Thriving Beyond The Operating System: Financial Threat Group Targets Volume Boot Record

December 07, 2015 | By Dimiter Andonov, Willi Ballenthin, Nalani Fraser, Will Matson, Jay Taylor | Threat Research, Advanced Malware

In September, Mandiant Consulting identified a financially motivated threat group targeting payment card data using sophisticated malware that executes before the operating system boots. This rarely seen technique, referred to as a 'bootkit', infects lower-level system components making it very difficult to identify and detect. The malware's installation location also means it will persist even after re-installing the operating system, widely considered the most effective way to eradicate malware.

We first identified this activity during a recent investigation at an organization in the financial industry. We identified the presence of a financially motivated threat group that we track as FIN1, whose activity at the organization dated back several years. The threat group deployed numerous malicious files and utilities, all of which were part of a malware ecosystem referred to as 'Nemesis' by the malware developer(s),[1] and used this malware to access the victim environment and steal cardholder data. FIN1, which may be located in Russia or a Russian-speaking country based on language settings in many of their custom tools, is known for stealing data that is easily monetized from financial services organizations such as banks, credit unions, ATM operations, and financial transaction processing and financial business services companies.

Nemesis, the malware ecosystem used by FIN1, includes comprehensive backdoors that support a variety of network protocols and communication channels for command and control (CnC). It provides a robust set of capabilities, including: file transfer, screen capture, keystroke logging, process injection, process manipulation, and task scheduling. The threat group continually updated the Nemesis malware during their ongoing access to the victim environment, deploying several different variants of the same tools and adding functionality between iterations. In early 2015, FIN1 updated their toolset to include a utility that modifies the legitimate system Volume Boot Record (VBR) and hijacks the system boot process to begin loading Nemesis components before the Windows operating system code. We refer to this utility as BOOTRASH.

A Brief Refresh

On a Windows system, the Master Boot Record (MBR) is critical to the boot process. The MBR stores information about the disk, including the number and layout of any partitions, and a small amount of code used during the boot process.[2] This code searches for the primary active partition, and passes control over to that partition's VBR.[3]

A VBR, on partitioned devices, is located in the first sector of an individual partition. The VBR contains machine code specific to the operating system or program on that partition. For example, in situations where more than one operating system is installed on a computer, each operating system is installed on a separate partition and each partition contains a VBR with instructions on how to start the respective operating system.[4] The VBR instructs the operating system code to begin the boot process, which involves loading the necessary software into memory.

Figure 1 depicts a simplified boot process. The MBR loads the VBR, which loads the operating system code. BOOTRASH hijacks this boot process in order to load the Nemesis payload before the operating system boots.

Figure 1. Simplified normal boot process

BOOTRASH Step by Step

To successfully hijack the boot process, the malware first uses a complex multi-step process to create a custom virtual file system[5] to store the Nemesis components in the unallocated space between partitions. It then hijacks the original VBR by overwriting the bootstrap code with its own malicious code. The bootstrap code calls the Nemesis bootkit, which intercepts certain boot process functions and injects the Nemesis components into the Windows kernel. Following is a description of the installation and hijacking process:

Step 1: System Checks

Prior to installation, the BOOTRASH installer gathers statistics about the system, including the operating system version and architecture. The installer is capable of deploying 32-bit or 64-bit versions of the Nemesis components depending on the system's processor architecture. The installer will install the bootkit on any hard disk that has a MBR boot partition, regardless of the specific type of hard drive. However, if the partition uses the GUID Partition Table disk architecture, as opposed to the MBR partitioning scheme, the malware will not continue with the installation process.

The malware checks to make sure a copy of the BOOTRASH installer is not already running on the system. It also checks to see if the Microsoft .NET 3.5 framework is installed on the system - a prerequisite for the malware. If the installer is already running or the .NET framework is not installed, the malware will quit.

Step 2: Available Space Calculations & Virtual File System Creation

BOOTRASH creates its own custom virtual file system (VFS) to store the components of the Nemesis ecosystem. The malware performs several calculations to determine the positioning of the file system and whether the system has enough space to fit the file structure.

To determine how much space is needed for the installation, the malware uses Windows Management Instrumentation (WMI) to query the system's boot disk and partition. The malware then calculates the total size of the 32-bit or 64-bit components to ensure there is enough space for the custom file system in the free space between the system's partitions.

Step 3: Boot Sector Hijacking

The installer reads the original boot sector into memory and saves an encoded backup copy of the VBR code at 0xE sectors from the start of the partition. Next, the malware applies two algorithms (CRC16-CCITT and MD5) to the original boot sector in order to check its integrity at a later time. With the original boot sector saved, the malware decodes the new bootstrap code from one of its embedded resources and overwrites the existing bootstrap code, effectively hijacking the boot process of the compromised system.

Step 4: Nemesis Component Installation

The installer always uses the virtual file system for saving the three components responsible for creating and installing the bootkit: vbr.bin, vbs.bin, and bootldr.sys. The rest of the components can be either saved in the virtual file system or as binary data within the HKCU\.Default\Identities registry keys. These components are responsible for the primary C2 functionality of the Nemesis ecosystem that includes: file transfer, screen capture, keystroke logging, process injection, process manipulation, and task scheduling.

Table 1 details the associated registry keys.

Registry Value Name	File Content
HKCU\.Default\Identities\{3D04DDFA-AE6F-4BC2-9AE6-4A76A471147A}	core.sys
HKCU\.Default\Identities\{424D9649-5B57-4C9B-A55A-00D6CD382092}	vfs.sys
HKCU\.Default\Identities\{AA1F2588-670C-450E-833B-C8CAAF26DA5E}	nemesis.sys
HKCU\.Default\Identities\{95E4F335-E152-4778-B3B0-3422B37B3A3D}	injproxy.dll
HKCU\.Default\Identities\{C34C01DC-8E9D-40AF-82FF-79DB2735C333}	loader.dll
HKCU\.Default\Identities\{4EED1600-54C9-471E-B74F-2A88BAC188B4}	nemesis.dll
HKCU\.Default\Identities\{36203264-DFF1-43AC-94AD-096592297776}	nmscfg.dat

Table 1: Nemesis components optionally saved as registry values

Step 5: Hijacked Boot Process

As previously discussed, during a normal boot process the MBR loads the VBR, which loads the operating system code. However, during the hijacked boot process, the compromised system's MBR will attempt to load the boot partition's VBR, which has been overwritten with the malicious BOOTRASH bootstrap code. This code loads the Nemesis bootkit components from the custom virtual file system. The bootkit then passes control to the original boot sector, which was saved to a different location on disk during the installation process. From this point the boot process continues with the loading and executing of the operating system software. Figure 2 illustrates the hijacked boot process.

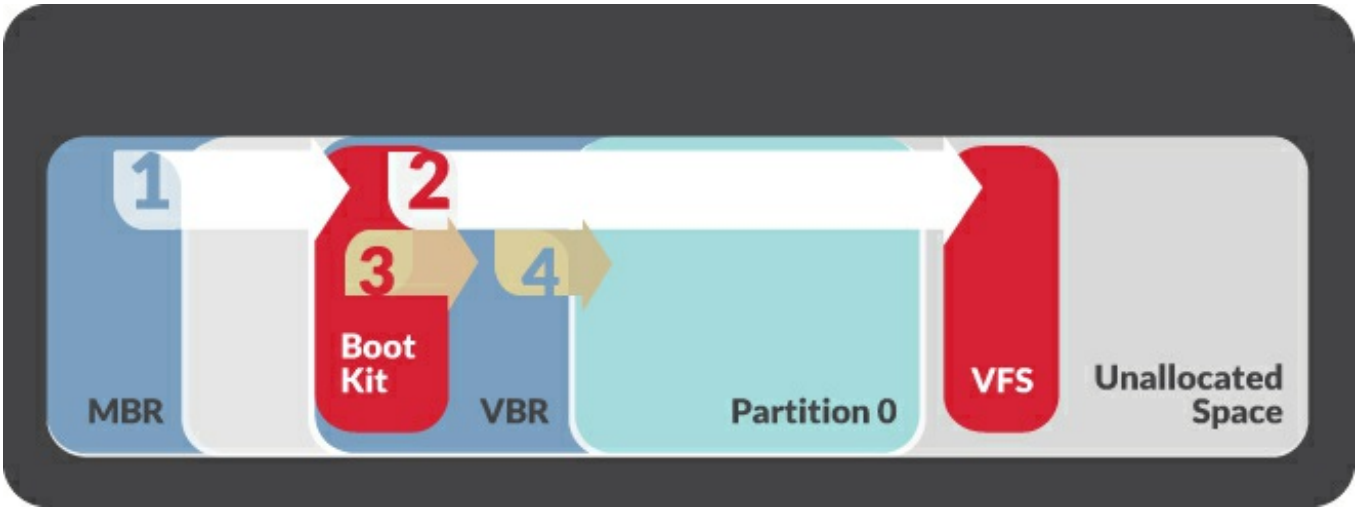

Figure 2. Simplified hijacked boot process

The bootkit intercepts several system interrupts to assist with the injection of the primary Nemesis components during the boot process. The bootkit hijacks the BIOS interrupt[6] responsible for miscellaneous system services and patches the associated Interrupt Vector Table entry so it can intercept memory queries once the operating system loader gains control. The bootkit then passes control to the original VBR to allow the boot process to continue. While the operating system is being loaded, the bootkit also intercepts the interrupt and scans the operating system loader memory for a specific instruction that transfers the CPU from real mode to protected mode.[7] This allows the bootkit to patch the Interrupt Descriptor Table each time the CPU changes from real mode to protected mode. This patch involves a modified interrupt handler that redirects control to the bootkit every time a specific address is executed. This is what allows the bootkit to detect and intercept specific points of the operating system loader execution and inject Nemesis components as part of the normal kernel loading.

Uninstall Option

BOOTRASH has a built in option for restoring the original boot sector, in the event that the threat actors wish to remove the hijacking process. This option only restores the original boot sector – it does not remove the custom virtual file system or the backup VBR that was created by BOOTRASH.

Other Bootkits

Malware that persists by compromising the MBR or VBR is unusual, but not unknown. ESET documented their technical research into bootkit malware families such as ‘TDL4’ (also known as ‘Olmarik’), ‘Necurs’, and ‘Rovnix’ in 2011,[8] and cataloged various MBR and VBR infection vectors in 2012.[9]

In addition, there have been reports of financially motivated malware utilizing bootkits. In 2013, RSA warned that a banking Trojan named ‘KINS’ had VBR bootkit functionality and was being advertised on a Russian-speaking online forum[10]. There was also reporting in 2013 that the source code for another banking Trojan, known as ‘Carberp,’ was publicly leaked. Reports stated the Trojan had been sold for $40,000 due to the addition of bootkit functionality[11]. Further reporting indicated the bootkit component might not have been completely operational[12]. Regardless, the high price commanded for the malware is indicative of the bootkit code complexity, as well as the demand for banking malware with this evasive capability.

Not Just a Financial Threat

In 2012, Mandiant observed a suspected China-based, Advanced Persistent Threat (APT) group utilizing a MBR (as opposed to VBR) bootkit that we call ROCKBOOT to establish persistence for backdoors at victim organizations within an industry unrelated to financial services. This group, like many of the threat groups we track, primarily uses more traditional techniques for ensuring their malware remains persistent, such as modifying Windows registry keys or using techniques like DLL search order hijacking. This threat group deployed the bootkit as part of the toolset used to steal intellectual property from the victim organization.

The selective use of bootkits for persistence suggests some threat actors may have access to more sophisticated toolsets. The threat actors may selectively deploy these advanced toolsets when the victim organization is difficult to penetrate or if the targeted data is of high value and the threat actors want to ensure continued access to the compromised environment.

Elevating Detection Capabilities

Bootkits, such as BOOTRASH, are very difficult to detect because they have the potential to be installed and executed almost completely outside of the Windows operating system.[13] Because the malicious boot loader executes before Windows itself is fully loaded, it is not subject to typical operating system integrity checks. The components used to load the malware payload are not scanned by anti-virus software, because they are stored in a VFS outside the Windows file system. In addition, the malware components themselves are stored either in the VFS or the Windows registry – another location not typically scanned by anti-virus. This leaves live memory as the only location where the malware is likely to be detected; and unless the bootkit and VFS components are removed, the malware will execute and load every time the system starts. Wiping the operating system partition and re-installing will not remove the bootkit or VFS components written to unallocated space.

During the investigation with BOOTRASH, we used Mandiant Intelligent Response (MIR), a proprietary host-based technology that provides raw disk access, to look for malware persistence outside of the operating system. This tool allowed us to identify systems that had a modified VBR.

Conclusion

The use of malware that persists outside of the operating system requires a different approach to detection and eradication. Malware with bootkit functionality can be installed and executed almost completely independent of the Windows operating system. As a result, incident responders will need tools that can access and search raw disks at scale for evidence of bootkits. Similarly, re-installing the operating system after a compromise is no longer sufficient. System administrators should perform a complete physical wipe of any systems compromised with a bootkit and then reload the operating system.

Appendix: Associated MD5 Hashes

MD5 Hash	Function
372f1e4d2d5108bbffc750bb0909fc49	BOOTRASH dropper
ac64ef80f8209ae7b67be0be9ea6400e	Windows 7 and later 32-bit modified VBR
073a2998a6f1ccf0ea89fe60ce4bdeaf	Windows 7 and later 64-bit modified VBR
c145624f148980ad026ea7b79e61212d	Windows XP 32-bit modified VBR
472926fe51fc6a4fdf687e8a4de64d78	Windows XP 64-bit modified VBR
1c17c92519523a129e9abd298bb78521	Bootstrap code for systems with NTFS/MBR

15de35de527ebe2115746b4fd4f1ba1d	32-bit Boot loader driver
012e6f3ee70d6558f8002d0efce5c9e0	64-bit Boot loader driver
dd366fcb810594e0620fdf672b03f4d5	32-bit Core services driver
fed12e07499e8cd3a5a47f1f7a8db0be	64-bit Core services driver
21cd4a30ac322bfc9bd2401ea17acfc0	32-bit Nemesis driver
76b6dc622264e3ad822a691a7ec68865	64-bit Nemesis driver
d0b9f9bccbc3725bfcc9546986982ff3	32-bit VFS driver
efbff3b08b5d368976eb4675bb4c000f	64-bit VFS driver

[1] The name ‘Nemesis’ is referenced in several build paths for the malware.

[2] https://technet.microsoft.com/en-us/library/cc976797.aspx

[3] https://en.wikipedia.org/wiki/Volume_boot_record

[4] http://windows.microsoft.com/en-us/windows/install-multiple-operating-system-multiboot#1TC=windows-7

[5] http://www.malwaretech.com/2014/11/virtual-file-systems-for-beginners.html

[6] A BIOS interrupt is a hardware or software driven condition requiring the interruption of the current code the processor is executing.

[7] Protected mode is an operating state that provides hardware-level protections for a system’s memory. Real mode does not provide this protection support.

[8] http://www.welivesecurity.com/2011/08/23/hasta-la-vista-bootkit-exploiting-the-vbr/

[9] http://www.welivesecurity.com/2012/12/27/win32gapz-new-bootkit-technique/

[10] https://blogs.rsa.com/is-cybercrime-ready-to-crown-a-new-kins-inth3wild/

[11] http://threatpost.com/carberp-source-code-leaked/101070/

[12] http://krebsonsecurity.com/tag/carberp-bootkit/

[13] Malware with bootkit functionality still relies on the operating system to initiate installation. However, the actual payload (the bootkit itself) can be allocated to disk outside of the operating system.

This entry was posted on Mon Dec 07 08:00:00 EST 2015 and filed under Advanced Malware, Blog, Dimiter Andonov, Jay Taylor , Latest Blog Posts, Malware, Mandiant, Nalani Fraser, Threat Research, Will Matson, Willi Ballenthin and bootkits.

Newsroom
Blog
Investor Relations

Incident?
Contact Us
Communication Preferences
Report Security Issue
Supplier Documents

Connect

Facebook
LinkedIn
Twitter
Google+
YouTube
Glassdoor

Privacy & Cookies Policy | Safe Harbor