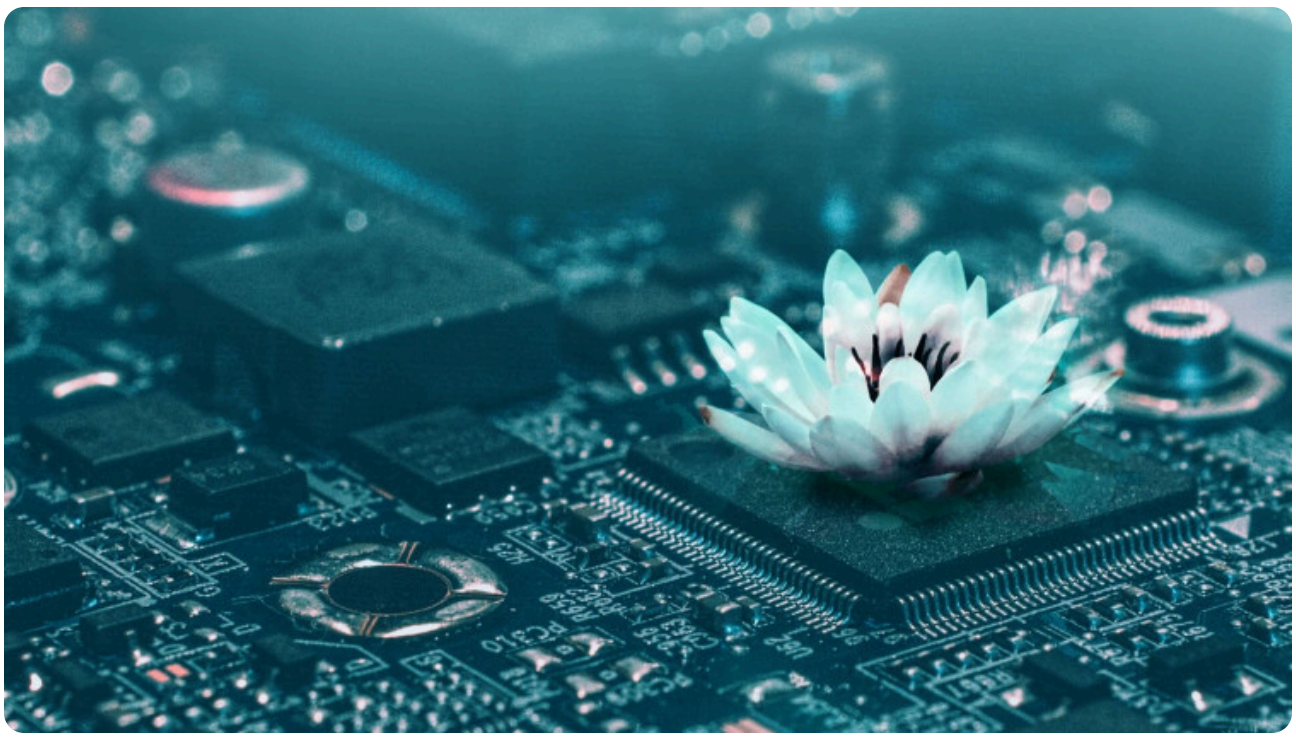
ESET RESEARCH

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

Martin Smolár

01 Mar 2023 • 40 min. read



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 for $5,000 since at least October 2022.

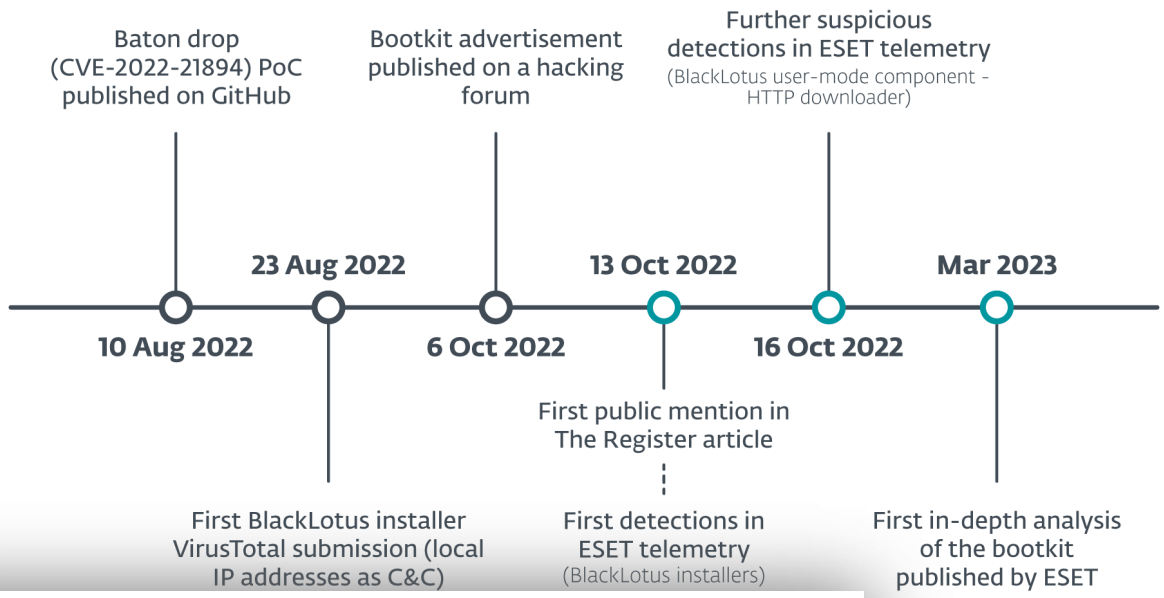
UEFI bootkits are very powerful threats, having full control over the OS boot process and thus capable of disabling various OS security mechanisms and deploying their own kernel-mode or user-mode payloads in early OS startup stages. This allows them to operate very stealthily and with high privileges. So far, only a few have been discovered in the wild and publicly described (e.g., multiple malicious EFI samples we discovered in 2020, or fully featured UEFI bootkits such as our discovery last year – the ESPecter bootkit – or the FinSpy bootkit discovered by researchers from Kaspersky).

UEFI bootkits may lose on stealthiness when compared to firmware implants – such as LoJax; 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.

- 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.



UEFI bootkit

ound forums since at
to identify, from our
bootkit to victims. The
tain, both from public
y threat actors have
le bootloaders that
ngs will change rapidly
rimeware groups,
ups' capabilities for
spreading malware using their botnets.

Is this really BlackLotus?

There are several articles or posts summarizing information about BlackLotus (here, here and here 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 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, 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*

t the HTTP downloader
ss. ✅

process context.

s a tiny bootkit with an

kB.

t it can disable built-in
ndows Defender, and

- 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 (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).

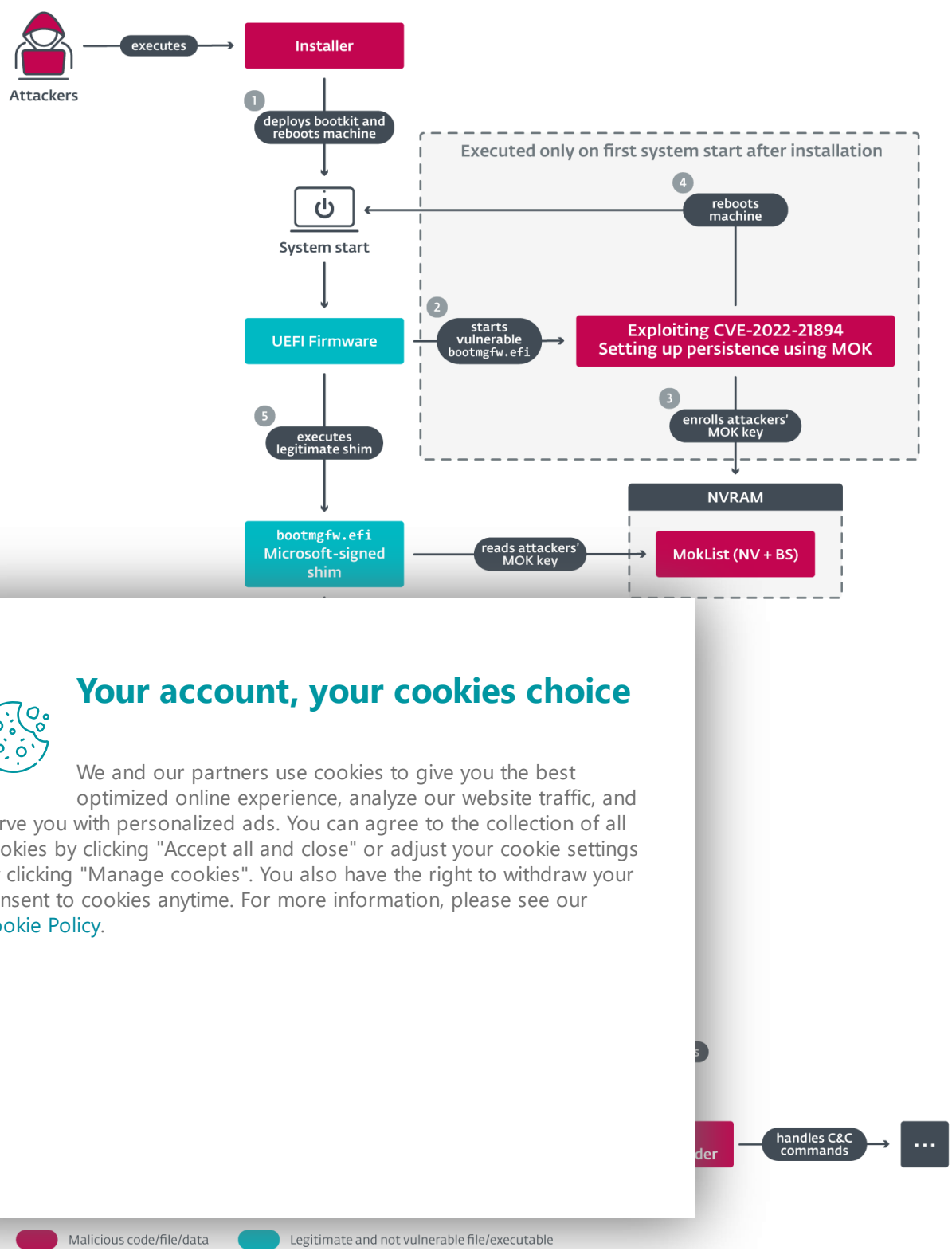


Figure 2. BlackLotus simplified execution overview

Interesting artifacts

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 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
```

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 is a well-known researcher and author of various malware-analysis tools), or just some random quotes from various songs, games, or series.

```
lea     r13d, [rdi+1]
mov     r8d, r13d
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
```

kLotus author

kit seems to be
– offline and online.
legitimate (but
re Boot.

aller

- 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/7144BCD31C0000/bootmgfw.efi`
- `https://msdl.microsoft.com/download/symbols/bootmgr.efi/98B063A61BC000/bootmgr.efi`
- `https://msdl.microsoft.com/download/symbols/hvloader.efi/559F396411D000/hvloader.efi`

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.

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 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"

…C communication

…ropping multiple files

`…2\` directories. While

…ter is a custom folder

…n of the role of each

…ain in detail how the

…timate Microsoft-

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

Folder	Filename	Description
ESP:\EFI\Microsoft\Boot	`grubx64.efi`	BlackLotus bootkit, malicious self-signed UEFI application.
	`bootload.efi`	Legitimate Microsoft-signed `shim` binary (temporary name, later replaces `bootmgfw.efi` after CVE-2022-21894 exploitation).
	`bootmgfw.efi`	Legitimate, but vulnerable (CVE-2022-21894) Windows Boot Manager binary, embedded in the installer or downloaded directly from the Microsoft Symbol Store.
	`BCD`	Attackers' custom Boot Configuration Data (BCD) store used in CVE-2022-21894 exploitation chain.
	`BCDR`	Backup of victim's original BCD store.
		Legitimate, but vulnerable (CVE-2022-21894) Windows Hypervisor Loader binary, embedded inside an installer or downloaded directly from the Microsoft Symbol Store.
		Legitimate, but vulnerable (CVE-2022-21894) Windows Boot Manager binary, embedded inside an installer or downloaded directly from the Microsoft Symbol Store.

`ESP:\system32`	`mcupdate_AuthenticAMD.dll`	Malicious self-signed native PE binary. This file is executed by the `hvloader.efi` after successful CVE-2022-21894 exploitation (on systems using an AMD CPU).
	`mcupdate_GenuineIntel.dll`	Malicious self-signed native PE binary. This file is executed by the `hvloader.efi` after successful CVE-2022-21894 exploitation (on systems using an Intel CPU).
	`BCD`	Attackers' custom BCD used in CVE-2022-21894 exploitation chain.

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
gues wrote a very
vers – Unguarded

features with the
rity (HVCI), which
grity in the kernel
vents vulnerable
load malicious
ems that malware
the main

ts the Enabled registry value under the `HypervisorEnforcedCodeIntegrity` registry key to zero.

```
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;
}
```

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. The reason for this is that BitLocker can be used in a combination with Trusted Platform Module (TPM) 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 and reboots the

g persistence

es persistence on
hain we are about to
es and then dig deeper

eature 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.

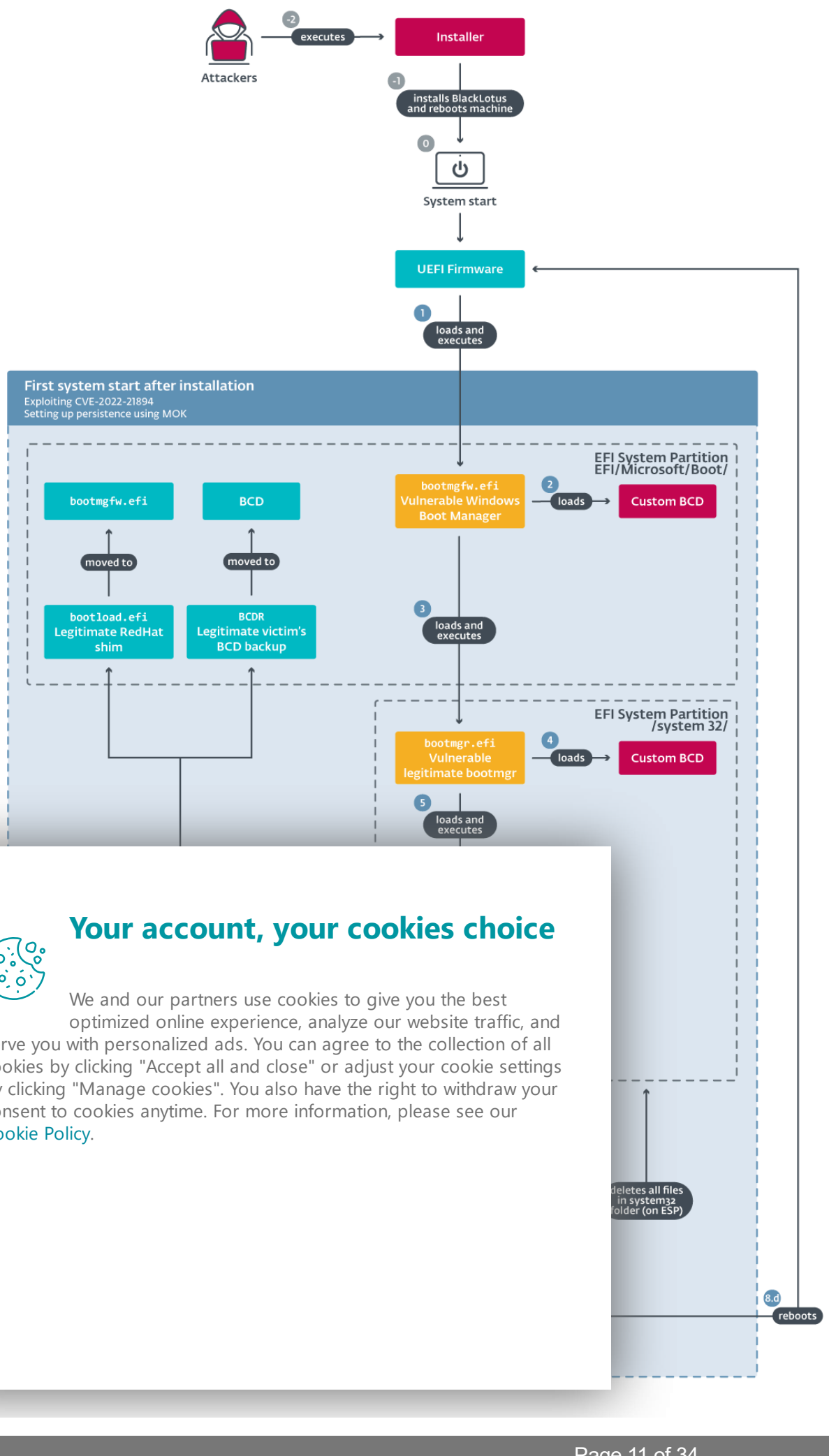


Figure 6. Bypassing Secure Boot and setting up persistence using MOK

Exploiting CVE-2022-21894

To bypass Secure Boot, BlackLotus uses the baton drop (CVE-2022-21894): Secure Boot Security Feature Bypass Vulnerability. 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. 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:

- 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.).

this vulnerability
Figure 6):

are proceeds with
rst boot option is by
soft/Boot` folder on
m's `bootmgfw.efi`
staller), the firmware

options, previously
f the legitimate BCD

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:

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

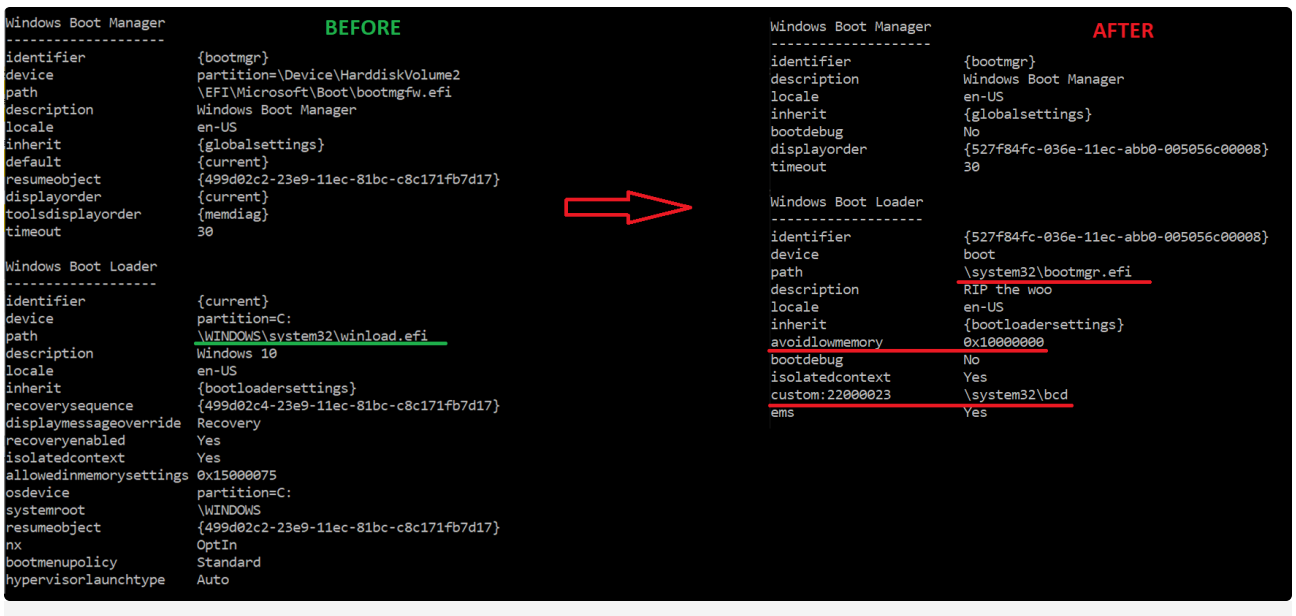
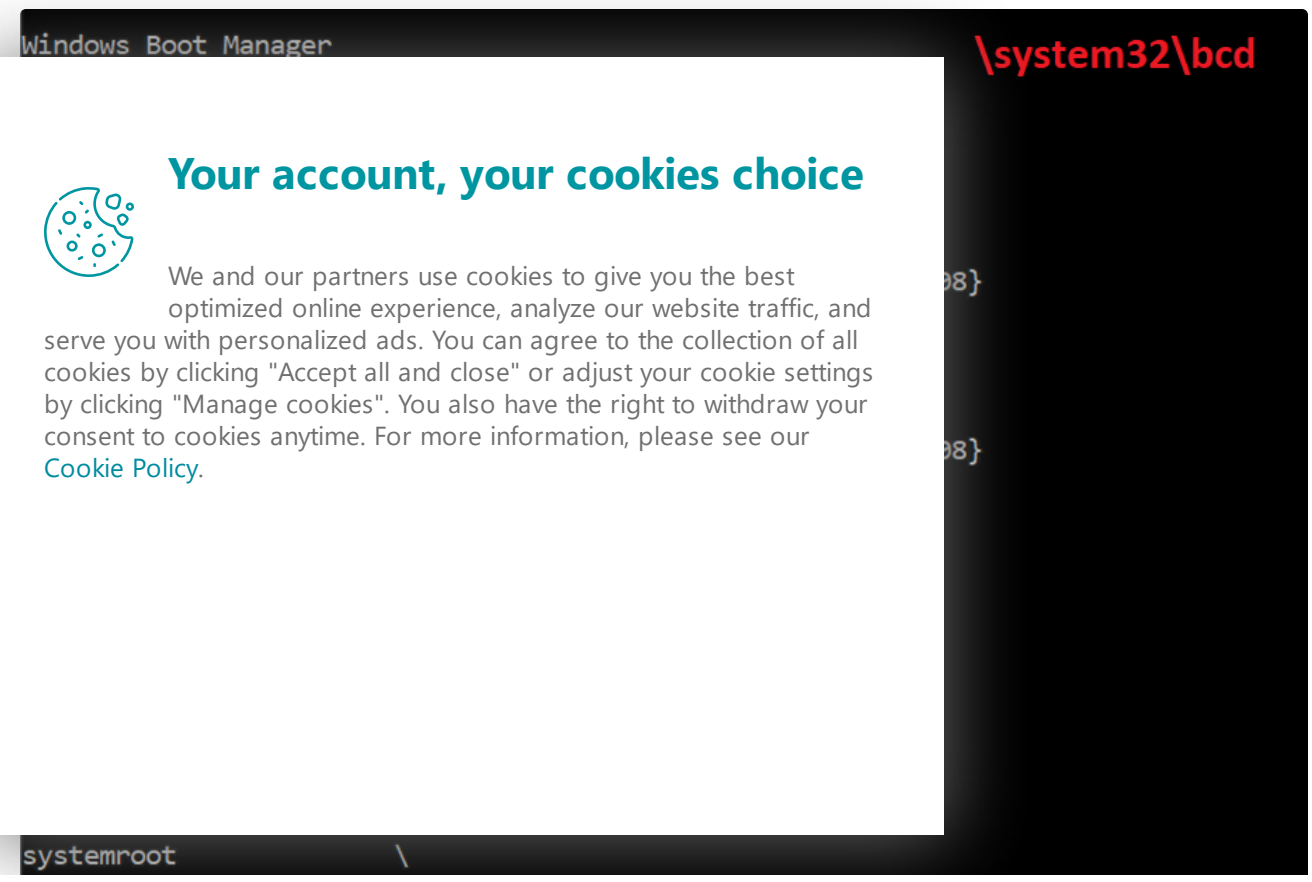


Figure 7. Legitimate BCD store (BEFORE) vs the one used by the BlackLotus installer (AFTER)

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.



```
ems                     Yes
```

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 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.

```
1  unsigned __int64 __fastcall BtLoadUpdateDll(__int64 a1, _QWORD *imageEP)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
                                                    date_<platform>.dll
                                                    9, &platform, a1, 1);
                                                    + 16));
                                                    ;
                                                    yPoint;
28 }
```

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.

```
// verify if HV loader imagebase found by checking if IMAGE_NT_HEADERS64.OptionalHeader.CheckSum  equals (0xEC35E)
if ( *(*(HvloaderImageBase + 0x3C) + HvloaderImageBase + 0x58) == 0xEC35E )
{
  EfiImageHandle = *(HvloaderImageBase + 0x113670);
  // 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);
      }
    }
  }
}
```

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 y that Microsoft signs ify the integrity of a le, and also embeds its ure that components cal, RedHat, etc.,) – are allows the use of an MOK list. Figure

e named `MokList`. ve, physical access is abled (it's available only es function lity, 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.

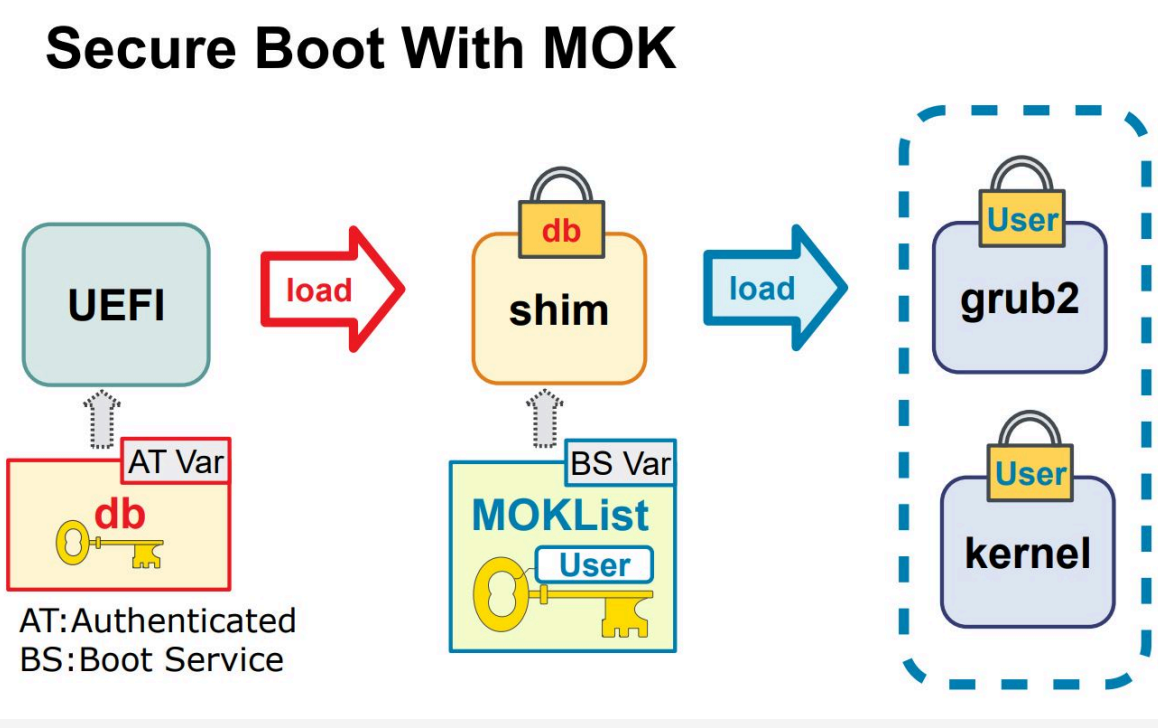


Figure 11. MOK boot process overview (image source)

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

s is shown in Figure 12.

```
"EFI\\Microsoft\\Boot\\BCD")

V_BS, 0x353i64, attackersCert);

.dll");
.dll");
```

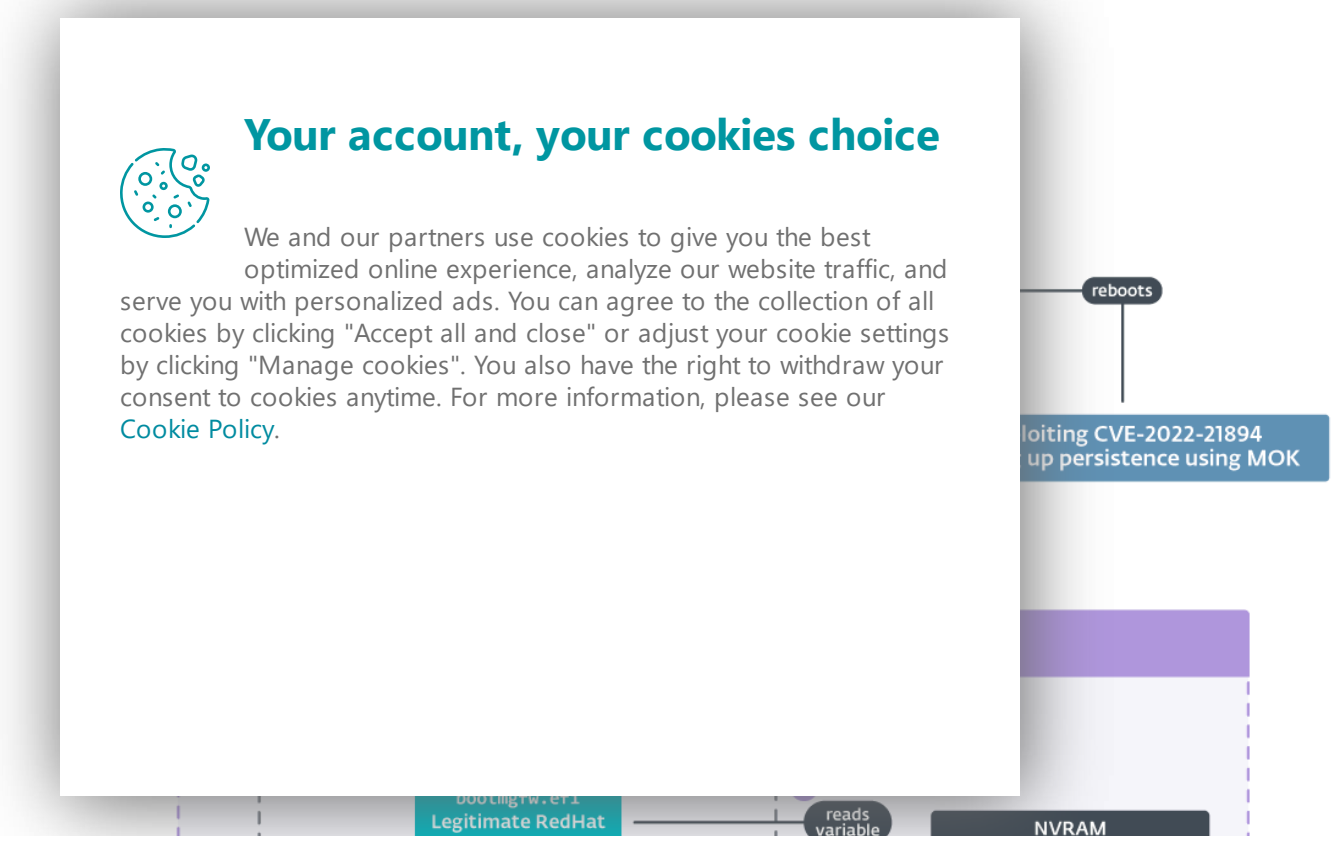
Figure 12. Hex-Rays decompiled code – MokInstaller UEFI app setting up persistence for the BlackLotus bootkit

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.



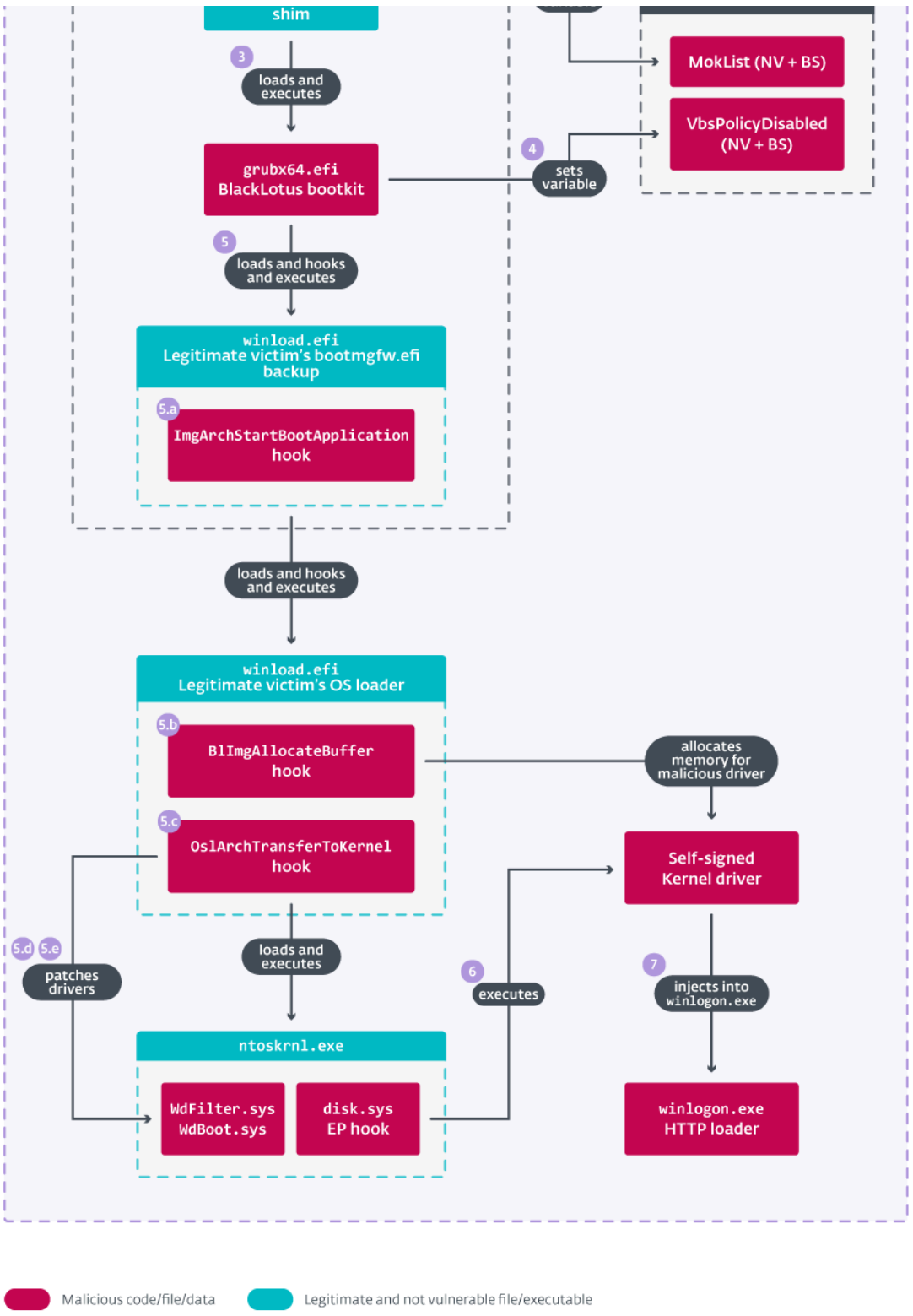


Figure 13. Diagram showing execution of the BlackLotus UEFI bootkit

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, `t\bootmgfw.efi`. As MokInstaller binary

M variable, and uses o verify the second- tkit located in

`cyDisable` NVRAM he Windows OS loader s HVCI and Credential

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 )
```

1);

rnel *hook – patching*

ys *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
es their handles, and
ng the

- thus protecting the
thwart any attempts
are protected:

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.
```

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

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:

- 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 system volume serial seen in Figure 16


```
"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 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.

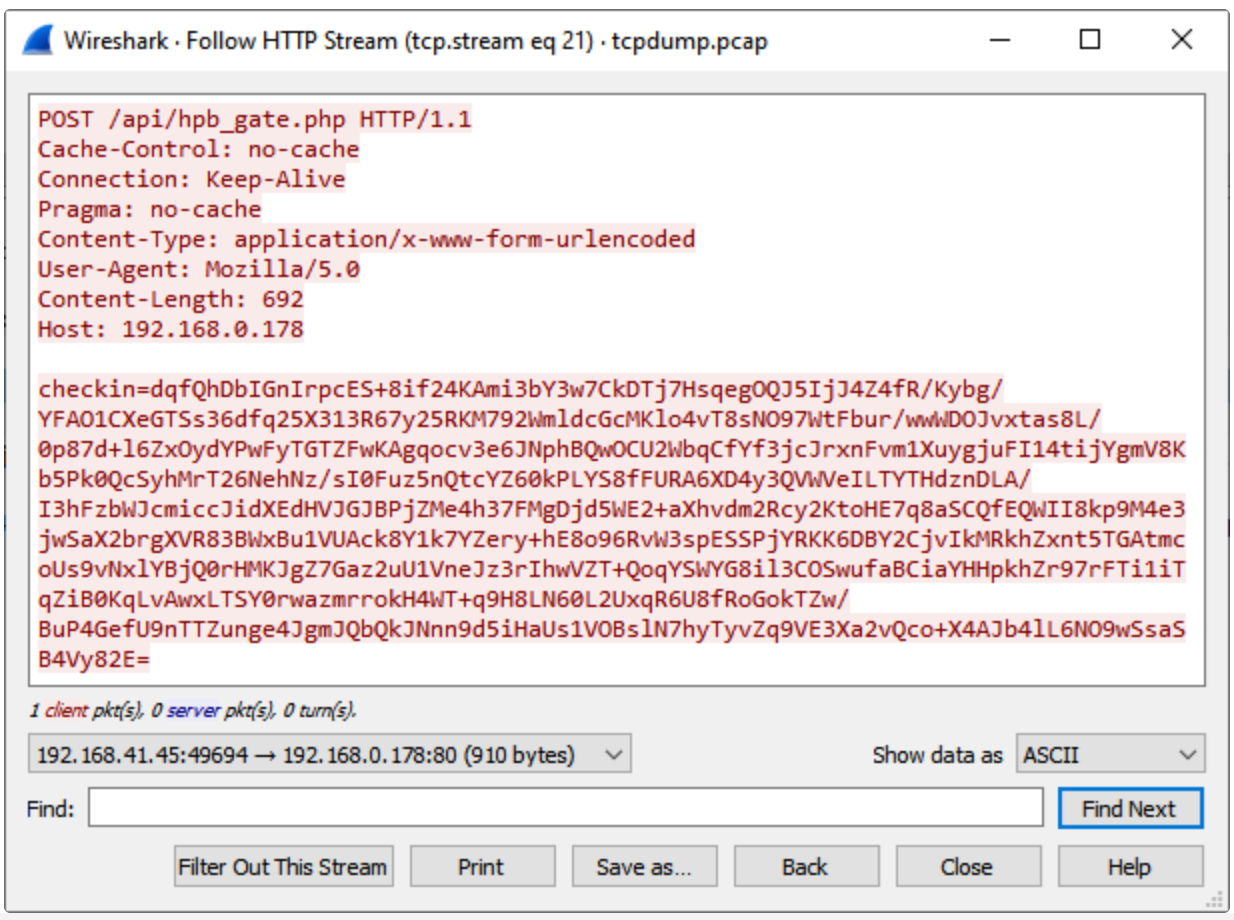


Figure 17. Example of a beacon HTTP POST message (generated by a sample from VirusTotal – the one with local IPs instead of real C&C addresses)

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:

…ported)

…ownload of payload data

… Table 2.

Table 2. C&C commands

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.
  - sed using LZMS.
  - resolved exclusively
  - mes are used to find the
  - sed to invoke the desired
  - 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
plexity of the whole
f these vulnerabilities
e vulnerabilities have
For a better image,
allowing UEFI Secure

ited by BlackLotus. One
are still not revoked,
stems with UEFI Secure

w, among other things,
orted by the OEM
so old – like 3-5 years at
the time of vulnerability disclosure). Read more in our blogpost: When "secure" isn't

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

- Later in 2022, we discovered a few other UEFI vulnerabilities, whose exploitation would also allow attackers to disable UEFI Secure Boot very easily. As pointed out by fellow researchers from Binarly, several devices listed in the advisory 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, 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 page.

IoCs

Files

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

0C0E78BF97116E781DDE0E00A1CD0C29E68D623D	N/A	Win64/BlackLotus.B	B
6D8CEE28DA8BCF25A4D232FEB0810452ACADA11D	N/A	Win64/BlackLotus.B	B
74FF58FCE8F19083D16DF0109DC91D78C94342FA	N/A	Win64/BlackLotus.B	B
ACC74217CBE3F2E727A826B34BDE482DCAE15BE6	N/A	Win64/BlackLotus.B	B
111C4998F3264617A7A9D9BF662D4B1577445B20	N/A	Win64/BlackLotus.B	B
17FA047C1F979B180644906FE9265F21AF5B0509	N/A	Win64/BlackLotus.C	B
1F3799FED3CF43254FE30DCDFDB8DC02D82E662B	N/A	Win64/BlackLotus.C	B
4B882748FAF2C6C360884C6812DD5BCBCE75EBFF	N/A	Win64/BlackLotus.C	B
91F832F46E4C38ECC9335460D46F6F71352CFFED	N/A	Win64/BlackLotus.C	B
994DC79255AEB662A672A1814280DE73D405617A	N/A	Win64/BlackLotus.C	B
FFF4F28287677CAABC60C8AB36786C370226588D	N/A	Win64/BlackLotus.C	B
71559C3E2F3950D4EE016F24CA54DA17D28B9D82	N/A	EFI/BlackLotus.C	B C st ir
D6D3F3151B188A9DA62DEB95EA1D1ABEFF257914	N/A	EFI/BlackLotus.C	B C st ir
547FAA2D64B85BF883955B723B07635C0A09326B	N/A	EFI/BlackLotus.A	B e
		EFI/BlackLotus.A	B e
		EFI/BlackLotus.A	B e M
		EFI/BlackLotus.A	B e M
		Win64/BlackLotus.D	B b

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

IP	Domain	Hosting provider	First seen
N/A	`xrepositoryx[.]name`	N/A	2022-10
N/A	`myrepositoryx[.]com`	N/A	2022-10
		oudflare, Inc.	2022-10
		gitalOcean, LLC	2022-10
		A VEESP	2022-10
		yLink Data enter BV	2022-10

190.147.189[.]122	egscorp[.]net	Telmex Colombia S.A.	2022-08

MITRE ATT&CK techniques

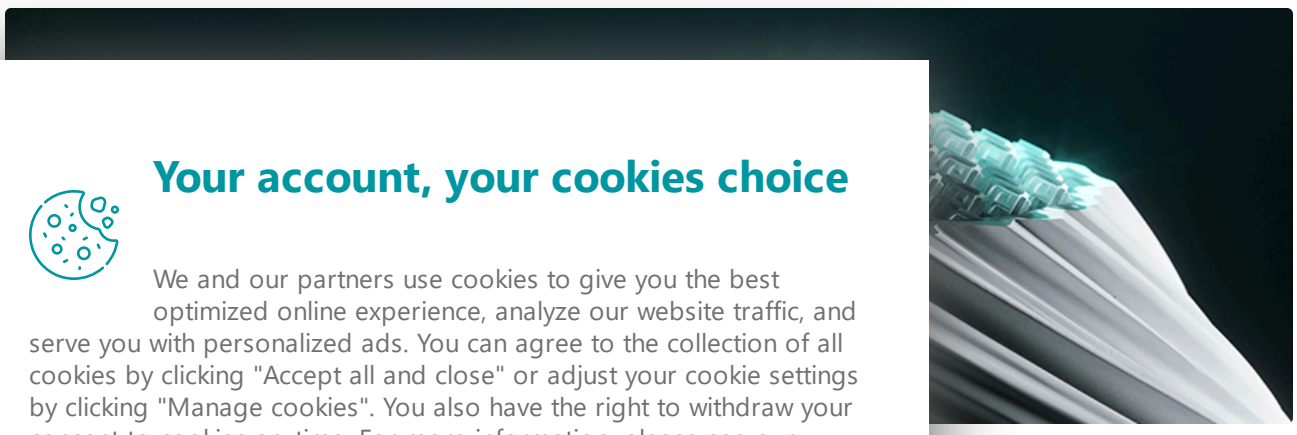
This table was built using version 12 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1587.002	Develop Capabilities: Code Signing Certificates	Some BlackLotus samples are signed with self-signed certificate.
	T1588.005	Obtain Capabilities: Exploits	BlackLotus used publicly known exploit to bypass UEFI Secure Boot.
Execution	T1203	Exploitation for Client Execution	BlackLotus installers can exploit CVE-2022-21894 to achieve arbitrary code execution on the systems with UEFI Secure Boot enabled.
	T1559	Inter-Process Communication	BlackLotus HTTP downloader uses named section to pass commands to the kernel-mode component.
	T1106	Native API	BlackLotus HTTP downloader uses various native Windows APIs to achieve code execution on the compromised machine.
	T1129	Shared Modules	BlackLotus HTTP downloader can load and execute DLLs received from the C&C server.
			us bootkit is deployed on ystem Partition and during the boot.
			us installer attempts to privileges by bypassing ount Control.
		Process with Token	us HTTP downloader can QueryUserToken and ocessAsUserW to execute downloaded payloads within a new

		process with local system privileges.
T1622	Debugger Evasion	BlackLotus components use various techniques to detect whether a kernel-mode or user-mode debugger is running on a victim.
T1574	Hijack Execution Flow	BlackLotus bootkit hijacks various components included in the early Windows boot process stages (Windows Boot Manager, Windows OS loader, Windows kernel and specific drivers) to avoid detection by deactivating various Windows security features (VBS, Windows Defender) and stealthily execute its kernel-mode and user-mode components
T1562	Impair Defenses	BlackLotus components can disable BitLocker and Windows Defender to avoid detection.
T1070.004	Indicator Removal: File Deletion	BlackLotus installer deletes itself after successfully deploying files to the EFI System partition. Also after successful CVE-2022-21894 exploitation, BlackLotus removes traces of exploitation by deleting all files included in exploitation chain from EFI System Partition.
T1070.009	Indicator Removal: Clear Persistence	BlackLotus can uninstall itself by removing all bootkit files from the ESP and restoring original victim's Windows Boot Manager.
		BlackLotus attempts to hide its oyed on the ESP by using e filenames, such as efi (if UEFI Secure Boot is on compromised machine) gfw.efi (if UEFI Secure sabled on compromised).
		us installer modifies registry to disable HVCI security feature.
		ll embedded strings in BlackLotus components are

T1027	Obfuscated Files or Information	encrypted using a custom combined cipher and decrypted only when needed.
T1027.007	Obfuscated Files or Information: Dynamic API Resolution	BlackLotus components use dynamic API resolution while using API names' hashes instead of names.
T1027.009	Obfuscated Files or Information: Embedded Payloads	Almost all embedded files in BlackLotus components are encrypted using AES.
T1542.003	Pre-OS Boot: Bootkit	BlackLotus bootkit is deployed on the EFI System Partition and executed during the early OS boot stages, and thus is capable of controlling the OS boot process and evading detection.
T1055.012	Process Injection: Dynamic-link Library Injection	BlackLotus HTTP downloader can inject a DLL into a newly created `svchost.exe` process using process hollowing.
T1055.002	Process Injection: Portable Executable Injection	BlackLotus driver injects the HTTP downloader portable executable into a `winlogon.exe` process.
T1014	Rootkit	BlackLotus kernel driver protects the bootkit files on the ESP from removal.
T1497.001	Virtualization/Sandbox Evasion: System Checks	BlackLotus employs various system checks including checking sandbox-specific registry values, to detect and avoid virtualization and analysis ents.
		us components use various es to detect whether a ode or user-mode is running on a victim.
		us collects system on (IP, GPU, CPU, memory, n) on a compromised host.
T1614	Discovery	us can exit if one of the system locales is identified on the compromised

			host: `ro-MD`, `ru-MD`, `ru-RU`, `uk-UA`, `be-BY`, `hy-AM`, `kk-KZ`.
Discovery	T1016	System Network Configuration Discovery	BlackLotus HTTP downloader can determine the public IP of a compromised host by requesting `api.ipify[.]org` service.
	T1016.001	System Network Configuration Discovery: Internet Connection Discovery	BlackLotus HTTP downloader checks the internet connection by querying Microsoft's `www.msftncsi[.]com/ncsi[.]txt`
	T1497.001	Virtualization/Sandbox Evasion: System Checks	BlackLotus employs various system checks including checking sandbox-specific registry values, to detect and avoid virtualization and analysis environments.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	BlackLotus uses HTTPS for communication with its C&C.
	T1132.001	Data Encoding: Standard Encoding	BlackLotus encodes encrypted data in C&C communication with URL-safe base64.
	T1573.001	Encrypted Channel: Symmetric Cryptography	BlackLotus uses 256-bit AES in CBC mode to decrypt messages received from its C&C.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	BlackLotus uses an embedded RSA public key to encrypt messages sent to C&C.



Related Articles

ESET RESEARCH
CloudScout: Evasive Panda scouting cloud services

ESET RESEARCH
ESET Research Podcast: CosmicBeetle

ESET RESEARCH
Embargo ransomware: Rock'n'Rust

Discussion