Chapter 11. Phase 5: Covering Tracks and Hiding

Every day, attackers take over Web sites by the dozens and tamper with their contents. A large number of such victims are archived at the Zone-H Web site (), which contains a virtual museum of Web vandalism attacks over the last several years. Some attackers want to create a big splash with a high-profile attack to establish a reputation, embarrass their victims, or to make a political point. Massive Distributed DoS (DDoS) attacks or vandalism of a major Web site can surely garner attention.

However, attackers who desire quiet, unimpeded access to computing systems and sensitive data conduct the vast majority of attacks. This class of attackers wants to stay hidden, so they can maintain covert control of systems for lengthy periods of time, stealing data, consuming CPU cycles, launching other attacks, or just keeping their valued access for use at a later time. In my experience, these silent system compromises far outnumber the instances of publicly observed attacks. With the large number of well-documented, high-profile Web tampering cases, consider that there are probably far more computer systems on the Internet that have been taken over by an attacker who silently hides in the background. Many companies, government agencies, universities, and other organizations are unwittingly providing a home on their computing systems for these silent attackers. In the course of investigating incidents, we routinely find networks of thousands or even tens of thousands of bots hidden from the owners of the host computers.

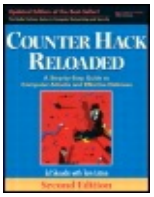
How are these attackers, who gain access on a system, hiding their tracks to avoid detection? In many cases, they don't have to hide. Over the past several years, the largest proportion of compromises have taken place on poorly maintained home computers connected to the Internet with broadband connections. These machines represent attractive targets for an attacker because they are often operated by individuals with little or no computer security expertise. However, public awareness of computer crime aimed at these "always on" home broadband machines is increasing. More and more home computer users are installing software designed to increase security without requiring any specialized computer security knowledge, such as antivirus, antispyware, and personal firewall tools. Although they certainly won't replace a knowledgeable system administrator, these "point-and-click" software security products have succeeded, to a greater or lesser degree, in somewhat increasing the security of the home computer market. The wild west isn't completely tamed, but the trajectory is improving.

Whereas a broadband-connected home computer might be a good target for an attacker intent on building a bot-net, the target of choice for the elite attacker is still a business network. Business networks, although providing

attractive targets, are also more closely monitored, requiring would-be attackers to hone their skills at covering their tracks. One of the main techniques for hiding on a system is utilizing a rootkit or backdoor program, as described in detail in Chapter 10, Phase 4: Maintaining Access. Beyond installing rootkits and backdoors to mask the changes made to the system, many attackers go further in covering their tracks, by modifying logs, creating hidden files, and establishing covert channels. This chapter describes these techniques for hiding on a system.

Counter Hack Reloaded, Second Edition: A Step-by-Step Guide to Computer Attacks and Effective Defenses

By Ed Skoudis, Tom Liston



.....

Publisher: **Prentice Hall**

Pub Date: **December 23, 2005**

Print ISBN-10: **0-13-148104-5**

Print ISBN-13: **978-0-13-148104-6**

Pages: **784**

Slots: **2.0**

Table of Contents | Index | Additional Reading

A **A** A

Hiding Evidence by Altering Event Logs

To avoid detection by system, network, and security administrators, many attackers alter the logs of their victim machines. Even though (as we discussed in the last chapter) the techniques used by rootkits are incredibly powerful and allow an attacker to mask practically all of their activities on the compromised machine, there will often be traces of the installation of the rootkit in the system's logs. Even an attacker who uses the most powerful and stealthy rootkit will need to remove particular events from the logs associated with gaining access, elevating privileges, and installing their backdoors or rootkits in the first place. Events such as failed login records, error conditions, stopped and restarted services, and file access and update times must be purged from the logs or altered to avoid having these activities spotted by an alert administrator.

Of course, on most systems, an attacker with sufficient access privileges (usually root or administrator) can completely purge or delete the log files. However, completely deleting the logs, blowing away all normal log data along with the insidious events, is very likely to be noticed. As the saying goes, even a blind squirrel finds an acorn once in a while. So it is with

system administrators: Even the worst one would probably notice a large chunk of time missing from the system logs. Attackers want to edit the system logs on a line-by-line basis to keep normal events in the logs, while removing the suspicious events generated by their activities. Obviously, the techniques used to modify system logs are very dependent on the system type. The techniques that an attacker will use on a Linux- or UNIX-based system will be dramatically different from those required for a Windows-based system, simply because the logging mechanisms themselves are quite different. We analyze attacks against logging in both Windows and Linux/UNIX. In Chapter 3, Linux and UNIX Overview, we briefly examined Linux and UNIX logging mechanisms, but before looking at how attackers manipulate and undermine logging on Windows systems, we'll first need to learn a little about how logging works under Windows.

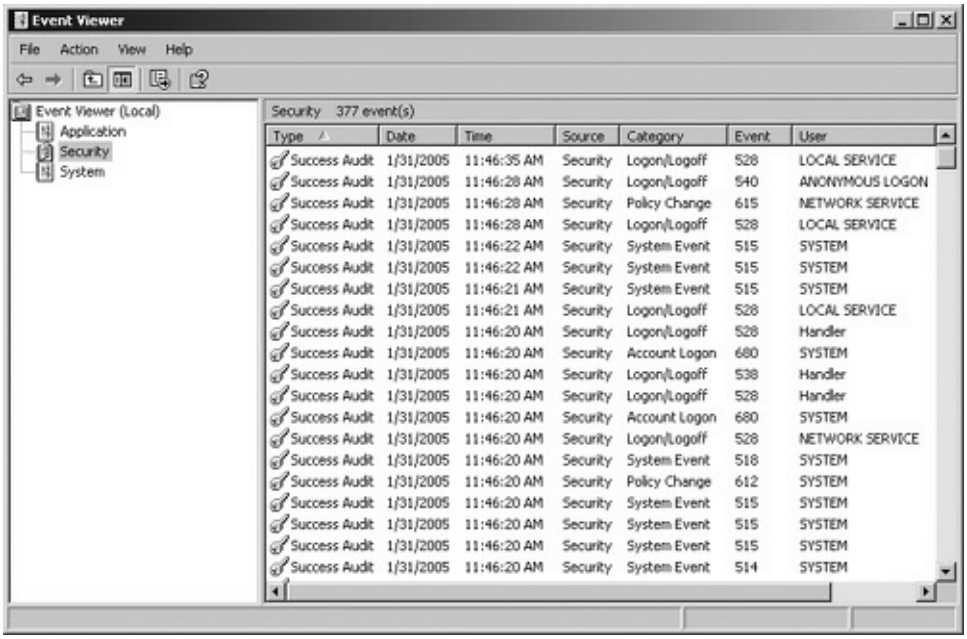
Attacking Event Logs in Windows

Event Logging in Windows

On modern Windows systems (that is, NT, 2000, XP, 2003, and later), the event logging service, known as EventLog, produces a set of files (with the suffix `.LOG`) where it temporarily places information about logged system and application events, such as a user logon, access control violation, service failure, and so on. This event information is constantly being written into files, which are named `SECURITY.LOG`, `SYSTEM.LOG`, and `APPLICATION.LOG`. The event information, however, doesn't stay in these `.LOG` files. Each of the `.LOG` files is periodically and automatically rewritten by Windows, which moves the event information into the system's main event logs: the `SECEVENT.EVT`, `SYSEVENT.EVT`, and `APPEVENT.EVT` files. It is actually these files that are the main event logs in Windows, and it is the `.EVT` files that are read by an administrator using the built-in Windows Event Viewer tool or a third-party log analysis tool. The Event Viewer tool, showing events from the `SECEVENT.EVT` file, is shown in Figure 11.1.

Figure 11.1. The Windows XP Event Viewer.

[View full size image]



The `SECEVENT.EVT` file stores security-related events, including failed logon attempts, policy changes, and attempts to access files without proper permission (if the system is configured to log such event types). The `SYSEVENT.EVT` file stores events associated with the system's functioning, and it is here that you'll find messages with details on the failure of a driver or the inability of a service to start. The `APPEVENT.EVT` file stores events associated with applications, such as databases, Web servers, or user applications. All of these files, which are written with a specific binary format, are what attackers want to target to cover their tracks. The `SECEVENT.EVT` file is most often targeted because it contains the majority of the events that attackers wants to remove, such as failed logon attempts and access violations that were triggered by their attempts to gain access to a system.

Altering Event Logs in Windows

To erase traces of activity, attackers would, at a minimum, want to alter `SECEVENT.EVT`. However, to be more confident that all traces of their activity are gone, the attackers would possibly want to alter the `SYSEVENT.EVT` and `APPEVENT.EVT` files as well. But all three `.EVT` files are "locked" on a running Windows machine, and cannot be opened or edited with a standard file-editing tool.

Completely deleting any `.EVT` file is no problem for anyone who has the proper rights (Manage Audit and Security Log) or permissions (such as Delete for the `\windows\system32\config` directory that holds these logs). But remember, a suddenly empty log should be highly suspicious and should attract the attention of even the most inattentive administrator.

Whereas a novice attacker might try to cover his or her tracks by simply deleting the .EVT files, a more experienced perpetrator will try to alter the event logs on a line-by-line basis.

With physical access to the Windows system, an attacker could simply boot the system from a CD-ROM and edit the log files on the main system partition using an editor with the capabilities of regenerating the correct binary format for the log data. The files are only "locked" and unalterable when the Windows system that generated them is running. As described in Chapter 7, Phase 3: Gaining Access Using Application and Operating System Attacks, a Linux boot CD-ROM image for editing the Windows password database can be found at . This tool allows an attacker to change the Windows administrator password by booting from a Linux CD-ROM. A boot disk for changing system logs on a line-by-line basis and regenerating the appropriate binary format for the .EVT log file could certainly be created using any of the Windows or Linux boot CDs available on the Internet, but there is currently no "prepackaged" tool like this in widespread use. Although not elegant and requiring a great deal of physical access, this technique could be remarkably effective in covering tracks.

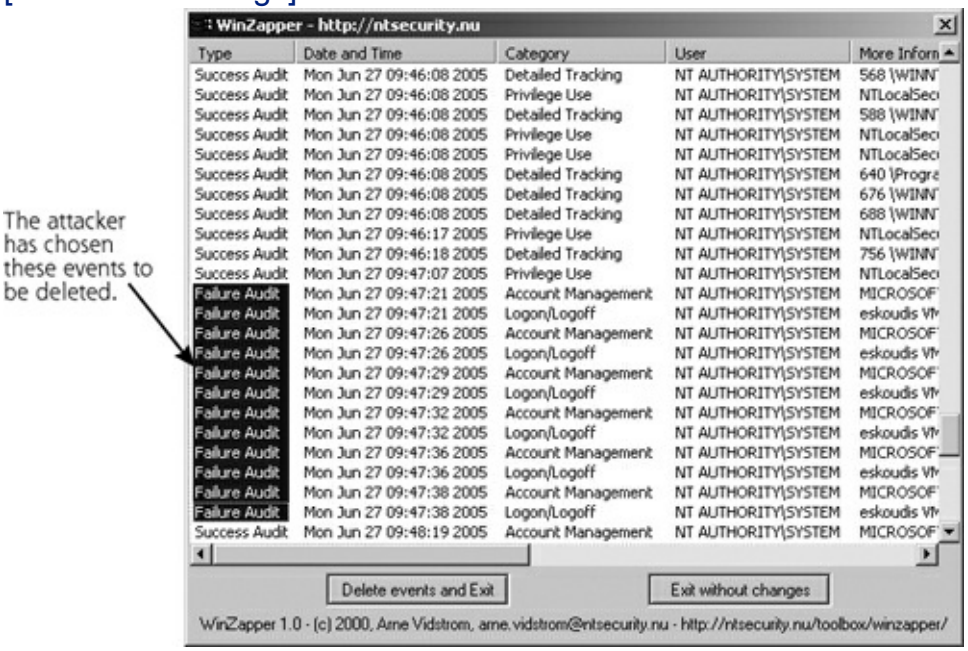
The most effective technique for altering system logs avoids booting the system from a CD-ROM and doesn't require physical system access. Event log editing tools are available that allow an attacker with administrator privileges to purge individual events from the SYSEVENT.EVT, SECEVENT.EVT, or APPEVENT.EVT file on a running Windows NT/2000 system (if you need to cover your tracks on a Windows XP or 2003 system, you're currently out of luck as there are no publicly released tools that work on those platforms ... yet). To accomplish this task for an attacker with administrative privileges, the tool first stops the Windows Event Logging service. It then changes the permissions on the .EVT files, and copies the data to memory for editing. The attacker can make any desired changes to the version of the event log in memory. The tool automatically calculates the new binary-formatted information (a crucial step in ensuring that the resulting event logs are not interpreted as corrupted by the Event Viewer). To clean up after the changes are made, the tool overwrites the .EVT files, resets their permissions, and restarts the Windows Event Logging service. When the administrators access the logs, they will see only the happy, pleasant image created by the attacker, with all suspicious events purged.

The WinZapper tool by Arne Vidstrom, allows an attacker to remove events selectively from the security logs of a Windows NT/2000 machine. Available at the WinZapper tool provides a point-and-click interface for deleting security events on a one-by-one basis. As shown in Figure 11.2, the attacker selects the specific events to delete, and clicks Delete Events and Exit. For

the changes that WinZapper makes to the event logs to take effect, however, the system must be rebooted to restart the EventLog service. There are other tools floating around in the computer underground that aren't really "public" that give an attacker the ability to alter the system logs without rebooting the machine. These tools typically focus on injecting code into the running EventLog service itself, giving the attacker the ability to alter the logs from within by undermining a piece of the operating system itself, in a fashion rather like the Windows rootkits we discussed in Chapter 10.

Figure 11.2. The WinZapper tool: Marked events will be selectively deleted from the Windows NT/2000 event logs.

[View full size image]



Attacking System Logs and Accounting Files in Linux and UNIX

Linux and UNIX System Logs

As described in Chapter 3, on Linux and UNIX systems, the vast majority of log files are written in standard ASCII text, not a specialized binary format like the logs of Windows machines. Thus, to edit Linux and UNIX logs, an attacker requires only root privileges or the privileges of a specific application that generates the logs, such as a Web server daemon. So, given this traditional Linux and UNIX log file environment, how do attackers cover their tracks? Some attackers employ automated scripts that pour through system logs, automatically deleting various items to cover their tracks. In the hands of an experienced attacker, these automated log editing scripts can quickly and efficiently hide any evidence of an attack. On the other hand, script kiddies often attempt to run such automated scripts on the wrong flavor of Linux or UNIX, resulting in attempts to edit or delete files

that do not exist on that particular flavor. This then creates a series of additional log entries, documenting these errors, making the attacker look pretty ridiculous in the process. Given the myriad differences in logging on various Linux and UNIX varieties, a standard log editing script will likely fail unless it is run on nearly the same version of the same Linux or UNIX variety for which it was designed.

How do more sophisticated attackers, the ones who don't need such scripts, cover their tracks? The attacker typically begins by looking at the `syslogd` configuration file, normally found in `/etc/syslog.conf`, to determine where the log files themselves are located. This configuration file tells `syslogd` where in the file system to put the logs. Once the log location is discovered, an attacker with root privileges (which might have been obtained through exploiting a buffer overflow or other attack) can directly edit the logs. Because the logs are plain ASCII text, with root privileges, attackers can alter the log files by using their favorite editor (such as vi, emacs, gedit, pico, or any other text editing tool). Sophisticated attackers will systematically go through the log files and remove entries associated with their gaining access to the system (such as failed login attempts or specific application error messages). Because the files are text, rather than a binary format, they can be altered and saved without any indication of file corruption.

Altering Accounting Entry Files in Linux and UNIX

Beyond the main log files, as described in Chapter 3, the main accounting files in Linux and UNIX are the `utmp`, `wtmp`, and `lastlog` files. Whereas the vast majority of Linux and UNIX log files are written in standard ASCII format, the `utmp`, `wtmp`, and `lastlog` files are written with a special binary format. The `utmp` and `wtmp` files are stored as so-called `utmp` structures, and `lastlog` is stored in a variety of different formats on different Linux and UNIX machines. If an attacker attempts to edit these files using a standard text editor, the files will appear corrupted and cannot be properly read by the system (using `who`, `last`, and other commands). Of course, because the files are written in a binary format, the attacker will only see gibberish anyway when opening them in a standard editor.

To edit these accounting files, an attacker must use a tool that can read and rewrite the special binary format that they use. An attacker can choose from several tools, with a complete inventory available at Particular tools are often fine-tuned for specific varieties of Linux and UNIX. In particular, the tool `remove`, written by Simple Nomad, allows for removing entries from `utmp`, `wtmp`, and `lastlog` for several UNIX systems. The `remove` program also allows an attacker to change the last login time, location, and status of

any users to whatever the attacker desires by editing the UNIX `lastlog` file. Other similar tools include wtmped, marry, cloak, logwedit, wzap, and zapper. Many of these log and accounting editing tools are included as standard components of the rootkit distributions discussed in Chapter 10.

Altering Linux and UNIX Shell History Files

One additional type of accounting and logging of particular concern to attackers is individual users' shell history files. The shell history file stores a list of all commands recently entered by the user into the command line. Whenever you type something at a Linux or UNIX command prompt, your shell (if it is configured properly) stores the command that you typed, maintaining a history of your interactions with the system. Usually, the shell history contains the previous 500 or so commands, although this is configurable. The command shell uses this history to allow the user easy access to previously entered commands, making repetitive command sequences much easier to enter.

If an attacker takes over a user's account, or creates a brand new account to attack from, the shell history file will contain a list of all commands entered by the attacker. Shell history files are typically stored in individual users' home directories, and have names such as `.bash_history`. For example, the following list shows the shell history from a user that has been messing around with the `/etc/shadow` file, where encrypted user passwords are stored:

```
ls
vi /etc/shadow
```

These commands were typed into the command line by the attacker and dutifully stored in the shell history file by the command shell program. We can see that the attacker first executed the `ls` command to get a listing of the contents of the current directory. Then, the attacker used the text editor, vi, to view and possibly alter the `/etc/shadow` file. The attacker might have changed a password or simply looked through the file for other account names and password hashes. After snagging a copy of the shadow file, the attacker might have started cracking the password representations.

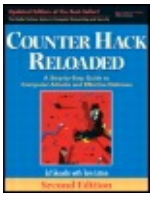
Like standard UNIX log files, shell histories are written in plain ASCII, and can be easily edited using the attacker's favorite text editing tool. Wiley attackers remove all lines associated with their nefarious activities to throw off administrators and investigators. Additionally, the attacker can configure the length of the shell history file to simply be "zero" so that no history will

be maintained for an account used for attacks. Shell history files with a length of zero could raise suspicions of system administrators, though, so the more careful attackers simply remember to remove the commands that could raise suspicion rather than completely deleting the history. Interestingly enough, attackers can even lines to another user's shell history file, possibly framing that user or diverting suspicion.

However, by simply opening the shell history file to edit it, the bad guy faces a problem. It's important to remember that shell history is written when the shell is exited. Therefore, you won't see your most recent commands in the shell history; they are stored in memory until the shell is exited gracefully. At that time, they are written to the shell history file. This has significant impact for attackers editing shell history. In particular, the attacker's command used to invoke the editor will be placed in the shell history file, so an investigator might see something like `vi .bash_history`. That's bad news for the attacker, because the investigator now knows the bad guy altered the shell history. To deal with this problem, the attacker could exit the shell, log back in, creating another shell, and then try to edit the shell history file again to remove the line about editing the shell history. But, then, when the attacker logs out, the most recent history will be written, along with the new command about editing the shell history file! It's a chicken-and-egg problem for the attacker.

With computers, if you ever face a chicken-and-egg problem, you need to find out how to kill the chicken or break the egg, solutions that lend themselves well to editing shell history. There are two widely used solutions to this dilemma for the attacker. First, the bad guy could simply set the shell history size to zero, as we discussed earlier (I suppose that's breaking the egg). But a more comprehensive way of dealing with the issue is to kill the chicken; that is, simply kill the shell instead of gracefully exiting it. Remember, shell history is written only when the shell gracefully exits. By killing the running shell process, the attacker deprives it of the ability to write its history. Therefore, instead of logging out, the attacker can kill the shell by simply running a command like this:

```
# kill -9 [pid_of_the_shell_process]
```



Counter Hack Reloaded, Second Edition: A Step-by-Step Guide to Computer Attacks and Effective Defenses

By Ed Skoudis, Tom Liston

Publisher: **Prentice Hall**
Pub Date: **December 23, 2005**
Print ISBN-10: **0-13-148104-5**
Print ISBN-13: **978-0-13-148104-6**
Pages: **784**
Slots: **2.0**

Table of Contents | Index | Additional Reading

Security Networking Ed Skoudis Tom Liston Prentice Hall Counter Hack Reloaded, Second Edition: A Step-by-Step Guide to Computer Attacks and Effective Defenses

Defenses Against Log and Accounting File Attacks

To mount an effective defense, it is critical to prevent attackers from having the ability to alter logs. Logs that have been tampered with are less than useless for investigative purposes, and conducting a forensic investigation without adequate logging is like trying to drive your car while wearing a blindfold: difficult if not impossible, and certainly messy. As with hardening any system, the amount of effort you will want to apply to defending a given system's log information depends on the sensitivity of the server. Clearly, for Internet-accessible machines with sensitive data, a great amount of care must be taken with the logs. For some internal systems, logging might be less important. However, for critical systems containing information about human resources, legal issues, and mergers and acquisitions, logs could make or break your ability to detect an attack and build a case for prosecution. Let's examine the techniques used to defend logs on Windows and Linux/UNIX, as well as other platforms.

Activate Logging, Please

The first step in ensuring the integrity and usefulness of your log files is quite simple: Activate logging on your sensitive systems! Quite often, I have been involved with a security investigation only to discover that by default, logging is deactivated on many of the servers that are included in the investigation. My heart drops when I come to this realization. Your organization must have a policy or standard specifying that logging must be done. Additionally, you should periodically audit your systems to ensure that logging is activated in accordance with your policy. It is especially important to check that adequate storage exists to house logging information. Windows systems are configured out of the box to limit each of the event logs to a paltry 512K, with the newest events overwriting the oldest events when that limit is reached. This limit can be changed through the "properties" item for each of the particular "classes" of events (security, system, or application) within Event Viewer. When deciding on these limits, it is important to consider just how quickly you believe your organization can respond to an event and track it back to a particular machine. Only slightly less frustrating than finding a compromised machine with logging disabled is finding a compromised machine where critical events have been overwritten, so give your logs plenty of disk space—hundreds of megabytes or even gigabytes of space—depending on the typical volume of logs for a given system.

Setting Proper Permissions

Another commonsense defense for protecting critical system logging and accounting information is to set proper permissions on the log files, as well as (for Linux and UNIX systems) `utmp`, `wtmp`, `lastlog`, and users' shell histories. Although particular permissions vary depending on the operating system, you should configure your system to allow for the minimum possible read and write access of log files. In particular, security and kernel logs should be set to be read and written only by root, if your Linux and UNIX flavor allows such tight permissions. Some variants of UNIX require that particular log files be writable by particular accounts other than root. If this is the case for your flavor of UNIX, make sure you configure the minimal permissions necessary for logging to function properly.

Using a Separate Logging Server

One of the most effective techniques for minimizing an attacker's capability to alter logs involves setting up a separate logging server. Your critical systems, such as your Internet-accessible DNS server, mail server, Web servers, and so on, should be configured to redirect their logs to a separate machine on your DMZ. Your critical internal systems should send their logs to a group of separate logging systems on the internal network. Not only does this technique help to centralize logs for better analysis, it also significantly limits an attacker's ability to monkey with the logs. If attackers

take over root on a Linux or UNIX system or an administrator account on a Windows box, they will not be able to alter the logs to cover their tracks, because the logs are elsewhere. The attacker will only be able to modify the logs by mounting a successful attack against the logging server. Therefore, by using the separate logging machine, we've just raised the bar. Of course, you must strongly secure the logging server. Make sure you apply system security patches, and close all unused ports on the logging server machine. Additionally, strongly resist the urge to use your logging server for any other purpose beyond aggregating logs. The more services you place on the logging box, the more vulnerable it becomes to attack.

Although you won't be able to capture shell histories, `utmp`, `wtmp`, and `lastlog` from Linux and UNIX systems on a separate server, you can still redirect all of the pure logs to a separate server. To configure a Linux or UNIX system to use a separate logging server, you must configure `syslogd` so it knows where to direct the logs. First, make sure there is a line in your `/etc/services` file associating `syslog` with its standard port, UDP port 514:

```
syslog 514/udp
```

Next, include an entry in the `syslog.conf` file that tells `syslog` to redirect particular message types to a remote server. For kernel-type messages, the following line should be placed in `syslog.conf`:

```
kern.* @[hostname_for_remote_logging]
```

Note that this type of configuration can be done in addition to local logging, rather than replacing local logging. That way, you'll get two sets of the same logs, which can act as corroborating evidence in an investigation. One set of logs comes from the local system, and another set comes from the log servers. Such a setup will also help you look for discrepancies when an attacker starts to change the local logs of a victim machine.

Just to be sure that an attacker cannot disable logging by attacking DNS, the logging server hostname listed above should be included in `/etc/hosts` so that it resolves locally. This local resolution of the log server name shouldn't present a major management headache, because your centralized log server will not be changing its IP address very often.

For particularly sensitive servers, I've also sent `syslog` information over a

serial connection to a local logging box with no network connectivity at all. The purpose of this log was to simply act as a local backup of information logged remotely, but it also provides a log that is virtually unalterable to anyone who does not have physical access to your location.

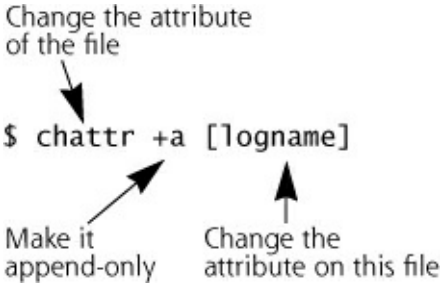
In Windows, the EventLog service can be replaced by a Windows-compatible version of syslog, with capabilities for centralizing log access. Several syslog-for-Windows tools are available, including the commercial tool SL4NT at and Kiwi syslog for Windows at . By using these tools, event logs can be sent to separate syslog servers from a Windows system.

Encrypting Your Log Files

Another very useful technique for log protection is to encrypt the log files. When attackers try to edit the files, they will not be able to alter them meaningfully without the decryption key. The attacker's only option will be to delete the log file, which is a very noticeable action. To encrypt log information as it passes across the network and is placed in the log files stored locally on the logging server, you could use Core Labs' free Secure Syslog tool available at . Of course, syslogging to a separate logging server can be combined with this log encryption technique to even further protect the system logs.

Making Log Files Append Only

On Linux and some UNIX systems, you might want to make your log files append only, particularly if you use a separate syslog server. To do this, use the change attribute command as follows:



If attackers try to edit a log file that has been set to be "append only," they will find it write protected, as it has been changed to allow operations only to append data to the file. This is, of course, only a speed bump, because any slightly sophisticated attacker with root privileges will notice this and simply change the attribute back to make the log file alterations. This is, however, a simple change that will flummox many of the log-cleaning scripts used by the rank-and-file script kiddie masses.

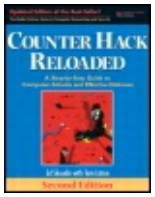
Protecting Log Files Using Write-Once Media

A more thorough way of protecting the logs on any type of system (Windows, Linux, UNIX, or others) is simply to store the logs on unalterable media, such as a nonrewriteable DVD or CD-ROM. The prices of both DVD recorders and media have dropped to the point over the past several years that this is certainly a viable option. The attacker cannot alter the logs because they are protected by the physical medium itself. Write-once media (like DVDs and CD-ROMs) unfortunately will always have lower performance when compared with a speedy hard drive, and might not be capable of sustaining real-time logging from several different sources simultaneously. Therefore, you might want to configure your logging server to flush logs periodically to the write-once media, such as once per day, or when specific file size thresholds are reached.

When all six of these techniques are applied together (activating logging, setting minimal permissions, using a separate logging server, encrypting the log files, setting the logs to append only, and storing them on write-once media), you can have a far better degree of confidence in the integrity of your log files. Of course, each of the techniques can be employed separately depending on your security needs.

Counter Hack Reloaded, Second Edition: A Step-by-Step Guide to Computer Attacks and Effective Defenses

By Ed Skoudis, Tom Liston



.....

Publisher: **Prentice Hall**

Pub Date: **December 23, 2005**

Print ISBN-10: **0-13-148104-5**

Print ISBN-13: **978-0-13-148104-6**

Pages: **784**

Slots: **2.0**

Table of Contents | Index | Additional Reading

A **A** A

Creating Difficult-to-Find Files and Directories

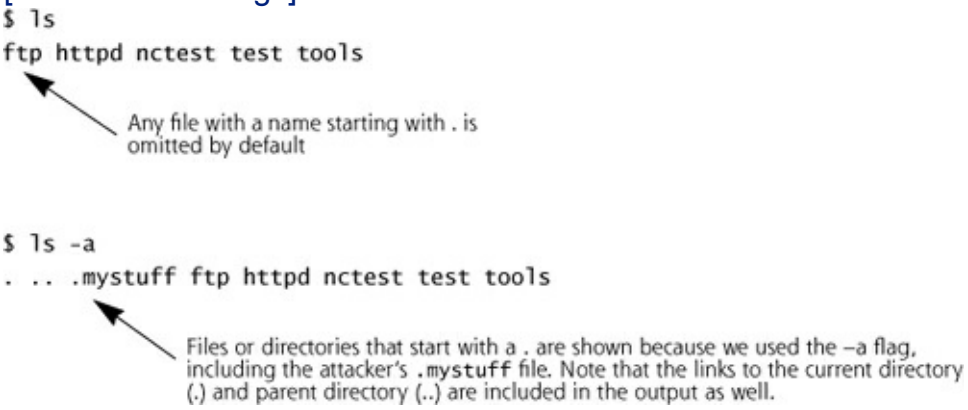
Another technique used by attackers to cover their tracks on a system involves creating files and directories with special names or other attributes that are easily overlooked by users and system administrators. Attackers often create "hidden" directories and files to store various attack tools loaded on the systems, save sniffed passwords, and store other information belonging to the attacker. Of course, as described in Chapter 10, rootkits can alter the function of critical system components to hide both files and directories. We have explained the techniques used by rootkits elsewhere, so we now turn our discussion toward other, nonrootkit options for hiding data. Let's explore the many ways to hide files and directories under UNIX and Windows using only the basic operating system features, without requiring the installation of a rootkit.

Creating Hidden Files and Directories in UNIX

On UNIX systems, attackers often name their files with a period (.) at the start of the name to make the files less likely to be noticed by users and system administrators. Why are such files less likely to be noticed? By

default, the standard UNIX `ls` command used for viewing the contents of a directory does not display files with names that begin with a period. This standard behavior was designed to keep directory listings from getting cluttered. An application can create a file or directory that is hidden from a user just by naming it `.[filename]`. Applications often use files or directories named in this way to store configuration information specific to an individual account, and there are usually many files of this type in each user's home directory. To view all files in a directory (including those files with names that begin with a period), the `ls` command must be used with the `–a` flag, which will show of the contents of the directory. Consider an example in which the attacker wants to hide information in the `/var` directory. The attacker can create a file or directory named `.mystuff` to hide stolen passwords or attack tools. When such a file is present, let's look at the difference between the output of the standard `ls` command and the `ls –a` command:

[View full size image]



An even subtler technique for hiding files on UNIX systems involves naming files or directories with a period followed by one or more spaces. As described in the Chapter 3 section titled "Linux and UNIX File System Structure," included inside every Linux and UNIX directory there are two links to other directories. One of these links is named., which refers to the directory itself. The other is .., which refers to the parent directory just above the given directory in the file system hierarchy. These conventions allow a user to refer to files in the local and parent directories with a convenient shorthand. An attacker will often name a file or directory period-space (.) or period-period-space (..) to hide it, making it appear just like the . and .. directories. Let's look at what happens when an attacker names a file period-space:

```
$ mkdir ". "
```

Make a directory with the name period-space.

```
$ ls -a
. . .. .mystuff ftp httpd nctest test tools
```

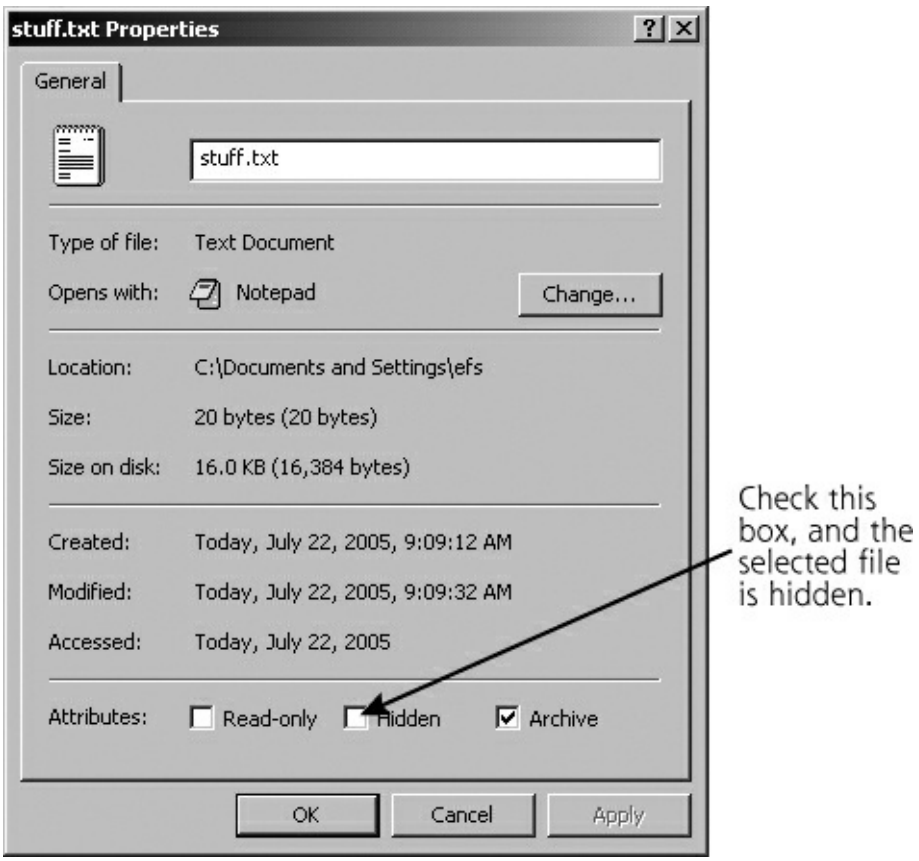
This is a file or directory where the attacker can hide items.

Most administrators looking at the output of this `ls` command would not see the name period-space in the output, effectively hiding the directory from view. The hidden directory is camouflaged and blends in with what an administrator would expect to see in the directory. Some attackers use other variants of this technique, naming a file or directory with just a space ( ) or with three dots (...).

Creating Hidden Files in Windows

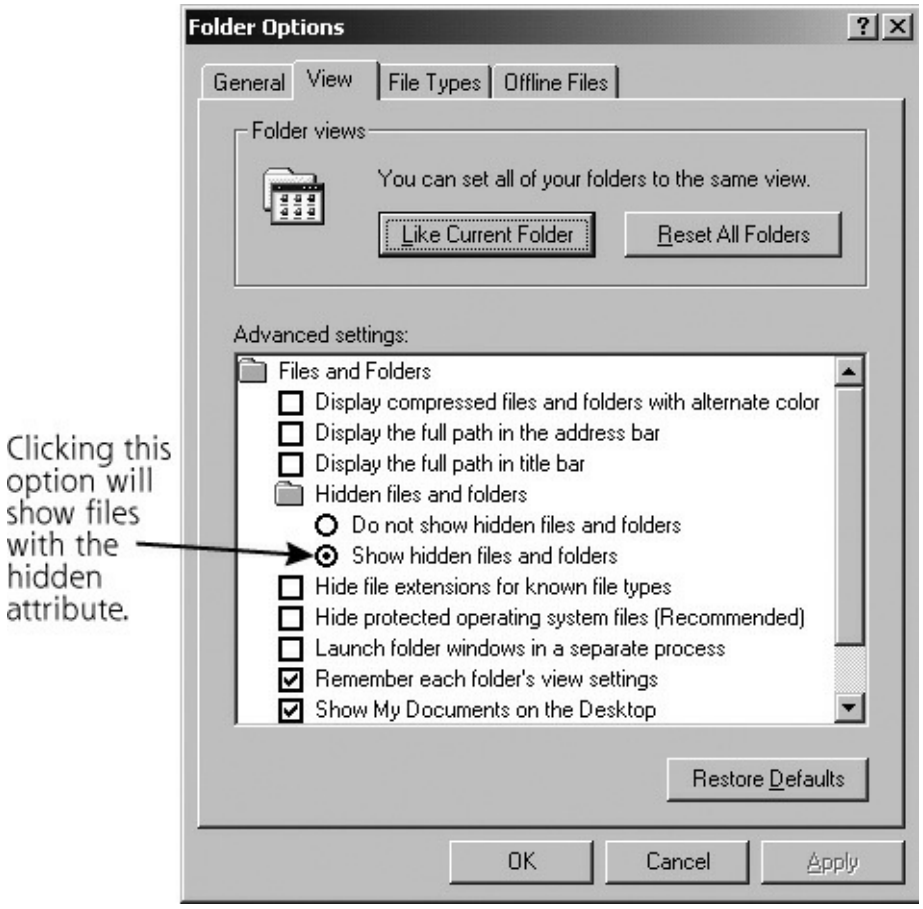
Techniques for hiding files are not limited to UNIX. Modern Windows systems offer users the option of setting a file or directory with the attribute "hidden" so that it will be omitted from view by default. By simply right-clicking on the file or directory in Windows Explorer and selecting Properties, the user is presented with an option to make the file hidden, as shown in Figure 11.3.

Figure 11.3. Setting the "hidden" attribute on a file or directory.



However, discovering files with the "hidden" attribute is actually quite easy. In Windows 2000 and XP, using the Folder Options panel in Windows Explorer, you can select the View tab and select Show All Files. The screen to configure this setting is shown in Figure 11.4.

Figure 11.4. Showing hidden files in Windows 2000/XP.



A far more powerful and subtle technique for hiding information in Windows involves using Alternate Data Streams (ADS), which relies on options included with the NTFS file system. The basic capabilities of NTFS are described in Chapter 4, Windows NT/2000/XP/2003 Overview. Beyond these basic capabilities, NTFS allows every file or directory to have multiple "streams" of data associated with it. These streams can store any type of information. In fact, the normal contents of a file that can be seen and accessed normally by users on the system is a stream itself. However, behind this normal stream, data can be stored in an arbitrary number of additional streams. Let's consider an example in which an attacker wants to hide data in a stream associated with the file `notepad.exe`. Of course, the attacker could hide data behind any file or directory on the system, but suppose they have chosen `notepad.exe`. The normal stream associated with `notepad.exe` contains the executable program for the simple Windows editor Notepad.

You might think that special programs are required to create and access ADS data, but our attacker can actually create another stream behind `notepad.exe` using only the built-in Windows commands coupled with input/output redirection. For our example, the attacker wants to take the file `stuff.txt` and hide it in a stream behind `notepad.exe`. The attacker could use this command:

```
C:\>type stuff.txt > notepad.exe:data.txt
```

This command copies the contents of the `stuff.txt` file into a stream named `data.txt` behind the file `notepad.exe`. The colon followed by a stream name indicates in which stream to put the data. The attacker could give the stream any name at all and create any number of streams for each file, as long as the partition on which `notepad.exe` resides is NTFS. If it is a FAT or FAT32 file system, streams are not supported, so an error message is displayed. But for NTFS-based file systems, the most common in use today for Windows systems, the new stream named `data.txt` is automatically created by this command and tacked onto the end of the `notepad.exe` file. After deleting the file `stuff.txt`, no remnants of the file `stuff.txt` will be visible in the directory. All of the contents of `stuff.txt` are hidden behind the Notepad executable. That's the beauty of ADS from an attacker's perspective: There is nothing built into Windows to locate these streams. Windows Explorer doesn't show them, nor does the `dir` command. They are, in fact, invisible on a stock Windows system. These streams act rather like a subterranean world burrowed under your file system. Remember, any file or directory can have an arbitrary number of streams underneath it.

Now, if anyone runs the `notepad.exe` program with our stream attached to it, only the normal executable will run, with no indication of the hidden file stream. When anyone on the system looks at the file size of `notepad.exe`, the size of the normal, executable program will be displayed, with no indication of the hidden stream of data. This stream is quite effectively hidden. At a later time, the attacker can come back to the system and retrieve the hidden data from the stream by using only built-in Windows commands again, as follows:

```
C:\>more < notepad.exe:data > stuff.txt
```

Now the `stuff.txt` file has been restored, and the attacker can access its contents.

It is important to note that the types of streams are independent of the parent file under which they are attached. For example, a `.txt` file can be embedded in a stream under an `.exe` file, or vice versa. You could store an `.exe` under a `.txt` file, and then even run the `.exe` from within the stream! Suppose we have created a stream called `evil.exe` underneath the file `good.txt`. We could create this situation with this simple command:

```
C:\>type evil.exe > good.txt:evil.exe
```

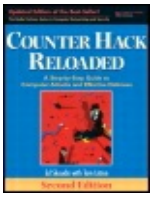
Now, we can run the executable from within the ADS by typing:

```
C:\>start .\good.exe:evil.exe
```

The evil executable runs just as if it were its own file, separated from the ADS itself.

Defenses from Hidden Files

To defend against these techniques for hiding files on sensitive systems, you should use file integrity checking tools that look at the contents of files and directories to make sure no additional data, files, or directories have been hidden in them. A file system integrity checker like Tripwire has this capability, as do numerous others that we discussed in Chapter 10 during our examination of rootkits. Additionally, host-based IDSs, which are described in more detail in Chapter 6, Phase 2: Scanning, as well as antivirus tools, can check the contents of directories to determine if a malicious hidden file is present and generate an alert message for a system or security administrator. If you use Windows, it is important to verify that the virus and spyware scanning tools you employ are ADS aware, because this method of hiding files and directories represents a huge untapped resource for malware authors. At the time of writing, there are few if any malicious programs that actually use ADSs, but look for that to change in the future. Additionally, there are specialized tools that will scan the entire file system looking for data stored in alternate streams. One such program is CrucialADS, a free ADS scanning program that can be found at Another is the free LADS tool (which stands for List Alternate Data Streams) by Frank Heyne at . It is important to remember that all ADS data isn't bad (notably, many graphic packages store metadata information about photos in an ADS) but you should carefully check any ADS data that a scanner uncovers to determine its origin, especially executables tucked in streams.



Counter Hack Reloaded, Second Edition: A Step-by-Step Guide to Computer Attacks and Effective Defenses

By Ed Skoudis, Tom Liston

.....
Publisher: **Prentice Hall**
Pub Date: **December 23, 2005**
Print ISBN-10: **0-13-148104-5**
Print ISBN-13: **978-0-13-148104-6**
Pages: **784**
Slots: **2.0**

A **A** A

Table of Contents | Index | Additional Reading

Security Networking Ed Skoudis Tom Liston Prentice Hall Counter Hack Reloaded, Second Edition: A Step-by-Step Guide to Computer Attacks and Effective Defenses

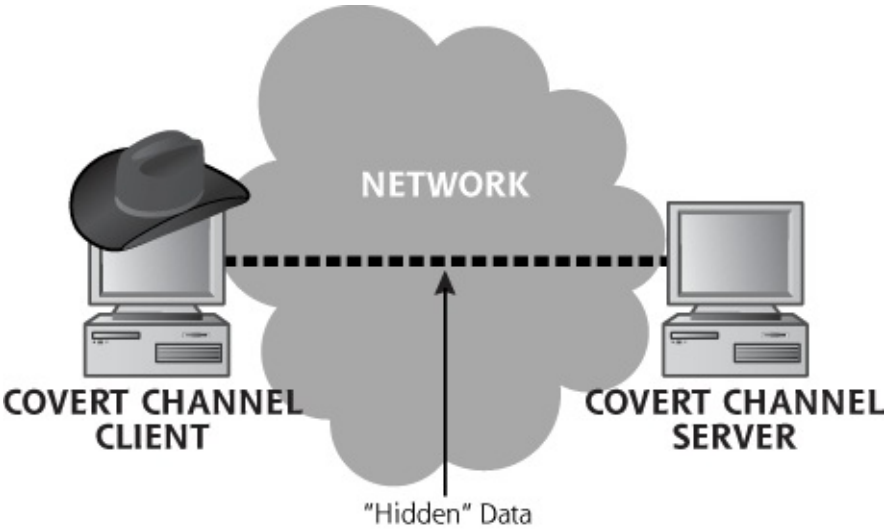
Hiding Evidence on the Network: Covert Channels

Once attackers have installed backdoor listeners on a system and cleaned up their tracks in the logs, they still need to communicate with their nefarious programs on the victim machine to control them. To avoid detection, some attackers utilize stealth mechanisms to communicate with the backdoor system across the network. Such disguised communication mechanisms are referred to as covert channels. Covert channels are essentially an exercise in hiding data while it moves. Whereas encryption mathematically transforms data into ciphertext so an adversary cannot understand its contents, covert channels hide the data so the adversary doesn't detect it in the first place. A truly paranoid attacker will use both a covert channel to hide information and cryptography to scramble the contents of the information as well.

The techniques we discuss for establishing covert channels across the network require both a client and a server. The server must be installed on a victim's system, acting as a sentinel, ready to exchange data with the client. The client packages up data using stealth techniques, and the server

unpackages the data and reacts to it. The covert channel can be used to control a system remotely, to transmit files secretly, or to hide any other application capability the attacker needs to disguise. Figure 11.5 depicts a typical generic exchange of data using a covert channel between a client and a server.

Figure 11.5. A covert channel between a client and a server.



How does the covert channel server acting as an endpoint for the covert channel get installed on a victim's machine in the first place? We have seen attackers employ countless techniques in real-world cases, including these scenarios:

- An attacker can take over a system and place a backdoor listener on it through a vulnerability such as a buffer overflow.
- The attacker could e-mail an unsuspecting internal user an executable Trojan horse program, worm, or virus, which implements a covert channel server.
- The attacker might be an ex-employee who had system administration privileges before being terminated. The attacker could leave the covert channel server as a way to keep unauthorized, lingering access.
- The attacker might have been a temp or contractor who signed on for a brief stint with the organization for the sole purpose of installing a backdoor agent on the internal network (and heck, to make a couple of bucks while on the victim's payroll).
- The attacker could have physically broken into a computing facility late at night, and installed an agent on a system. In some environments, nighttime is not even a necessary ingredient. By simply walking in the

front door and acting confident enough, an attacker can pretend to be a vendor or use some other ruse to gain access to computing systems to install internal covert channel servers.

Any of these mechanisms can be used to gain access. Once access is obtained, the covert channel allows the attacker to work in stealth mode remotely.

Tunneling

Covert channels often rely on a technique called tunneling, which allows one protocol to be carried over another protocol. Any communications protocol can be used to transmit another protocol. Information theory says it must be so. Consider a hypothetical protocol called TCP/CP. TCP/CP marries a modern-day computer protocol to an ancient mechanism for delivering messages, resulting in a slow, yet remarkably effective communication tool for intermediate distances.

In a real-world example of tunneling techniques, the SSH protocol can be used legitimately to carry other TCP-based services. Originally, SSH focused on providing strongly authenticated, encrypted command shell access across a network, still probably its most popular use today. However, through tunneling, its use has been greatly expanded. With a rock-solid SSH session in place, any other TCP services, such as telnet, FTP, or even an X-Window session, can be transmitted securely over SSH. The information comprising the telnet, FTP, X, or other session is simply written into SSH messages and transmitted across the authenticated, encrypted SSH pipe. This SSH tunneling technique is frequently used to create VPN-like access across untrusted networks for TCP services. Although SSH tunneling only works with TCP connections, there are other tunneling protocols that are designed to handle UDP traffic. But, if you're ever in a jam, remember that our old friend Netcat (see the section titled "Netcat: A General-Purpose Network Tool" in Chapter 8) can be used to create a UDP listener to grab traffic, which can be piped into a Netcat client creating a TCP stream, allowing you to pass it through an SSH tunnel.

What Is TCP/CP?

The Transmission Control Protocol (TCP), transmitted via Carrier Pigeon (CP), of course. The higher layer application (which could be Web browsing, telnet, FTP, SSH, or any other TCP-based application) passes data down its protocol stack. The TCP layer formats the packet, and instead of sending it to the IP layer, it prints each TCP packet on a tiny sheet of paper. Each packet is then wrapped around the leg of a carrier pigeon. The pigeon is released, carrying the printed sheet to its destination. At the destination, the data is retyped into a computer, passed up through the TCP layer, and sent to the receiving application. Pigeons are then fitted with responses, and interactive communication occurs. Although not terribly efficient (downloading the latest MP3, in addition to outraging the recording industry, has the unwanted side effect of exhausting fleets of pigeons), TCP/CP shows how any protocol, no matter how bizarre or awkward, can be used to carry another protocol through tunneling. Another bird-related transport protocol was defined by the IETF in RFCs 1149 and 2549. Check out for more information about how to transmit IP over avian carriers.

An SSH tunnel and protocol tunneling in general are powerful methods to allow confidential traffic to flow through an untrusted network, but like most good things, they can be abused as well. Attackers have harnessed the power of these tunneling techniques to remain undetected as they communicate with their backdoor listeners. Several tools are widely exchanged within the computer underground based on these techniques. We'll look at a few of the most widely used tools for tunneling covert information: Loki and Reverse WWW Shell.

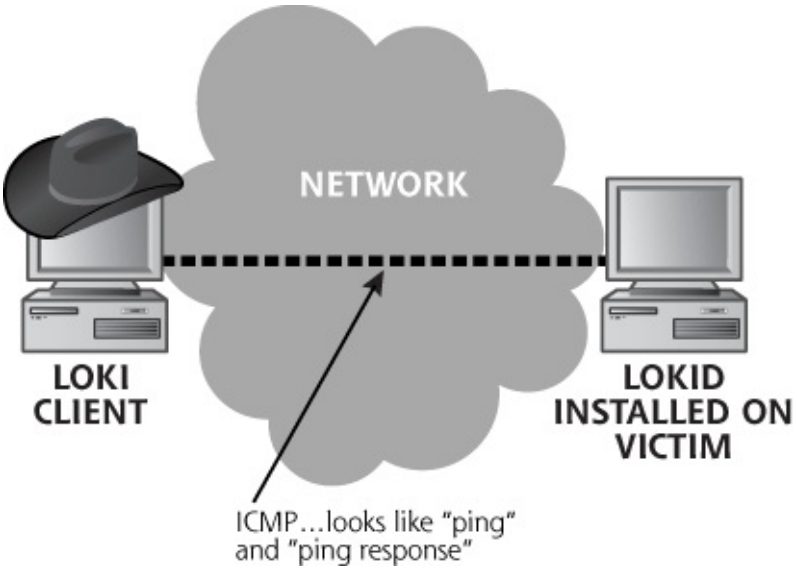
Loki: Covert Channels Using ICMP

Many networks allow incoming ICMP packets so users can ping or traceroute to their Web sites for troubleshooting. Suppose an attacker takes over such a Web server, installs a backdoor listener, and wants to communicate with it. Sure, the bad guy could set up a backdoor listener on a specific port, but that might be detected. A more stealthy approach would be to utilize ICMP as a tunnel to carry on interactive communications with the backdoor listener. Tunneling the communication over ICMP has several advantages, including the fact that ICMP messages don't require an open port that might be detected by a curious system admin using the `netstat` or `lsof` commands we discussed in Chapter 6. Numerous tools have been released that utilize tunnels over ICMP to establish a covert channel, and one of the

most popular is Loki, pronounced "Low-Key."

Loki was written by daemon9 to provide shell access over ICMP, making it much more difficult to detect than other (TCP- or UDP-based) backdoors. Loki was originally described in Phrack issue 49, with source code available in Phrack 51 (both at). The tool runs on Linux, FreeBSD, OpenBSD, and Solaris systems and although there are rumors that it has been ported to Windows, if it has, it certainly isn't in widespread distribution. As shown in Figure 11.6, the attacker types in commands at a prompt into the Loki client. The Loki client wraps up these commands in ICMP and transmits them to the Loki server (known as "lokid" and pronounced "Low-Key-Dee"). Lokid unwraps the commands, executes them, and wraps the responses up in ICMP packets. All traffic is carried in the ICMP payload field. The Lokid responses are transmitted back to the client, again using ICMP. Lokid executes the commands as root, and must be run with root privileges, so it can snag the ICMP packets from the kernel and extract the commands.

Figure 11.6. Loki hides data inside ICMP messages.



As far as the network is concerned, a series of ICMP packets are shot back and forth: Ping, Ping-Response, Ping, Ping-Response. As far as the attacker is concerned, commands can be typed into the Loki client that are executed on the server machine, yielding a very effective covert communication session.

System administrators often use the familiar `netstat -na` command to show which processes are listening on which TCP and UDP ports. In addition to running `netstat`, system administrators can periodically port scan their systems to detect backdoor listeners using a tool like Nmap, as described in Chapter 6. However, as stated earlier, ICMP does not include the concept of

a port, and is therefore not detected using `netstat` and will not show up in a port scan. Loki therefore foils these two detection techniques, flying under the radar screens of the common system administrator backdoor detection techniques. The only trace of the Loki daemon on the internal system is a root-level process running, and ICMP packets going back and forth.

Loki also has an option to run over UDP port 53, thereby disguising its packets as DNS queries and responses. These packets are not properly formatted DNS queries and responses, however. Instead, Loki just uses the same port as DNS traffic. Loki supports on-the-fly protocol switching to toggle between ICMP and UDP port 53. When in UDP mode, Loki will show up in the output of the `netstat -na` command, and can be identified during a port scan. Additionally, to further stealthify the connection, Loki supports end-to-end encryption of the ICMP payload information using the Blowfish algorithm for encryption and Diffie-Hellman for key exchange.

This technique of transporting covert communication via ICMP is by no means limited to Loki. There are several other tools that can be used to tunnel communications over various protocols. The Covert Channel Tunneling Tool (CCTT) can tunnel communication using ICMP, TCP, and UDP packets. MSNShell is a tool that tunnels shell commands to and from a Linux machine using Microsoft's MSN protocol. Both tools are projects of Gray World Net Team and available from These tools and others like them are currently used by the underground to provide covert communication with backdoors installed on compromised systems.

Reverse WWW Shell: Covert Channels Using HTTP

"Loki is interesting," you might say, but you are far too smart to allow incoming or outgoing ICMP on your network. Sure, blocking pings is an inconvenience for users, but security is paramount, for goodness sakes. So, because ICMP is blocked at your border, you're secure against covert channels, right?

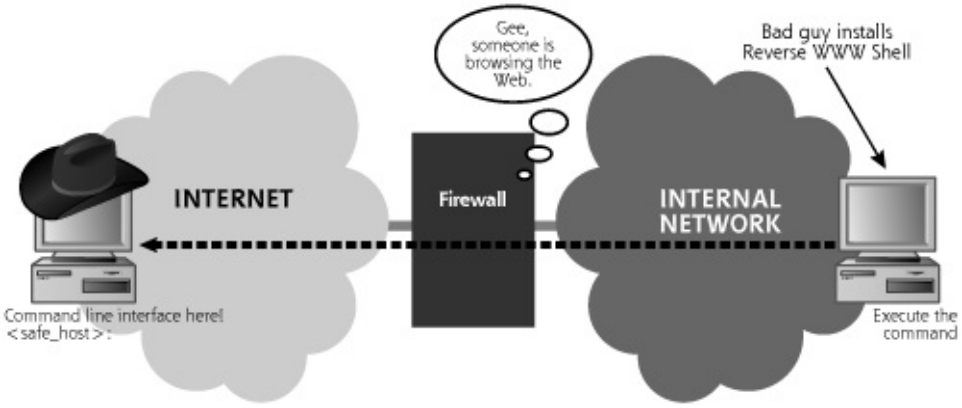
Well, unfortunately, Loki and ICMP tunneling are but a small area in an enormous universe of covert channel choices for an attacker. Another particularly insidious technique is to carry shell-type commands using HTTP, which has been implemented in the aptly named Reverse WWW Shell tool.

Reverse WWW Shell allows an attacker to access a machine with a command shell on your internal network from the outside, even if it is protected with a firewall. It was written by van Hauser (who also wrote THC-Scan, the war dialer described in Chapter 6—clearly a talented individual) and is available at The attacker must install (or get one of your users to install) a simple program on a machine in your network, the Reverse WWW Shell server.

On a regular basis, usually every 60 seconds, the internal server tries to access the external master system to pick up commands, essentially calling home. If the attacker has typed something into the master on the external system, this command is retrieved and executed on the internal system. The next communication from the internal agent will carry the results of this command, and a request for the next command. This is the "reverse" part of Reverse WWW Shell: The server goes to the master to pull commands, executes them, and pushes the results. This polling technique is called a reverse shell, or, more colorfully, shoveling shell, as we discussed in the context of Netcat in Chapter 8, Phase 3: Gaining Access Using Network Attacks. Figure 11.7 shows the operation of Reverse WWW Shell in more detail. Therefore, we have simply pushed out shell access, an impressive feat, but by no means revolutionary, right?

Figure 11.7. Reverse WWW Shell looks like outgoing Web access, but is really incoming shell access.

[View full size image]



But wait ... there's more! From a network perspective, the internal (victim) machine appears to be surfing the Web. The Reverse WWW Shell server uses standard HTTP GET messages sent to the attacker's external system across the network, where the Reverse WWW Shell master is running. When it accesses the master, the Reverse WWW Shell server pushes out the command-line prompt from the server, tunneled in HTTP requests and responses. So, the internal agent looks like a browser surfing the Web. The external master looks like a Web server. All outgoing data is transmitted from a high source port (greater than 1024), to a destination TCP port of 80. All responses come back from TCP port 80 to the high-numbered port.

So the packets have HTTP characteristics, but, even worse, the shell data is formatted as HTTP GET commands. Therefore, even a proxy firewall that enforces the use of HTTP on TCP port 80, carefully combing the protocol to make sure it's HTTP, is befuddled. The firewall and other network components view the traffic as standard outgoing HTTP, something that most networks allow. In fact, the covert channel is incoming shell access, allowing

the attacker to execute any command on the internal system.

From the attacker's point of view, using Reverse WWW Shell is rather annoying; the cadence of entering in commands, waiting for the server to come and retrieve them, execute them, and send the response can be cumbersome and frustrating. The attacker types in a command, waits 60 seconds, and then gets the response. The attacker can then type another command, wait 60 more seconds, and get the response. Although annoying, the tool is still incredibly useful for an attacker, and the 60 seconds can be set to a lower value. Making it too low, however, would not look as much like normal HTTP traffic. If you saw a browser going to the same Web server every three seconds, you might be suspicious. Of course, to make Reverse WWW Shell even stealthier, the attacker can randomize this period between accesses.

Unfortunately, you are still not safe if you require HTTP authentication with static passwords to get out of your firewall. Many organizations only allow outgoing Web browsing if a user authenticates to a Web proxy with a user ID and password, a reasonable increase in security and auditability under most circumstances. However, Reverse WWW Shell allows the attacker to program the system with a user ID and password that will be given to the outgoing Web proxy firewall for authentication.

From an implementation perspective, the Reverse WWW Shell client and server are the same program, with different command-line parameters. The single client/server program is written in Perl, so a Perl interpreter is required on both the inside and outside machines. Additionally, several folks have developed similar functionality for tools that use HTTPS.

Unfortunately, the ideas behind Reverse WWW Shell didn't stay confined to the computer underground. Currently, there are some commercial services that implement remote GUI access to the desktop via HTTP, with one of the most popular named GoToMyPC.com. It's very scary from a security perspective, letting your users (and evil attackers) anywhere on the Internet control machines remotely via outgoing HTTP that is secured only by a user-chosen password. If users choose weak passwords, an attacker might be able to take over their internal systems by riding across the outbound HTTP access of GoToMyPC. This security administrator's nightmare even offers a free trial period, and claims it takes only two minutes to install. And people wonder why some security folks have thinning hair!

Other protocols besides ICMP and HTTP are being used to tunnel covert data. Attackers have created tools that utilize SMTP, the protocol used to transport e-mail across the Internet, to carry shell access and transfer files. Of course, the latency of using a store-and-forward application like e-mail for transmitting commands and results is even more painfully slow than