[in]security

by Gary McGraw

Innovative Rootkits: The Ultimate Weapon?

An attacker's job isn't complete until he or she can return to a compromised computer and use it at will—all while remaining deep in the shadows. To achieve this, attackers install rootkits on target systems. Malicious hackers call this "owning" your machine. While these nefarious tools have always been a problem, today's rootkits are becoming more sophisticated, harder to discover, and more difficult to remove. Here's what you need to know about them.

KNOW YOUR ROOTS

Rootkits occupy the pinnacle of the attacker's formidable tool set because they yield ultimate control over a target machine. They can be installed locally, or they can arrive via some other vector, such as a worm. In fact, virus code, worms, and rootkits have many things in common. They're all typically very small pieces of code that's extremely tightly written. They all employ stealth techniques, and they often use tricks such as call hooks, trampoline tables, and patches to obtain their goals.

Because worms are really a category of mobile code, worm payload often uses many of these tricks to infect a target system. A worm usually infects a target and leaves code behind, in effect becoming a rootkit.

Early rootkits involved making subtle changes to standard executables on a system. A classic Unix rootkit from the 1980s involved installing a compromised version of the directory utility ls, a much-used command to list the contents of a directory. An attacker could return to a system and gain privilege by passing in the proper secret arguments and executing the backdoor planted in ls. Other commonly accessed executable files, such as ps and netstat, were also targeted.

Because this technique involved changing the size and makeup of target executables, early rootkits could be detected by file integrity-checking software such as Tripwire, but many of these defenses no longer work today.

INTO AND UNDER THE KERNEL

Today's more sophisticated rootkits are harder to detect. Kernel rootkits, widely available on the Internet (see http://rootkit.com), are installed as loadable modules or device drivers and provide hardware-level access to a machine. Because these programs are fully trusted by the kernel, they can hide from any other software running on the machine. For example, when an application asks the kernel to list all running processes, a rootkit can make sure its processes are never revealed by scrubbing the list on its way back to the calling application. Kernel rootkits commonly hide both files and running processes to provide an undetectable backdoor into the owned machine.

This means aside from running your own rootkit that installs itself first, there's no way for you to tell if your machine has been victimized. More reactive approaches, such as monitoring your network for strange behavior and looking for weird ports being opened, can help, but these aren't a complete solution. OSs that sign kernel code and check kernel integrity are promising, but not yet standard issue.

Particularly nasty rootkits have even been known to ensconce themselves in the very chips of a computer. Consider that many modern PCs have around 2Mbytes of unused EEPROM space on the motherboard that can be accessed through software. If you're owned by one of these rootkits, even completely reinstalling the OS won't clear things up. Short of reflashing your EEPROM, you're fresh out of options.

> "Particularly nasty rootkits are able to ensconce themselves in the very chips of a computer."

One potential solution is to employ compartmentalized hardware. However, the delayed attempt to commercialize hardware security apparatuses in the form of Intel's LaGrange, Microsoft's Palladium/NGSC, and other systems shows how far we have to go to defeat the most sophisticated rootkits. Without specialized hardware as a defense, today's killer rootkits may be the ultimate weapon.

Gary McGraw is CTO of Cigital, a software quality management consultancy. He is co-author of *Exploiting Software* (Addison-Wesley, 2004), *Building Secure Software* (Addison-Wesley, 2001), and *Java Security* (Wiley, 1996). Reach him at gem@cigital.com.