4
EVOLUTION OF THE BOOTKIT

This chapter introduces you to the *bootkit*, a malicious program that infects the early stages of the system startup process, before the operating system is fully loaded. Bootkits have made an impressive comeback after their use diminished due to changes in the PC boot process. Modern bootkits use variations on old stealth and persistence approaches from these early bootkits to remain active on a target system for as long as possible without the system user's knowledge.

In this chapter, we take a look at the earliest bootkits; trace the fluctuating popularity of bootkits, including their spectacular comeback in recent years; and discuss modern boot-infecting malware.

The First Bootkits

The history of bootkit infections dates back to before the IBM PC hit the shelves. The title of "first bootkit" is usually bestowed upon Creeper, a self-replicating program discovered around 1971. Creeper ran under the TENEX networked operating system on VAX PDP-10s. The first known antivirus was a program called Reaper designed to remove Creeper infections. In this section, we'll look at early examples of bootkits from Creeper onward.

Boot Sector Infectors

Boot sector infectors (BSIs) were among the earliest bootkits. They were first discovered in the days of MS-DOS, the nongraphical operating system that preceded Windows, when the PC BIOS's default behavior was to attempt to boot from whatever disk it found in the floppy drive. As their name suggests, these malicious programs infected the boot sectors of floppy diskettes; the boot sectors were located in the first physical sector of the disk.

At bootup, the BIOS would look for a bootable diskette in drive A and run whatever code it found in the boot sector. If an infected diskette was left in the drive, it would infect the system with a BSI even if the disk wasn't bootable.

Although some BSIs infected both the diskette and the operating system files, most BSIs were *pure*, meaning they were hardware specific, with no OS component. Pure BSIs relied solely on BIOS-provided interrupts to communicate with the hardware and infect disk drives. This meant an infected floppy would attempt to infect IBM-compatible PCs regardless of the OS being run.

Elk Cloner and Load Runner

BSI viral software first targeted the Apple II microcomputer, whose operating system was usually entirely contained within the diskettes. Credit for the first virus to infect the Apple II goes to Rich Skrenta, whose Elk Cloner virus (1982–1983)[1] used an infection method, employed by BSIs, though it preceded PC boot sector viruses by several years.

Elk Cloner essentially injected itself onto the loaded Apple OS in order to modify it. The virus then resided in RAM and infected other floppies by intercepting disk accesses and overwriting their system boot sectors with its code. At every 50th bootup, it displayed the following message (sometimes generously described as a poem):

```
Elk Cloner:
The program with a personality

    It will get on all your disks
```

1. David Harley, Robert Slade, and Urs E. Gattikerd, *Viruses Revealed* (New York: McGraw-Hill/Osborne, 2001).

```
      It will infiltrate your chips
        Yes it's Cloner!


    It will stick to you like glue
      It will modify ram too
        Send in the Cloner!
```

The next known malware to affect Apple II was Load Runner, first seen in 1989. Load Runner would trap the Apple reset command triggered by the key combination CONTROL-COMMAND-RESET and take it as a cue to write itself to the current diskette, allowing it to survive a reset. This was one of the earliest methods of malware persistence, and it foreshadowed more sophisticated attempts to remain on a system undetected.

The Brain Virus

The year 1986 saw the appearance of the first PC virus, Brain. The original version of Brain affected only 360KB diskettes. A fairly bulky BSI, Brain infected the very first boot sector of a diskette with its loader. The virus stored its main body and the original boot sector in the available sectors on the diskette. Brain marked these sectors (that is, sectors with the original boot code and the main body) "bad" so that the OS wouldn't overwrite the space.

Some of Brain's methods have also been adopted in modern bootkits. For one, Brain stored its code in a hidden area, which modern bootkits typically do. Second, it marked the infected sectors as bad to protect the code from the housekeeping done by the OS. Third, it used stealth: if the virus was active when an infected sector was accessed, it would hook the disk interrupt handler to ensure that the system displayed the legitimate boot code sector instead. We'll explore each of these bootkit features in more detail over the next few chapters.

The Evolution of Bootkits

In this section, we'll look at how the use of BSIs declined as operating systems evolved. Then we'll examine how Microsoft's Kernel-Mode Code Signing Policy rendered previous methods ineffective, prompting attackers to create new infection methods, and how the rise of a security standard called *Secure Boot* presented new obstacles for modern bootkits.

The End of the BSI Era

As operating systems became more sophisticated, pure BSIs began to confront some challenges. Newer versions of operating systems replaced the BIOS-provided interrupts used to communicate with disks that had OS-specific drivers. As a result, once the OS was booted, the BSIs could no longer access BIOS interrupts and so could not infect other disks in the system. An attempt to execute a BIOS interrupt on such systems could lead to unpredictable behavior.

As more systems implemented a BIOS that could boot from hard drives rather than disks, infected floppies became less effective, and the rate of BSI infection began to decline. The introduction and increasing popularity of Microsoft Windows, along with the rapid decline of floppy disk use, dealt the death blow to old-school BSIs.

The Kernel-Mode Code Signing Policy

Bootkit technology had to undergo major revision with the introduction of Microsoft's Kernel-Mode Code Signing Policy in Windows Vista and later 64-bit versions of Windows, which turned the tables on attackers by incorporating a new requirement for kernel-mode drivers. From Vista onward, every system required a valid digital signature in order to execute; unsigned malicious kernel-mode drivers simply wouldn't load. Finding themselves unable to inject their code into the kernel once the OS was fully loaded, attackers had to look for ways to bypass integrity checks in modern computer systems.

We can divide all known tricks for bypassing Microsoft's digital signature checks into four groups, as shown in Figure 4-1.

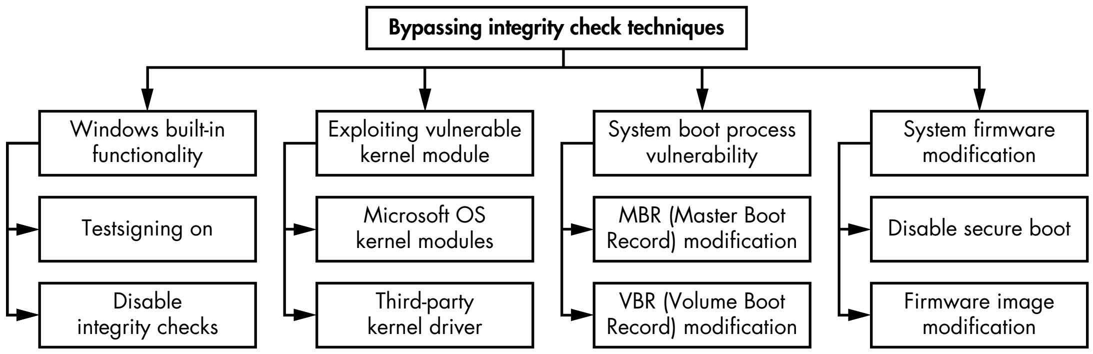
Bypassing integrity check techniques
Windows built-in functionality
Testsigning on
Disable integrity checks
Exploiting vulnerable kernel module
Microsoft OS kernel modules
Third-party kernel driver
System boot process vulnerability
MBR (Master Boot Record) modification
VBR (Volume Boot Record) modification
System firmware modification
Disable secure boot
Firmware image modification

Figure 4-1: Techniques for bypassing the Kernel-Mode Code Signing Policy

The first group operates entirely within user mode and relies on built-in Microsoft Windows methods for legitimately disabling the signing policy in order to debug and test drivers. The OS provides an interface for temporarily disabling driver image authentication or enabling test signing by using a custom certificate to verify the digital signature of the drivers.

The second group attempts to exploit a vulnerability in the system kernel or a legitimate third-party driver with a valid digital signature, which allows the malware to penetrate into kernel mode.

The third group targets the OS bootloader in order to modify the OS kernel and disable the Kernel-Mode Code Signing Policy. The newer bootkits take this approach. They execute before any OS component is loaded so they can tamper with the OS kernel to disable security checks. We'll discuss this method in detail in the next chapter.

The fourth group aims to compromise system firmware. As with the third group, its goal is to execute on the target system before the OS kernel does in order to disable security checks. The only major difference is that these attacks target firmware rather than bootloader components.

In practice, the third method—compromising the boot process—is the most common, because it allows for a more persistent attack. As a result, attackers returned to their old BSI tricks to create modern bootkits. The need to bypass integrity checks in modern computer systems has heavily influenced bootkit development.

The Rise of Secure Boot

Today, computers increasingly ship with functional Secure Boot protection. Secure Boot is a security standard designed to ensure the integrity of the components involved in the boot process. We'll look at it more closely in Chapter 17. Faced with Secure Boot, the malware landscape had to change again; instead of targeting the boot process, more modern malware attempts to target system firmware.

Just as Microsoft's Kernel-Mode Code Signing Policy eradicated kernel-mode rootkits and initiated a new era of bootkits, Secure Boot is currently creating obstacles for modern bootkits. We see modern malware attacking the BIOS more often. We'll discuss this type of threat in Chapter 15.

Modern Bootkits

With bootkits, as in other fields of computer security, *proofs of concept (PoCs)* and real malware samples tend to evolve together. A PoC in this circumstance is malware developed by security researchers for the purpose of proving that threats are real (as opposed to the malware developed by cybercriminals, whose goals are nefarious).

The first modern bootkit is generally considered to be eEye's PoC BootRoot, presented at the 2005 Black Hat conference in Las Vegas. The BootRoot code, written by Derek Soeder and Ryan Permeh, was a *Network Driver Interface Specification (NDIS)* backdoor. It demonstrated for the first time that the original bootkit concept could be used as a model for attacking modern operating systems.

But while the eEye presentation was an important step toward the development of bootkit malware, it took two years before a new malicious sample with bootkit functionality was detected in the wild. That distinction went to Mebroot, in 2007. One of the most sophisticated threats at the time, Mebroot posed a serious challenge to antivirus companies because it used new stealth techniques to survive after reboot.

The detection of Mebroot coincided with the release of two important PoC bootkits, Vbootkit and Stoned, at the Black Hat conference that same year. The Vbootkit code showed that it was possible to attack Microsoft's Windows Vista kernel by modifying the boot sector. (The authors of

Vbootkit released its code as an open source project.) The Stoned bootkit, which also attacked the Vista kernel, was named after the very successful Stoned BSI created decades earlier.

The release of both PoCs was instrumental in showing the security industry what sort of bootkits to look out for. Had the researchers hesitated to publish their results, malware authors would have succeeded in preempting a system's ability to detect the new bootkit malware. On the other hand, as it often happens, malware authors reused approaches from PoCs presented by security researchers, and new in-the-wild malware emerged shortly after the PoC presentation. Figure 4-2 and Table 4-1 illustrate this co-evolution.

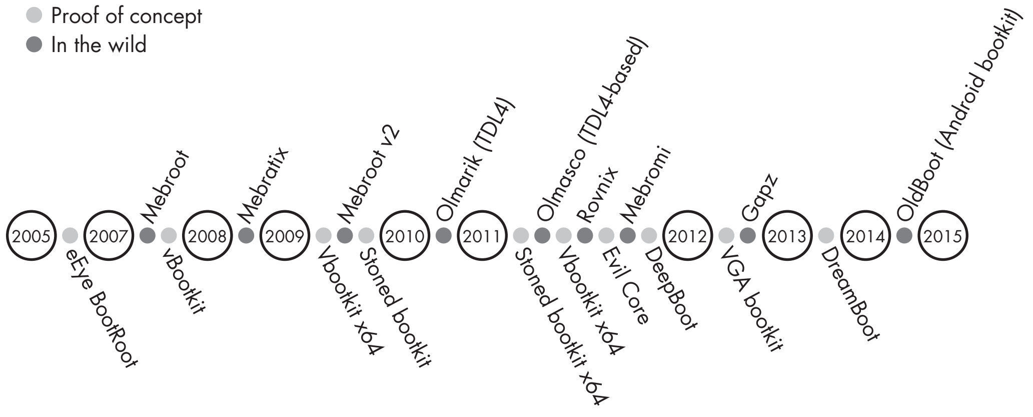

Figure 4-2: Bootkit resurrection timeline

Table 4-1: Evolution of Proof-of-Concept Bootkits vs. Real-World Bootkit Threats

Proof-of-concept bootkit evolution	Bootkit threat evolution
eEye BootRoot (2005) The first[1] MBR-based bootkit for Microsoft Windows operating systems	**Mebroot (2007)** The first well-known modern MBR-based bootkit (we'll cover MBR-based bootkits in detail in Chapter 7) for Microsoft Windows operating systems in the wild
Vbootkit (2007) The first bootkit to abuse Microsoft Windows Vista	**Mebratix (2008)** The other malware family based on MBR infection
Vbootkit 2 x64 (2009) The first bootkit to bypass the digital signature checks on Microsoft Windows 7	**Mebroot v2 (2009)** The evolved version of Mebroot malware
Stoned (2009) Another example of MBR-based bootkit infection	**Olmarik (TDL4) (2010/11)** The first 64-bit bootkit in the wild
Stoned x64 (2011) MBR-based bootkit supporting the infection of 64-bit operating systems	**Olmasco (TDL4 modification) (2011)** The first VBR-based bootkit infection
Evil Core[3] (2011) A concept bootkit that used SMP (symmetric multiprocessing) for booting into protected mode	**Rovnix (2011)** An evolved VBR-based infection with polymorphic code

Proof-of-concept bootkit evolution	Bootkit threat evolution
DeepBoot[4] **(2011)** A bootkit that used interesting tricks to switch from real mode to protected mode	**Mebromi (2011)** The first exploration of the concept of BIOS kits seen in the wild
VGA[5] **(2012)** A VGA-based bootkit concept	**Gapz**[6] **(2012)** The next evolution of VBR infection
DreamBoot[7] **(2013)** The first public concept of a UEFI bootkit	**OldBoot**[8] **(2014)** The first bootkit for the Android OS in the wild

1. When we refer to a bootkit as being "the first" of anything, note that we mean the first *to our knowledge*.

2. Nitin Kumar and Vitin Kumar, "VBootkit 2.0—Attacking Windows 7 via Boot Sectors," HiTB 2009, *http://conference.hitb.org/hitbsecconf2009dubai/materials/D2T2%20-%20Vipin%20and%20Nitin%20Kumar%20-%20vbootkit%202.0.pdf.*

3. Wolfgang Ettlinger and Stefan Viehböck, "Evil Core Bootkit," NinjaCon 2011, *http://downloads.ninjacon.net/downloads/proceedings/2011/Ettlinger_Viehboeck-Evil_Core_Bootkit.pdf.*

4. Nicolás A. Economou and Andrés Lopez Luksenberg, "DeepBoot," Ekoparty 2011, *http://www.ekoparty.org//archive/2011/ekoparty2011_Economou-Luksenberg_Deep_Boot.pdf.*

5. Diego Juarez and Nicolás A. Economou,"VGA Persistent Rootkit," Ekoparty 2012, *https://www.secureauth.com/labs/publications/vga-persistent-rootkit/.*

6. Eugene Rodionov and Aleksandr Matrosov, "Mind the Gapz: The Most Complex Bootkit Ever Analyzed?" spring 2013, *http://www.welivesecurity.com/wp-content/uploads/2013/05/gapz-bootkit-whitepaper.pdf.*

7. Sébastien Kaczmarek, "UEFI and Dreamboot," HiTB 2013, *https://conference.hitb.org/hitbsecconf2013ams/materials/D2T1%20-%20Sebastien%20Kaczmarek%20-%20Dreamboot%20UEFI%20Bootkit.pdf.*

8. Zihang Xiao, Qing Dong, Hao Zhang, and Xuxian Jiang, "Oldboot: The First Bootkit on Android," *http://blogs.360.cn/360mobile/2014/01/17/oldboot-the-first-bootkit-on-android/.*

We'll go over the techniques used by these bootkits in later chapters.

Conclusion

This chapter has discussed the history and evolution of boot compromises, giving you a general sense of bootkit technology. In Chapter 5, we'll go deeper into the Kernel-Mode Code Signing Policy and explore ways to bypass this technology via bootkit infection, focusing on the TDSS rootkit. The evolution of TDSS (also known as TDL3) and the TDL4 bootkit neatly exemplifies the shift from kernel-mode rootkits to bootkits as a way for malware to persist undetected for longer on a compromised system.