Detecting UEFI Bootkits in the Wild (Part 1)

By Takahiro Haruyama

Published: 2021-06-15 · Archived: 2026-04-05 14:05:42 UTC

Threat actors are continually looking for ways to improve the persistence of their malware and implants. Bootkits, meaning rootkits running at the firmware level, have been utilized for this purpose. Once bootkits are installed, it can be extremely difficult to detect or remove versus OS-level rootkits as they are executed prior to the actual OS boot process.

The approach of using bootkits is not a new concept. According to Rootkits and Bootkits, there have been several PoCs and threats targeting legacy BIOS boot systems since 2005. As UEFI boot systems are going mainstream, the bootkits are also shifting to an implementation of infecting firmware in a flash chip on the motherboard instead of the MBR/VBR on the hard drive. The first PoC of UEFI bootkits was presented in 2013 and the threats have been observed in the wild since 2018.

In this article, the VMware Carbon Black Threat Analysis Unit (TAU) will describe the current threat landscape of UEFI bootkits then discusses how to detect these threats. OS-level security products are not effective to examine firmware as bootkits can take over the interfaces utilized by the detections (e.g., FSMIE bit in HSFC SPI register). Therefore, TAU focuses on the detections in the installation phase.

UEFI Bootkit Comparison

TAU analyzed the known UEFI bootkit samples in the wild (LoJax, MosaicRegressor, and TrickBoot) and summarized the characteristics, as displayed in Table 1. We also included the Hacking Team's Vector-EDK bootkit in the table as the leaked source code has been heavily reused by the threat actors.

	Hacking Team's Vector-EDK (source code leakage)	**Sednit's LoJax**
UEFI bootkit type	DXE driver (unsigned)	DXE driver (unsigned)
callback event	EFI_EVENT_GROUP_READY_TO_BOOT	EFI_EVENT_GROUP_READY_TO_BOOT
OS-level executables for installation	N/A	**info_efi.exe**: application for UEFI firmware reconnaissance, **ReWriter_read.exe**: application dumping SPI flash memory, **ReWriter_binary.exe**: application adding SecDxe.efi to UEFI firmware
UEFI bootkit modules	**Ntfs.efi**: DXE driver for NTFS filesystem read/write, **rkloader.efi**: DXE driver setting a callback for fsbg.efi, **fsbg.efi**: UEFI application running the main bootkit code, **ReSetfTA.efi**: UEFI application resetting the infection marker for debug	**SecDxe.efi**: bootkit DXE driver with Ntfs.efi

target hardware platform	N/A	misconfigured or fairly old systems (on motherboards older than Platform Controller Hub chipsets introduced around 2008) without the SMM_BW configuration bit
reconnaissance	N/A	BIOS control register LE/WPD(WE)/EISS(SMM_BWP) bits, SPI Protected Ranges (PR registers PR0–PR4), NVRAM UEFI variables "*SecureBoot/SetupMode/AcpiGlobalVariable*" (SeSystemEnvironmentPrivilege required), etc.
installation technique	N/A	changing BIOS control register, exploiting vulnerabilities: VU#766164, CVE-2017-3197 (potentically)
OS infection marker	NVRAM UEFI variable "fTA" = 1	N/A
OS persistence technique	Startup folder	registry value "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Ses Manager\BootExecute" = '*autocheck autoche* *' (offline detection only)
code reuse	N/A	RWEverything kernel driver (**RwDrv.sys**), Hacking Team's **Ntfs.efi**: the inline DXE driver is 93% matched
obfuscation/compression	N/A	Tiano compression (**SecDxe.efi**)

Table 1: UEFI Bootkit Comparison

Infection Process

The components, used by the UEFI bootkits, are classified into two categories: OS-level executables for a bootkit installation (kernel driver and user-mode application) and UEFI modules (DXE driver and UEFI application if any). The infection vector of MosaicRegressor is unknown and TrickBoot contains no UEFI module. Only LoJax has both. The rough LoJax bootkit installation flow and the relationship between the components are as follows.

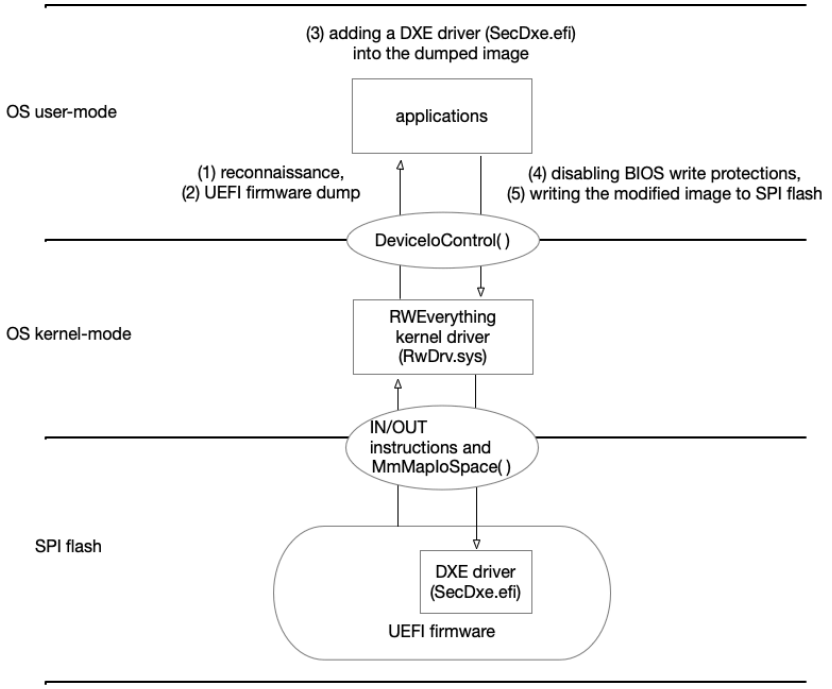


Figure 1: LoJax UEFI bootkit installation process

1. One of the user-mode applications (exe) collects UEFI firmware information such as BIOS write protection settings and SPI registers.
2. exe saves the UEFI firmware image extracted from a SPI flash memory.
3. exe not only dumps the firmware image but also adds a DXE driver (SecDxe) into the image.
4. The same executable disables the BIOS write protections before the firmware modification by just changing the BIOS control register or exploiting known vulnerabilities like VU#766164 if needed.
5. The same executable writes the modified firmware image to the SPI flash.

All of these applications abuse a kernel driver (RwDrv.sys) from the third-party tool RWEverything.

After a bootkit installation, the bootkit execution starts with DXE drivers in all cases. The drivers register a callback to intercept the *EFI_EVENT_GROUP_READY_TO_BOOT* event. The interception enables attackers to take control before the OS bootloader runs. The drivers are unsigned. Thus, users can prevent the execution by just enabling UEFI Secure Boot. The TrickBoot sample only implements the platform identification and BIOS write protection status check functionalities for the reconnaissance.

Code Reuse

Threat actors generally take the easiest approach to complete their intended objective. The UEFI bootkit implementation is also no exception. As described in the previous section, a kernel driver is required to access the SPI flash memory during the bootkit installation. LoJax and TrickBoot reuse RWEverything's RwDrv.sys as there are some open source implementations communicating with the driver already (e.g., CHIPSEC and fwexpl). TrickBoot additionally reutilizes the fwexpl code in the user-mode application.

Another reused code is Hacking Team's Vector-EDK. MosaicRegressor is almost the same except for minor changes. LoJax also reutilizes the DXE driver (Ntfs.efi) for NTFS filesystem read/write. The bootkit drops the agent program onto the disk where the driver will be used.

Obfuscation and Compression

Except the TrickBoot's DLL module, no obfuscation technique is generally used by droppers. The module utilizes the open source obfuscation library ADVobfuscator. Most strings are obfuscated by the library. The obfuscation technique is specifically a combination of dynamic string construction in a stack area and a decode routine dedicated for each string. As it's time-consuming to defeat the compiler-level obfuscation by static-analysis only, TAU has released the IDAPython script de-obfuscating the strings based on the code emulator (FireEye's flare-emu).

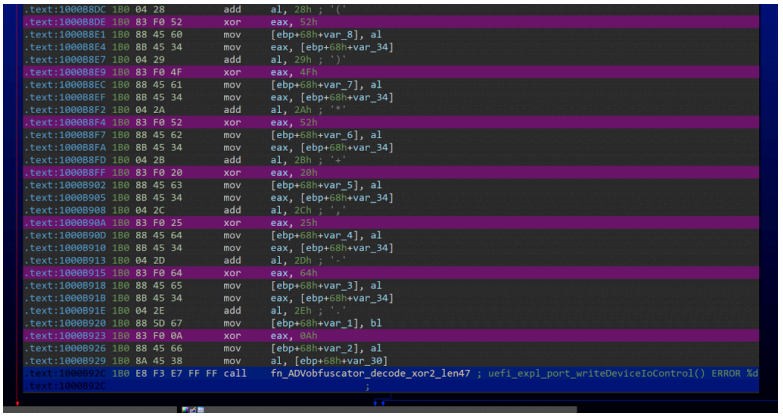
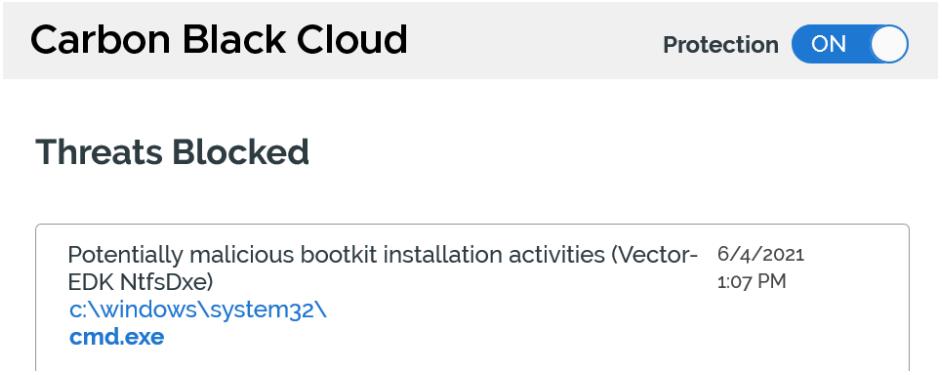


Figure 2: One example of strings obfuscated by ADVobfuscator

It should be noted that the LoJax samples utilize the Tiano compression algorithm on the DXE driver (SecDxe.efi). The compressed driver is written to the SPI flash without the decompression as the algorithm is natively-supported by the EDK II (UEFI build system).

Our Detection Approach

TAU has developed approaches to detecting and blocking common technicques used to install UEFI bootkits. Both approaches focus on the behaviors outlined above. The screenshot below shows how Zero Touch Prevention can detect and stop activities related to the installation of bootkits.



Wrap-up

TAU reviewed the UEFI bootkits in the wild then discussed the detection rules focusing on the common characteristics. The techniques used by the bootkits are nothing new, but we should pay more attention to the threat as they are starting to be observed in commodity cybercrime malware like TrickBot. TAU will dig into the bootkit's low-level behavior in the installation for a generic rule creation next time.

Last but not least, please note that the fundamental solution against UEFI bootkits is to enable UEFI Secure Boot or Platform Secure Boot (Verified and Measured Boot). The UEFI Secure Boot function authenticates UEFI modules with digital signatures then takes actions according to the policy if the authentication fails. Verified and Measured Boot like Intel BootGuard does the same thing, however the root of trust for the authentication depends on an immutable hardware logic. TAU recommends that customers utilize either of them (the latter one if possible). If customers can't enable it for some reason, our detection rules will be beneficial.
