Kernel Manipulation Impact

Neo: This isn't real…

Morpheus: What is "real"? How do you define "real"? If you're talking about what you can feel, what you can smell, what you can taste and see, then "real" is simply electrical signals interpreted by your brain…

—Dialogue from the movie *The Matrix,* 1999

What happens if some bad guy starts manipulating the kernel itself? Because the kernel is all about control, by modifying the kernel, an attacker can change the system in a fundamental way. To apply changes to the kernel, the attacker first requires superuser privileges on the machine. To manipulate the kernel, root-level access is needed on UNIX machines, and administrator or system access is required on Windows systems. Once installed, a kernel-mode RootKit replaces or modifies components of the kernel. These alterations might make everything on the system appear to be running perfectly well, but the operating system is really rotten to the core. The attacker can change the kernel so that it lies about the status of the machine.

For example, the administrator might run a command looking to see if any backdoor processes are running. This command calls the kernel to get a list of running processes. However, the bad guy changed the kernel so that it lies, and doesn't show the attacker's backdoor process, as illustrated in Figure 8.2. Alternatively, an administrator might run a file integrity checker to see if some critical files on the machine have been changed. The deceiving kernel tells the administrator that no files have been altered; everything looks wonderful.

Figure 8.2. Manipulating the kernel to hide processes.

Using kernel manipulation, the attackers can alter the kernel so that it thoroughly hides the attacker's activities on the machine. Most kernel-mode RootKits include the following types of subterfuge:

- *File and directory hiding.* Most kernel-mode RootKits hide files and directories from users and system administrators. When a file is hidden, the kernel will lie to any program that

comes looking for the file.

- *Process hiding.* By hiding a process using a kernel-mode RootKit, the attacker can create an invisible backdoor that cannot be discovered using process analysis tools.

- *Network port hiding.* By hiding listening TCP and UDP ports so that local programs cannot see them, the bad guy's backdoor is even stealthier.

- *Promiscuous mode hiding.* The attacker doesn't want an administrator to detect a sniffer running on the box in promiscuous mode, so most kernel-mode RootKits lie about the promiscuous status of the network interface.

- *Execution redirection.* With this feature of many kernel-mode RootKits, when a user or administrator runs a program, the kernel pretends to run the requested program. However, the kernel really substitutes a different program in a bait-and-switch maneuver. Users and system administrators think they are running one program, but are really executing some other program of the attacker's choosing. For example, instead of relying on user-mode RootKit techniques to replace the secure shell daemon (sshd) on a victim machine, with a kernel-mode RootKit, an attacker can just redirect execution of the sshd executable to another version with a backdoor. The administrator can even check the integrity of the sshd file. However, the file will look completely intact, because it *is* intact. However, when a user or administrator tries to execute the sshd file by remotely logging in, the backdoor version will be executed, giving the bad guy remote access to the victim machine.

- *Device interception and control.* Using a kernel-mode RootKit, an attacker can intercept or manipulate data sent to or from any hardware device on the machine. For example, a bad guy could modify the kernel to record any keystrokes typed into the system in a local file on the machine, thereby implementing a very stealthy keystroke logger [1]. Alternatively, attackers have implemented kernel alterations that let them spy on users' terminal sessions (TTYs), observing and even injecting keystrokes, as well as the responses generated by the system [2].

Think about this from the attacker's point of view. With a user-mode RootKit, like those we covered in Chapter 7, the attacker has to break into the box and modify a bunch of programs to hide and implement a backdoor. On a UNIX system, the attacker might break in, start up a backdoor shell listener, and then use a tool like URK to replace `ps`, `ls`, `netstat`, and several other commands. The attacker then has to run the fix routine to set the modification dates and file lengths of these commands to the appropriate values. Then, the drudgery continues as the attacker configures the various hiding components and backdoors of URK. After all of this tiring work, the attacker still has to worry about a suspicious system administrator showing up with a CD-ROM full of statically linked binaries, such as Bill Stearns' static tools for Linux at *www.stearns.org/staticiso*, which won't lie about the system state. These user-mode RootKits are a lot of work, and aren't very stealthy if the administrators bring their own programs on a CD.

However, with a kernel-mode RootKit, the whole equation changes in favor of the attacker. Instead of modifying a bunch of individual programs, the attacker modifies the underlying kernel that these programs all rely on. To hide a file, the bad guy won't change `ls`, `find`, `du`, and other commands. Instead, the attacker just modifies the kernel so that it lies to *any* particular command or program run by the administrator looking for that file. In this way, kernel-mode RootKits are far more efficient for the attacker.

With a kernel-mode RootKit, the attacker morphs the system so that administrators and users are in a prison, but don't even realize it. You might think you are running certain programs or looking at the status of your machine, but you don't know that you are viewing a fantasy concocted by the attacker and implemented with a kernel-mode RootKit. What you see is not

really your operating system, but only a dream world designed to hide you from the truth: the truth that your operating system is really completely owned by the attacker. Without even being aware of your prison, you blithely go on living your life, managing your system, and unwittingly letting the attackers control everything.

Have you ever seen the movie *The Matrix*? If you haven't, I'll be careful not to give away any spoilers for those few souls who haven't yet seen the movie or its sequels. For those who have seen it, the movie provides some excellent illustrations that help make the ideas behind kernel-mode RootKits more concrete. You know, some people have compared *The Matrix* to the ultimate Rorschach test. Looking into and interpreting the meaning of the inkblot that is *The Matrix* really reveals your own philosophy and worldview. Some fans think the movie is about Buddhism, Christianity, Gnosticism, Hinduism, Islam, or Judaism. Others think it's a great flick about martial arts or firearms. But I'm here to tell you what *The Matrix* is really all about: kernel-mode RootKits.

In the movie, some pretty evil beings manipulate their victims so that they are wired into a virtual reality simulation that looks like the real world. With their brains wired into the Matrix, the victims believe they are living normal lives, paying their taxes, going to church, and taking out their landladies' garbage. However, the victims are really lying in pods full of pink goo, completely unaware of their real physical circumstances. The virtual reality image of their lives is merely a mirage, designed to enslave the victims so that the evil beings could use their resources. With a kernel-mode RootKit, you think you are looking at your real system, but the attackers have altered the kernel so that they can use your system resources without your knowledge. You might not realize it, but, with a kernel-mode RootKit, your computer is living a lie. Your computer is an attacker-controlled Matrix and you are unknowingly trapped inside.

Because various operating system kernels vary so significantly, we'll break the remainder of this chapter into two sections. First, we'll look at the Linux kernel and how bad guys manipulate it, and then we'll address Windows kernel-mode RootKits in the latter half of the chapter.

Keep in mind that for each of the concepts and attacks we discuss for Linux and Windows, analogous ideas apply to other operating systems. Given the differences in the kernel implementations of various UNIX variants (and our desire to keep this chapter under 200 pages), we need to pick one specimen from the UNIX world to analyze in more detail. We'll focus on Linux as one of the most common representatives of UNIX and UNIX-like operating systems. In addition to Linux, we'll look at the Windows kernel because of its widespread deployment and popularity as a target for kernel-mode RootKits. However, keep in mind that similar kernel-mode RootKit concepts have been implemented for other operating systems, including Solaris [3], FreeBSD [4], and others. By analyzing the details of kernel attacks on Linux and Windows, we can not only understand how they work in detail on the most popular platforms, but also get a high-level view of similar techniques that are used against other systems.

PREV < Day Day Up > NEXT