# Windows User-Mode RootKits

For many years, user-mode RootKits were focused primarily on UNIX systems. With these origin accident that the word *root* (the superuser account of UNIX) is prominently featured in the word may have been born and grown up on UNIX, but user-mode RootKit techniques have been adapt platforms as well, especially over the past few years. In particular, there are a handful of interes RootKits for Windows machines, which we'll discuss in this section. Like their UNIX counterparts, RootKits on Windows modify critical operating system software to allow an attacker to gain acce a machine. Note that we're still focused on user-mode RootKits (which, by our definition, involve operating system executables and not the kernel).

User-mode RootKit techniques are used in a some of tools on Windows, but we should note that as many solid and popular user-mode RootKits on Windows as there are on UNIX. I've observed computer attack cases I've handled myself. On UNIX, user-mode RootKits are very frequently er Windows systems, on the other hand, user-mode RootKits are less frequently used. There are se this phenomenon, including these:

- *Early on, application-level backdoors proliferated on Windows systems.* Throughout the mi much of the work in creating backdoor tools for Windows focused on creating application-l such as the port listeners and remote GUI tools we discussed in Chapters 5 and 6. Most at perfectly satisfied with the remote control GUI features of tools like VNC, Back Orifice, and didn't need to modify operating system components when they could easily rely on all of t offered by these application-level tools. If a bee-bee gun accomplishes your goal, there's n Howitzer.

- *Later, many RootKits on Windows focused on manipulating the kernel and not general syst* in large part to research by Greg Hoglund and other folks at the well-named *www.rootkit.c* Windows RootKits jumped to the kernel level in the late 1990s. So, a lot of backdoor devel Windows jumped from focusing on the application level right into the kernel itself, with litt operating system executables and the associated user-mode RootKits that sit in between. tired of bee-bee guns and some guy is giving away cruise missiles for free, you don't need We'll zoom in on these kernel-mode RootKits for Windows in Chapter 8.

- *Windows File Protection (WFP) hinders the replacement of executables.* With the release o Microsoft started building functionality into the operating system that automatically scans looking for unexpected changes to critical executables and libraries. If the WFP feature fin critical system file, it restores the original file automatically: No fuss, no muss. WFP doesn possibility of a user-mode RootKit on Windows, but it does raise the bar somewhat. If they implement a RootKit that replaces executable or library files, attackers have to work a tiny defeat WFP. We'll discuss how they accomplish this feat later in the chapter.

- *Windows is a closed-source operating system, so creating a user-mode RootKit on Window work.* The source code for several UNIX variations is widely available, allowing attackers t backdoor functionality to existing programs. Much of the real work is already done for the operating system developer in creating the actual programs in the system. The attacker m evil features to the already existing source code to create the fakes included in user-mode UNIX. On the other hand, with Windows, Microsoft maintains tighter (but not perfect) cont source code. An attacker trying to create replacements for existing Windows binary execut will have to reverse-engineer the Windows functionality, without being able to review the s

[1] Please note that I'm not arguing here that a closed-source model is inherently more or less secure than an o development model. I'm merely discussing the relatively more difficult task of creating a user-mode RootKit on we'll see throughout the rest of this chapter, although creating RootKits on Windows is more of a challenge with code available, attackers have more than met that challenge. Without source code, they've implemented some powerful user-mode RootKit tools. I personally believe that the closed-source and open-source software develop are tied from a security perspective.

- *Windows isn't as well documented as UNIX.* This issue goes hand-in-hand with the precedi discussion. Because UNIX has been around for a longer time, and the source code is more both the good guys and the bad guys understand its features in far more detail. When tryi how a UNIX binary executable works, an attacker can simply ask a question in a public for Internet search engine like Google to find the answer. In contrast, in the Windows operatir good deal of functionality just isn't documented. The attackers have to figure out how Win through trial and error, reviewing compiled binary code, and decompiling code.

Now that we've got a feel for some of the challenges faced by developers of user-mode RootKits let's jump in and start analyzing how they surmount these challenges. We'll cover three different implementing user-mode RootKits on Windows, and look at example tools that rely on each tech three different Windows RootKit implementation techniques are contrasted in Figure 7.12.

**Figure 7.12. Three different techniques for implementing user-mode R Windows systems.**

First, a user-mode RootKit on Windows could interface with existing Microsoft operating system undermine their security. Microsoft has devised several interfaces in Windows for extending its b functionality through third-party tools. A user-mode RootKit could exploit these interfaces by ins the defined interfaces between existing Microsoft programs, instead of overwriting Windows code chapter, we'll discuss a tool called FakeGINA that uses this technique.

Next, a user-mode RootKit on Windows could just overwrite existing executable files and librarie machine, much like the user-mode RootKits on UNIX that we discussed in the first half of this ch accomplish this task, an attacker must first disable the WFP feature that prevents changes to va operating system files in Windows. We'll analyze how they shut off WFP so they can overwrite th and then look at a particular example of malware that conducts such an attack: the Code Red II

Finally, an attacker could implement a RootKit that uses a set of very popular techniques to injec running processes and overwrite their functionality. Instead of manipulating files on the hard dri mode RootKits shoot their code right into running processes' memory using techniques called DL API hooking. At the end of this chapter, we'll explore this technique, as well as a tool based on it Windows RootKit.

## Manipulating Windows Logon with FakeGINA

In an effort to be flexible, Microsoft has designed some components of Windows to be easily mo party tools can extend the operating system. Some parts of Windows are highly modular, allowir administrator (or evil attacker) to add components to the system using well-defined interfaces c Microsoft. In particular, the user logon process is one of the most important areas, from a securi that can be extended in this way by adding libraries to the system. The basic logon process can third-party tools, so that nifty new authentication mechanisms, such as biometrics, public key in other tools, can be easily deployed. Unfortunately, this flexibility offers a foothold for the bad gu could subvert this process using user-mode RootKit mechanisms by adding malicious code to the

To understand how, we need to discuss the process of logging on to a Windows system. When y on to Windows, the operating system invokes the Winlogon process. This process collects your a credentials (e.g., a user ID and password) and verifies them so you can get access to the machi is illustrated in Figure 7.13.

**Figure 7.13. The normal Winlogon process.**

A user initiates the logon process by conducting what Microsoft calls a secure action sequence, s of Figure 7.13. The most common secure action sequence is hitting the Ctrl+Alt+Delete keys sim which some people refer to, tongue in check, as the Microsoft three-fingered salute. The Winlogc Step 2, invokes a GINA (usually pronounced "jeena"), which is a special library of code designed authentication. GINA is an abbreviation for Graphical Identification aNd Authentication. In Step 3 the user for authentication credentials, such as a user ID and password. The GINA then package credentials for the appropriate authentication mechanism (e.g., the Local Security Authority), an user's environment if the credentials are authentic, in Steps 4 and 5.

By default, Windows systems are shipped with a Microsoft-provided GINA called, appropriately e Msgina.dll. If you've ever logged in locally to a Windows machine (and who hasn't), you've seen GINA in action. It's the code that displays the standard logon dialog box on Windows, asking for password.

Now, to support different authentication mechanisms, Windows allows system administrators to party GINAs. In fact, instead of completely writing a GINA from scratch, a developer could even code between the Winlogon process and the existing Msgina.dll, in essence wrapping the current the existing logon functionality is preserved, and new capabilities can be easily added. That's the this nifty GINA feature. However, as you might expect, bad guys abuse this capability by creatin GINA code, thereby employing user-mode RootKit techniques on Windows.

One of the most popular GINA attack tools is named FakeGINA, which is pronounced "Fake-jeen circles. FakeGINA runs on Windows NT and 2000 (XP and 2003 are not supported). It was writte Vidstrom and is available at *http://ntsecurity.nu/toolbox/fakegina*. Now, FakeGINA is not a full u RootKit by itself. However, it uses RootKit-like techniques to undermine the authentication proce included as an element in a more full-featured RootKit package.

As illustrated in Figure 7.14, FakeGINA sits in between the Win-logon process and the existing M purpose of FakeGINA is not to give backdoor access. Instead, it records the passwords typed in the system, storing them in a file for the attacker. Step 1 works just like before; the Winlogon p activated by a user. In Step 2, however, the Winlogon process calls the FakeGINA tool instead o When the user types in a user ID and password, the FakeGINA tool secretly writes them to the a Steps 3 and 4. After writing them to the file, FakeGINA passes the authentication credentials to

the system, Msgina.dll. The real GINA completes the authentication process just as before, in St

**Figure 7.14. FakeGINA secretly gathers all user IDs and passwords l between winlogon.exe and msgina.dll.**

To install FakeGINA, an attacker must set a registry key indicating which GINA the system shoul located at HKEY_LOCAL _MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon, GinaDLL. If this registry key is not set, the default Msgina.dll is used by the system. To install Fa attacker simply sets this key so that it contains the name of the FakeGINA.dll, which must be ins System32 directory. After the system is rebooted, FakeGINA will begin its nefarious password sn all user IDs and passwords to a file named passlist.txt in the System32 directory.

To install FakeGINA, an attacker needs administrator privileges to alter the registry, as well as tc GINA into the System32 directory. You might wonder why an attacker who already has administ on a system would want to get a list of passwords on the machine. After all, you might think, if t have conquered the machine already, why do they bother collecting passwords for the system? A passwords could prove to be very valuable for several reasons. First, the attacker might have ex process on the machine running with administrator or system privileges via a buffer overflow or The bad guy can execute commands with these privileges, thereby installing FakeGINA, without the administrator's password. After installing FakeGINA using this access, however, the attacker back and wait for the real administrator to log on. At that point, the attacker will know the admi password, and can directly log on to the box without exploiting the buffer overflow again.

Another reason that FakeGINA is so valuable to attackers who already have administrator privile common practice of users manually synchronizing their passwords across multiple systems. By s passwords of users on one system using FakeGINA, the attacker can try those same passwords t systems. If the users have the same passwords on different machines, the attacker will be succe attacker might get really lucky and get the password for a user who has administrative privilege machines.

Finally, you might ponder the use of a password-cracking tool instead of FakeGINA. After all, if a administrator permissions, he or she could just dump the local stored encrypted passwords from and crack them using a password-cracking tool. Password crackers guess password after passwc each guess. If the encrypted guess matches the encrypted password, the attacker now knows th Depending on how difficult the passwords are to guess, cracking could take between seconds an FakeGINA, on the other hand, passwords materialize instantly in the attacker's file as each user consuming cracking is required, speeding up the attacker's job.

## WFP: How It Works and Attacks Against It

Although FakeGINA can certainly be damaging, it only alters Windows functionality that Microsof designed to be changed through special interfaces. Suppose an attacker wants to alter other bin or libraries on the hard drive in a user-mode RootKit attack. The attacker will have to contend w feature built into Windows 2000, XP, and 2003. Windows Me includes a similar capability called S Protection (SFP), which works like a junior version of WFP. We'll focus on WFP, because Windows 2003 are in far more widespread use than Windows Me.

When any directory containing sensitive Windows files (e.g., the System32 directory) is changed signals WFP, invoking its functionality to check the digital signature of the changed file. WFP mo programs, libraries, and configuration files to look for changes. On a stock Windows 2000 systen more than 1,700 files, a pretty large number. If it detects a change in any one of these files, WF digital signature of the changed file to the original file. If the signature doesn't match a Microsoft stored in the registry, WFP replaces the file with the proper Microsoft version of the file. This fea seriously impact an attacker's user-mode RootKit, automatically uninstalling the tool before the a has a chance to use it. Note that WFP focuses on checking for changes to existing files; if a new sensitive directory, WFP neither prevents nor logs the fact. Its sole focus is on stopping changes Windows files that Microsoft considers sensitive, acting as a built-in file integrity checking tool.

Microsoft created WFP to serve both stability and security needs. From a stability perspective, sc software installation programs inadvertantly modify or corrupt critical Windows files, possibly ma crash. In the security realm, attackers might try to alter critical system files with user-mode Roo case, WFP restores the original file, often without the user or system administrator even realizing saved from certain doom. WFP acts almost like Big Brother sitting over your shoulder. You might delete a WFP-protected file, replacing it with some new version. Thirty seconds later, though, the version mysteriously appears again, raised from the dead by WFP. The invisible hand of WFP trie straight, whether you wanted it to or not.

When it detects a change in a critical system file, WFP searches the system for a Microsoft-autho that file so that it can switch things back. WFP looks in the following locations, in this order, to lo version of the altered file:

- The Dllcache directory, which is stored by default in the System32 directory (usually C:\Wi \Dllcache on Windows 2000).
- The Driver.cab file, which is stored in the Driver Cache directory (C:\Winnt\Driver Cache\I by default on a Windows 2000 box).
- The original Windows 2000 installation, which could be stored on the hard drive itself or or directory accessible via Windows File Sharing, if the operating system was originally install network.
- A CD-ROM inserted in the local system.

When a good version of the program matching the appropriate digital signature is located, WFP suspect version, restoring the system to its original Microsoft-approved state. If a suitable good cannot be found, WFP notes this fact in the system logs and prompts the user via a dialog box, i error. This dialog box is shown in the bottom right-hand corner of Figure 7.15.

**Figure 7.15. Deleting tftp.exe from the dllcache and its original location pop up a dialog box.**

WFP typically runs in the background, cruising around the file system every few minutes, but an can manually force a WFP check to occur immediately. Using the command-line tool called the S Checker (SFC), an administrator can kick off a WFP check right away or at the next system boot command starts the process.

With WFP working its magic, you might wonder how an administrator would alter any of the legit the system, such as installing a patch. If WFP undoes all changes to these files, how can you pat Well, WFP allows files to be altered, provided that the alteration occurs using one of the following approved mechanisms:

- Windows Service Pack installation, using the program Update.exe.
- Hotfix distributions installed using the program Hotfix.exe.
- Operating system upgrade, using the program Winnt32.exe.
- Windows Update Feature.
- Windows Device Installer.

Each of these programs works with WFP to make sure that changes are allowed on the system. \ allow some changes to the system from time to time in the normal course of business.

## WFP in Action

To get a feel for WFP in action, let's pretend we're a user-mode RootKit and try to change a pret on a Windows system. You can try this on your own machine, if you'd like. We're going to experi called tftp.exe that isn't too important, so removing it shouldn't damage your machine.[2] Tftp.ex the Trivial File Transfer Protocol (TFTP), a younger sibling to the more robust FTP. TFTP allows u files around without providing any authentication, but it is seldom used legitimately on a Window attackers use TFTP on Windows systems to transfer backdoors to the machine. Because of this, remove it from my systems so that an attacker cannot exploit it. I feel a lot safer on a system w been deleted.

[2] However, if you do try this at home, you might want to back up your system, just in case. You should always keep a of a critical system handy. You've been warned!

By default, tftp.exe is located in the System32 directory. On this very machine where I'm typing just deleted it! Yikes… Is this the end of the world? Will I lose this document in a fiery flash of bi Not at all. After 30 seconds or so, WFP restored the file, no questions asked. My system is fine, a

previous condition, thanks to Big Brother, as manifested by WFP.

Now, I know what you are thinking. You're wondering what happens if we remove the file in the directory that WFP uses to restore tftp.exe. I like the way you think! That's what a RootKit might we'd better check it out. This time, I first removed the tftp.exe copy used by WFP in the Dllcache I deleted the regular tftp.exe, as shown in Figure 7.15.

How does WFP react to this situation? If it cannot find the file in the Dllcache folder, it checks to copy of the Windows install media on the hard drive, on the network, or in the CD-ROM drive. If good version of tftp.exe anywhere, WFP pops up a dialog box, shown in the bottom corner of Fig

WFP really wants to get a copy of that file, but cannot find it in the Dllcache directory. Therefore for the installation CD-ROM. Most users would hit the Retry button. The same message pops up repeating this Retry task a few times, most users will just hit Cancel, not realizing that their ope has been altered. Unfortunately, this warning dialog box isn't very helpful, as it doesn't even sho been altered. The user can only guess at which files are causing problems. Clicking More Informa either, as it, too, fails to show the name of the offending file. The More Information button just c system to display the following quite useless text:

```
Possible reasons for this problem:

- You have inserted the wrong CD (i.e., a different

  Windows 2000 product CD than the version installed).

- The CD-ROM drive in your system is not functioning.
```

Thank you so much, Big Brother. Although the dialog boxes are of limited value, when the user a gives up and clicks Cancel, WFP logs this event. Therefore, a diligent system administrator can f discover what really happened. I'm happy to say that the event log does indicate which file was as the user name that clicked Cancel in the dialog box, as shown in Figure 7.16.

**Figure 7.16. WFP logs an event indicating that it cannot restore tf**

Now, in our example, our alteration of the system, removing tftp.exe, was very small, and actua improved the security of the system. But an attacker could do even nastier things, such as instal user-mode RootKit, overwriting normal files all over the operating systems to mask the attacker' get backdoor access to the machine.

## Controlling Big Brother: WFP Settings

The WFP configuration is stored in the system registry, where most Windows operating system s found. There are several registry keys associated with WFP, all located under the registry locatio HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\. The WFP r include the SFCScan, SFCQuota, SFCDllCacheDir, SFCShowProgress, and SFCDisable keys. If the explicitly set in the registry, the system operates with the default behavior for each item.

The SFCScan value determines how WFP will act during the system boot process. This key has th settings:

- *0:* The default value of 0 does not do a comprehensive scan of all protected files at reboot, WFP in the background continuously after the box is booted up.
- *1:* The value 1 instructs WFP to check the integrity of all protected files at every reboot, si lowering system performance after booting up. With this setting, your system might be ve or 20 minutes after reboot.
- *2:* Setting this key to a value of 2 causes WFP to do a comprehensive scan once after the r all subsequent reboots, the tool will run continuously in the background.

It's important to note that this registry setting controls the behavior of WFP only during the boot Regardless of this key's value, WFP will always run continuously in the background after the syst

The SFCQuota key sets the maximum size, in megabytes, that the Dllcache directory can hold. T for this key is 400 MB. The value 0xFFFFFFFF allows all critical system files to be stored in the Dl regardless of the total size. The SFCDllCacheDir setting establishes where the Dllcache folder is l default on a Windows 2000 machine, it is set to C:\Winnt\System32\Dllcache. The SFCShowProg specifies whether a progress meter should be displayed on the screen during a WFP scan.

The SFCDisable key is very, very important, as it can be used to enable or disable WFP. There ar documented values for this key:

- *0:* This default value means that WFP is active, always running in the background.
- *1:* This setting will disable WFP, but prompt an administrator to re-enable it during the nex
- *2:* This value will disable WFP for the next reboot only, without prompting the user. For sul reboots, WFP will be automatically reactivated.
- *4:* The value 4 enables WFP, but disables all pop-up windows warning the user that files ha

It should be noted that for values 1 or 2 to take effect, a kernel debugger must be installed and system. If a kernel debugger is not in use, WFP will not be disabled.

So, Microsoft has publicly documented these four values for this supervital key. Note that the nu missing from this list. This setting, although undocumented, leaves WFP enabled. So, that round through 4, but, as we shall see shortly, there's another undocumented value for SFCDisable that for attackers.

## Attacking WFP

Now, how can an attacker manipulate WFP to install user-mode RootKit replacement programs o machine? The attacker has at least four options. First, the attacker could simply implement the t earlier, deleting the Dllcache backup version of the file first, and then replacing the actual file. Al

would certainly work, it has a downside for the bad guy: The user or administrator will see the w on the screen. Still, if the attacker expects the user to just hit Cancel, this approach works like a

Another approach an attacker could use to undermine WFP involves altering the location of the D modifying the SFCDllCacheDir registry key. A bad guy could simply create a new Dllcache folder the system to use it. WFP would then not be comparing the RootKit files to the original Microsoft technique doesn't work that well, however, as WFP checks the digital signatures of all protected appropriate digital signatures from Microsoft-certified programs, WFP won't be able to ensure th the attacker's programs. Therefore, this approach would generate many error log messages, as about, wondering why none of the signatures of files in the new Dllcache folder are correct.

A third approach is to set the SFCDisable registry key to the Microsoft-recommended value for tu By setting this registry value to 1 (and installing a kernel debugger), an attacker can disable WF the system boots up. The change isn't immediate, however. Until the system reboots, WFP is stil Therefore, an attacker would have to change this registry key, install a kernel debugger, force th reboot, and then install a user-mode RootKit. There is a downside to this approach for the attack next time the administrator logs into the system, the SFCDisable registry setting value of 1 mak system bring up a dialog box with this text:

```
Warning! Windows File Protection is not active on this

system. Would you like to enable Windows File Protection

now? This will enable Windows File Protection until the

next system restart. <Yes> <No>.
```

With a diligent system administrator, this message could trigger an investigation. However, som administrators might simply click No. We should note that the SFCDisable value of 2 isn't all that RootKit-wielding attacker, as WFP is only disabled for the very next reboot. For all subsequent re would be reactivated, scrubbing the attacker's RootKit from the system.

As a fourth option, the attacker could set the SFCDisable registry key to a different value, undoc Microsoft. I hinted at this possibility earlier. Several reverse engineers were able to determine th SFCDisable value of 0xFFFFFF9D would completely disable WFP on Windows 2000. At the next re be started. That's a profound discovery, with nary a mention from Microsoft in the WFP documer using this setting, the system won't print any dialog boxes indicating that WFP has been shut off though, the following message will be written to the system log: "Windows File Protection is not system." It's subtle, but Windows will tell you when WFP has been shut off, thankfully.

This fourth option is a little more complicated on Windows 2000 Service Pack 2 or later, Window Windows 2003. On these types of systems, in addition to changing the SFCDisable registry key attacker must alter a specific library on the system, called Sfc.dll. Using a hexadecimal editor, th change four hex digits in this library to activate the SFCDisable key. The particular four digits an depends on the patch level of the operating system [3].

After implementing these changes, whether by hand or using an automated program, the attack modify any file on the system. The disabled WFP will not interfere with the attacker's actions, ma ready for planting a user-mode RootKit.

### WFP Attack Example: Code Red II

One tool that implements this type of attack against WFP is the Code Red II worm, which attack systems running Microsoft's IIS Web server. In August 2001, this worm started its spread across following in the footsteps of the original Code Red worm. "Wait a minute," I can see you saying, Red II a worm, and not a RootKit?" Actually, it uses both types of techniques. Code Red II used mechanisms to carry elements of a user-mode RootKit to victim Windows machines. With this ca Red II was potentially far more damaging than the original Code Red worm, which focused only

flooding the White House Web site and not distributing backdoors. Code Red II, on the other har style tactics. We'll look at other specimens of combination malware in more detail in Chapter 9.

The Code Red II worm scanned for and penetrated victim IIS servers by exploiting a buffer over vulnerability. This worm used the exact same buffer overflow exploit that the earlier original Cod exploited to spread on the Internet a month before. Once running on the victim machine, the Co made copies of the Windows Cmd.exe command shell in several locations, including C:\Inetpub\ This directory is where the IIS server stores Web-accessible scripts. Code Red II copied the norn command shell (Cmd.exe) there, renaming it Root.exe. With a copy of the command shell execu this standard IIS scripts directory, an attacker can easily send commands to this shell remotely itself. There's a backdoor for you. Now, the worm had to conceal this backdoor by hiding the Roc

To accomplish this, Code Red II inserted a Trojan horse program called explorer.exe into the C:\ directories. The normal Windows Explorer implements the standard Windows GUI, painting prett desktop and accepting user input from the keyboard and mouse. Unfortunately, because of an e unpatched Windows machines called the Relative Path Vulnerability, a file named explorer.exe a directory structure (in C:\ or D:\) will be run by default whenever a user logs into the box. So, w in, the worm's code is executed when the GUI is initiated.

The worm's version of explorer.exe was simply a filtering wrapper that executed the real explore filter built into the tool, however, masked all references to the Root.exe and explorer.exe files cr worm. That's how the evil explorer.exe hid itself and the command shell backdoor in the Web se directory. The evil version of explorer.exe also had one other very interesting other feature. It al of the SFCDisable registry key, changing it to 0xFFFFFF9D so that WFP is disabled. That way, an access the system and make changes to files without having to worry about WFP restoring them user-mode RootKit-style techniques, Code Red II spread to thousands of systems very quickly in However, its damage was limited by the fact that a good number of administrators had already systems against the buffer overflow used by the original Code Red worm. Still, these WFP-disabl likely be used again in future tools.

## DLL Injection, API Hooking, and the AFX Windows RootKit

We've seen how an attacker places code like FakeGINA in between existing Windows component critical system files by disabling WFP. Next, we'll talk about another user-mode RootKit techniqu that is both pernicious and popular. Instead of messing around with Windows features for exten operating system or overwriting files, attackers are increasingly injecting their malicious code rig memory space of running processes on a machine. On a running Windows machine, several doze processes are executing at any given time, each with its own memory space. Some processes ar applications, such as Winword, the familiar Microsoft Word program. Others are associated with system, such as the Winlogon process used for authenticating users. With the proper privileges an attacker can inject malicious code into any already-running process on the box, overwrite exi that target process, and activate the attacker's code to run *inside* of the other process. Now, tha this form of user-mode RootKit on Windows, instead of replacing files on the hard drive, attacke malicious code into running processes.

### Windows Code: EXEs vs. DLLs

To understand how this code injection technique works, we need to discuss two of the most pror code on Windows machines: executable programs (EXEs) and dynamic link libraries (DLLs). EXE pronounced "E-X-Eees" or even "Ek-sees") are simply programs that can run on the machine. Of familiar with EXEs. You start them all the time, either by double-clicking them or typing their na command prompt. When an EXE starts to execute, it creates a running process on the system. D other hand, are not directly executed by themselves. Instead, they provide functions to a runnin that it can take some action.

DLLs are little bundles of code, broken up into several different functions. All of the related funct one or more DLLs are called an application programming interface (API). Each individual functior some action on the system. The running EXE processes on the system load DLLs into their mem

EXEs share the DLLs with each other and use them to accomplish various actions, such as displa window or sending information across the network. In fact, the Windows operating system itself a collection of EXEs, DLLs, the kernel, and some device drivers, all implemented in tens of millio code.

You can think of the relationship between EXEs and DLLs using the analogy of people and hand t like people. By themselves, they can accomplish various straightforward, self-contained goals. A around the block. An EXE can do basic mathematical operations, such as counting numbers. DLL hand tools, extending the reach of the EXE so it can accomplish more complex goals, interacting operating system and environment. Using a hammer and nails, a person can build a house. Or, a person can cut down a tree. Similarly, using a DLL, an EXE can display data in a window on the Furthermore, just as people can share tools, various EXE programs can share DLLs. A single DLL information on the screen can be utilized by thousands of different EXEs, all to benefit from the functionality.

## DLL Injection and API Hooking

So, an EXE program loads the various DLLs it requires and relies on them to take actions on the Attackers use a technique called *DLL injection* to force an unsuspecting running EXE process to a it never requested. Very rudely, an attacker injects code in the form of a DLL directly into the vic process's memory space. DLL injection requires several steps to be taken by the attacker [4], in

- *Allocating space in the victim process for the DLL code to occupy.* Microsoft has included a in the Windows API to accomplish this task, called VirtualAllocEx.

- *Allocating space in the victim process for the parameters required by the DLL to be injecte* can be done using the built-in Windows VirtualAllocEx function call.

- *Writing the name and code of the DLL into the memory space of the victim process.* Again, includes an API with a function for doing this step, too. The WriteProcessMemory function to write arbitrary data into the memory of a running process.

- *Creating a thread in the victim process to actually run the newly injected DLL.* As you migh by now, Windows includes an API with this capability, too. Microsoft has made this entire p easier with these various API calls. The CreateRemoteThread starts an execution thread in which will run any code already in that process, including a newly injected DLL.

- *Freeing up resources in the victim process after execution is completed.* If the attacker is e she can even free up the resources consumed by this technique after the victim thread or running, using the VirtualFreeEx function.

The attacker runs a DLL injection tool that creates an attacking process to utilize these Windows Like a snake injecting venom into a victim, the attacker's DLL injection process then inserts func other currently running process. No predefined functionality is required in the victim process. In process really doesn't have a say in the matter. The attacker's process uses the various Window calls to inject the code and make the victim process run it. So, for example, using this technique code for implementing a network backdoor shell listener inside of your running Notepad.exe proc currently editing a file. *Any* type of code can be injected into *any* running process. We should no these Microsoft-provided functions could be used in a legitimate fashion to extend running proce dynamically or to debug programs. As usual, attackers abuse this powerful capability to achieve

To perform each of these DLL injection steps, the attacker must have the *Debug Programs* right This privilege is normally used to attach a program debugger to running processes, so a system software developer can troubleshoot a problem by looking at running programs in detail. To carr debugger requires detailed access to a running program's memory structures, including all data as code. With this great level of access and control, the Debug Programs right is typically very ca on the system, given only to administrators, or to no one at all. However, by taking over a victin

conquering system or administrator privileges, attackers can exploit this capability by giving the Debug Programs right. Of course, beyond looking at the guts of running processes, attackers use Programs right to perform DLL injection.

Employing this technique, the attacker can inject code into an unsuspecting process. But what ty the attacker inject? Here's where the *API hooking* concept comes into play, allowing the bad guy mode RootKit-style techniques. The attacker *overwrites* code from existing DLLs already loaded i victim process. The bad guy hooks certain functions in the API offered by a legitimate DLL to ma provided by the attacker, as illustrated in Figure 7.17. That is, these function calls will no longer legitimate code in the DLL. Instead, when the running EXE process makes a certain function call action, such as displaying information in a window on the screen, the attacker's injected DLL will attacker's own code will then decide whether to accurately display the information, filter the outp some completely different activity. Alternatively, instead of wholesale replacement of the existin attacker could be more efficient by just wrapping existing DLL code inside the attacker's own inje functionality. Such wrapping lets the attacker write less code by relying on existing features in th of the work. Using this API hooking in coordination with DLL injection, an attacker can replace or system functionality, gaining backdoor access and hiding on a system. In other words, using the an attacker can implement a user-mode RootKit on Windows.

**Figure 7.17. Attackers use DLL injection to hook APIs in a victim p**

This entire multistep process of injecting DLLs and hooking APIs sounds pretty complex, with lot attacker to perform. However, attackers don't run these steps by hand. Instead, they use autom to conduct the entire process without much manual intervention at all. Indeed, specific DLL injec hooking tool suites have been developed, simplifying the entire process. MadCodeHook, develop called "Madshi," is one of these API tools that includes code that can inject DLLs into a variety of operating systems, including Windows 95/98/NT/2000/XP/2003. According to its author, MadCo injecting DLLs into already running processes as easy as possible." The attacker just writes a litt calls Madshi's code, passing it a handle to the running process, as well as the DLL to be injected. care of the rest. A list of various DLL injection and API hooking projects, including Madshi's ware Table 7.3.

Another developer, called EliCZ, released a similar tool, called EliRT. Building on EliCZ's work, ye developer called Aphex has created a really easy-to-use DLL injector. Aphex's tool is called Injec that summarizes its functionality quite well. The attacker runs Inject.exe at the command line, g parameters: the name of the running process to receive the DLL, as well as the name of the DLL So, to inject a hypothetical RootKit named RootKit.dll into the Winlogon authentication process u I'd simply have to type:

**Table 7.3. Various DLL Injection and API Hooking Tools**

| Tool Name | Author | Feature Summary | Location |
|---|---|---|---|
| MadCodeHook | Madshi | Extremely well- | *www.madshi.net/olddlp6.htm* |

| | | | |
|---|---|---|---|
| | | documented, full-featured DLL injection and API hooking libraries | |
| APIHijack | Wade Brainerd | Library for simplifying API hooking | www.codeproject.com/dll/apihijack.asp |
| EliRT | EliCZ | API that implements VirtualAllocEx, CreateRemoteThread, and other functions so they transparently function across older Windows platforms (Win95/98/Me) and newer systems (NT/2000/XP/2003) | www.anticracking.sk/EliCZ/export/EliRT.zip |
| Inject.exe | Aphex | Command-line executable, based on EliRT | www.megasecurity.org/Programming/StealthD |

```
C:\> inject.exe winlogon "C:\My Documents\RootKit.dll"
```

That's it. The functionality of RootKit.dll will now be inserted inside the active Winlogon process c
If RootKit.dll is any good, I've completely subverted the box. Implementing DLL injection and AP
never been so easy.

## AFX Windows RootKit: Using DLL Injection and API Hooking

Now, that RootKit.dll we discussed in the last section was completely hypothetical, but let's look
particularly powerful RootKit that utilizes DLL injection and API hooking: the AFX Windows Rootk
developed by Aphex, and distributed at www.iamaphex.cjb.net, this user-mode RootKit is focuse
things on all types of Windows systems, including Win95/98/Me/NT/2000/XP/2003. That pretty r
the major Windows releases over much of the past decade. Unlike the UNIX RootKits we discuss
AFX Windows RootKit doesn't include any functionality to implement a backdoor. AFX focuses sol
things. The attacker is expected to bring a separate backdoor tool to the party, such as a Netcat
any other tool that gives remote backdoor access. Once that separate remote backdoor access t
the AFX Windows RootKit hides the backdoor by utilizing DLL injection and API hooking to subve
programs on the Windows machine.

The AFX Windows RootKit is capable of masking four different aspects of backdoor programs: ru
files on the hard drive, registry keys, and TCP or UDP ports. An attacker would first take over th
install a separate backdoor tool. Then, the bad guy would install and use the AFX Windows Root
traces of that backdoor on the system. You can think of the AFX Windows RootKit as a cloaking f
attacker can deploy around backdoors on the system. With the cloaking field in place, administra
will not be able to see evidence of the backdoor's process, file, registry settings, or network conr

The tool consists of only one executable program, the AFX Windows RootKit Configuration Conso
to configure and generate custom RootKits based on the attacker's needs. The attacker activates
Configuration Console on a local system owned by the attacker. The Configuration Console does
on the victim machine. Using the Configuration Console, the attacker configures the RootKit and
an executable file to deploy to and run on the victim machine, a process illustrated in Figure 7.1

**Figure 7.18. Using the AFX Windows RootKit to generate an executable the target machine.**

Using the simple, highly intuitive Configuration Console GUI shown in Figure 7.19, the attacker c rules to mask various elements that will be invisible on the target system. Now, you might obser looks like it comes from an Apple Macintosh system. However, don't be fooled by the aqua look screen capture is really from my very own Windows 2000 machine. The author of the tool just g looking skin, even though it runs on Windows. The attacker selects each of the four tabs on the defines hiding rules for processes, files, registry keys, and network connections.

**Figure 7.19. The Configuration Console of the AFX Windows RootKit is u hiding rules.**

The hiding rule syntax for the AFX Windows RootKit is very straightforward. The attacker identifi processes, files, registry keys, and port numbers, which should be filtered out and not shown to administrator. The attacker can even employ the wildcard (*) symbol to match all substrings in t numbers. Using this syntax, I've defined two network connection filters for the RootKit in Figure hiding rule, which specifies *TCP*, will hide all TCP connections. My second rule, *UDP*:2222*: connections associated with local UDP port 2222. Check out that UDP syntax. Essentially, I'm de the output of the Windows `netstat` command, which shows listening ports using the following fc

```
Protocol   LocalAddress:Port   ForeignAddress:Port   State
```

By specifying *UDP*:2222*:*, I've said that I want to hide any usage of the UDP protocol for an using port 2222 connecting to any foreign address on any port. Strings matching my filter just w the output of `netstat`. After the RootKit is installed, `netstat` won't ever show the usage of UDP

The AFX Windows RootKit even includes a handy help function for defining these masking rules. friendly help button in the GUI, the screen shown in Figure 7.20 appears, offering coaching in de rules.

**Figure 7.20. The AFX Windows RootKit Help screen offers tips for defi rules.**

After setting up all of the hiding rules, the attacker clicks *Generate* in the GUI. The AFX RootKit ( Console then creates an executable RootKit file. The attacker moves this file to the target systen voilà! The target system suddenly hides everything defined in the hiding rules by the attacker.

The process used by the RootKit executable to install itself is shown in Figure 7.21. When it runs machine, the RootKit executable first makes a copy of itself in the System32 directory. Then, in Figure 7.21, it creates two other files in the same directory: *iexplore.dll* and *explorer.dll*. Gee, wi that, these files sure look like they belong on the machine, don't they? They look kind of like son might think are associated with the legitimate programs Internet Explorer (iexplore.exe) and Wi (explorer.exe). But pay careful attention to the file suffixes here; the RootKit creates iexplore.dll On a stock Windows machine, there just aren't any files named iexplore and explore with a DLL pretty tricky, and reminiscent of the naming games we discussed in Chapter 6.

**Figure 7.21. The interplay between iexplore.exe, iexplore.dll, explore explorer.dll when the AFX RootKit for Windows is executed.**

After writing these DLLs in the System32 directory, the RootKit executable injects explorer.dll in processes named explorer.exe, in Step 3. The explorer.exe process is the legitimate running pro displays the Windows GUI to the user. Once inside the legitimate explorer.exe process, the malic then does API hooking. It grabs the code inside of iexplore.dll in Step 4. To finish the process, in explorer.dll then injects iexplore.dll into the explorer.exe process, overwriting function calls asso displaying processes, files, registry keys, and connections. When a standard Windows tool, such Manager, File Explorer, Registry Editor, or `netstat` command are executed, the malicious API co the legitimate Windows Explorer filters the hidden stuff from the output. In this way, the attacke deeds are hidden on the machine.

If all of these different references to iexplore.exe, iexplore.dll, explore.exe, and explorer.dll are don't worry. That's what the attackers intended! However, by inspecting Figure 7.21, you can ge what's really happening with this RootKit.

Unfortunately, if you happen to stumble across this RootKit by observing iexplore.dll and explore System32 directory, you cannot uninstall the RootKit by simply deleting the DLLs. If you try to d Windows will bark at you, telling you that these files are in use and cannot be deleted. As long a system is running, it will not allow these DLLs to be removed from the system.

To show you how the AFX Windows RootKit works, I've installed it on one of my own machines, rules for network connections to all TCP ports and UDP port 2222 that we defined earlier. Figure results of running the `netstat` command on this system before and after installing the RootKit. enough, all of my TCP port usage has suddenly disappeared. Also, any usage of UDP port 2222 as well.

**Figure 7.22. Before installing the AFX Windows RootKit, netstat shows Afterward, all TCP port usage is hidden.**

Although Figure 7.22 shows only port usage, the same hiding technique can be applied to proces names, and registry keys using the AFX Windows RootKit. Truly, this capability is quite useful to

## User-Mode RootKit Defenses on Windows

How can you defend yourself from the scourge of user-mode RootKit attacks against your Windo Happily, the defensive tools and techniques required for these tools map very closely to those w UNIX systems. As we saw with UNIX, the user-mode RootKit defenses fall into three areas: prev detection, and response. Let's go over each, citing specific tools you can use to protect your Win

### User-Mode RootKit Prevention on Windows

As with UNIX RootKits, attackers require superuser privileges to implement each of the Windows techniques we've discussed in this chapter. Therefore, you need to harden and patch your syste ensure that an attacker cannot get administrator or System privileges on your machines. To har systems, there are a variety of guides and programs available. However, one of my favorites is t Pro Gold Template. To get a feel for this tool, consider the Windows security template features. XP, and 2003 all support the concept of a security template, which is just a file containing variou settings for the box. Security templates can be used to bundle together settings for account per settings, password controls, and logging, as well as a myriad of other Windows security configur applying the same template file to many systems, you can be sure that the overall security stan your environment meets a standard baseline. Administrators can apply these security template f using a variety of mechanisms, including the Secedit.exe command-line tool, the Security Config Analysis Tool GUI, or, if you've deployed Active Directory, via a Group Policy Object.

Microsoft ships Windows with a variety of security template files for workstations, servers, and d controllers. However, these built-in security templates tend to be either way too weak so that ar slice through them, or so strong that they render the system unusable in a real-world environme world needs is a reasonable security template that isn't too weak or too strong, but just right for environments.

The Center for Internet Security (CIS), the National Security Agency, and the SANS Institute, to variety of other organizations, embarked on creating just such a template. They spent several m standard that would meet the most pressing needs of all of these organizations. Finally, they ach and released their work, the Win2K Pro Gold Template, available at *www.cisecurity.org*. This tem reasonable baseline of security for Windows 2000 workstations for most organizations. It serves starting and reference point for your security configuration. You can tweak it to make it stronger restrictions for your environment. Note that, as of this writing, this standard applies only to Winc Windows 2000 Professional, and Windows 2000 Server machines. However, these same organiza working on templates for other versions of Windows machines right now.

CIS has also released a free scoring tool so you can check to see how well your security settings template, such as the Win2K Pro Gold Template. You run the scoring tool on a local system, and security stance to a baseline template, giving you a summary score between 0 and 10. The high the more closely you match the template used for comparison. To generate the score, the tool u algorithm that analyzes the Service Packs and Hotfixes installed on the machine, the account an settings, other security settings, and available services. This score is quite useful for comparing security stance of several different systems, but I don't get too hung up on the absolute score. Y that to support a given environment's needs, the maximum score you can get from the CIS scor of 10. That might sound pretty bad, but you might require those security settings for the service That's why I use the CIS scoring tool as a relative measure of security among several machines. scores a 5, but another one ranks a 3, I know that the latter system deviates more from the bas scoring tool, shown in Figure 7.23, is available for free at *www.cisecurity.org*.

**Figure 7.23. The CIS scoring tool scores a Windows system against a template.**

## User-Mode RootKit Detection on Windows

*Trust but verify.*

—Ronald Reagan

Prevention is a good thing, but no defense is completely impermeable. Therefore, you need solid capabilities for user-mode RootKits on your Windows systems. As with UNIX, file integrity checki of the best methods for looking for the malicious changes introduced by user-mode RootKits on built-in WFP provides a modicum of security, and you should be alert to any dialogue boxes or lo WFP telling you that a critical system file has been altered. You should investigate immediately if messages shown in Figures 7.15 and 7.16 earlier in this chapter.

Although WFP provides some protection against file changes, you need to go further, employing integrity checking tools for your critical systems. These tools scan the system, looking for chang system files based on cryptographic hashes of known good files and settings. Fcheck is a free to such functions on Windows, available at *www.geocities.com/fcheck2000/fcheck.html*. Additionall commercial version of the Tripwire tool also runs on Windows systems, available at *www.tripwire* bonus, on Windows, Tripwire also looks for alterations to critical registry settings, such as the SF that controls WFP and numerous other security configuration elements on the box. Several other integrity checking tools are available for Windows, including GFI LANguard System Integrity Mon Data Sentinel.

Beyond file integrity checkers, antivirus programs, such as those we discussed in Chapter 2, can the user-mode RootKits when they are loaded onto a system, before they are installed. Most ant have signatures for several different user-mode RootKits on Windows. For example, when I first Windows RootKit to my machine for testing purposes, my antivirus tool totally freaked out, prev program from being accessed. Only by disabling the antivirus tool was I able to install the AFX to purposes. So, by using antivirus tools, you'll raise the bar against casual attackers wielding user The bad guy will have to be smart enough to first disable the antivirus tool, or modify its signatu installing the RootKit.

Furthermore, you should carry around a CD-ROM with third-party tools you can use to analyze y Include programs that look for strange port usage, such as the Fport and TCPView tools we discu 5. The AFX Windows RootKit is powerful, but it only hides information using those components o knows to alter. If you show up with a separate tool that you run from a CD-ROM, you will be mu get the real scoop on what's happening on your system. Interestingly enough, William Salusky's CD-ROM, FIRE, also includes a few Windows tools on the same disk. Although the basic CD rema centric, this handful of Windows tools can be used to back the system up and conduct a forensic NTFS partition.

## User-Mode RootKit Response on Windows

After you've backed up the RootKit-infected system for a forensics analysis, you'll really need to scratch, using the base operating system install plus all patches and Hotfixes. Remember, you ca the system from the base install packages without patching it. If you do, the attackers will likely back into the machine using the same vulnerability they employed to get on the box in the first system is back in production, you need to very carefully monitor it, using network- and host-bas Detection Systems (IDSs). Also, monitor the logs of the machine very closely so you can quickly guy returns.

PREV < Day Day Up > NEXT