BOOTKITS: PAST, PRESENT & FUTURE

Eugene Rodionov
ESET, Canada

Alexander Matrosov
Intel, USA

David Harley
ESET North America, UK

Email rodionov@eset.com; alexander.matrosov@intel.com; david.harley.ic@eset.com

ABSTRACT

Bootkit threats have always been a powerful weapon in the hands of cybercriminals, allowing them to establish a persistent and stealthy presence in their victims' systems. The most recent notable spike in bootkit infections was associated with attacks on 64-bit versions of the *Microsoft Windows* platform, which restrict the loading of unsigned kernel-mode drivers. However, these bootkits are not effective against UEFI-based platforms. So, are UEFI-based machines immune against bootkit threats (or would they be)?

The aim of this presentation is to show how bootkit threats have evolved over time and what we should expect in the near future. First, we will summarize what we have learned about the bootkits seen in the wild targeting the *Microsoft Windows* platform: from TDL4 and Rovnix (the one used by the Carberp banking trojan) up to Gapz (which employs one of the stealthiest bootkit infection techniques seen so far). We will review their infection approaches and the methods they have employed to evade detection and removal from the system.

Secondly, we will look at the security of the increasingly popular UEFI platform from the point of view of the bootkit author as UEFI becomes a target of choice for researchers in offensive security. Proof-of-concept bootkits targeting *Windows 8* using UEFI have already been released. We will focus on various attack vectors against UEFI and discuss available tools and what measures should be taken to mitigate against them.

INTRODUCTION

The first bootkits started to emerge on the malware scene as cybercriminals realized that bootkit development was a way in which they could increase the profitability of a kernel-mode rootkit by widening the range of its targets to include users of 64-bit machines. This resulted in a trend whereby rootkit developers began to focus on bootkits.

The main obstacle to 64-bit development was the need to bypass the *Microsoft* kernel-mode code signing policy for system drivers, and this is the rationale behind modern bootkit development. However, the history of the bootkit begins much earlier than that.

BOOTKIT EVOLUTION

The first IBM-PC-compatible boot sector viruses from 1987 used the same concepts and approaches as modern threats, infecting boot loaders so that the malicious code was launched even before the operating system was booted.

In fact, attacks on the PC boot sector were already known from (and even before) the days of MS-DOS, and these have a part to play in our understanding of the development of approaches to taking over a system by compromising and hijacking the boot process.

The first microcomputer to have been affected by viral software seems to have been the *Apple II*. At that time, *Apple II* diskettes usually contained the disk operating system. Around 1981 [1], there were already versions of a 'viral' DOS reported at *Texas A&M*. In general, though, the 'credit' for the 'first' *Apple II* virus is given to Rich Skrenta's Elk Cloner (1982–3) [2, 3].

Although Elk Cloner preceded PC boot sector viruses by several years, its method of boot sector infection was very similar. It modified the loaded OS by hooking itself, and stayed resident in RAM in order to infect other floppies, intercepting disk accesses and overwriting their system boot sectors with its own code. The later Load Runner (1989), affecting *Apple IIGS* and *ProDOS* [2], rarely gets a mention nowadays, but its speciality was to trap the reset command triggered by the key combination CONTROL+COMMAND+RESET and take it as a cue to write itself to the current diskette, so that it would survive a reset. This may not be the earliest example of 'persistence' as a characteristic of malware that refused to go away after a reboot, but it's certainly a precursor to more sophisticated attempts to maintain a malicious program's presence.

© Brain damage

The first PC virus is usually considered to be Brain, a fairly bulky Boot Sector Infector (BSI), which misappropriated the first two sectors for its own code and moved the original boot code up to the third sector, marking the sectors it used as 'bad' so that they wouldn't be overwritten.

Brain had some features that significantly prefigured some of the characterizing features of modern bootkits. First, the use of a hidden storage area in which to keep its own code (though in a much more basic form than TDSS and its successors). Secondly, the use of 'bad' sectors to protect that code from legitimate housekeeping by the operating system. Thirdly, the stealthy hooking of the disk interrupt handler to ensure that the original, legitimate boot sector stored in sector three was displayed when the virus was active [2].

The volume of boot sector infectors and infection first began to decline when it became possible to change the boot order in setup so that the system would boot from the hard disk and ignore any left-over floppy. However, it was the increasing take-up of modern *Windows* versions and the virtual disappearance of the floppy drive that finally killed off the old-school BSI.

BOOT INFECTION REBOOTED

Windows – and hardware and firmware technology – has moved on since Brain and its immediate successors, and boot infection

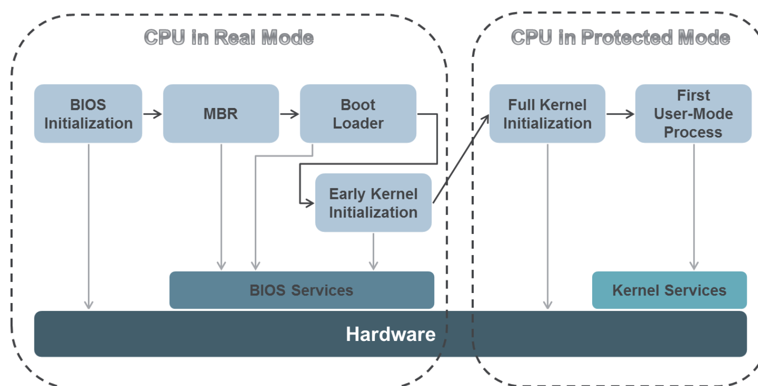

Figure 1: The system booting flow.

has evolved into new types of attack on operating system boot loaders, especially since *Microsoft* started to use a kernel-mode code signing policy in its 64-bit operating systems.

All bootkits aim to modify and subvert operating system components before the OS can be loaded. The most interesting target components (Figure 1) are as follows: BIOS/UEFI, MBR (Master Boot Record) and the operating system boot loader.

The harbinger of modern bootkits is generally considered to be *eEye*'s proof of concept (PoC) BootRoot [4], which was presented at BlackHat 2005. BootRoot was an NDIS (Network Driver Interface) backdoor demonstrating the use of an old vector as a model for modern OS attacks.

At BlackHat 2007, Vbootkit [5] was released. This PoC code demonstrated possible attacks on the *Windows Vista* kernel by modifying the boot sector. The authors of Vbootkit released its code as an open-source project, and that release coincided with the initial detection of the first malicious bootkit, Mebroot.

This unusually sophisticated malware offered a real challenge for anti-virus companies because it used new stealth techniques for surviving after a reboot. The Stoned bootkit [6] was also released at BlackHat, apparently so named in homage to the much earlier, but very successful Stoned BSI.

These proof-of-concept bootkits are not the direct cause for the coinciding release of unequivocally malicious bootkits such as Mebroot [7]. Malware developers were already searching for new and stealthy ways to extend the window of active infection before security software detected an infection. In addition, in 2007 *Microsoft Windows Vista* enforced a kernel-mode code signing policy on 64-bit operating systems, regulating the distribution of system drivers. This triggered the resurrection of stealth implementation by subversion of the boot process, in the form of modern bootkits.

All known bootkits conform to one of two categories. The first group consists of proof-of-concept demonstrations developed by security researchers, and the second consists of the real and unequivocally malicious threats developed by cybercriminals (see Table 1).

Evolution of proof-of-concept bootkits	**Evolution of bootkit threats**
eEye BootRoot – 2005 The first MBR-based bootkit for *MS Windows* operating systems.	Mebroot – 2007 The first MBR-based bootkit in the wild.
Vbootkit – 2007 The first bootkit to target *Microsoft Windows Vista*.	Mebratix – 2008 The other malware family based on MBR infection.
Vbootkit x64 – 2009 [8] The first bootkit to bypass the digital signature checks on *MS Windows 7*.	Mebroot v2 – 2009 The evolved version of the Mebroot malware.
Stoned Bootkit – 2009 Another example of MBR-based bootkit infection.	Olmarik (TDL4) – 2010/11 The first 64-bit bootkit in the wild.
Stoned Bootkit x64 – 2011 MBR-based bootkit supporting the infection of 64-bit operating systems.	Olmasco (TDL4 modification) – 2011 The first VBR-based bootkit infection.
DeepBoot – 2011 [9] Used interesting tricks to switch from real-mode to protected mode.	Rovnix – 2011 The evolution of VBR-based infection with polymorphic code.
Evil Core – 2011 [10] This concept bootkit used SMP (symmetric multiprocessing) for booting into protected-mode	Mebromi – 2011 The first exploration of the concept of BIOSkits seen in the wild.
VGA Bootkit – 2012 [11] VGA-based bootkit concept.	Gapz – 2012 [12] The next evolution of VBR infection
DreamBoot – 2013 [13] The first public concept of UEFI bootkit.	OldBoot - 2014 [14] The first bootkit for the *Android* operating system in the wild.

Table 1: The chronological evolution of PoC bootkits versus real-world bootkit threats.

Bootkit classification

The main idea behind bootkits is to abuse and subvert the operating system in the course of the initial boot process. At the

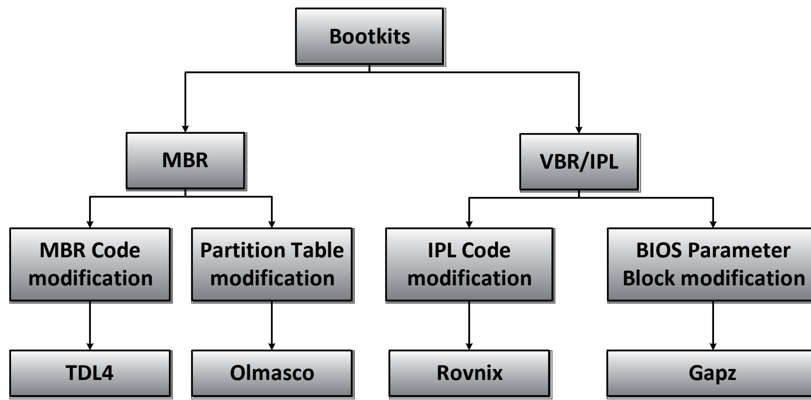

Figure 2: Bootkit classification by type of boot sector infection.

very beginning of the bootup process, the BIOS code reads the Master Boot Record at the first sector of the bootable hard drive, to which it transfers control. The MBR consists of the boot code and a partition table that describes the hard drive's partitioning scheme. Modern bootkits can be classified into two groups, according to the type of boot sector infection employed: MBR and VBR (Volume Boot Record) bootkits (see Figure 2). The more sophisticated and stealthier bootkits we see are based on VBR infection techniques.

TDL4 and Olmasco

TDL4 [15] and Olmasco [16] bootkits both target the MBR of the bootable hard drive – however, they differ in that TDL4 overwrites MBR code, whereas Olmasco modifies the MBR's partition table. Both infection approaches have the same result. Malicious components are initialized at boot time in order to load the malicious kernel-mode driver from the hidden storage area, and thus bypass the *Microsoft* kernel-mode code signing policy enforced on x64 platforms. In Table 2, we show the modules stored in the hidden file system of the TDL4 and used in the boot chain.

File name	Description
mbr	Original contents of the infected hard drive boot sector
ldr16	16-bit real-mode loader code
ldr32	Fake kdcom.dll for x86 systems
ldr64	Fake kdcom.dll for x64 systems
drv32	The main bootkit driver for x86 systems
drv64	The main bootkit driver for x64 systems

Table 2: TDL4 boot components.

Figure 3 summarizes the boot process followed by the TDL4 bootkit on *Windows Vista* and *Windows 7* operating systems.

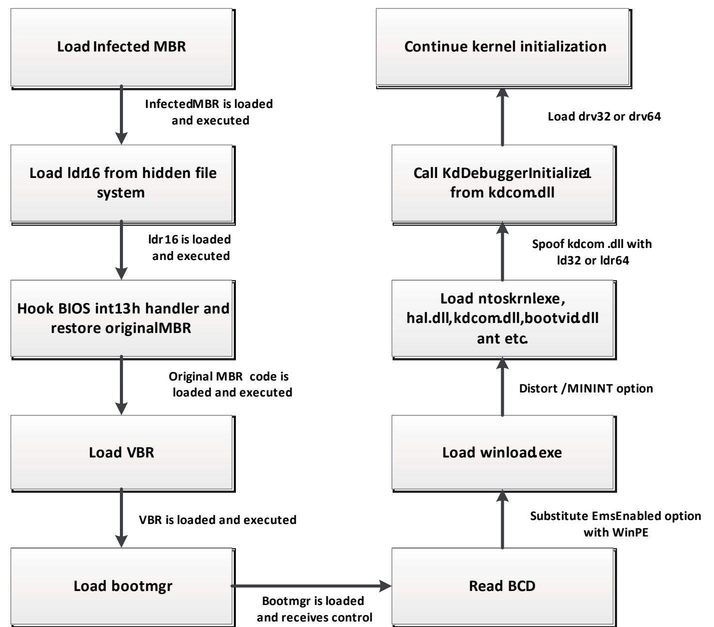

Figure 3: TDL4 bootkit workflow.

Rovnix

Win32/Rovnix is the first known bootkit to target the VBR. Its infection routine reads the 15 sectors following the VBR, which contain the Initial Program Loader (IPL) code. These sectors are compressed and appended to the malicious bootstrap code. The resulting code is then written to the 15 sectors that follow the VBR, as shown in Figure 4. Consequently, on the next system start-up, the malicious bootstrap code receives control.

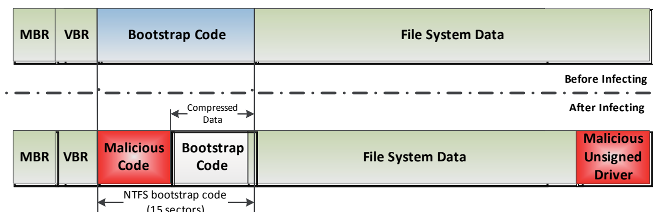

Figure 4: Win32/Rovnix approach to infection.

When the malicious bootstrap code is executed it hooks the Int 13h handler in order to patch ntldr/bootmgr system components so as to gain control after the boot loader

components are loaded. After that it decompresses and returns control to the original bootstrap code.

In order to load its malicious unsigned driver into kernel-mode address space and bypass the kernel-mode code signing policy, Win32/Rovnix employs the following technique. First, in order to propagate itself through processor execution mode switching (from real mode into protected mode), it uses the IDT (Interrupt Descriptor Table). This is a special system structure which is used in protected mode and consists of interrupt handler descriptors. The malware copies itself over the second half of the IDT, which is not used by the system. Secondly, it hooks the int 1h protected mode handler and sets hardware breakpoints so as to be able to receive control at specific points of the OS kernel loading process. By using debugging registers dr0–dr7, which are an essential part of the x86 and x64 architectures, the malware gets control at some point during the kernel initialization and loads its own malicious driver manually, thus bypassing the kernel-mode code integrity check.

Gapz

Historically, there are two modifications of the bootkit Win32/Gapz implementing different infection methods. The first version of the malware acted like a traditional MBR infector, while the other version employed a rather sophisticated stealth approach to infecting the VBR. For this reason, in this section we will focus on the latter, more interesting approach. What is remarkable about this technique is that only a few bytes of the original VBR are affected. The essence of this approach is that Win32/Gapz modifies the 'Hidden Sectors' field of the VBR, while all the other data and code of the VBR and IPL remain untouched.

The field that is targeted by the malware is located in the Volume Parameter Block (VPB), which is a special data structure located in the VBR and describing the attributes of the NTFS volume. The purpose of the 'Hidden Sectors' fields is to provide an offset in sectors to the Initial Program Loader (IPL) from the beginning of the volume, as illustrated in Figure 5.

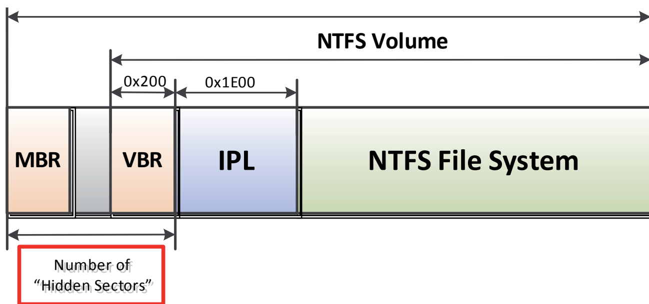

Figure 5: 'Hidden Sectors' field of BPB.

The IPL code is loaded and executed by the VBR: thus, by modifying value of the 'Hidden Sectors' field, the malware is able to intercept execution flow at boot time, as shown in Figure 6. The next time the VBR code is executed, it loads and executes the bootkit code instead of the legitimate IPL. The bootkit image is written either before the very first partition or after the last partition of the hard drive.

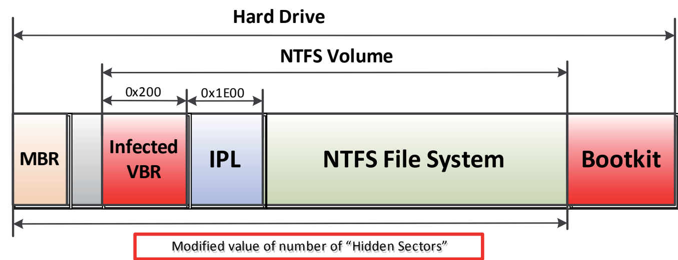

Figure 6: Layout of a hard drive infected by Win32/Gapz.

The main purpose of the bootkits considered above is to load and pass control to the malware's kernel-mode module without being noticed by security software. The kernel-mode module of Win32/Gapz isn't a conventional PE image, but is composed of a set of blocks with position-independent code, each block serving a specific purpose as described in Table 3.

Block #	Implemented functionality
1	General API, gathering information on the hard drives, CRT string routines, etc.
2	Cryptographic library: RC4, MD5, SHA1, AES, BASE64, etc.
3	Hooking engine, disassembler engine.
4	Hidden storage implementation.
5	Hard disk driver hooks, self-defence.
6	Payload manager.
7	Payload injector into processes' user-mode address space.
8	Network communication: data link layer.
9	Network communication: transport layer.
10	Network communication: protocol layer.
11	Payload communication interface.
12	Main routine.

Table 3: Win32/Gapz blocks description.

Win32/Gapz: hidden storage implementation

So as to store payload and configuration information secretly Win32/Gapz implements hidden storage. The image is located in a file named

'\??\C:\System Volume Information\{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}'

where X signifies hexadecimal numbers generated based on configuration information. The file is formatted as a FAT32 volume.

To keep the information stored within the hidden storage secret, its content is encrypted. The malware utilizes AES with key length 256 bits in CBC (Cipher Block Chaining) mode to encrypt/decrypt each sector of the hidden storage. As IV (Initialization Value) for CBC mode, Win32/Gapz utilizes the number of the first sector being encrypted/decrypted. Thus, even though the same key is used to encrypt every sector of the hard

drive, using different IVs for different sectors results in different ciphertexts each time.

Win32/Gapz: network communication

In order to communicate with C&C servers, Win32/Gapz employs a rather sophisticated network implementation. The network subsystem is designed in such a way as to bypass personal firewalls and network-traffic-monitoring software running on the infected machine. These features are achieved due to customized implementation of TCP/IP stack protocols in kernel mode, the implementation being based on the miniport adapter driver. According to the NDIS specification, the miniport driver is the lowest driver in the network driver stack – thus, using its interface makes it possible to bypass network-traffic-monitoring software, as shown in Figure 7.

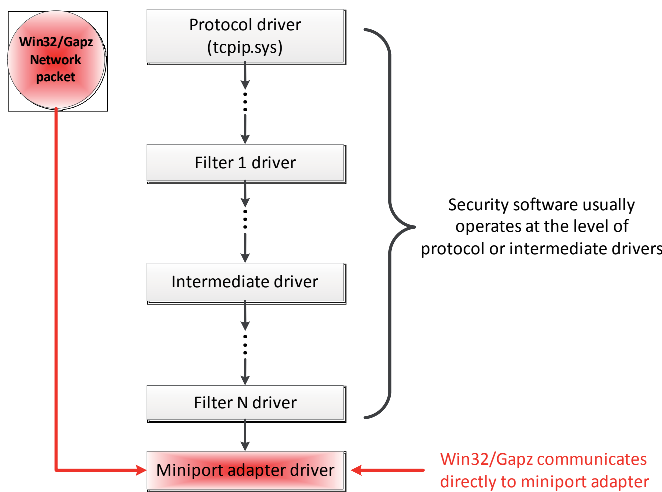

Figure 7: Win32/Gapz custom network implementation.

The malware obtains a pointer to the structure describing the miniport adapter by inspecting the NDIS library (ndis.sys) code manually. The routine responsible for handling NDIS miniport adapters is implemented in block #8 of the kernel-mode module. The architecture of the Win32/Gapz network subsystem is presented in Figure 8.

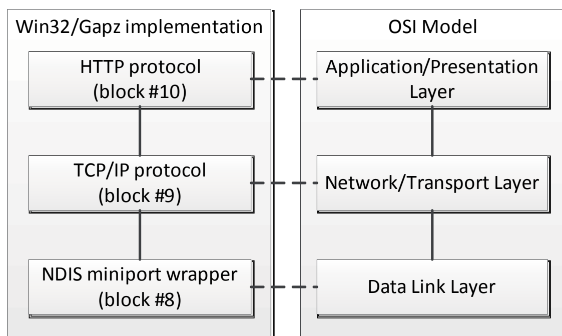

Figure 8: Win32/Gapz network architecture.

This approach allows the malware to use the socket interface to communicate with the C&C server without being noticed.

Communication with C&C servers is performed over HTTP. The malware enforces encryption to protect the confidentiality of the messages being exchanged between the bot and C&C server and to check the authenticity of the message source (to prevent subversion by commands from C&C servers that are not 'authentic' – a technique often used by security researchers to disrupt a malicious botnet). The main purpose of the protocol is to request and download the payload and report the bot status to the C&C server.

UEFI SECURITY

UEFI stands for Unified Extensible Firmware Interface: the specification was originally developed to replace legacy BIOS boot software. The boot process in UEFI is substantially different from that in the legacy BIOS environment: there is no longer any MBR and VBR code, which on older systems eventually load bootmgr and winload; these components are replaced with the UEFI boot code. Instead of an MBR-based partitioning scheme, the GPT (GUID Partition Table) partitioning scheme is used as the layout of the hard drive. The UEFI bootloader is loaded from the special partition, referred to as the EFI System Partition, formatted using the FAT32 file system (FAT12 and FAT16 are also possible). The path to the bootloader is specified in a dedicated NVRAM variable. For instance, for *Microsoft Windows 8*, the path to the bootloader looks like this: '\EFI\Microsoft\Boot\bootmgfw.efi'. The purpose of this module is to locate the OS's kernel loader (winload.efi for *Microsoft Windows 8*) and transfer control to it. The functionality of winload.efi is essentially the same as that of winload.exe – that is, to load the OS kernel image.

UEFI bootkit: Dreamboot

As noted in Table 1, Dreamboot is the first public proof-of-concept bootkit targeting UEFI and *Windows 8*. The bootkit infection results in the replacement of the original UEFI bootloader with a malicious substitute. When this is executed by UEFI boot code, it looks for the original bootloader (bootmgfw.efi), loads it and hooks the Archpx64TransferTo64BitApplicationAsm routine. The hook allows it to receive control at the time when the OS kernel loader – winload.efi – is in memory, but before the loader is executed. At this point the malware sets up another hook in winload.efi on the OslArchTransferToKernel routine: the name is self-explanatory. The latter hook is triggered when the OS kernel image has been mapped into system address space and Dreamboot patches it in order to disable kernel-mode security checks (PatchGuard and so on). Figure 9 summarizes the Dreamboot boot process.

FUTURE THREATS AND FUTURE TOOLS

Implementing forensics procedures for the UEFI platform is a problem because popular forensic software is not covered. However, proof-of-concept UEFI bootkits have already been presented at many security conferences in the last few years, some of them with source code. In the previous case of bootkits targeting legacy bootstrap code in the MBR, it was two years from the release of the first publicly known PoC code to

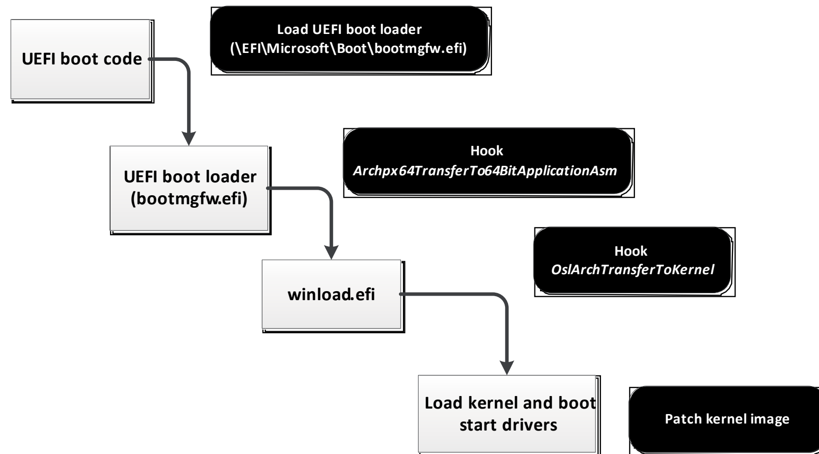

Figure 9: Dreamboot boot process.

malware samples being seen in the wild. The motivation for attacks on UEFI is growing every year because the number of PCs and laptops with a legacy BIOS is decreasing year on year. The number of people using *Microsoft Windows 8* is also growing, which means that the number of users with active Secure Boot is increasing.

UEFI malware infection can attack by way of a number of different vectors:

- The first type uses the same approach as the Dreamboot bootkit, based on replacing the original *Windows* Boot Manager and adding a new boot loader (Figure 10).
- The second approach is by directly abusing the UEFI DXE (Driver execution Environment) driver [17] (Figure 11).
- The third method is to patch the UEFI 'Option ROM': for example, the DXE Driver in Add-On Card (Network, Storage …), which isn't embedded in the firmware volume in ROM [18, 19] (Figure 12).

The Secure Boot implementation in the latest version of *Microsoft Windows* protects the booting process from malware

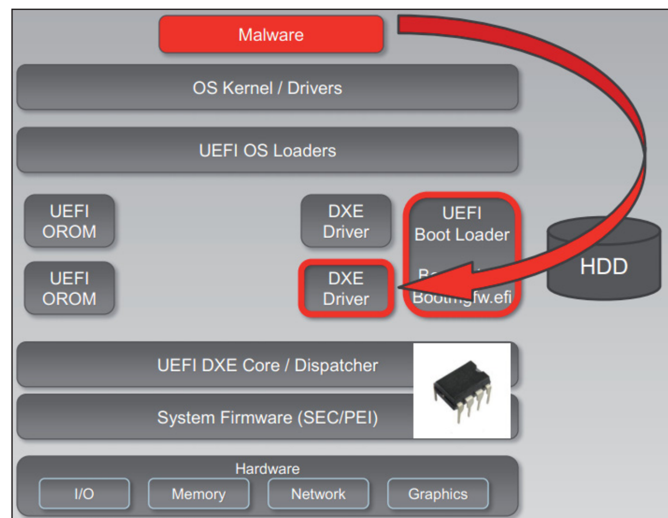

Figure 11: UEFI infection by abusing DXE driver.

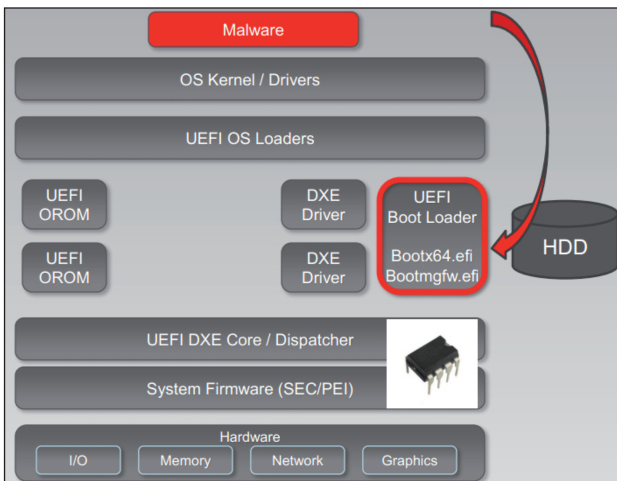

Figure 10: UEFI infection by replaced boot loader.

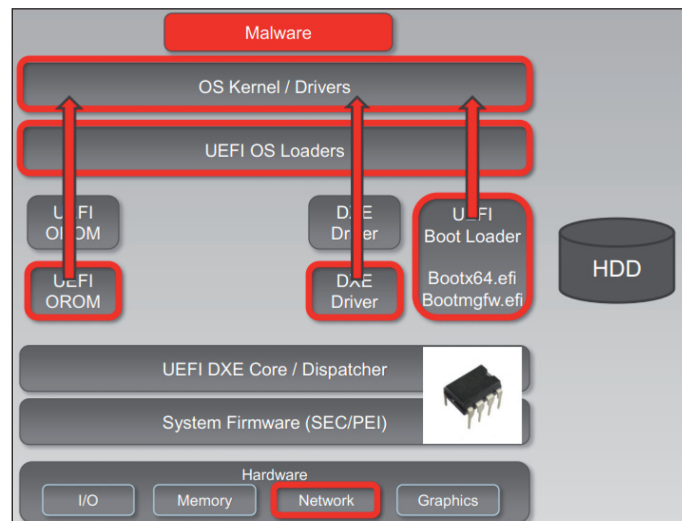

Figure 12: Types of UEFI bootkit infection.

modifications. However, researchers are trying to understand the methods attackers could use to bypass Secure Boot exploiting vulnerabilities in BIOS/UEFI implementations in order to infect the machine, and to mitigate against them [20, 21].

CHIPSEC

Intel has developed a CHIPSEC framework especially for BIOS/UEFI security assessment. This is an open-source framework for analysing the security of PC platforms covering hardware, system firmware including BIOS/UEFI, and the configuration of platform components. It allows the creation of a security test suite, security assessment tools for various low-level components and interfaces, as well as forensic capabilities for firmware. CHIPSEC is a framework developed in Python but with some parts coded in C++ for deeper integration with the hardware at operating system level. Besides *Microsoft Windows* and *Linux* operating systems, CHIPSEC can also run from a UEFI shell. The framework architecture is presented in Figure 13.

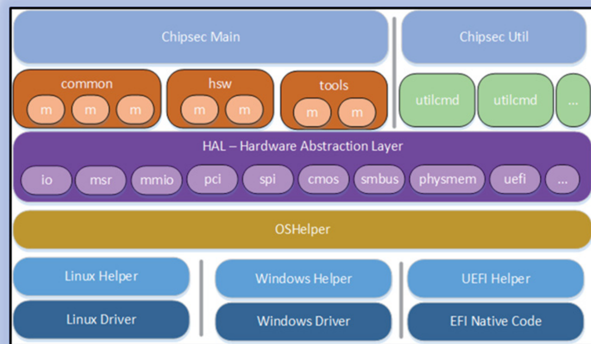

Figure 13: CHIPSEC framework architecture.

The CHIPSEC framework can be used as a security testing tool for searching for BIOS and UEFI firmware vulnerabilities. Also, the functionality of this tool covers forensic approaches for live/offline firmware analysis from CHIPSEC modules [22]. This tool includes modules for hidden file system forensics directly from the UEFI shell without the need to boot the operating system. In addition, CHIPSEC has basic heuristics for detecting BIOS/UEFI bootkit infection.

HIDDEN FILE SYSTEM READER TOOL

Implementing hidden storage makes forensic analysis more difficult because:

- Malicious files are not stored in the file system (difficult to extract)
- Hidden storage cannot be decrypted without malware analysis
- Typical forensic tools do not work out of the box.

To tackle the problem of retrieving the contents of the hidden storage areas, one needs to perform malware analysis and reconstruct the algorithms used to handle the stored data. In the course of our research into complex threats, we developed a tool some time ago [23] which is intended to recover the contents of hidden storage used by such complex threats as:

- TDL3 and its modifications
- TDL4 and its modifications
- Olmasco
- Rovnix.A
- Rovnix.B
- Sirefef (ZeroAccess)
- Goblin (XPAJ)
- Flame (dump decrypted resource section)

The tool is very useful in incident response, threat analysis and monitoring. It is able to dump the malware's hidden storage, as well as to dump any desired range of sectors of the hard drive. A screenshot of the tool's output is shown in Figure 14.

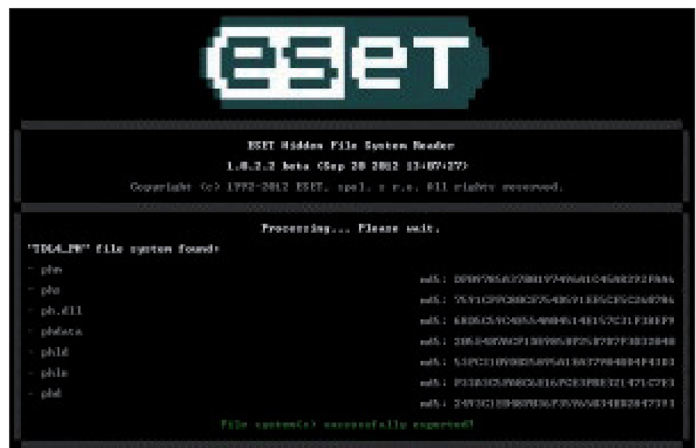

Figure 14: Hidden file system reader.

CONCLUSION

Microsoft claimed that the release of the Secure Boot technology heralded the end of the bootkit era. In practice, Secure Boot just switched the focus of the attackers towards a change in infection strategy. There are still many active machines in the world with old operating systems where Secure Boot is not supported. For non-targeted attacks, just intended to build botnets, cybercriminals will continue to use old bootkits and bootkit techniques for MBR/VBR infection until a critical mass of users have switched to modern hardware and operating systems.

In targeted attacks on *Microsoft Windows 8*, however, attackers will use vulnerabilities in the most common BIOS/UEFI firmware. The security life cycle for BIOS/UEFI is totally different from that in operating systems or popular software. This presents a problem because the BIOS/UEFI firmware on end-users' machines has sometimes never been updated since the first day they were used. We do not see a unified updating process embedded in the operating system because different firmware vendors use different schemes for the delivery of updates. Modern security software does not yet operate at the

level of BIOS/UEFI firmware protection. In the opinion of the authors of this paper [24], an interesting future lies ahead, possibly starting with targeted attacks: who's to say that they haven't already started?

REFERENCES

[1] Slade, R. Robert Slade's Guide to Computer Viruses. 2nd Edition, Springer, 1996. http://www.amazon.com/Robert-Slades-Guide-Computer-Viruses/dp/0387946632.

[2] Harley, D.; Slade, R.; Gattiker, U. Viruses Revealed. Osborne, 2007. http://www.amazon.com/Viruses-Revealed-David-Harley/dp/B007PMOWTQ.

[3] Ször, P. The Art of Computer Virus Research and Defense. Addison Wesley, 2005. http://books.google.co.uk/books/about/The_Art_of_Computer_Virus_Research_and_D.html?id=XE-ddYF6uhYC&redir_esc=y.

[4] Soeder, D.; Permeh, R. eEye BootRoot. BlackHat, 2005. http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-soeder.pdf.

[5] Kumar, N.; Kumar V. Vbootkit. BlackHat 2007. https://www.blackhat.com/presentations/bh-europe-07/Kumar/Whitepaper/bh-eu-07-Kumar-WP-apr19.pdf.

[6] Kleissner, P. Stoned Bootkit. BlackHat 2009. http://www.blackhat.com/presentations/bh-usa-09/KLEISSNER/BHUSA09-Kleissner-StonedBootkit-PAPER.pdf.

[7] Florio, E.; Kasslin, K. Your Computer is Now Stoned (...Again!): The Rise of MBR Rootkits. Symantec, 2013. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/your_computer_is_now_stoned.pdf.

[8] Kumar, N.; Kumar V. 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.

[9] Economou, N.; Luksenberg A. Deep Boot. Ekoparty 2011. http://www.ekoparty.org//archive/2011/ekoparty2011_Economou-Luksenberg_Deep_Boot.pdf.

[10] Ettlinger, W.; Vieböck, S. Evil Core Bootkit: Pwning Multiprocessor Systems. NinjaCon, 2011. http://downloads.ninjacon.net/downloads/proceedings/2011/Ettlinger_Viehboeck-Evil_Core_Bootkit.pdf.

[11] Diego, J.; Economou, N.A. VGA Persistent Rootkit. Ekoparty 2012. http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=publication&name=vga_persistent_rootkit.

[12] Rodionov, E.; Matrosov, A. Mind the Gapz: The most complex bootkit ever analyzed? ESET, 2013. http://www.welivesecurity.com/wp-content/uploads/2013/05/gapz-bootkit-whitepaper.pdf.

[13] Kaczmarek, S. UEFI and Dreamboot. HiTB 2013. http://www.quarkslab.com/dl/13-04-hitb-uefi-dreamboot.pdf.

[14] Zihang, X. Oldboot: the first bootkit on Android. 360, 2014. http://blogs.360.cn/360mobile/2014/01/17/oldboot-the-first-bootkit-on-android/.

[15] Rodionov, E.; Matrosov, A. The Evolution of TDL: Conquering x64. ESET, 2011. http://www.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf.

[16] Matrosov, A. Olmasco bootkit: next circle of TDL4 evolution (or not?). ESET, 2012. http://www.welivesecurity.com/2012/10/18/olmasco-bootkit-next-circle-of-tdl4-evolution-or-not-2/.

[17] Intel® Platform Innovation Framework for UEFI Specification. http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-specifications-general-technology.html.

[18] UEFI Validation Option ROM Validation Guidance. Microsoft, 2014. http://technet.microsoft.com/en-us/library/dn747882.aspx.

[19] Loukas, K. Mac EFI Rootkits. Black Hat 2012. http://ho.ax/De_Mysteriis_Dom_Jobsivs_Black_Hat_Paper.pdf.

[20] Bulygin, Y.; Furtak, A.; Bazhaniuk, O. A tale of one software bypass of Windows 8 Secure Boot. Black Hat 2013. https://media.blackhat.com/us-13/us-13-Bulygin-A-Tale-of-One-Software-Bypass-of-Windows-8-Secure-Boot-Slides.pdf.

[21] Kallenberg, C.; Bulygin, Y. All Your Boot Are Belong To Us Intel, MITRE. CanSecWest 2014. https://cansecwest.com/slides/2014/AllYourBoot_csw14-intel-final.pdf.

[22] Intel CHIPSEC. https://github.com/chipsec/chipsec.

[23] ESET Hidden File System Reader. http://www.eset.com/int/download//utilities/detail/family/173/.

[24] Matrosov, A.; Rodionov, E.; Harley, D. Rootkits and Bootkits: Advanced Malware Analysis. No Starch, 2015 (in preparation).