to catch any attacker when he or she makes a mistake of that magnitude, and, thankfully, I have found several instances of attackers calling a backdoor Netcat or even VNC. However, we can't expect all of our adversaries to make such trivial errors, so let's investigate their naming games in more detail.

Playing with Windows Suffixes

One very simple Trojan horse naming technique used by attackers against Windows systems is to trick victims by creating a file name with a bunch of spaces in it to obscure the file's type. As you no doubt know, the three-letter suffix (also known as an "extension") of a file name in Windows is supposed to indicate the file's type and which application should be used to view that file. For example, executables have the .EXE suffix, whereas text files end in .TXT. The information security business has done a good job over the last decade of informing our users not to run executable attachments included in e-mail or those that appear on their hard drive. "Unknown EXE files cause trouble," we lecture our users, with furled eyebrows and a deep voice to emphasize the importance of this lesson. So, given users' fright and awe in the presence of EXE files, how could a malicious executable program be disguised as something benign, such as a simple text file? An attacker could confuse a victim by naming a file with a bunch of spaces before its real suffix, like this:

```
just_text.txt                    .exe
```

That .EXE at the end of the name after all of the spaces makes the program executable, but the unwary user might not notice the .EXE suffix. If users look at such a file with the Windows Explorer file viewer, it'll appear that the file might just be text, as shown in Figure 6.1. For comparison to a benign file, the first line in Figure 6.1 shows a normal text file, with a normal text file icon and a file type of Text Document. Most users would have no qualms about double-clicking such a nice-looking, happy file. The second line, however, is far more evil. It shows an executable file with a name of "just_text.txt   .exe". Note that the display shows the name of the file as just_text.txt followed by "…". Those innocent-looking dots mean that the file name is actually longer than what is displayed.

A normal text file, named "just_text.txt"

A text file, named "just_text.txt   .exe"

A file named "just_text.shs" (By default, the shs suffix is not displayed by the system.)

A file named "just_text.txt   .shs"

Figure 6.1
Hiding the EXE extension after several spaces.

Of course the Explorer file viewer shows the second file's type as Application and displays an executable's icon next to the name instead of a text file icon. Still, the vast majority of users would never notice these somewhat subtle distinctions. If this is a huge concern for the attackers, they could even configure the system so that an executable program type's icon actually appears as a .TXT icon. This can be accomplished by altering the icon using one of a variety of tools, such as the free E-Icons program available at *www.deepgls.com/eicons/.* Alternatively, an attacker could choose a file type that is both executable *and* has an icon that looks quite similar to a text file, such as the Shell Scrap Object file type, with an .SHS extension. These .SHS files are used to bundle together commonly copied and pasted text and pictures, as well as commands, for various Windows programs. The third line of Figure 6.1 shows a typical .SHS file. The fourth line of the figure shows a combination of these techniques: a .SHS file is given a name of "just_text.txt   .shs", which includes several spaces to make it appear as a .TXT file. You can easily see how a user could get duped into executing this type of file.

Numerous file suffixes could be used to deliver and contain malicious code on a target machine. Table 6.1 shows the different file types developers use to hold binary, scripts, and other types of executable code. Many, but certainly not all, of these script types are tied to Windows machines, as the Windows operating system is freakishly obsessed with a file's type being stored in the suffix. However, the phenomenon is not limited to Windows. On UNIX systems, some program types are also indicated with a suffix, including .sh, .pl, and .rpm files. It's important to note, however, that UNIX doesn't put any special meaning into a file's suffix, unlike Windows. In Windows, the operating system uses the suffix to determine which application to use when opening a document. On UNIX machines, this suffix is just a handy reference for users; UNIX won't run a specific application based merely on the file

suffix. Still, any one of these file types in Table 6.1 could be abused to spread malicious code. For a detailed description of any type of file suffix, you can refer to the very handy Filext Web site, at *http://filext.com.*

Table 6.1
Useful File Extensions to Filter at an Internet Gateway

File Extension	Purpose of This Type of File
.API	Acrobat Plug-in, for extending the capabilities of Adobe's Acrobat file viewing tool.
.BAT	Batch processing file, used to execute a series of contained commands in sequential order.
.BPL	Borland package libraries, containing chunks of shared code used in programs developed within the Delphi software language and environment.
.CHM	Compiled HTML Help file, which could include a link that would download and execute malicious code on a victim machine.
.COM	Command file, containing scripts or even executables for DOS and Windows systems.
.CPL	Windows Control Panel Extension, allowing new capabilities in your previously dull and monotonous control panels.
.DLL	Dynamic Link Library, executable code that is shared by other programs on the system.
.DPL	Delphi Package Library, used to add bundled together shared libraries of code developed in the Delphi programming environment.
.DRV	Device driver, used to extend the hardware support of a Windows machine, but could be abused to modify the kernel and completely control the victim machine.
.EXE	Windows binary executable program.
.HTA	HyperText application, a file that can run applications from an HTML document.
.JS	JavaScript, a scripting language that can be embedded in HTML or run through any JavaScript interpreter, including the Windows Scripting Host built into most Windows systems.
.OCX	Object Linking and Embedding (OLE) control, used to orchestrate the interaction of several programs on a Windows machine.

Table 6.1
Useful File Extensions to Filter at an Internet Gateway (Continued)

File Extension	Purpose of This Type of File
.PIF	Program Information File, used to tell Windows how to run a non-Windows application.
.pl	Perl script, a powerful, high-level scripting language supported on most UNIX systems and some Windows machines.
.SCR	Screen saver program, which includes binary executable code.
.SHS	Shell Scrap Object file, a format used to hold frequently repeated commands, text, and pictures for Windows programs.
.SYS	System configuration file, normally used to establish system settings, but could be used by an attacker to reconfigure a victim machine.
.VBE	VBScript Encoded Script file, used to carry Visual Basic Scripts.
.VBS	Visual Basic Script, a scripting language built into many Windows machines.
.VXD	Virtual device driver, a device driver with direct access into a Windows kernel.
.WMA	Windows Media Audio file, used to store audio data, but has been exploited to carry a buffer overflow designed to execute malicious code embedded in the file.
.WSF	Windows Script File, designed to carry a variety of Windows script types.
.WSH	Windows Script Host Settings file, used to configure the script interpreter program on a Windows machine.
.rpm	Red Hat Package Manager, used to bundle libraries, configuration files, and code for simpler installation on Linux systems.
.sh	A UNIX shell script or shell archive file, used to carry sequences of commands for a UNIX shell, usually the Bourne shell (sh) or Bourne again shell (bash)

There sure are many suffixes that could contain executable code of some form. Your users are not going to be able to memorize every single item in this massive list. Still, they should be wary of the biggies that are most often abused by bad guys, such as .EXE, .COM, .BAT, .SCR, .PIF, and .VBS.

Mimicking Other File Names

These Trojan horse naming issues go beyond just putting a bunch of spaces between the name and its file extension on Windows systems. We've just barely scratched the surface. Often, to fool a victim, attackers create another file and process with exactly the same name as an existing program installed on the machine, such as the UNIX init process. Init normally starts running all other processes while the system boots up. In this type of naming attack, you could actually see two processes named init running on your system: your normal init that's supposed to be there, and another Trojan horse named init by the attacker. This is a particularly bizarre circumstance, kind of like waking up and finding that you have two noses.

Similarly, on a Windows machine, you could notice that there are two running processes called iexplore. A bunch of such naming schemes are possible. Table 6.2 lists common programs expected to be running on Windows and UNIX operating systems whose names are frequently borrowed by attackers for malicious code. **It is hugely important to note the following:** There are often *supposed* to be processes running on your machine with these names. Don't freak out if you see a running program named init or iexplore! In all likelihood, these are merely the legitimate programs that should be on your system. If these are legitimate processes, you should not kill them, as your machine requires them to function properly. We're discussing this issue because attackers sometimes impersonate these vital programs using Trojan horses that have the same name.

Of course, the list in Table 6.2 is not comprehensive, as tens of thousands of possible programs and variations would fill this whole book. Still, I want to give you a flavor for the types of Trojan horse naming attacks I'm seeing in the wild in the incidents I handle. If you investigate computer attacks, expect to see these exact names, subtle variations on these names, and a variety of other similar tricks.

I remember a particularly compelling Trojan horse naming attack attempted against me recently. I saw this technique at a SANS security conference, where I run a hacker tools workshop about once per month. In these workshops, student attendees get the opportunity to break into several experimental machines I build and maintain for the class. Students learn the mindset and skills of an attacker, and I get to have fun watching them repeatedly smash into my systems. During one workshop, I received an urgent e-mail from one of my students. The e-mail extolled the virtues of a very exciting new game, named

Table 6.2
Common Names Given to Trojan Horses to Blend In

Name Given to Trojan Horse	Operating System	Legitimate Program That the Trojan Horse Is Trying to Look Like
init	UNIX	During the UNIX system boot sequence, this process runs first and initiates all other processes running on the box.
inetd	UNIX	This process listens on the network for connection requests for various network services, such as Telnet and FTP servers.
cron	UNIX	This process runs various programs at pre-scheduled times.
httpd	UNIX	On a UNIX Web server, several copies of this process typically run to respond to HTTP requests.
win	Windows	Typically there is no legitimate process by this name on a Windows box. However, attackers take advantage of the fact that many administrators might *expect* to see a process with this name.
iexplore	Windows	This executable is Microsoft's Internet Explorer browser. On most Windows systems, a spare browser running every once in a while would go unnoticed.
notepad	Windows	This familiar editor frequently used on Windows systems is an ideal program for an attacker to impersonate. Several backdoor tools attempt to impersonate notepad.exe.
SCSI	Any	Attackers sometimes name their Trojan horse processes SCSI, attempting to dupe an administrator into thinking that the program controls the SCSI chain. An administrator will hesitate to kill a process named SCSI for fear that it might disable the hard drive.
UPS	Any	Sometimes, attackers name their processes UPS to fool administrators into thinking the program controls the uninterruptible power supply.

Vixens with No Clothes, or VNC for short. The sender detailed all of the enticing blockbuster action in this exciting game, which I was invited to install free of charge! How could any reasonable person pass up such an incredible opportunity? In keeping with the fun atmosphere of the workshop, I decided to take the bait knowingly and installed this supposedly nifty game. However, as you might expect, not only were there no

clothes … there were no vixens either! I watched as the keyboard and mouse on my screen began to move by themselves, while squeals of joy erupted from my attacker on the other side of the computer lab! Of course, this was all just a little game. Real-world attackers might not be so blatant, but this example really helps illustrate the concept of using deceptive naming to achieve installation of a Trojan horse backdoor.

For another more real-world example, check out Figure 6.2. You can see the familiar Windows Task Manager on my Windows 2000 system. By hitting Ctrl-Alt-Delete, selecting Task Manager, and then looking at the Processes tab, I can see the various processes running on my box. The list of Figure 6.2 look pretty reasonable. In particular, you can see that I'm running one instance of the Internet Explorer browser (iexplore.exe).
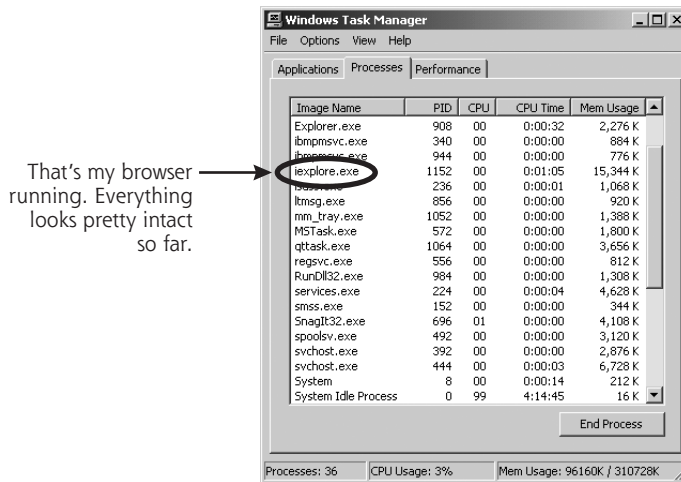
That's my browser running. Everything looks pretty intact so far.

Figure 6.2
Normal Windows Task Manager: Here is what I expect to be running on my Windows 2000 system.

Now, to illustrate a Trojan horse name-based attack, check out Figure 6.3. Here, we see an attacker copying the Netcat program, giving it the rather curious name of iexplore.exe. That's pretty nasty, but rather common. After creating the copy of Netcat, our intrepid attacker, evil dude that he is, sets up a backdoor listener with the copy. The backdoor is waiting with a command shell on TCP port 2222. However, if you look at the Task Manager now, it appears that there is just another copy of iexplore.exe, the Internet Explorer browser, running on my machine.
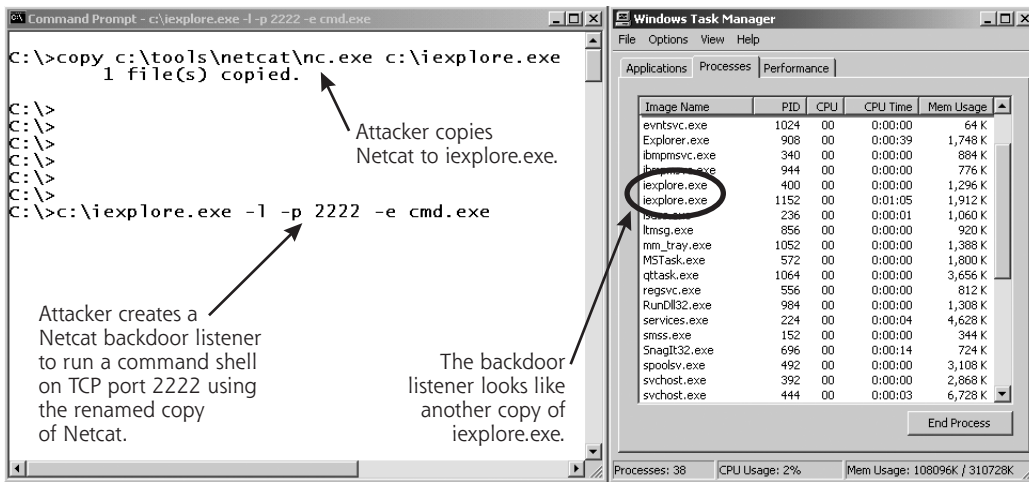
Figure 6.3
Bad guy runs Netcat. Now, the evil attacker creates a copy of Netcat called iexplore.exe and runs a backdoor listening on TCP port 2222.

Users or administrators searching for a malicious process would likely overlook this extra little goodie running on the box, as it looks completely reasonable.

Giving a backdoor a name like iexplore.exe is pretty sneaky. However, an attacker could do something even worse by taking advantage of an interesting characteristic of Windows 2000, XP, and 2003. In these operating systems, the Task Manager won't allow you to kill processes that have certain names [1]. If a process is named winlogon.exe, csrss.exe, or any other name shown in Table 6.3, the system automatically assumes that it is a sensitive operating system process based solely on its name. These names are all used for very important processes on a Windows machine [2], but attackers can use the exact same name for a backdoor. We'll discuss the interplay between many of these processes in more detail in Chapter 8.

Table 6.3
Windows Process Names That Cannot Be Killed with Task Manager

Windows Process Name	Purpose of Legitimate Process with This Name
csrss.exe	This is the environment subsystem process, which supports creating and deleting processes and threads, running 16-bit virtual DOS machine processes, and running console windows.
services.exe	This process is the Windows Service Controller, which is responsible for starting and stopping system services running in the background.
smss.exe	The Session Manager SubSystem on Windows machines is invoked during the boot process. Among numerous other tasks, it starts and supports the programs needed to implement the user interface, including the graphics subsystem and the log on processes.
System	This process includes most kernel-level threads, which manage the underlying aspects of the operating system.
System Idle Process	On a Windows system, this process is just a placeholder to indicate all of the CPU cycles consumed by idle tasks, when no specific other processes have a pressing need.
winlogon.exe	This process authenticates users on a Windows system by asking for user IDs and passwords, and interacting with other components to verify their validity.

If an attacker gives a backdoor a name from Table 6.3, Task Manager will refuse to kill it. The system gets confused, believing the backdoor process is really the vital system process. The system is overprotective. To prevent a user from accidentally killing a vital process and making the system unstable, Windows goes overboard by preventing users from killing any process with such a name. To illustrate this concern, in Figure 6.4, I created a copy of Netcat named winlogon.exe, executed it as a backdoor listener, and tried to kill this imposter using Task Manager. The system instantly popped up a dialog box saying, "This is a critical system process. Task Manager cannot end this process." You might think that Windows would be smart enough to differen-
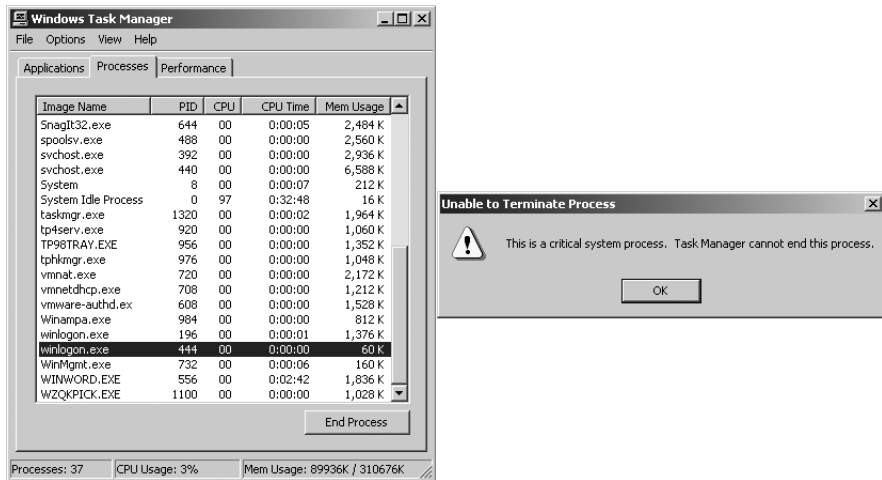
Figure 6.4
On Windows, backdoors that have the same names as vital system processes cannot be
killed by Task Manager.

tiate vital system processes from imposters by looking at the file on the
hard drive the process was started from, or even its process ID number.
However, Windows doesn't do this, and just assumes that any process
named winlogon.exe or csrss.exe must be okay. Therefore, unfortu-
nately, these names are just perfect for Trojan horse backdoors, because
they are more difficult for a system administrator to terminate, if they are
ever discovered.

   As an additional concern, under certain circumstances, you might
legitimately have multiple copies of both csrss.exe and winlogon.exe
running on a machine. If you use Windows Terminal Services or Cit-
rix to allow multiple users to simultaneously log on to virtual desktops
on a single Windows machine, each user will have a csrss.exe and win-
logon.exe. So, if there are two or more copies of these two processes
running, you might not have been attacked; you're just looking at the
processes created for different users. For the other processes listed in
Table 6.3, however, only a single instance of the process should show
up in Task Manager.

The Dangers of Dot "." in Your Path

Another issue associated with Trojan horse names involves the setting
of the path variable for users and administrators. On Windows and

UNIX, most running programs, including command shells and even GUIs, have the concept of a path. This variable just contains a list of directories that are searched in order from start to finish when a new program or command name is executed. For example, on my UNIX machine, I can view my path by typing:

```
$ echo $PATH
```

The default path for users on my UNIX box includes a variety of directories, such as /bin, /usr/bin, /usr/local/bin, and so on. These directories are the locations of the commands commonly run by users on UNIX machines.

On Windows, you can view your path by using the set command and searching for the word *Path*, as follows:

```
C:\> set Path
```

My default path on Windows includes folders such as C:\WinNT \System32, C:\WinNT, and others.

Whenever I type a program's name at a command prompt, my system starts combing through the directories in my path, one by one, until it finds the command and runs it. If it cannot find the command in my path, the system responds with an error message, saying that the program or command could not be found.

On UNIX systems, by default, your current working directory, referred to as "." and usually pronounced "dot," is not in your path. So, if you change to a directory, and type the name of a program in that directory, you'll get a "Command not found" error, even though you are in the same directory as the program you are looking for. This can be frustrating for new UNIX users, but not having the current working directory in your path is a very good thing from a security perspective!

Suppose someone misconfigured your UNIX account, and "." was in your path. Also, suppose that an attacker gains low-privileged access to your machine, but hasn't yet conquered superuser privileges on the box. This bad guy could name an evil Trojan horse program `ls`, and put it in some world writable directory on the machine. The `ls` command is used to get a listing of the contents of a directory. With "." in your path, if you ever changed directories into the attacker's trap directory and ran the `ls` command to get a directory listing, you'd run the evil Trojan horse! This Trojan horse might instantly give the attacker all of your permissions on the machine. If you have superuser privileges, the

attacker now has such privileges as well, having successfully launched a privilege escalation attack using a Trojan horse version of `ls`.

Or, similarly, an attacker could create a backdoor with a name that matches a commonly mistyped command, such as `ipconfig`. The normal UNIX command for viewing network interface information is `ifconfig`, with an f instead of a p. However, users sometimes type `ipconfig` instead, given that a similar command with that name is available on Windows. If I create a Trojan horse named `ipconfig` on your UNIX machine, I can sit back and wait for an administrator to accidentally type `ipconfig` while in the wrong directory. For this reason, "." isn't in the path on UNIX machines by default, and you shouldn't reconfigure your shell to add it. In this case, the default path setting for UNIX is quite reasonable. So, do yourself a favor, and leave it as is. Also, if you do have "." in your path, consider removing it by editing the various start-up scripts associated with your login shell, which depend on the particular shell you are using.

However, not having "." in your path also means that if you change directories to a place where a program file is located, you cannot just type the program's name to run it. Instead, to run the program, you have to type `./[program_name]` to execute the program. So, if the system ever complains that it cannot find a file, but you can see the file in the current working directory using `ls`, use the "./" notation to start the program. It's not too much of a burden.

This matter differs markedly on Windows systems. In the Windows command shell, the current working directory is implicitly in your path. Even though the `set` command doesn't show a "." in your path, it's still there, implicitly represented, just because you are using Windows. Therefore, if you change to a directory with an executable inside and then type the executable's name on Windows, the executable runs. The system automatically finds it, because "." is implicitly at the very beginning of your path. Yes, it's convenient, as you don't have to ever mess with the "./" notation. However, having "." in your path is also a security hole.

If an attacker gets low-privileged access to your machine, and then tricks an administrator into running a command, the attacker can escalate privileges. One of the most common tricks attackers utilize in Windows is to create a privilege-escalating Trojan horse named `cp`. On Windows, the `copy` command is used to copy a file, and there is no default command named `cp`. However, users sometimes mistakenly type `cp` when they try to copy files. If they type `cp` in a directory where

the attacker placed a Trojan horse with that name, the attacker could easily get that user's privileges on the machine.

Unfortunately, you cannot easily remove "." from your path on a Windows machine. It's built into the operating system itself right at the start of your path. Remember, the system searches for commands starting from the beginning of your path, running the first matching program that it finds. Mistyping a command name could lead to a privilege escalation attack on a Windows system, so be careful when typing commands with an account with administrator privileges.
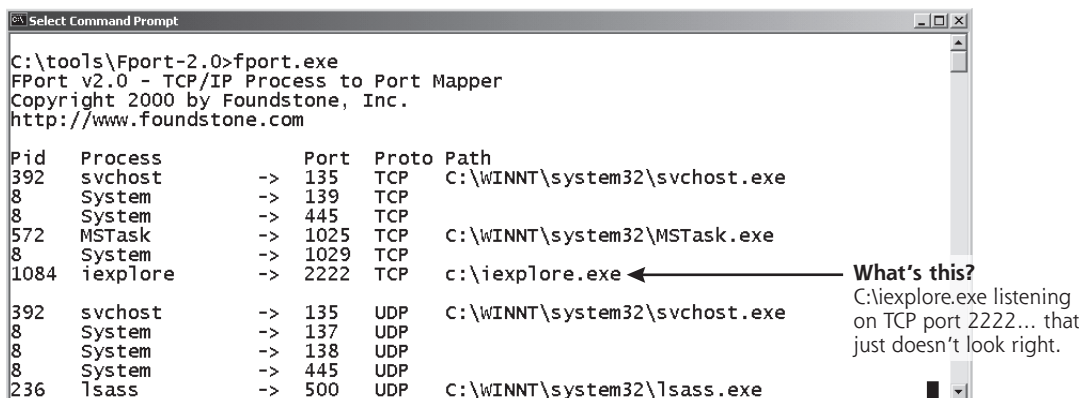
Trojan Name Game Defenses

So, in light of these deviously named Trojan horses, what can we do to defend ourselves? First, we must keep the malicious code off of our systems in the first place by employing the antivirus tools described in Chapter 2 and the backdoor defenses described in Chapter 5.

Also, you should be ready to kill suspicious processes that usurp the names of legitimate processes. Even though Task Manager cannot kill processes with certain names, you can deploy a free tool called PsKill from the PsTools package, available for free at *www.sysinternals.com.* PsKill can shut down any running process, regardless of its name. However, be careful with this tool! If you shut down a legitimate process, you could cause your system to be unstable or even create an instant crash. Therefore, you need to research each process of concern in more detail before shutting it down.

To conduct this research, you can use some tools we initially discussed in Chapter 5. Remember our good friends, lsof and Fport? As you might recall, Fport, run on a regular basis by diligent system administrators on Windows machines, will help you discover strange port usage associated with Trojan horses on your system. For each running process that has an open TCP or UDP port on the network, Fport shows the process ID, process name, port number, and the full pathname of the file that the process ran from on the hard drive. Fport is very simple, yet highly effective. On UNIX machines, you can use the `lsof` command to achieve similar functionality to Fport, as we discussed in Chapter 5.

Remember our example in which the attacker renamed Netcat so that it appeared as iexplore.exe? In Figure 6.5, we can see how Fport displays this subterfuge.

```
Select Command Prompt                                              _|□|×|
C:\tools\Fport-2.0>fport.exe
FPort v2.0 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com

Pid    Process         Port   Proto Path
392    svchost    ->   135    TCP   C:\WINNT\system32\svchost.exe
8      System     ->   139    TCP
8      System     ->   445    TCP
572    MSTask     ->   1025   TCP   C:\WINNT\system32\MSTask.exe
8      System     ->   1029   TCP
1084   iexplore   ->   2222   TCP   c:\iexplore.exe

392    svchost    ->   135    UDP   C:\WINNT\system32\svchost.exe
8      System     ->   137    UDP
8      System     ->   138    UDP
8      System     ->   445    UDP
236    lsass      ->   500    UDP   C:\WINNT\system32\lsass.exe
```

What's this?
C:\iexplore.exe listening
on TCP port 2222... that
just doesn't look right.

Figure 6.5
Using Fport. Why is iexplore.exe listening on TCP port 2222 and why is it running from
C:\iexplore.exe? That looks like a problem!

Fport tells us that there are a variety of programs using ports on this machine. All of these ports are pretty normal on a Windows machine, except for the one with a Process ID (Pid) of 1084. It's called iexplore.exe, but is listening on TCP port 2222 and running out of C:\iexplore.exe. That just doesn't look right!

Using Fport, we can differentiate between the real browser, which should have a path of C:\Program Files\Internet Explorer\iexplore.exe, and the attacker's backdoor, which runs from C:\iexplore.exe. Unfortunately, this kind of analysis requires an administrator to be intimately familiar with what is supposed to be running on the system. That way, if a counterfeit pops up, an administrator can quickly identify it and investigate. This can be very difficult, but rock-solid system administrators should have a gut feel for what is installed and running on critical systems. If an experienced system administrator notifies you that "something just doesn't look right with this program," you ignore their concerns at your own peril. Your best bet is to analyze suspect programs in a laboratory environment to determine if they are attempting to access files or the network unexpectedly. In Chapter 11, we'll discuss a recommended laboratory environment and analysis process you can use to analyze problematic software.

Another defense for these Trojan naming schemes is to block executable e-mail attachments at your Internet gateway. You should filter out all programs that are potentially executable. These include the familiar EXE programs, but go well beyond that, too. In reality, you

should filter out at least all of the program types described in Table 6.1. For more information about these and other file extension types, you should check out the File Extension Source Web site at *http://filext.com*.

Wrap Stars

Be afraid. Be very afraid.

—The movie, *The Fly*, 1986

Bad guys' Trojan horse ruses aren't limited to just playing games with names. Many attackers also combine their malicious code with an innocuous program to create a nice, cozy-looking package. By grafting together two programs, one malicious and one benign, an attacker can more easily trick unsuspecting users or administrators into running or ignoring the combined result. When unsuspecting victims receive the combined package and run it, the malicious executable embedded in the package will typically run first. Of course, the vast majority of backdoors don't display anything on the screen, so the victim will not see anything during this step, which usually takes less than a second. After the backdoor is firmly lodged on the victim machine, the benign program runs. For example, an attacker might take the Tini backdoor we briefly mentioned in Chapter 5 and combine it with Internet Explorer. Given Tini's small size, the resulting program would be only 3 kilobytes larger than the original browser.

To marry two executables together, an attacker uses a wrapper tool. The computer underground uses several terms to refer to these tools, including *wrappers*, *binders*, *packers*, *EXE binders*, and *EXE joiners*. Figure 6.6 illustrates how an attacker uses a wrapper program. In essence, these wrappers allow an attacker to take *any* executable backdoor program and combine it with *any* legitimate executable, creating a Trojan horse without writing a single line of new code! Even the most inexperienced attacker can easily create Trojan horses using this technique. This is the stuff script kiddie attackers fantasize about.

For an analogy of the operation of wrapper programs, consider the classic movie *The Fly*. As you might recall, in that epic feature, a scientist tests his new teleporter invention to whisk himself across his laboratory at the speed of light. Sadly, a simple housefly zooms into the teleporter pod just as he initiates his first short journey. The machine cannot handle
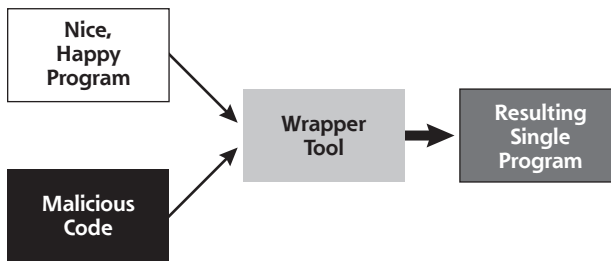
Figure 6.6
Wrapper programs: Two programs enter and one program leaves with the combined
functionality of both input programs.

two living beings, so it just combines the scientist and the fly at their
most fundamental level into one very ghastly mutant combination of
the two. That's essentially what wrapper tools do: combine two or more
separate programs at a fundamental level into one package.

Wrapper Features

Some wrappers allow for combining two, six, nine, or even an arbitrary
number of programs together. Others allow for the addition of static
files into the mix. When the wrapper is run, it executes all included pro-
grams, and also unloads the bundled static files into the attacker's cho-
sen places on the file system. With such capabilities, these wrappers are
actually becoming the functional equivalent of souped-up install shields
and Setup programs.

For most of the popular wrapper tools available today, when a
combined package file is executed, the malicious program and benign
program will each show up as separate running processes in Windows
Task Manager or Fport output. The two programs only live together in
the file on the hard drive. When a user is duped into running the pack-
age, the two wrapped programs become two separate processes. There-
fore, to hide the malicious processes, attackers use wrappers together
with the deceptive naming schemes we discussed in the last section.

Some wrappers go even further by encrypting the malicious code
portion of the resulting package, so that antivirus programs on the tar-
get system have more difficulty detecting the malicious program. Of
course, to make the malicious program run on its target, the wrapper
must add a decryption routine to the resulting package. Antivirus pro-
grams therefore look for the decryption code added by these popular
wrapping tools. Attackers raise the bar by morphing the decryption

code so that it dynamically alters itself to evade detection, using poly-
morphic coding techniques, as we discussed in Chapter 2.

The computer underground has released dozens of wrapper pro-
grams available for free download from the Internet. Table 6.4 shows some
of the most popular and powerful wrapper programs available today. To
analyze these and other wrapper tools in more detail, you can check out
www.tlsecurity.net/exebinder.htm, a comprehensive Web site devoted to the
fine art of wrappers. It's important to note that not all of these programs are
inherently evil. They also have a variety of entirely legitimate uses for pack-
aging and distributing useful software, not just Trojan horses.

Table 6.4
Popular Wrapper Tools

Wrapper Tool Name	Function of Wrapper Tool
AFX File Lace	This wrapper encrypts an executable and appends it to the end of another, unencrypted executable.
EliteWrap	This program is the premier wrapper tool, with gobs of features, including: • The ability to bind together an unlimited number of executables. • A function to start programs in a specified order, with each program waiting for the other programs ahead of it to finish running before executing itself. • Built-in integrity checks to make sure the package hasn't been altered.
Exe2vbs	This tool converts executable programs (in EXE format) into Visual Basic Scripts (VBSs or VBScripts). By packing the EXE inside of a VBScript, the attacker might be able to transmit a Trojan horse through e-mail filtering programs that block standard EXEs, but allow VBScripts to pass through.
PE Bundle	This program bundles together an executable with all the DLLs required by that executable to run. With this combined package, the malicious software will be able to run on the target system even if some critical DLLs are not installed there.
Perl2Exe	Using this tool, a developer can create standalone programs originally written in the Perl scripting language that do not require a Perl interpreter to run. Also, the original Perl code isn't included inside the resulting executable, making reverse engineering the functionality of the executable code significantly more difficult than simply analyzing more easily understood Perl scripts. This nifty tool is available for both Windows and UNIX, turning a Perl script into an executable binary program. Binary executables can be created that will run on Windows or UNIX.

Table 6.4
Popular Wrapper Tools (Continued)

Wrapper Tool Name	Function of Wrapper Tool
Saran Wrap	This easy-to-use GUI-based wrapper combines two executables together.
TOPV4	This so-called Teflon Oil Patch program combines up to nine executables together and sports a simple GUI.
Trojan Man	This wrapper combines two programs, and also can encrypt the resulting package in an attempt to foil antivirus programs.

Wrapper Defenses

To defend your systems against attacks involving Trojan horses created with wrappers, antivirus tools are really your best bet. By detecting the malicious code wrapped into a combination package and preventing its installation, antivirus tools stop the vast majority of these problems. Following the antivirus recommendations we discussed in Chapter 2 goes a long way in dealing with this problem.

Trojaning Software Distribution Sites

The woman said, "The serpent deceived me, and I ate."

—Genesis 3:13

So, we've seen how attackers use name trickery and wrapper programs to create and disguise their backdoors. Now, let's discuss a far nastier Trojan horse technique that is greatly increasing in popularity: Trojaning software distribution sites. Increasingly, some attackers are aiming beyond the individual software loaded on your system, and going upstream by attacking the Internet sites used to distribute software. What better way could there be to get widespread dispersal of malicious code than to put a Trojan horse version of a popular program on a Web site used by millions of people around the world? Everyone who downloads and installs the tool would be impacted by such a Trojan horse.

Trojaning Software Distribution the Old-Fashioned Way

There is an admittedly lower tech precedent to this trend. Over the last two decades, attackers would sometimes send software updates containing malicious code via the snail-mail postal service. A package would arrive containing a tape or CD of supposedly crucial software updates, claiming to be from a legitimate vendor. Some administrators and users would fall for the trick, and blindly load the software onto their systems. Bingo! The attacker's backdoor would be loaded onto the system by the administrators or users themselves. Of course, such an attack could constitute mail fraud, a felony in some countries.

Sending Trojan horse updates with backdoors via the postal service still works today. If several administrators in your organization received an official-looking package claiming to be from Microsoft Corporation, Sun Microsystems, or even Ed's Linux Software and Chop Suey Take Out Service, would they install it? Similarly, what would happen if some of your telecommuters received a CD in the mail at home with a note on company letterhead describing an important update? Unfortunately, in most organizations, at least some administrators and users would install the package without a second thought. All it takes is one mistake for the attacker to get a foothold in the organization. Of course, if any users start asking questions about the mysterious new package that arrived in the mail, the attacker's subterfuge should be quickly detected.

Popular New Trend: Going after Web Sites

While the snail-mail technique works like a charm, attackers don't want to have to pay postage. Instead, they've set their sights on higher targets with a wider spread of dispersal possibilities, such as the Web servers used to distribute new software and updates across the Internet. These attacks are particularly pernicious, as they could impact thousands or millions of unsuspecting administrators and users who are simply trying to download the latest versions of popular programs. One of the earliest attacks of this kind involved the Washington University at St. Louis FTP server (wu-ftpd), which was Trojanized way back in April 1994 [3]. In January 1999, a similar attack occurred involving the TCPWrapper distribution, which is, rather ironically, a security tool [4]. However, much more recently, we've seen a rash of successful attacks against Web sites, including these:

- *Monkey.org:* In May 2002, someone broke into the Web site that distributes the popular security and hacking tools written by Dug Song. Attackers modified the Dsniff sniffing program, as well as the Fragroute and Fragrouter IDS evasion tools distributed through Monkey.org. The attacker replaced each tool with a Trojan horse version that created a backdoor on the systems of anyone who downloaded and installed these tools. This attack was especially insidious, considering the widespread use of these tools by security professionals and computer attackers alike.

- *Openssh.org:* From July 30 to August 1, 2002, an attacker loaded a Trojan horse version of the Open Secure Shell (OpenSSH) security tool onto the main OpenSSH distribution Web site. OpenSSH is widely used to provide rock-solid security for remote access to a system. However, diligent administrators who tried to protect their systems by downloading this security tool in late July 2002 unwittingly installed a backdoor. Sadly, this tool often utilized to protect systems against attack included its own backdoor for this short period of time.

- *Sendmail.org:* This one is just plain evil. From September 28 until October 6, 2002, a period of more than one week, the distribution point for the most popular e-mail server software on the Internet was subverted. The main FTP server that distributes the free, open-source Sendmail program was Trojanized with a nasty backdoor.

- *Tcpdump.org:* From November 11 to 13, 2002, tcpdump, the popular sniffing program, and libpcap, its library of packet capture routines, were replaced with a Trojan horse backdoor on the main tcpdump Web site. Not only is the tcpdump sniffer widely used by security, network, and system administrators around the world, but the libpcap (pronounced using the elegant term *lib-pee-cap*, which is short for "library for packet capture") component is a building block for numerous other tools. Administrators who installed tcpdump, libpcap, or any other package built on top of libpcap during this time frame were faced with a backdoor running on their systems.

Some pretty big names have fallen to this attack! This list contains some pretty important software, used by millions of people each and every day. Heck, I *personally* use Dsniff, OpenSSH, and tcpdump all the time, to say nothing of Sendmail. With all of these attacks over a six-month period, I began to take this whole thing very much to heart. In most of these attacks, the bad guys manipulated the install program

associated with each tool so that it created a backdoor listener on the machine where the program was configured and compiled. In these cases, the compiled binary executable itself wasn't altered; the installation program was modified to include the backdoor. The great similarities in each of these attacks could indicate that a single perpetrator committed all of these dastardly deeds, or the actions could merely have been copycat crimes.
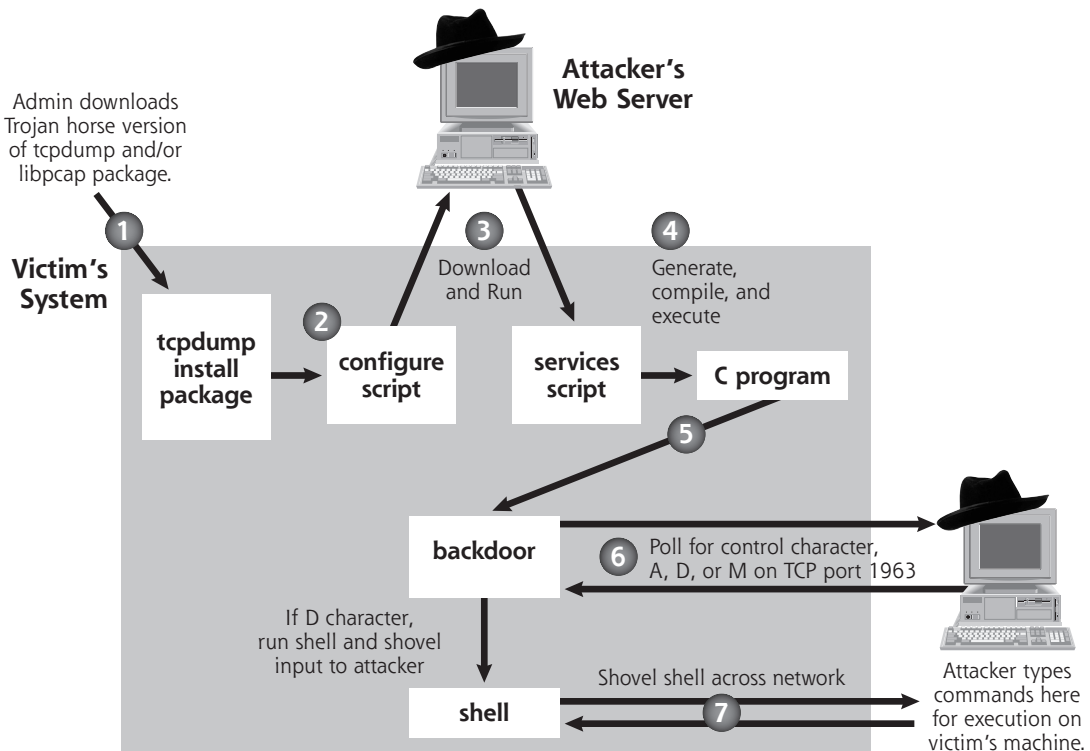
The Tcpdump and Libpcap Trojan Horse Backdoor

To understand the nature of the Trojan horses bundled with these programs, let's look at the functionality of the malicious code included in the tcpdump and libpcap distribution during that fateful week in November 2002. This Trojan horse was similar to the one used in the Monkey.org, Sendmail, and OpenSSH attacks, so analyzing it will help us better understand this whole class of attacks.

To install an up-to-date version of tcpdump, an administrator typically downloads the latest package from the tcpdump Web site. This package includes a script called "configure" that analyzes the system used to compile the tool, typically an administrator's machine. The configure script verifies that certain required compiler options, libraries, and other programs needed for building tcpdump are included on the system. The script then devises a plan for compiling the software on that particular machine. After configure runs, the administrator can compile the tool.

However, the version of the configure script distributed with tcpdump and libpcap included a nasty yet invisible surprise. The whole process is illustrated in Figure 6.7, starting with the download of the Trojan horse version of the installation package in step 1. The administrator runs the configure script in step 2. While the configure script checks the system configuration as expected, it also attempts to connect to a Web server operated by the attacker to grab a copy of another script, named "services," shown in step 3. With a simple name like services, it sounds pretty innocuous, huh?

Step 3 is a somewhat risky move for the attacker, because the victim's machine will send out an HTTP request to the attacker's machine. It is conceivable, although highly unlikely, that an administrator might notice this request on the network, and trace it down to a Web site controlled by the attacker. Still, this Web request to download the services script gives the attacker flexibility. Rather than bundling a set of fixed backdoor functionality into the installation package, the attacker can add new capabilities to the backdoor and load it on a Web site. Then,

Figure 6.7
The tcpdump and libpcap Trojan horse backdoor.

the attacker can just sit back and wait for a new set of victims to inadvertently install the updated functionality of the backdoor. After downloading the services script, the configure script executes it. In step 4, the services script, in turn, creates a small amount of C code for a backdoor, which it compiles and executes.

This little compiled C program is really a simple backdoor, which starts running in step 5. The backdoor then makes a connection across the network to the attacker's own machine. In step 6, the backdoor polls the attacker's system on TCP port 1963 to retrieve a single character indicating what the backdoor should do. This request for a command is sent every few minutes. The backdoor responds to three possible control characters:

- The A character indicates that the backdoor program should stop running.

- The D character tells the backdoor program to create a shell and shovel this shell to the attacker. It uses the same shell-shoveling technique we discussed in Chapter 5. The attacker can then type any commands into the shell for execution on the victim machine, shown in step 7. If tcpdump or libpcap was installed by an administrator, these commands would run with root privileges. Otherwise, the commands would still run, but with the privileges of a more limited account. Of course, most people who compile and install tcpdump or libpcap do so with root permissions.
- The M character tells the backdoor tool to sleep for one hour, and then poll for another control character.

After the attacker finishes executing commands on the victim, the shell is terminated and the backdoor's polling for A, D, or M commands continues. At a later time, the attacker can fire up the shell shoveler again, and access the system.

There are a couple of interesting little twists in this Trojan horse backdoor. First, look at those control characters: A-D-M. A rather famous group of hackers calls itself the ADM Crew, known for writing some seriously powerful computer attack tools. Is this a mere coincidence? That's highly doubtful, as the odds that someone would randomly select control characters of A, D, and M are very slim. Did ADM perpetrate the attack, or was someone trying to frame them? At the time of this writing, the information security community at large just doesn't know the answers to these questions. Given the secrecy in certain quarters of the computer underground, we might never know the full truth.

A second twist in this tcpdump Trojan horse involves alterations to the sniffer tools themselves. The attacker manipulated the source code of the libpcap library so that any sniffer that uses it will not show any traffic destined for TCP port 1963. That way, if administrators run a sniffer built from the compromised program on the compromised machine, they won't see the polling request for the A-D-M control characters, or the traffic going to and from the shell! If you are going to Trojanize a sniffer with an embedded backdoor, you might as well make the sniffer itself hide the backdoor's traffic. This certainly helps to mask the attacker's activity. Not only does the Trojan horse tcpdump distribution open up a backdoor, it also installs a Trojan version of a sniffer to hide that very same backdoor quite effectively. Any sniffer built on the system that relies on the modified libpcap package, such as tcpdump, Snort, Ethereal, or others, would likewise ignore this traffic.

Unfortunately, this trend of Trojanizing software distribution Web sites didn't end with the Trojan horse version of tcpdump. Attackers are certainly setting their sights on even larger prey. I'm sure they are constantly scanning large-scale software distribution sites, such as Microsoft's own Windows Update servers, various Linux software distribution sites, and other popular software depots to find flaws and upload their malicious wares. On the plus side, these sites are usually quite carefully secured, and software vendors such as Microsoft are increasingly using digital signatures to ensure the integrity of their patches. On the negative side, a single error in any of these schemes could lead to Trojan horse backdoors installed on millions of systems. That's not a happy thought.

Defenses against Trojan Software Distribution

Defenses against this type of attack fall into three categories: user awareness, administrator integrity checks, and carefully testing new software. First, you and your organization must be aware of the threat. Without fundamental knowledge of what you're up against, you're guaranteed to lose. Your policies must clearly state that users are strictly forbidden from installing unauthorized programs on your organization's systems. Users should not install any unexpected software updates that arrive in the mail, no matter how "official" they appear to be. I don't care if the package included the company logo; it should never be installed. If any updates do arrive, they should immediately be forwarded to the security team. If you need to update users' systems, you should have a formalized plan announcing how you'll be distributing software to them. This plan should be included in user awareness materials.

Furthermore, put together an awareness campaign to let your computer users and administrators know that attackers sometimes distribute nasty software via the Internet or even via snail mail. Dress up your awareness efforts by setting up a booth outside of a cafeteria with colorful signs and balloons. I call this the froo-froo components of a security awareness campaign, because it's neither deep nor technical. Still, the froo-froo is important, as it gets users' attention. Distribute simple pamphlets with silly cartoons to your user base to let them know how to do the right thing. Although a solid security awareness program takes a lot of work, it can be fun. In fact, it'll be far more effective if it's entertaining and full of froo-froo rather than just the same old droning on about policy this blah-blah-blah policy that blah-blah-blah. Typical users rap-

idly tune out any dialogue they don't understand or care about, but if it has cool balloons and cartoons, they just might listen.

Another important area for defending against these attacks involves administrative procedures for checking the integrity of the packages you download. Whenever I upgrade a software tool across the Internet, I always download copies from at least three different mirrors. I then verify the integrity of the programs using a cryptographically strong hash against each mirror's copy to make sure they all match. You can create an MD5 hash, kind of like a digital fingerprint, for any file using the great md5sum program included in most Linux distributions. On Windows, you can use the free md5summer program written by Luke Pascoe, available at *www.md5summer.org*. Because MD5 is a one-way hash function, an attacker would find it very, very difficult to create a Trojan horse with the exact same hash as the legitimate program. By difficult, I mean that they would require a supercomputer running for thousands of years to create an evil program that has the exact same hash as your good program. At least, that's the idea if these one-way algorithms are as good as we hope they are.

A lot of Web sites that distribute software include a file containing the MD5 hash of the latest version on the site itself. However, I'm uncomfortable downloading a program from just a single mirror and checking this single hash from the exact same site. Think about it. If attackers could compromise a single Web site and Trojanize the software, of course they could alter the file containing the hash on that same Web server. The idea here is that an attacker would have a more difficult time compromising several mirrors of the code, and therefore I'll be able to catch their treachery by observing different versions on the mirrors. By downloading from multiple mirrors and checking for consistency across them, I get much better odds that the attacker hasn't compromised them all, and I'll have an intact program to run. Unfortunately, if the mirrors are automatically updated from a single central server, I'd still lose if the bad guy contaminates the code on the main server. I've raised the bar some by comparing hashes across multiple mirrors, but the bad guys could still leap over the higher bar.

Some software download sites go beyond hashes and include a digital signature of the software, using a public key encryption package such as Pretty Good Privacy (PGP). If you download any software with such signatures, you should verify those signatures using an appropriate package, such as the open source clone of PGP called "Gnu Privacy Guard," available for free at *www.gnupg.org*. Of course, an attacker could modify the digital signature or even replace the key used to sign

the package. However, such attacks would be much more difficult, and are therefore far less likely.

Finally, you should always test new tools before rolling them into production. Such a test process not only gives you a chance to detect the malicious software in advance, but it also gives you some precious time for others to discover the problem before you blindly put code into production. I was working with one bank whose bacon was saved simply because they spend at least one month reviewing any new release of Sendmail before putting it into production. I'd love to tell you that they discovered the Sendmail backdoor while they were looking through the program in their evaluation network. However, they didn't find it. Still, while they were analyzing the new release to make sure it met corporate functionality requirements, other folks had discovered and publicized the backdoor in October 2002. When the bank heard about the discovery of a backdoor in this version of Sendmail, they yanked it from their test systems and never rolled it into production. The built-in lag of their analysis process certainly helped this organization avoid catastrophe. For critical security patches, rapid deployment is crucial. For simple upgrades or new features, a few weeks lag can actually help improve security.

Poisoning the Source

> *Most software sucks.*
>
> —Jim McCarthy, founder of a software quality training company, as quoted in *Technology Review Magazine,* July/August, 2002

So, we've seen a variety of techniques bad guys use to squeeze Trojan horse functionality into our systems. However, perhaps the most worrisome Trojan horse vector involves inserting malicious code into a software product before it's even released. Attackers could Trojanize programs during the software vendor's development and testing processes. Suppose an attacker hires on as an employee at a major software development shop or volunteers to contribute code to an open source software project. The target could be anything; a major operating system, a widely used enterprise resource planning tool, or even a very esoteric program used by banks to manage their funds transfer would all make juicy targets. As a developer or even a tester, the attacker could insert a relatively small backdoor of less than 100KB of code inside of hundreds of megabytes of legitimate code. That's really a needle in a

haystack! Any users purchasing the product would then unwittingly be buying a Trojan horse and installing it on their systems. The whole software product itself becomes the Trojan horse, doing something useful (that's why you buy or download it), yet masking this backdoor.

Ken Thompson, noted UNIX cocreator and C programming language guru, discussed the importance of controlling source code and the possibility of planting backdoors in it in his famous 1984 paper titled "Reflections on Trusting Trust." In that classic paper, Thompson described modifying the source code for a compiler so that it built a backdoor into all code that it compiles [5]. The proposed attack was particularly insidious, as even a brand new compiler that is compiled with a Trojan version of the old compiler would have the backdoor in it, too. This avenue of attack has long been a concern, and is an even bigger potential problem today.

This concern is even more disturbing than the Trojaning of software distribution sites that we discussed in the last section. When an attacker Trojanizes a software distribution site, the developers of the software at least have a clean version of the software that they can compare against to detect the subterfuge. Backing out problems is relatively easier after discovery, as a clean version of the software can be placed on the Web site for distribution. On the other hand, if an attacker embeds a Trojan horse during the software development process, the vendor might not even have a clean copy. If the attackers are particularly clever, they will intertwine a small, inconspicuous backdoor throughout the normal code, making eradication extremely difficult. The software developer would have to scan enormous quantities of code to ensure the integrity of a whole product. The larger the software product, the more difficult detection and eradication become. Let's analyze why this is so.

Code Complexity Makes Attack Easier

Most modern software tools are vast in scope. Detecting bugs in code, let alone backdoors, is very difficult and costly. To Trojanize a software product, an evil employee doesn't even have to actually write an entire backdoor into the product. Instead, the malicious developer could purposefully write code that contains an exploitable flaw, such as a buffer overflow, that would let an attacker take over the machine. Effectively, such a purposeful flaw acts just like a backdoor. If the flaw sneaks past the software testing team, the developer would be the only one who

knows about the hole initially. By exploiting that flaw, the developer could control any systems using his or her code.

To get a feel for how easily such an intentional flaw or even a full Trojan horse could squeak past software development quality processes, let's consider the quality track record of the information technology industry over time. Software quality problems have plagued us for decades. With the introduction of higher density chips, fiber-optic technology, and better hard drives, hardware continues to get more reliable over time. Software, on the other hand, remains stubbornly flawed. Watts Humphrey, a software quality guru and researcher from Carnegie Mellon University, has conducted surveys into the number of errors software developers commonly make when writing code [6]. Various analyses have revealed that, on average, a typical developer accidentally introduces between 100 and 150 defects per 1,000 lines of code. These issues are entirely accidental, but a single intentional flaw could be sneaked in as well.

Although many of these errors are simple syntactical problems easily discovered by a compiler, a good deal of the remaining defects often result in gaping security holes. In fact, in essence, a security vulnerability is really just the very controlled exploitation of a bug to achieve an attacker's specific goal. If the attacker can make the program fail in a way that benefits the attacker (by crashing the system, yielding access, or displaying confidential information), the attacker wins. Estimating very conservatively, if only one in 10 of the defects in software has security implications, that leaves between 10 and 15 security defects per 1,000 lines of code. These numbers just don't look very heartening.

A complex operating system like Microsoft Windows XP has approximately 45 million lines of code, and this gigantic number is growing as new features and patches are released [7]. Other operating systems and applications have huge amounts of code as well. Doing the multiplication for XP, there might be about 450,000 security defects in Windows XP alone. Even if our back-of-the-envelope calculation is too high by a factor of 100, that could still mean 4,500 security flaws. Ouch! Indeed, the very same day that Windows XP was launched in October 2001, Microsoft released a whopping 18 megabytes of patches for it.

Don't get me wrong; I love Windows XP. It's far more reliable and easier to use than previous releases of Windows. It's definitely a move in the right direction from these perspectives. However, this is just an illustration of the security problem inherent in large software projects. It isn't just a Microsoft issue either; the entire software industry is introducing larger, more complex, ultra-feature-rich (and some-

times feature-laden) programs with tons of security flaws. Throughout the software industry, we see very fertile soil for an attacker to plant a subtle Trojan horse.

Test? What Test?

Despite these security bugs, some folks still think that the testing process employed by developers will save us and find Trojan horses before the tainted products hit the shelves. I used to assuage my concerns with that argument as well. It helped me sleep better at night. But there is another dimension here to keep in mind to destroy your peaceful slumber: *Easter eggs*. According to The Easter Egg Archive™, an Easter egg is defined as:

> *Any amusing tidbit that creators hid in their creations. They could be in computer software, movies, music, art, books, or even your watch. There are thousands of them, and they can be quite entertaining, if you know where to look.*

Easter eggs are those unanticipated goofy little "features" squirreled away in your software (or other products) that pop up under very special circumstances. For example, if you run the program while holding down the E, F, and S keys, you might get to see a dorky picture of the program developer. The Easter Egg Archive maintains a master list of these little gems at *www.eeggs.com*, with more than 2,775 software Easter eggs on record as of this writing.

What do Easter eggs have to do with Trojan horses in software? A lot, in fact. If you think about our definition of a Trojan horse from early in this chapter, an Easter egg is really a form of Trojan horse, albeit a (typically) benign one. However, if software developers can sneak a benign Easter egg past the software testing and quality assurance teams, there's no doubt in my mind that they could similarly pass a Trojan horse or intentional buffer overflow as well. In fact, the attacker could even put the backdoor inside an Easter egg embedded within the main program. If the testing and quality assurance teams don't notice the Easter egg or even notice it but let it through, they likely won't check it for such hidden functionality. To me, the existence of Easter eggs proves quite clearly that a malicious developer or tester could put nasty hidden functionality inside of product code and get it through product release without being noticed.

To get a feel for an Easter egg, let's look at one embedded within a popular product, Microsoft's Excel spreadsheet program. Excel is quite famous for its Easter eggs. An earlier version of the program,

Figure 6.8
The game hidden inside of the Microsoft Excel 2000 spreadsheet application.

Excel 97, included a flight simulator game. A more recent version, Excel 2000, includes a car-driving game called Dev Hunter, which is shown in Figure 6.8.

For this Easter egg to work, you must have Excel 2000 (pre Service Release 1), Internet Explorer, and DirectX installed on your computer. To activate the Easter egg and play the game, you must do the following:

- Run Excel 2000.
- Under the File menu, select Save as Web Page.
- On the Save interface, select Publish and then click the Add Interactivity box.
- Click Publish to save the resulting HTM page on your drive.
- Next, open the HTM page you just created with Internet Explorer. The blank spreadsheet will appear in the middle of your Internet Explorer browser window.
- Here's the tricky part. Scroll down to row 2000, and over to column WC.
- Now, select the entirety of row 2000 by clicking on the 2000 number at the left of the row.
- Hit the Tab key to make WC the active column. This column will be white, while the other columns in the row will be darkened.
- Hold down Shift+Ctrl+Alt and, at the same time, click the Microsoft Office logo in the upper left corner of the spreadsheet.
- In a second or two, the game will run.
- Use the arrow keys to drive and steer and the spacebar to fire. The O key drops oil slicks to confound the other cars. When it gets dark, you can use the H key to turn on your headlights.

If the game isn't invoked on your system, it is likely because you have Service Release 1 or a later version of Microsoft Excel installed on your machine, which doesn't include the Easter egg. You could hunt down an earlier version of Microsoft Excel, or just take my word for it.

Now, mind you, this "feature" is in a spreadsheet, an office productivity program. Depending on your mindset, it might be quirky and fun. However, how does such a thing get past the software quality process (which should include code reviews) and testing team? Maybe the quality assurance and testing personnel didn't notice it. Or, perhaps the quality assurance folks and testers were in cahoots with the developers to see that the game got included into the production release. Either way, I'm concerned with the prospects of a Trojan horse being inserted in a similar way at other vendors.

Again, I'm not picking on just Microsoft here. In fact, Microsoft has gotten better over the past couple of years with respect to these concerns. New service packs or hot fixes frequently and quickly squash any Easter eggs included in earlier releases. Microsoft's Trusted Computing initiative, although often derided, is beginning to bear some fruit as fewer and fewer security vulnerabilities and Easter eggs appear to be coming to market in Microsoft programs. However, I say this with great hesitation, as another huge gaping egg could be discovered any day. Still, underscoring that this is not a Microsoft-only issue, many other software development shops have Easter eggs included in their products, including Apple Computer, Norton, Adobe, Quark, the open source Mozilla Web browser, and the Opera browser. The list goes on and on, and is spelled out for the world to see at *www.eeggs.com.*

The Move Toward International Development

A final area of concern regarding malicious software developers and Trojan horses is associated with code being developed around the world. Software manufacturers are increasingly relying on highly distributed teams around the planet to create code. And why not? From an economic perspective, numerous countries have citizens with top-notch software development skills and much lower labor rates. Although the economics make sense, the Trojan horse security issue looms much larger with this type of software development.

Suppose you buy or download a piece of software from Vendor X. That vendor, in turn, contracts with Vendors Y and Z to develop certain parts of the code. Vendor Z subcontracts different subcomponents of the work to three different countries around the globe. By the time the

product sits on your hard drive, thousands of hands distributed across the planet could have been involved in developing it. Some of those hands might have planted a nasty backdoor. Worse yet, the same analysis applies to the back-end financial systems used by your bank and the database programs housing your medical records. Information security laws and product liability rules vary significantly from country to country, with many nations not having very robust regulations at all.

This concern is not associated with the morality of the developers in various countries. Instead, the concern deals with the level of quality control that can be applied with limited contract and regulatory supporting structures. Also, the same economic effects that are driving development to countries with less expensive development personnel could exacerbate the problem. An attacker might be able to bribe a developer making $100 a week or month into putting a backdoor into code for very little money. "Here's 10 years' salary … please change two lines of code for me" might be all that it would take. We don't want to be xenophobic here; international software development is a reality with significant benefits in today's information technology business. However, we must also recognize that it does increase the security risks of Trojan horses or intentional software flaws.

Defenses against Poisoning the Source

How can you defend yourself from a Trojan horse planted by an employee of your software development house? This is a particularly tough question, as you have little control over the development of the vast majority of the software on your systems. Still, there are things we can all do as a community to improve this situation.

First, you can encourage your commercial vendors to have robust integrity controls and testing regimens for their products. If they don't, beat them up[1] and threaten to use other products. When the marketplace starts demanding more secure code, we'll gradually start inching in that direction. Additionally, if you use a lot of open source software, support that community with your time and effort in understanding software flaws. If you have the skills, help out by reviewing open source code to make sure it is secure.

Next, when you purchase or download new software, test it first to make sure it doesn't include any obvious Trojan horse capability. Use

---

1. I don't mean to beat them up literally. I don't want to incite violence, for goodness sakes. By "beat them up," I mean give them a hard time. Challenge them. Yell at them. Let your software development vendors know how important secure code is to your operations.

the software tests we described in Chapter 11 to look for unusual open ports, strange communication across the network, and suspect files on your machine. With a thorough software test and evaluation process in house, you might just find some Trojan horses in your products before anyone else notices them. Communicate this information to the vendor to help resolve the issue.

If your organization develops any code in house, make sure your software testing team is aware of the problems of Easter eggs, Trojan horses, and intentional flaws. Sadly, software testers are often viewed as the very bottom tier of importance in the software development hierarchy, usually getting little respect, recognition, or pay. Yet, their importance to the security of our products is paramount. Train these folks so that they can quickly spot code that doesn't look right and report it to appropriate management personnel. Reward your testers when they find major security problems before you ship software. Be careful, though. You don't want to have testers working with developers to game the system and plant bugs so they can make more money. That's like having a lottery where people can print their own winning tickets. Carefully monitor any bug reward programs you create for such subterfuge.

Furthermore, ensure that your testers and developers can report security concerns without reprisals from desperate managers trying to meet a strict software deadline. Depending on the size of your organization and its culture, you might even have to introduce an anonymous tipline for your developers to report such concerns. By giving this much-needed additional attention to your software testers, you can help to squelch problems with Trojan horses as well as improve the overall quality of your products.

To infuse this mindset throughout the culture of your software development teams, consider transforming your test organization into a full-fledged quality assurance function. The quality assurance organization should be chartered with software security responsibility as a facet of quality. Build your quality assurance process into the entire cycle of software development, including design, code reviews, and testing. You should also impose careful controls on your source code, requiring developers to authenticate before working on any modules. All changes should be tracked and reviewed by another developer. Only with thorough quality processes and source code control can we improve the situation associated with untrustworthy source code.

Co-opting a Browser: Setiri

You know, attackers don't have to poison source code to implement a trojan. Instead, they can co-opt software already installed on a system. As we saw in the section on deceptive naming, impersonating an Internet browser is a very useful Trojan horse technique, but the issue goes way beyond mere name games. In February 2002, two very bright developers pushed this trend of Trojanizing browsers to the extreme by creating a tool that they later named Setiri. After installing Setiri on a victim machine, a bad guy can remotely control the system, executing arbitrary commands on the victim's box. In that regard, Setiri is a pretty standard backdoor, like many of the specimens we discussed throughout Chapter 5. However, the tool goes a lot further than most backdoors and Trojan horses in the way that it hides the communication channel with the attacker. These extreme hiding techniques make detecting and blocking the backdoor very challenging, and finding the actual location of the attacker highly difficult.

Setiri represents an extremely stealthy Trojan horse backdoor that works by co-opting the Internet Explorer browser included on most Windows machines. Setiri hasn't been released to the public yet, thankfully. However, its authors, Roelof Temmingh and Haroon Meer, have demonstrated their code at a variety of information security and hacker conferences. Others have independently implemented similar ideas, such as the IEEvents.pl tool by Dave Roth at *www.roth.net/perl/scripts/scripts.asp?IEEvents.pl*. In fact, the very clever techniques implemented in Setiri are just starting to trickle down into other tools that are being used in real-world attacks.

Setiri Components

So, what are these clever techniques? First, the Setiri code consists of two components, as shown in Figure 6.9: the connection broker code and the Setiri backdoor code. The connection broker is installed on a Web server of the attacker's choosing anywhere on the Internet. This system could be the attacker's own Web server, or, better yet (from the attacker's perspective), it could be on someone else's Web server conquered by the attacker. The connection broker code is simply a few Common Gateway Interface (CGI) scripts, installed on the Web server. These scripts do not impair the normal functioning of the Web server, and could be added to any Web server the attacker has conquered or has been given the privileges necessary to write these scripts. As we shall see, the connection broker will be used to temporarily hold the attacker's commands and responses, as well as obscure where the
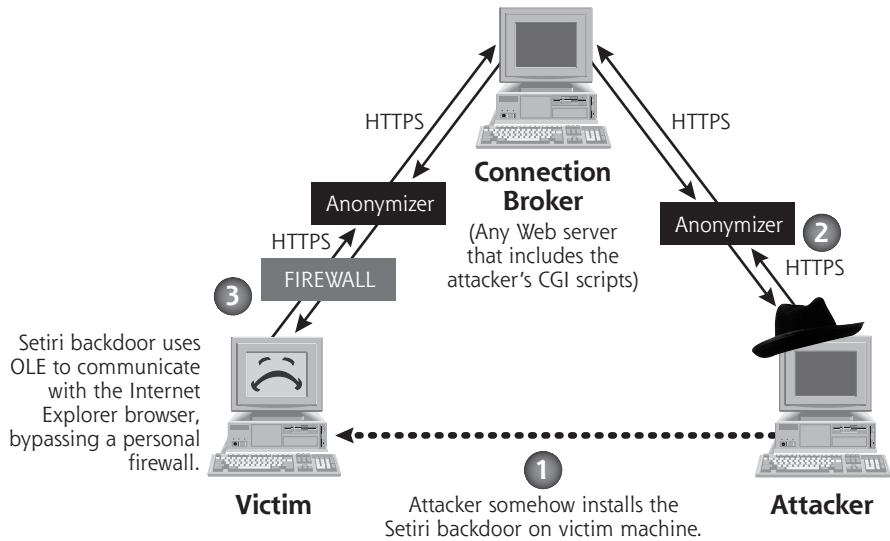
Figure 6.9
The Setiri Trojan horse browser architecture: This tool represents a new level of Trojan horse stealthiness.

attacker comes from. Attackers use the connection broker to launder their actual location on the Internet, making them virtually untraceable.

The second component of Setiri is the backdoor itself, which is installed on the victim's computer, shown with a sad face in Figure 6.9. In step 1, the attacker could install this code on the victim machine by tricking a user into running an executable built with a wrapper tool. Alternatively, attackers could install the Setiri backdoor on the victim themselves, given physical access to the machine or through any attack that executes a command on the victim machine, such as a buffer over-flow exploit.

Setiri Communication

The attacker accesses the connection broker using a standard browser on the attacker's machine. All communication occurs via the HTTPS protocol, which encrypts the data in transit across the network. Furthermore, the attacker uses an anonymizing Web surfing service, such as the one available at *www.anonymizer.com*, to strip all information going to the connection broker about where the attacker is located. These anonymizing services hide a Web surfer's location from Web servers by removing all information associated with the browser, such

as the source IP address, browser type, and any user profiling information stored in cookies. Essentially, these services function as intelligent Web proxies that users surf through to hide their identity and location.

In step 2 of Figure 6.9, the attacker surfs to the connection broker and types commands into HTML forms generated by the CGI scripts on the connection broker machine. These commands will be executed later by the Setiri backdoor. There are only three commands supported by Setiri:

- Upload a file.
- Execute a program.
- Download a file.

That's it! Although these commands might seem pretty simple, they really are all an attacker needs to have complete domination of a victim system. With the ability to upload files, an attacker can install a variety of other attack tools on the victim machine, such as the Netcat program we discussed in Chapter 5. The attacker can also execute any local commands on the victim machine and store the results in a file on the victim. Then, by downloading the file, the attacker can get the results of the commands.

Things get really interesting in step 3. To retrieve the attacker's commands from the connection broker, the Setiri backdoor code uses Microsoft's Object Linking and Embedding (OLE) technology to interact with the Internet Explorer browser on the victim. OLE is a framework that lets different objects and applications running on a machine communicate with each other. The Setiri backdoor uses OLE to send messages to Internet Explorer running in an invisible mode, telling the browser to surf to the connection broker and retrieve a command. The Internet Explorer browser supports both visible and invisible window panes on the system's GUI. Invisible browser windows are a rather dubious function that allows the browser to access information from the Internet using a whole new window without crowding the user's screen. Some Web applications use these invisible panes to make connections, run scripts, or conduct other activities that don't need to interact with the user. The Setiri backdoor uses an invisible browser window to poll the connection broker for commands at a periodic interval configured by the attacker, usually every 60 seconds or so. In effect, the backdoor on the victim machine uses Internet Explorer to surf out to the connection broker to pick up the attacker's commands.

So far, you might be thinking that this sounds like a pretty standard backdoor, like those we discussed in Chapter 5. "What's the big deal?"

you might ask. The big deal involves Setiri's use of Internet Explorer to retrieve commands, and how this operation bypasses many widely used security tools. Many users and organizations are deploying personal firewalls on desktop and laptop systems to limit the flow of data into and out of those machines. As we saw in Chapter 5, personal firewalls block unauthorized access by controlling which applications can send and/or receive data on the network. Many personal firewalls include a list of applications that can use the network on specific ports; all others are blocked.

Here's the rub. Most personal firewalls are configured to allow an *Internet browser* to access the network. After all, without allowing the browser to access the Internet, the user couldn't surf the Internet, severely limiting the usefulness of the computer. However, as long as the victim machine's browser can access the Internet, the Setiri backdoor can use the browser to reach across the network and get the attacker's commands from the connection broker! In this way, Setiri bypasses personal firewalls, Network Address Translation (NAT) devices, proxies, and stateful firewalls by running an invisible browser on the victim's PC. These security components do not know whether a user is accessing the network or the Setiri backdoor is retrieving commands from the connection broker. As an added bonus, Setiri hides the victim's location from the connection broker by using the Anonymizer Web site as well. To completely confound the victim, all communication between the Setiri backdoor and connection broker is encrypted using HTTPS.

Let's analyze what the victims of this Setiri Trojan horse would see. First, suppose someone installs the Setiri CGI scripts on your Web server. You'd see a few extra scripts in your CGI directory, as well as Web access via HTTPS through the Anonymizer service. You wouldn't be able to determine the location of the attacker or the Setiri backdoor.

Next, consider what the backdoor victim sees. On the end system running the backdoor, the victim would not be able to see the Setiri client or the invisible browser on the GUI, as each runs hidden in the background. Fport wouldn't show the Setiri client, as it isn't receiving or sending data on the network itself. It's only using OLE to communicate with the browser, which is expected to be using TCP ports to transmit data. Fport can show a browser process communicating across the network, but that's a pretty common occurrence. From a network perspective, all data would be masked via HTTPS. However, the network firewall on the victim's machine would be able to see the connection going to the Anonymizer Web site. This latter element is really the only item that indicates something fishy might be going on, depending on how commonly the Anonymizer Web site is used at this organization.

Setiri Defenses

So how do you defend against Setiri and other tools that borrow its ideas? To get started, you should configure your firewall and/or outgoing Web proxies to block access to various anonymizing Web sites, such as those shown in Table 6.5. The vast majority of Internet users in your organization have no business masquerading their Internet browsing activities. Now, depending on your particular industry and individual job roles, a handful of users in your organization might in fact require access to anonymizing services. For instance, your organization might have some select employees whose jobs require them to visit the competitions' Web sites, foreign government sites, or even hacking tool distribution centers to conduct research covertly. You can configure your filters to allow this limited number of employees to access specific sanctioned anonymizer sites.

Table 6.5
A Brief List of Anonymizing Web Sites*

Service Name	URL	Services Provided
Anonymizer	*www.anonymizer.com*	This service was one of the first anonymizers, and remains one of the most popular. It offers free anonymizing services, which are extremely slow, as well as much higher bandwidth commercial services. Both HTTP and HTTPS access are available.
idMask	*www.idmask.com*	This site provides free and commercial services, but currently supports only HTTP (not HTTPS).
SamAir Resources	*www.samair.ru/proxy/*	This free site maintains a giant list of thousands of free, anonymous proxies located around the world, supporting both HTTP and HTTPS access.
Anonymity 4 Proxy	*www.inetprivacy.com/a4proxy/*	This site provides commercial software that a user loads onto a machine that automatically directs all HTTP and HTTPS requests to an updated list of free proxy services.

Table 6.5
A Brief List of Anonymizing Web Sites* (Continued)

Service Name	URL	Services Provided
The Cloak	*www.the-cloak.com*	This free service offers both HTTP and HTTPS access.
JAP	*anon.inf.tu-dresden.de*	This is another anonymous proxy, hosted out of Germany.
Megaproxy™	*www.megaproxy.com*	This commercial anonymizer offers monthly or quarterly subscriptions.

* This list is by no means exhaustive, but it lists the most popular Web sites that strip off the source IP address and other ways of identifying the source of Web traffic.

To accomplish this filtering, you can block individual sites by loading their domain name and/or IP address ranges into your firewall or Web proxies. Alternatively, you could deploy software that filters out Web requests for sites that your users shouldn't be accessing, such as porn, games, hacking sites, and anonymous Web services. Many such tools are available, but the market leader for such Web filtering software is the commercial tool SurfControl, which includes a specific filtering category called "Remote Proxies." SurfControl includes a nifty free feature on its Web site that allows anyone on the Internet to check if a given URL is included in their filtering rules and to determine which type of rule the given Web site triggers. You can check out this feature at *http://mtas.surfcontrol.com/mtas/MTAS.asp.* I've frequently used this free service to get a feel for the nature of some URLs without having to actually surf to the possibly malicious Web site.

Of course, none of these filtering solutions will stop access to every single anonymous Web service on the planet. Highly intelligent users and attackers continuously find creative ways to dodge such filters. Vast numbers of small, private Web anonymizers are continually being added to the Internet, as indicated by an amazingly huge list of these sites at *www.samair.ru/proxy/.* An attacker could even reconfigure a Setiri-like tool so that it surfs directly to the connection broker instead of using an anonymizer. So, although you cannot use filtering to *completely* squelch this problem, you'll still get rid of much of the riffraff by strictly controlling access to the most popular anonymizing services. Also, when a user tries to get access to one of these popular blocked sites, the log of that attempt will alert you in advance to a possible problem with that employee. You can then, with appropriate writ-

ten permission from your Human Resources (HR) organization, keep a closer watch on other potentially malicious activities associated with that employee. Make sure HR signs off on monitoring that targets any individual person, though, or else you could get into serious trouble both inside your organization and possibly with the law for privacy violations!

In addition to blocking anonymizing Web sites, other Setiri defenses include keeping your antivirus tools widely deployed and up to date, as we discussed in detail in Chapter 2. Setiri has not yet been released publicly, so there aren't any antivirus detection signatures for it at this point. However, antivirus vendors do a pretty decent job at keeping their tools up to date with the latest malicious software. I expect antivirus tool vendors to release signatures for Setiri soon after a public release. Before that time, however, there are a lot of other Trojan horse backdoors with lesser functionality than Setiri that antivirus tools can detect today. With up-to-date antivirus tools, you can prevent their installation and detect attackers' attempts to use these tools in your organization.

Another possible longer term defense against Setiri involves changes to the fundamental functionality of the Internet Explorer browser itself. Sadly, you can't make these changes yourself, because they require the browser vendor to modify source code and release a new browser version. Remember, Setiri works by creating an invisible browser window pane to retrieve commands across the network.

If Microsoft altered Internet Explorer to limit the actions of an invisible browser, a significant component of this problem would go away. Why should an invisible browser window be able to surf anywhere on the Internet in the first place? This capability seems to have very limited benefit and enormous security risks. There are rumors in the computer underground that Microsoft is considering implementing such a solution in future versions of Internet Explorer, although Microsoft hasn't made a public comment on the issue as of this writing. In the meantime, make sure you keep your browsers patched, applying the latest service packs and fixes regardless of which browser you use (Internet Explorer, Conqueror, Netscape, Mozilla, Opera, Lynx, etc.)

Another interesting option for dealing with code like Setiri involves a concept we originally discussed in Chapter 4, namely cross-site scripting. We might be able to turn the tide against the bad guys and utilize cross-site scripting to undermine their own technology and pierce the cloaking features of Setiri. Suppose you discover a Setiri-like program running on one of your machines. You could send a little snippet of Java-

Script to the connection broker as the result of a command. When the attacker retrieves the results of the command from the connection broker using a browser, the JavaScript would run in the attacker's browser itself, provided that the attacker's browser is configured to automatically run JavaScript. We could create a JavaScript that e-mails law enforcement agents a message saying, "Come and arrest me, big guy!" This e-mail, created by the JavaScript running in the attacker's browser, would originate at the attacker's machine, and could include information about the attacker, such as the source address. Although I've never seen this technique used by law enforcement, and significant civil liberties issues are involved, it still remains an intriguing possibility.

Hiding Data in Executables: Stego and Polymorphism

So far in this chapter, we have focused on Trojan horses that masquerade some sort of remote control or command shell backdoor, but that's not the full extent of what Trojan horse techniques could disguise. Beyond hidden executables for remotely taking over a system, attackers could embed hidden messages inside programs. The program looks like a nice, happy executable, but in fact contains a hidden message. Therefore, this executable fits our definition of a Trojan horse, and also acts as a covert channel for communication.

The art and science of hiding messages is called *steganography*, from the Greek words for hidden writing. Steganography is often referred to as *stego* for short. To get a feel for its use, consider this scenario. Suppose a military general wants to send the message "Attack at dawn" to another general without their mutual adversary knowing about their communication. Of course, they could just encrypt the message so the adversary wouldn't know for sure whether the message says "Attack at dawn" or "Gee, you smell funny." Still, by analyzing the traffic between the two generals and seeing the encrypted message sent across the network, the adversary could figure out that something significant is afoot.

Traditional cryptography mathematically transforms the message so the adversary cannot read its contents, but can still see that some form of information is being exchanged. Steganography conceals the message so that the adversary doesn't even know that there is data being exchanged in the first place. Of course, clever generals would use steganography to hide a message *and* cryptography to transform the message just in case it is discovered. Detecting and eliminating all such covert communication is an extremely difficult endeavor.

Steganographic techniques have been used for thousands of years. However, in the field of computer science, they've really gotten a lot more attention in just the last few years. Typical computer steganography techniques hide information in pictures, such as BMP, JPEG, or GIF files. Other techniques hide information in sound files, such as MP3, WAV, or other formats. However, newer techniques stash information inside of computer executable programs without altering the program's function or size.

Hydan and Executable Steganography

In February 2003, Rakan El-Khalil released a program called Hydan to stash messages inside of executable programs written for x86 processors, such as Intel's or AMD's popular chips. The tool stores hidden information inside of executables for the Linux, Windows, NetBSD, FreeBSD, and OpenBSD operating systems. Available at *www.crazyboy.com/hydan*, Hydan implements this steganography by using polymorphic coding techniques. There's that fancy-sounding word again: polymorphic. We saw it before in Chapter 2 associated with viruses, and in Chapter 3 on worms. Remember, polymorphic code simply means that you can have multiple different pieces of computer code that all do the exact same thing. By carefully selecting certain variations of that functionally equivalent code, we can transmit a message in the executable. In other words, there's more than one way to skin a cat, and Hydan embeds messages by selecting specific cat-skinning techniques. Figure 6.10 illustrates how Hydan works.

The process starts with an executable program, such as a word processor, backdoor, or operating system command. Really, any x86 executable will do. Hydan's not too picky. Hydan also needs some secret information to hide, such as a message, a picture, some other executable code, or anything else. The user feeds both the executable and the secret information into the Hydan tool. Hydan prompts the user, asking for a pass phrase that can be used to encrypt the message before the stego process ensues. Hydan first encrypts the message with the blowfish encryption algorithm using this passphrase as an encryption key.

Hydan then works its magic by embedding the encrypted secret information inside the executable program. For this embedding, Hydan defines two different sets of CPU instructions that have exactly the same function, Set 0 and Set 1. For example, when you add two numbers, you can use the `add` or `subtract` instructions. You could add X and Y, or you
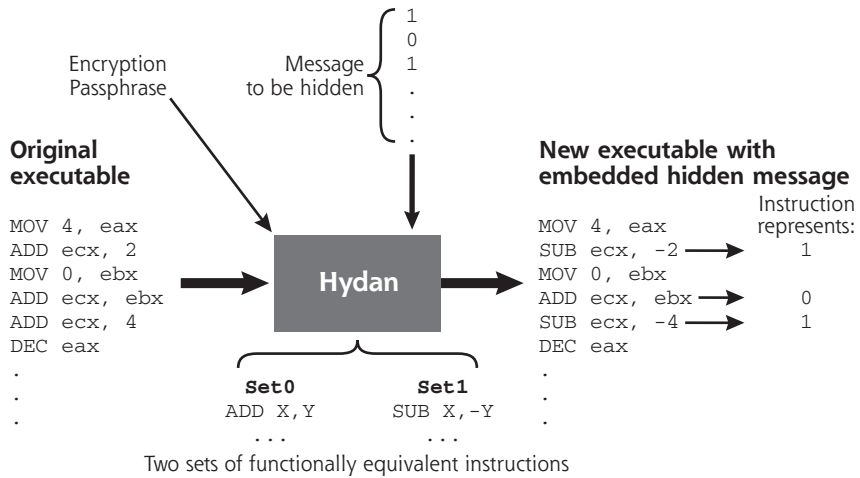
Figure 6.10
How Hydan embeds data using polymorphic coding techniques.

could subtract negative Y from X. If you remember your high school algebra class, these two different instructions have the exact same result. So, we could put the `add` instruction into Set 0 and the `subtract` instruction into Set 1. Hydan takes the original executable and rebuilds it by choosing instructions from Set 0 or Set 1 based on the particular bits from the secret information to hide. It looks for the first instruction in the executable that is represented in one of the sets, such as an `add` instruction. If a given bit to be hidden is a zero, we will choose an instruction from the Set 0 group of instructions to replace the existing instruction. If the bit is a one, we will choose a functionally equivalent instruction from Set 1.

Then, after the entire code is rebuilt with instructions from these two sets, the new executable is rewritten to the hard drive. Because each instruction in Set 0 is chosen so that it has the same size as its functionally equivalent counterpart in Set 1, the resulting executable program has exactly the same size, and exactly the same function! However, it is a brand new piece of code. Most important, by using Hydan again in reverse mode, the original secret information can be retrieved from the resulting executable if the proper passphrase is typed in.

Hydan's stego technique, implemented with polymorphic instructions, isn't the only way to hide messages, of course. Data can be embedded inside of nonexecutable files as well, such as pictures, sounds, and other data types. For these other types of files, the stego

technique might alter the color or sound frequency distribution of the image or other mathematical properties to hide data, using techniques analogous to Hydan's instruction substitution. Because our focus in this book is on malware (e.g., malicious programs), we've addressed hiding data inside of programs. For more information about stego techniques for other types of files, I highly recommend that you consult Eric Cole's book, *Hiding in Plain Sight* [8].

Hydan in Action

Look at Figure 6.11 to get a feel for Hydan in action on Linux. The Windows version of Hydan is virtually identical to this Linux version. In this example, I created a small file called hideme.txt that contains my super-secret text. I then used Hydan to embed hideme.txt inside a GUI calculator named xcalc. Note that it put 40 bytes into the file, but it could have stored up to 72 bytes. The total storage capacity of an executable is based on the number of adds and subtracts, as well as other
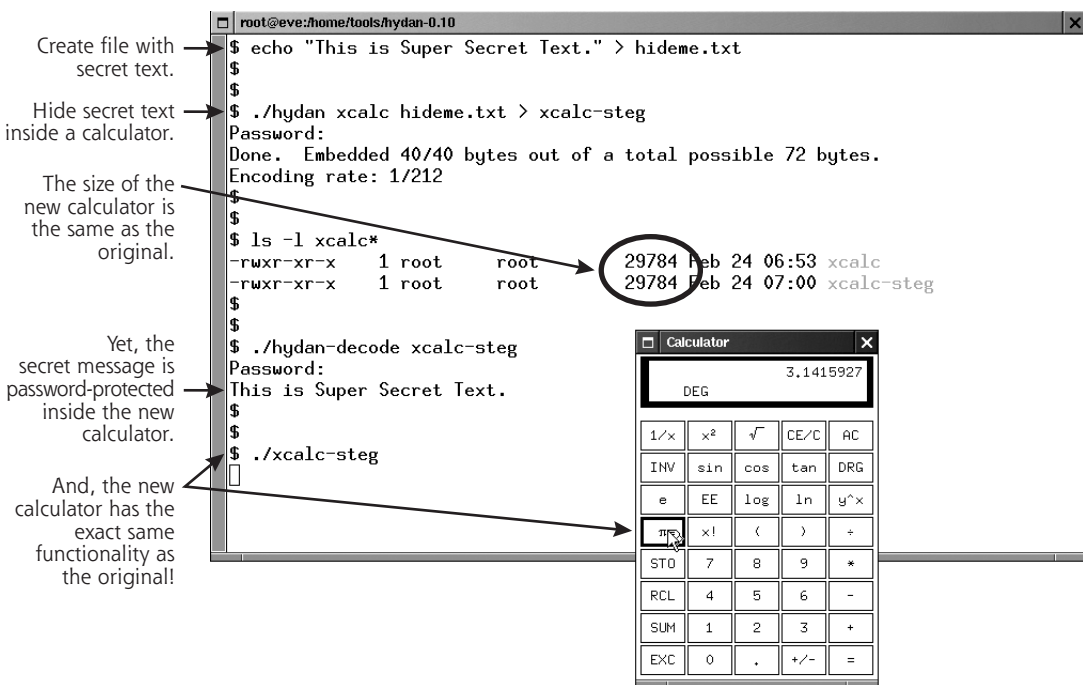

Figure 6.11
Hydan in action on Linux: Hydan encrypts and hides a message inside of a calculator program.

related polymorphic instructions, in that executable. After it ran, Hydan generated a new copy of the xcalc tool, which I named xcalc-steg. This version is exactly the same size (29,784 bytes) and has the same functionality as the original xcalc. I ran a copy of the new calculator so you can see that it is, in fact, a calculator. However, this xcalc-steg also includes my hidden super-secret message. By using the hydan-decode routine, I can recover my original message, the contents of hideme.txt. So, the new calculator program is now a Trojan horse: It still runs as a program, but I could send this program to other people to transmit my secret information.

Hydan is capable of stashing one byte of the secret information in approximately 150 to 250 bytes of executable code, depending on the particular instructions used by that executable. That's not nearly as efficient as more traditional stego techniques for hiding data inside of pictures (which often get up to one byte hidden in 20 bytes of image). Still, it's not a bad ratio for hiding data.

It's also important to note that Hydan does alter the statistical distribution of instructions used in the Trojan horse executable. By creating a histogram showing how frequently various instructions are used in that executable, an investigator could determine that the program just doesn't look right. For an analogy, think of the use of various letters of the alphabet in standard English text: There are many uses of e and t, but not very many uses of q or z. We could graph the relative occurrences of letters to create a histogram. By analyzing the histogram of a sample file, we could get a good feel for whether the sample is English text or something else, such as an encrypted file, an executable, or even non-English text. If the histogram matches what we'd expect for the alphabetic distribution for English, it's probably an English text file.

You could do a similar analysis with x86 instructions. "Normal" programs have a certain predictable usage pattern for various instructions. There are lots of `add` and `move` instructions, but somewhat fewer `subtract`s. In this way, an analyst or automated tool might be able to detect the presence of hidden data in an executable without knowing what that hidden data is. This statistical analysis technique would certainly work, but no current tool is available for such analysis on executable programs. For similar types of analysis of images with hidden data, however, there is a popular analysis tool called StegDetect by Niels Provos available at *www.outguess.org/detection.php*.

You might be wondering what an attacker could do with a Hydan-generated program containing hidden text. There are several possibilities, including the following:

- *Hiding Information for Covert Communication:* Two people might have login access to a single machine somewhere on the Internet. One user could cram secret information inside a user program, service, or even a kernel module and install the resulting program on the shared machine. The other user could log in, analyze the appropriate executable, and retrieve the message. An eavesdropper looking to see if the two parties are communicating might not notice this subtle covert channel.
- *Watermarking or Signing an Executable:* By using Hydan, a software developer could mark an executable with an identification code unique to that instance of the program so that a copy of the program can be easily correlated with the original. Furthermore, by using Hydan to embed a digital signature inside the executable, a user can verify that he or she was the author of an executable. Suppose I'm a software vendor. If I ever want to prove that I was the one who compiled a particular version of a program, I can digitally sign a document saying so, and then embed this document inside of the executable itself. When I want to prove that I compiled the executable, I could extract the document and show that it was signed with my own key. This technique could be applied to copyrighting mechanisms and digital rights management for executables.
- *Evading Signatures:* Finally, and perhaps most ominously, the technique could be extended to implement evasion of signature-based antivirus tools and network-based IDS tools. Many antivirus and IDS tools look for specific sequences of bits to identify malicious software. By using the polymorphic techniques included in Hydan, an attacker can morph an executable so that it no longer matches the signatures and therefore evades detection. Simply embedding a different hidden message totally alters an executable so it won't match an existing signature. It's important to note that Hydan doesn't yet do this. It lacks enough different types of polymorphic substitutions to do effective signature evasion. When Hydan is used, enough of the original program survives so that signature matching still works. However, in the near future, these Hydan concepts could be extended to achieve true signature evasion … stay tuned!

Hydan Defenses

To check if someone has been altering your critical executables with a tool like Hydan, you really need to use a file integrity checking tool, such as Tripwire or AIDE as we've highlighted in Chapters 2 and 5. We'll discuss these tools briefly here, but will cover them in far more depth in Chapter 7 when we deal with RootKits. At this point, though, we need to note that these file integrity checking tools create a database of hashes of your critical system files, which you can store on secure media (e.g., a write-protected floppy disc or write-once CD-ROM). Then, you run a check against this database on a regular basis (every hour, day, or week) to see if someone has altered your files. If you spot changes, you need to figure out whether a system administrator or an attacker made them. If an attacker tries to use Hydan to embed data in any of your critical executables, you'll notice the change the next time you run the file integrity checker. Of course, this technique will only detect problems associated with those programs that you actually analyze with the file integrity checking tool, such as your operating system commands and important applications. Changes to any other programs on your system would fly under your file integrity checking radar.

Conclusions

In battle, soldiers use camouflage and stealth to evade detection by their adversaries and gain the upper hand in a conflict. Trojan horses provide a similar kind of cover in the world of computer attacks. From the simple name games we discussed at the start of this chapter to the highly sophisticated Setiri methods of co-opting browsers, Trojan horses let bad guys gain access to and operate on your computer systems without your knowledge. Because they can be so effective, we see numerous attacks in the wild using the techniques described throughout this chapter. Indeed, more often than not, attackers use at least some form of Trojan horse subterfuge to hide.

However, if you look at the Trojan horse techniques described in this chapter, they all rely on adding software to the victim machine to accomplish the attacker's goal. In our discussion so far, the attackers place new programs on the victim machine and disguise them as legitimate code. In the next chapter, we'll move beyond this use of additional disguised programs into the area of RootKits, an even nastier form of Trojan horse. With a RootKit, attackers don't add new programs to your machine. Instead, they replace or modify the existing

programs on your box, especially those associated with your operating system. By supplanting your existing programs with malicious code, RootKits are far more insidious than anything we've covered so far. So, go grab a latte, fasten your seat belt, and get ready for RootKits.

Summary

This chapter discussed Trojan horses, which are computer programs that appear to be benign, but really include hidden malicious functionality. The term Trojan is often abused, being applied to any type of backdoor. However, the term should only apply if that backdoor is disguised as some benign program. Attackers use Trojan horses to sneak onto systems and hide there, without triggering the suspicion of administrators or users.

One of the simplest Trojan horse strategies involves giving a malicious program the name of a benign program. By including many spaces between the program's name and suffix on a Windows machine, such as "just_text.txt   .exe," an attacker can trick some users into running an executable application, thinking it's just text. Also, attackers choose program suffixes or names from those programs that would normally be installed and running on the victim machine, such as init, inetd, iexplore, and notepad. To defend against this technique, system administrators must become very familiar with their systems, so that they know what programs should normally be running on them. With this detailed familiarity, a counterfeit can be spotted and investigated. The Fport tool helps this process by showing which programs are listening on TCP and UDP network ports. Additionally, filter .EXE, .COM, .SCR, and other related programs at your Internet gateway.

Attackers also use wrapping programs to combine two or more executables into a single package. The victim is duped into thinking that the combined package is sweet and innocent. When it's run, however, the package first installs the malicious code, and then executes a benign program. Wrappers let an attacker create Trojan horses by marrying malicious code to benign programs, without writing a single line of code themselves. Antivirus tools are one of the best defenses against wrapper programs.

Attackers are also increasingly targeting software distribution channels to distribute Trojan horses, including snail-mail and Web site downloads. The main OpenSSH, sendmail, and tcpdump Web sites were all conquered by an attacker and used to distribute malicious

code. The Trojan horse built into the tcpdump distribution communicated with an attacker across the network and supported shoveling a shell back to the attacker. To defend against this type of attack, make sure you check the integrity of all downloaded software across multiple mirrors using MD5 hashes. Also, test software before putting it into production to look for squirrelly functionality, such as backdoor listeners and sniffers.

If attackers get jobs with or break into software development firms, they could even Trojanize the source code of a product, infecting unsuspecting users of the code with malware. This trend is exacerbated by the enormous complexity of today's software, the limitations of software testing (as exemplified by the large number of Easter eggs), and the move toward international software development. To defend against this attack vector, make sure you have strong integrity controls and test regimens for software used in your environment.

The Setiri tool is an extremely powerful Trojan horse. Although it was never publicly released, concepts from Setiri are trickling into other Trojan horse tools. The Setiri code runs an invisible Internet Explorer window to send requests for commands through a personal firewall and any network filtering devices to a connection broker. The attacker plants commands on the connection broker for the Setiri victim to execute. To defend against Setiri and related tools, make sure to keep antivirus programs up to date and consider blocking access to the more popular anonymizing Web surfing proxies.

The Hydan tool embeds messages of any kind inside of executable programs using polymorphic coding techniques. Hydan stores data by selecting from different sets of functionally equivalent instructions. To defend against tools like Hydan, guard the integrity of your critical system files using tools such as Tripwire and AIDE.

References

[1]   "Win2K Processes," *http://users.aber.ac.uk/anw1/processes.html.*

[2]   David A. Solomon and Mark E. Russinovich, *Inside Microsoft Windows 2000, Third Edition, Microsoft Press, 2000.*

[3]   CERT Coordination Center, "Wuarchive Ftpd Trojan Horse," April 6, 1994, *www.cert.org/advisories/CA-1994-07.html.*

[4]   CERT Coordination Center, "Trojan Horse Version of TCP Wrappers," January 21, 1999, *www.cert.org/advisories/CA-1999-01.html.*

[5]   Ken Thompson, "Reflections on Trusting Trust," *Communication of the ACM*, Vol. 27, No. 8, August 1984, pp. 761–763, *www.acm.org/classics/sep95/*.

[6]   Watts S. Humphrey, "Bugs or defects?" *http://interactive.sei.cmu.edu/news@sei/columns/watts_new/1999/March/watts-mar99.htm#humphrey*.

[7]   Kathryn Balint, "Software Firms Need to Plug Security Holes, Critics Contend," San Diego Union-Tribune, *www.signonsandiego.com/news/computing/personaltech/20020128-9999_mz1b28securi.html*.

[8]   Eric Cole, *Hiding in Plain Sight: Steganography and the Art of Covert Communication,* Wiley, 2003.