black hat ASIA 2018
MARCH 20-23, 2018
MARINA BAY SANDS / SINGAPORE

I Don't Want to Sleep Tonight:

Subverting Intel TXT with S3 Sleep

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

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

#BHASIA / @BlackHatEvents

Who Are We?

- Senior security researcher at NSR (National Security Research Institute of South Korea)
- Speaker at Black Hat Asia 2017 and HITBSecConf 2016/2017
- Author of the book series titled “64-bit multi-core OS principles and structure, Vol.1&2”
- a.k.a kkamagui, @kkamagui1

- Senior security researcher at NSR
- Embedded system engineer
- Interested in firmware security and IoT security
- a.k.a davepark, @davepark312

Goal of This Presentation

- **We present new attack vector, "S3 Sleep" to subvert hardware-based security**
  - S3 sleeping state cuts off the power of CPU and devices
  - We intercept control flow while system wakes up and subvert hardware-based security
- **We present new vulnerability, "Lost Pointer" (CVE-2017-16837)**
  - "Lost pointer" is a software vulnerability in tBoot which is a reference implementation of Intel TXT
  - We explain the vulnerability in detail and show mitigation

Background

Boot Protection Mechanisms

Intel Trust Execution Technology (TXT) and tBoot

Lost Pointer Vulnerability and Demo.

Mitigation and Conclusion

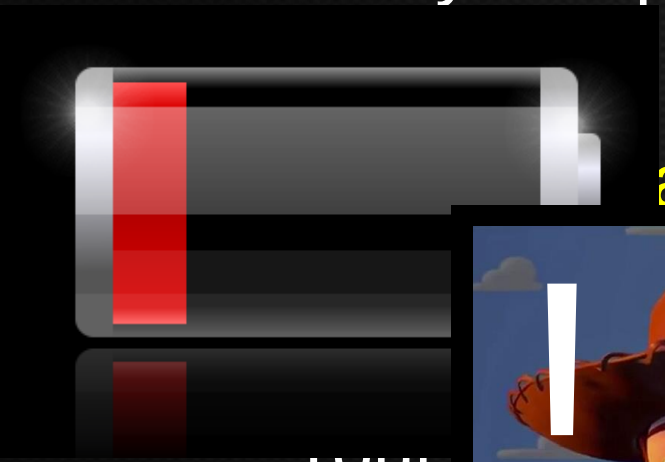
(with Black Hat Sound Bytes)

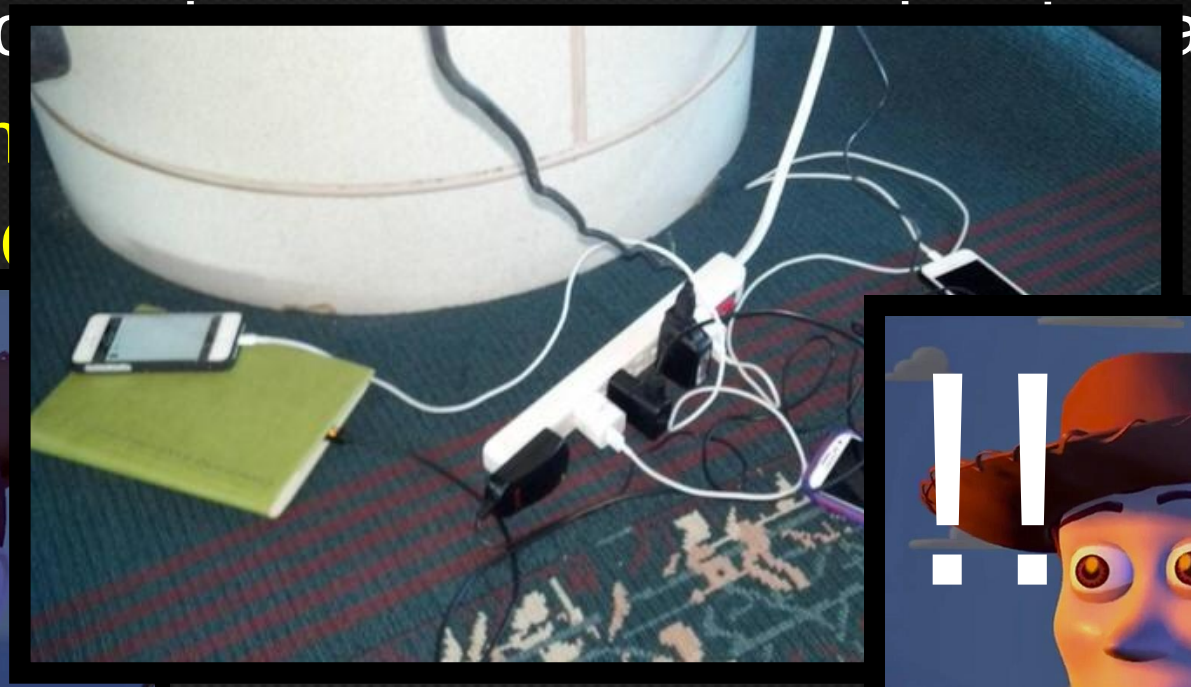
Saving Power is Important!

- **Power Consumption == Cost**
  - Many companies worry about power consumption, because …
  - Low Power Consumption = Low Electricity Fee!
- **Power Consumption == Running Time**
  - Your laptop works on BATTERY!
  - Low Power Consumption = Long Running Time = Inner Peace!
  - We wish the battery could last a day, but, …

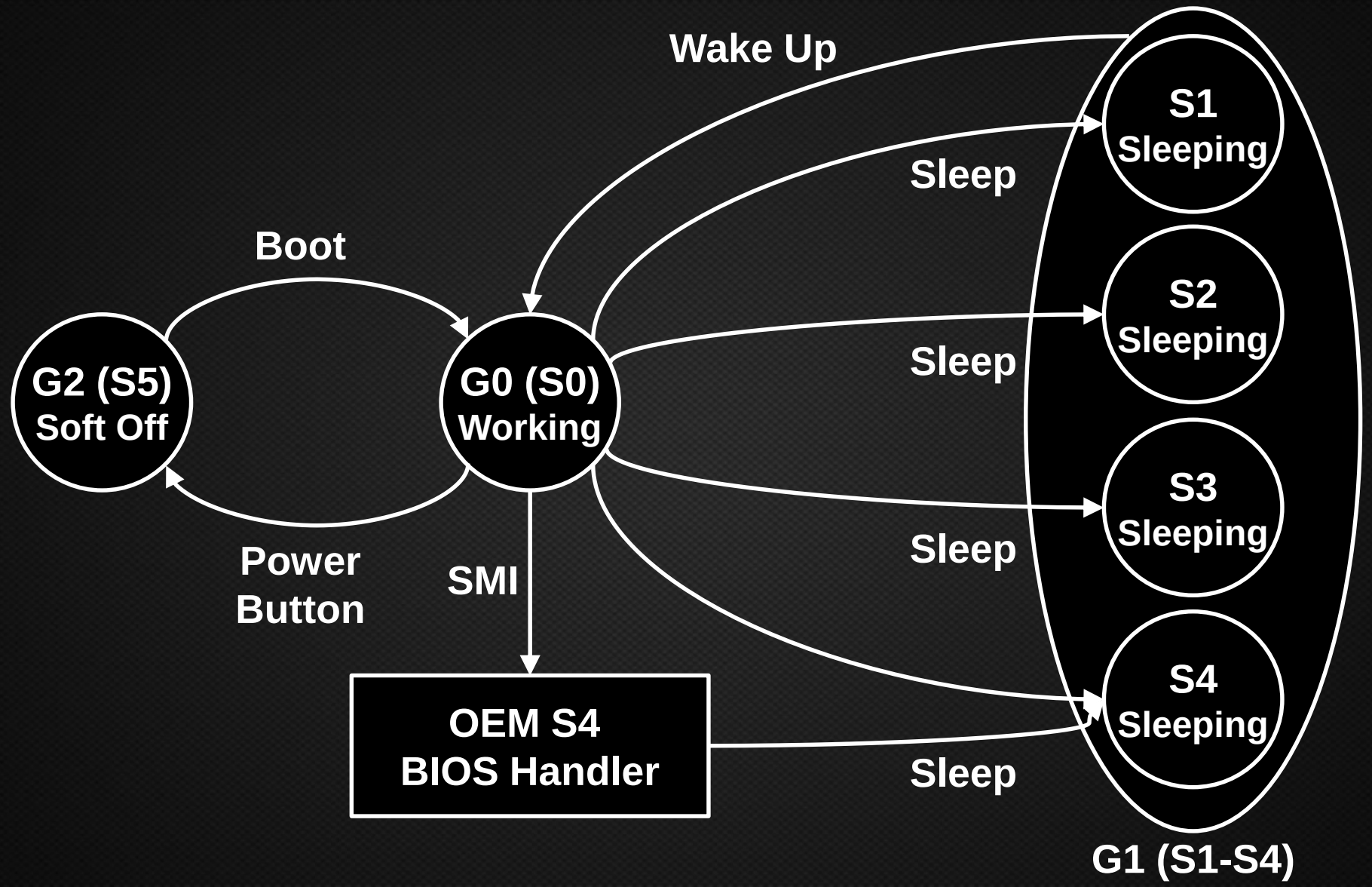
Saving Power is Important!

- **Power Consumption == Cost**
  - Many companies wo ause …
  - Consum
  - acturing
  - Your
  - Low
  - We w d last a day, but, …





Advanced Configuration and Power Interface (ACPI) Sleeping States



Impacts of ACPI Sleeping States

- **Impacts of ACPI sleeping states are as follows:**
  - 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

Impacts of ACPI Sleeping States

- **Impacts of ACPI sleeping states are as follows:**
  - 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

POWER OFF ?!

ACPI Sleeping State and Security

- Sleeping states cut off the power of CPU, devices, and RAM!
- If the OS works with sleeping states, security hardware must do so
- Because of power off, their states need to be restored and reinitialized for waking up

If we intercept sleep and wake up, we can do something interesting!

Attack Surface of S3 Sleeping State in Linux Kernel

Notify Power Manager Chain(Sleep)
- pm_notifier_call_chain()
- 64bit Mode, Multiple CPU

Notify Device Chain(Sleep)
- dpm_prepare(), dpm_suspend(), dpm_suspend_late()
- 64bit Mode, Single CPU

Sleep, De-Power of CPU and Devices
- acpi_os_enter_sleep()

Wake Up (Vector of ACPI) and Resume Kernel
- Real Mode, Single CPU

Notify Device Chain(Resume)
- dpm_resume_noirq(), dpm_resume()
- 64bit Mode, Single CPU

Notify Power Manager Chain(Resume)
- pm_notifier_call_chain()
- 64bit Mode, Multiple CPU

Attack Surface of S3 Sleeping State in Linux Kernel

Notify Power Manager Chain(Sleep)
- pm_notifier_call_chain()
- 64bit Mode, Multiple CPU

Notify Device Chain(Sleep)
- dpm_prepare(), dpm_suspend(), dpm_suspend_late()
- 64bit Mode, Single CPU

Sleep, De-Power of CPU and Devices
- acpi_os_enter_sleep()

Wake Up (Vector of ACPI) and Resume Kernel
- 64bit Mode, Single CPU

Notify Device Chain(Resume)
- dpm_resume_early(), dpm_resume()
- 64bit Mode, Single CPU

Notify Power Manager Chain(Resume)
- pm_notifier_call_chain()
- 64bit Mode, Multiple CPU

You can intercept control flow by registering a callback function!

Now,
we got the power to change control flow!

Let’s get into
a more privileged level!

Background

Boot Protection Mechanisms

Intel Trust Execution Technology (TXT) and tBoot

Lost Pointer Vulnerability and Demo.

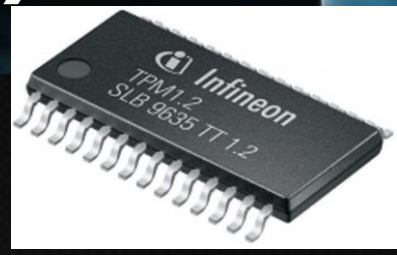
Mitigation and Conclusion

(with Black Hat Sound Bytes)

Boot Protection Mechanisms

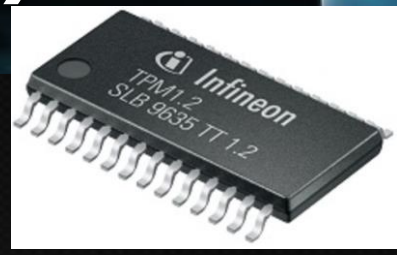
- **Secure boot of Unified Extensible Firmware Interface (UEFI)**
  - CHECKS a cryptographic signature of the binary executed in the next step PRIOR TO the execution
  - Stops the execution if the executable file has no valid signature
- **Measured boot**
  - MEASURES a hash of the binary executed in the next step PRIOR TO the execution, and then stores the measurement to the secure storage of Trusted Platform Module (TPM)
  - CHECKS the platform hashes in TPM AFTER the boot process

Trusted Platform Module (TPM)



- **TPM is designed to provide hardware-based security functions and build up a trusted platform**
  - It is a tamper-resistant device and has a random number generator, encryption functions, Platform Configuration Registers (PCRs), etc.
  - It has been widely deployed in commercial products
  - Trusted Computing Group makes MANY specifications about TPM
- **PCRs of TPM store hashes and reveal the system status**
  - They can be used to seal data (like BitLocker) and verified by remote attester

Trusted Platform Module (TPM)



- TPM is designed to provide **hardware-based security functions** and build up a **trusted platform**
  - It is a tamper-resistant device and has a random number generator, encryption functions, Platform Configuration Registers (PCRs), etc.
  - It has been widely deployed in commercial devices
  - Trusted Computing Group has MANY specifications about TPM
- PCRs of TPM store hashes and reveal the system status
  - They can be used to seal data (like BitLocker) and verified by remote attester

PCRs are stored safely and…

They cannot be reset

while the system is running!

Static and Dynamic Root of Trust for Measurement

- **Root of Trust for Measurement (RTM) of TCG architecture is the trust anchor of a measurement chain**
- **Static RTM makes a chain from Core RTM (CRTM)**
  - CRTM in protected firmware block starts at power-on state or restart
- **Dynamic RTM makes a chain at runtime without power-on or restart**
  - It executes a special code module (DRTM Configuration Environment, DCE) and it ensures the platform is in a trustworthy state
  - DCE executes a Dynamically Launched Measured Environment (DLME)

Background

Boot Protection Mechanisms

Intel Trust Execution Technology (TXT) and tBoot

Lost Pointer Vulnerability and Demo.

Mitigation and Conclusion

(with Black Hat Sound Bytes)

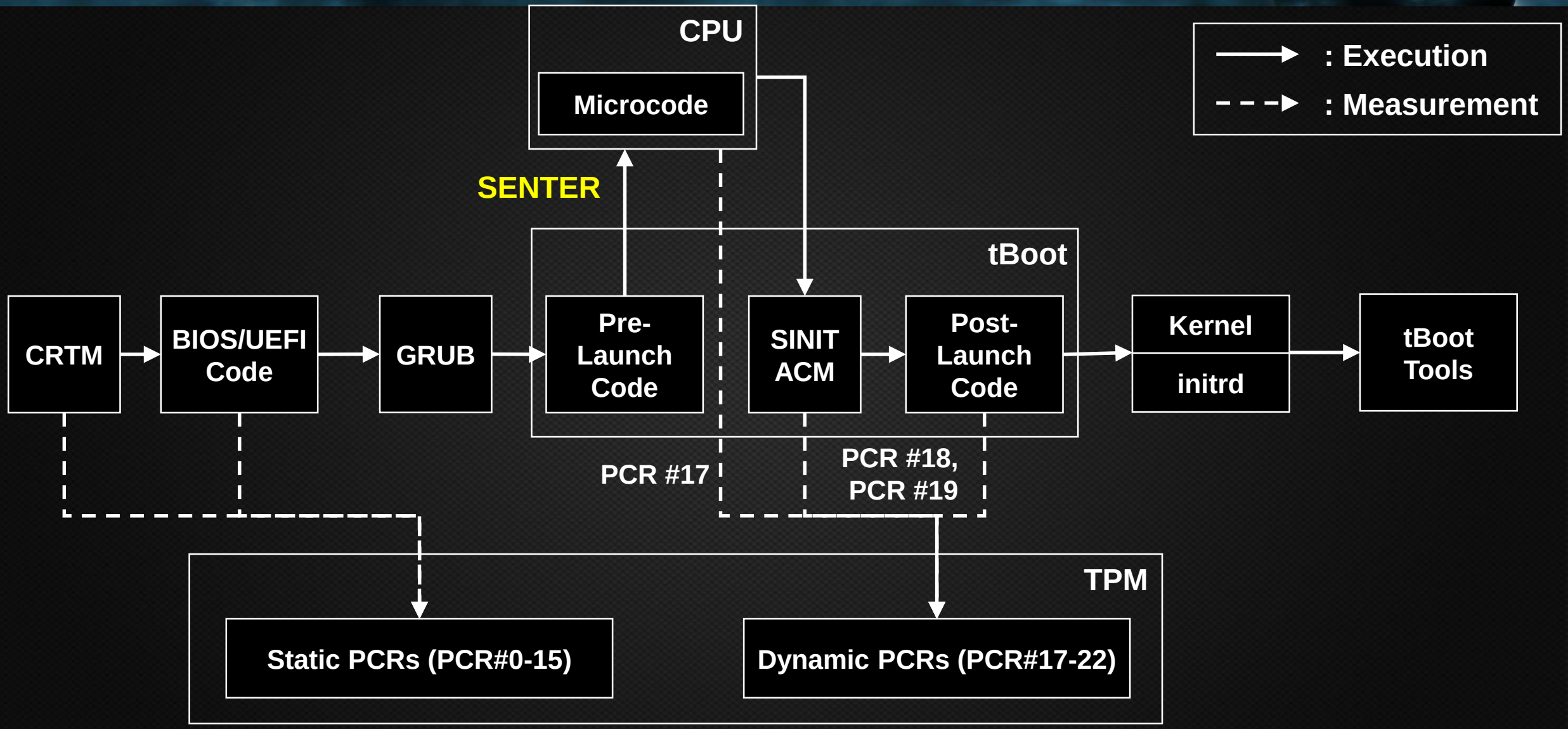
Intel Trusted Execution Environment

- **Intel Trusted Execution Technology (TXT) is the DRTM tech.**
  - Intel just uses different terminologies
  - ex) DCE = Secure Initialization Authenticated Code Module (SINIT ACM)
    DLME = Measured Launched Environment (MLE)
  - It extends hashes of SINIT ACM and MLE to TPM
- **Intel TXT has a special command (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

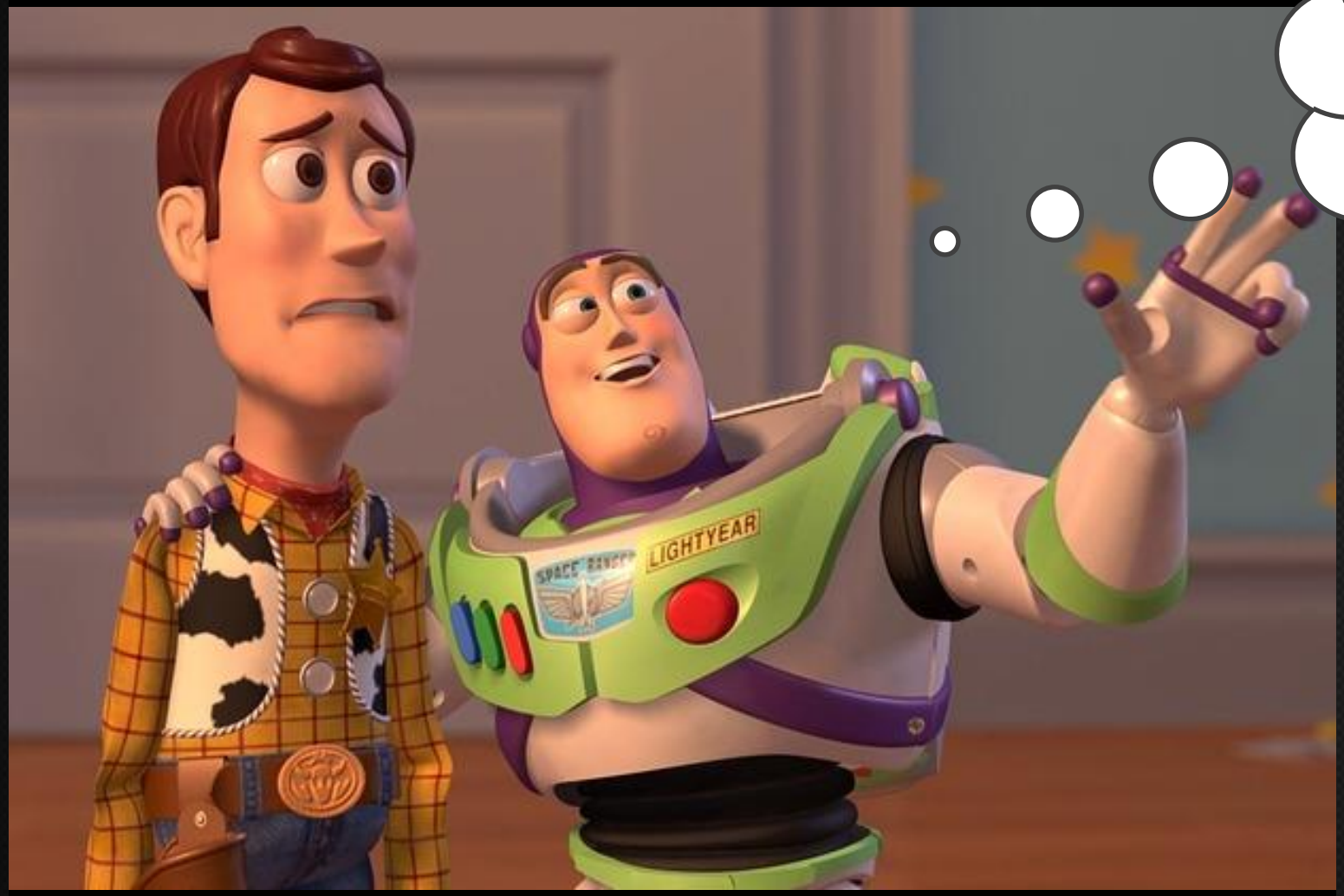
Trusted Boot (tBoot)

- **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
- **tBoot can verify OS and VMM**
  - It measures TXT and OS components and stores hashes to TPM
  - Measured results in PCRs of TPM can be verified by remote attestation server such as Intel Open CIT
  - It is typically used in server environments

Boot Process with tBoot



IMAGINE SECURE WORLD



WITH TRUSTED BOOT!

TRUSTED COMPUTING GROUP®

intel®

Background

Boot Protection Mechanisms

Intel Trust Execution Technology (TXT) and tBoot

Lost Pointer Vulnerability and Demo.

Mitigation and Conclusion

(with Black Hat Sound Bytes)

Previous Works on tBoot (1)

- **"Attacking Intel trusted execution technology", Black Hat DC (2009)**
  - System Management Mode (SMM) code attack
- **"Another way to circumvent Intel trusted execution technology", Invisible Things Lab (2009)**
  - DMA remapping table (DMAR table) attack
- **"Attacking Intel TXT via SINIT code execution hijacking", Invisible Things Lab (2011)**
  - Also DMAR table attack

Previous Works on tBoot (2)

- **"Breaking hardware-enforced security with hypervisor", Black Hat USA (2016)**
  - SENTER emulation with malicious hypervisor

Many researchers have aimed at Intel TXT and tBoot!

But, they only focused on the boot process!

Sleep and Waking Up Sequence of 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. De-Power of 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 and Waking Up Sequence of 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()

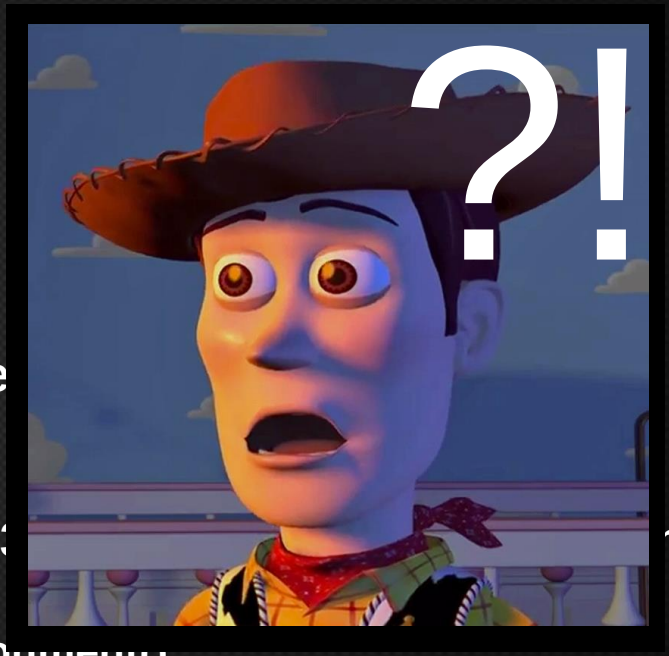
Shutdown Intel TXT
- txt_shutdown()

Sleep. De-Power of CPU and the TPM!
- shutdown_system()

Wake Up, Restore Static PCRs, and Re
- Real Mode, Single CPU

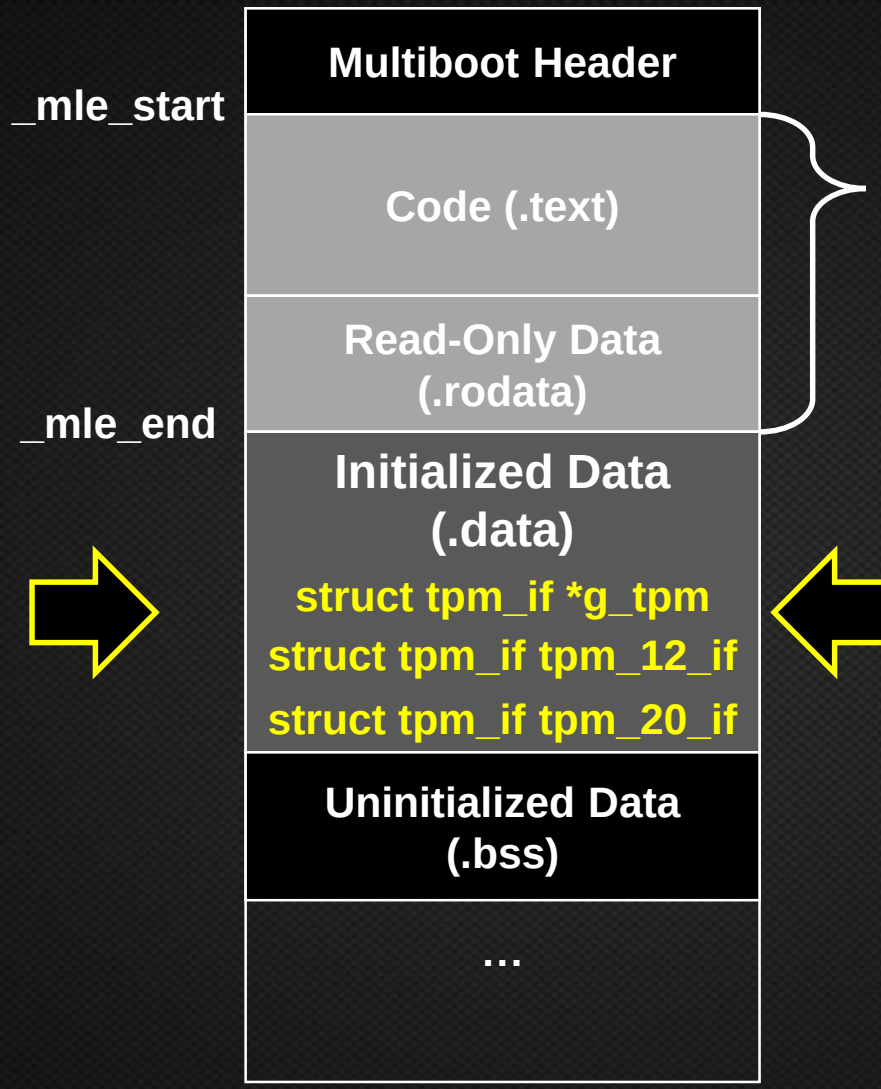
Launch MLE again and then, Unseal S
PCRs
- begin_launch() → txt_s3_launch_environment()
- post_launch() → s3_launch() → verify_integrity() → g_tpm->unseal()

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
```

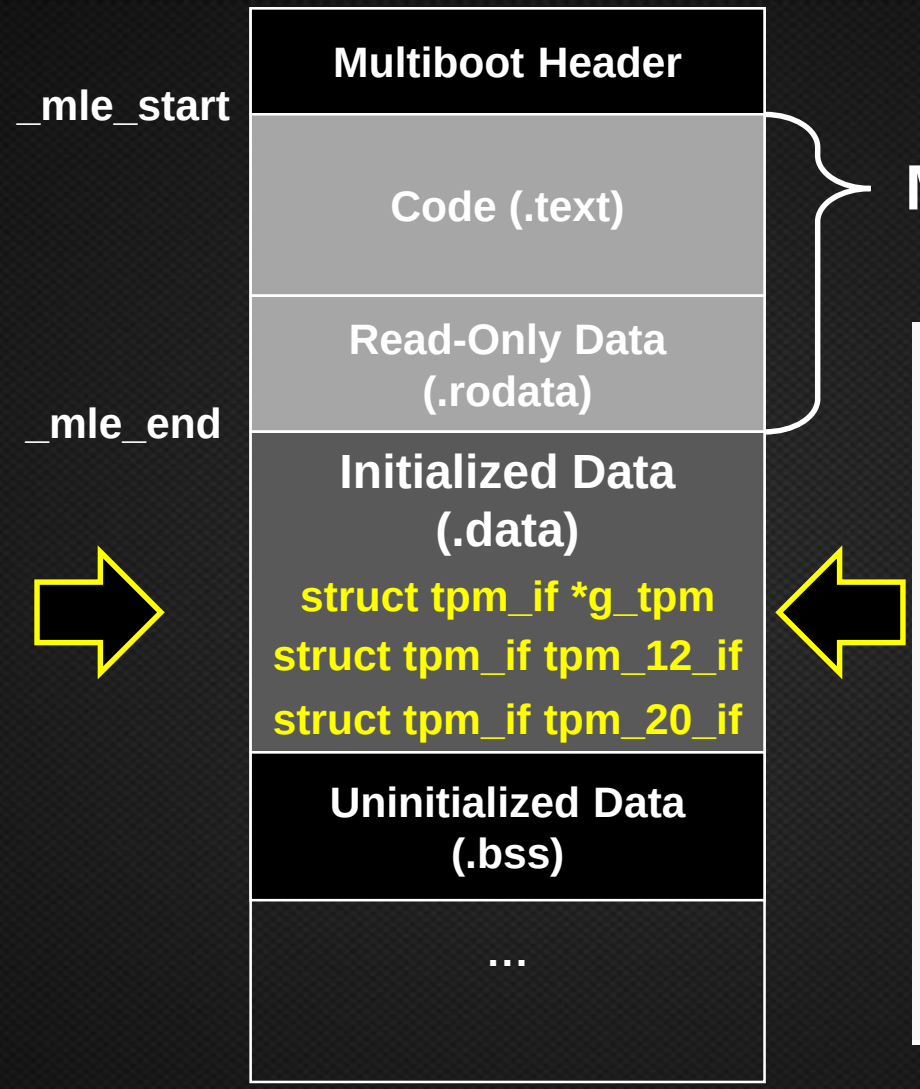
Memory Layout of tBoot

Measured by Intel TXT!

```
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,
};
```

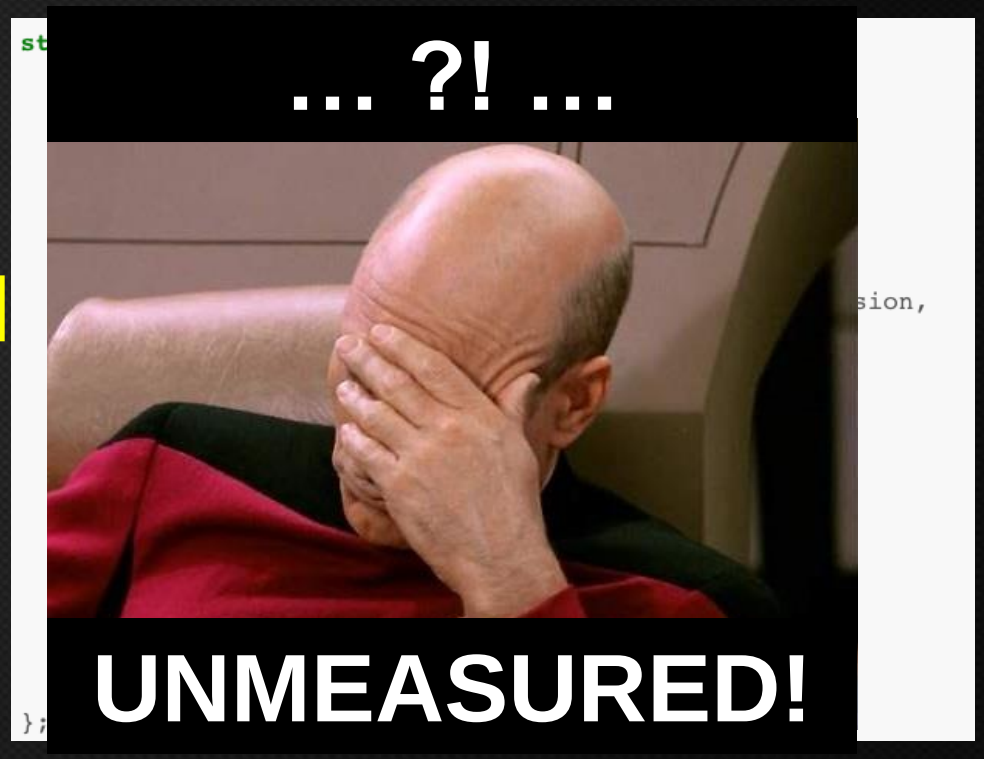
“Lost Pointer” Vulnerability
(CVE-2017-16837)


```
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
```

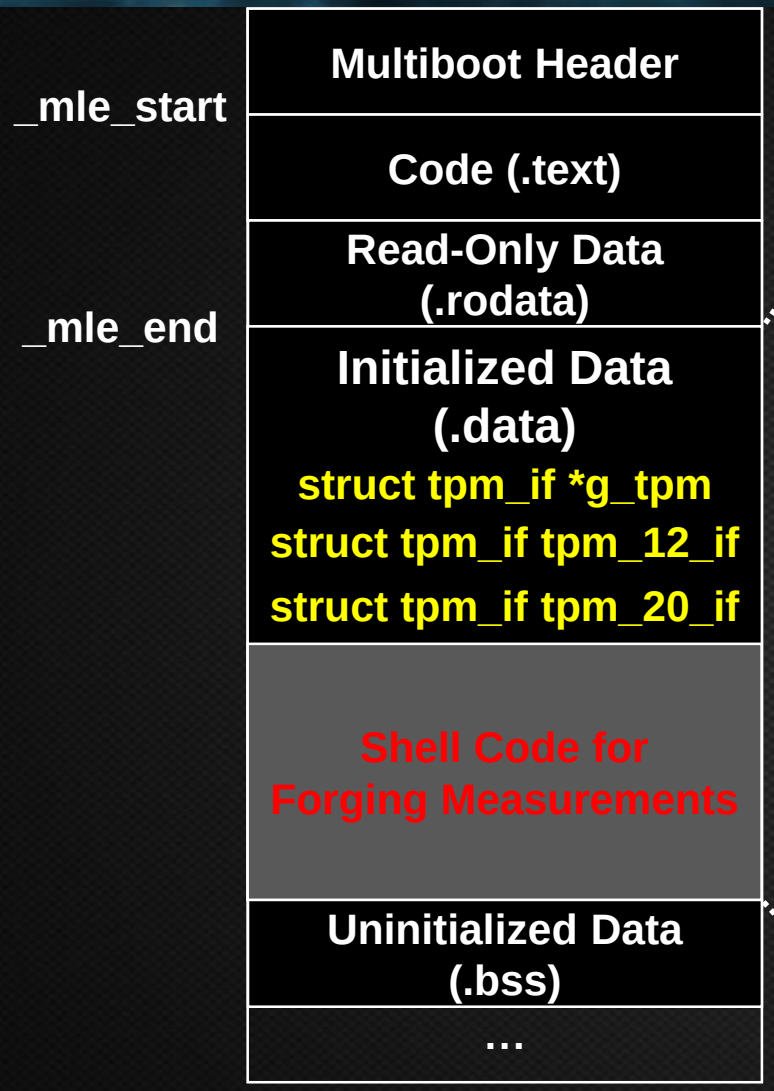


Memory Layout of tBoot

Measured by Intel TXT!



Hijack the Control Flow and Exploit tBoot!



Memory Layout of tBoot

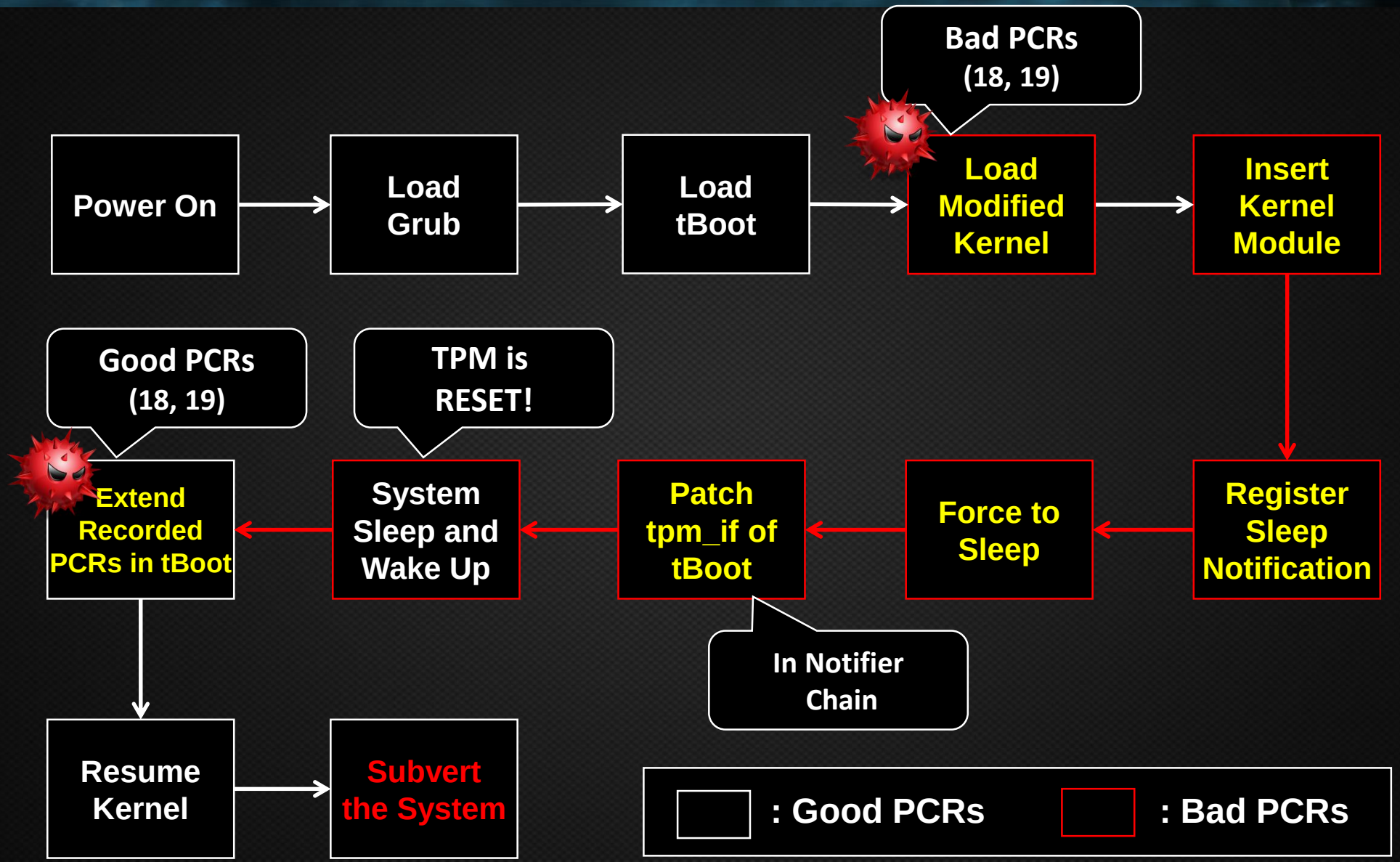
Hook tpm_12_if structure for g_tpm (extend_pcr → hook_extend_pcr)

Hook tpm_20_if structure for g_tpm (extend_pcr → hook_extend_pcr)

```
bool hook_extend_pcr(pcr, hash)
        for tpm_12_if and tpm_20_if
{
    hash18 = a good hash for PCR 18
    hash19 = a good hash for PCR 19

    if (pcr == 18)
        return org_extend_pcr(pcr,
                    hash18);
    else if (pcr == 19)
        return org_extend_pcr(pcr,
                    hash19);
    else
        return org_extend_pcr(pcr,
                    hash);
}
```

Put it all together!



IMAGINE SECURE WORLD

… AND KEEP IT UP, DUDE!

TRUSTED COMPUTING GROUP®

intel®

with Your Belief!

Load
odified
ernel

Module

Register
Sleep
Notification

rce to
Sleep

: Bad PCRs

DEMO

```
PCR-00: 5C A8 82 83 6E 28 0D 8E 2C F3 8C D5 46 92 35 72 82 49 C8 93
PCR-01: 3A 3F 78 0F 11 A4 B4 99 69 FC AA 80 CD 6E 39 57 C3 3B 22 75
PCR-02: 8B 14 FE 1B 2F 64 BC 01 65 44 BA 02 23 8F F0 5F 5A E0 19 4A
PCR-03: 3A 3F 78 0F 11 A4 B4 99 69 FC AA 80 CD 6E 39 57 C3 3B 22 75
PCR-04: 6D 10 9E [illegible] AC 43 FF FD 07
PCR-05: 4A D1 66 [illegible] 13 E9 11 4F 9E
PCR-06: 3A 3F 78 [illegible] 57 C3 3B 22 75
PCR-07: 97 B0 DE [illegible] 23 EC 14 68 41
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 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-10: 47 FA 34 84 16 11 12 6F A8 1B 52 49 F6 1E 39 B3 BC 6C 1D E8
PCR-11: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-12: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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: 8A F3 11 F3 B5 B4 CA 74 7C 5F D0 6C FE DD CD 86 C0 8C 38 25
PCR-18: B0 A5 47 68 36 C0 08 00 F6 C0 BC 6C AA F7 F5 36 5E 45 CB 43
PCR-19: 6B 0C 61 65 23 3D 55 9C 8A 9A 5D E9 8B D6 72 86 F0 6A 0A 8B
PCR-20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-21: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-22: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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
```

Background

Boot Protection Mechanisms

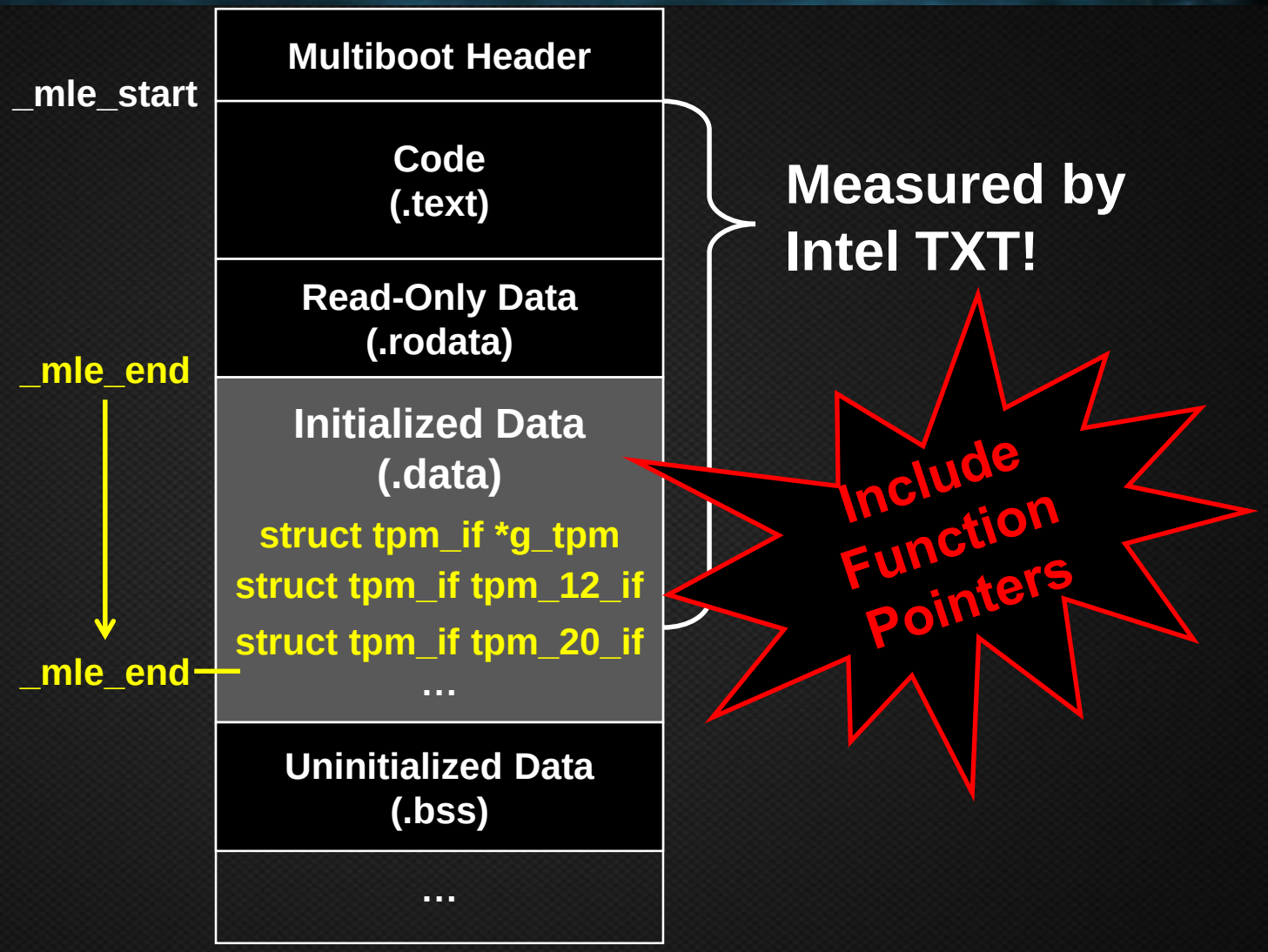
Intel Trust Execution Technology (TXT) and tBoot

Lost Pointer Vulnerability and Demo.

Mitigation and Conclusion

(with Black Hat Sound Bytes)

Mitigation – Measure Function Pointers (1)



Memory Layout of tBoot

Mitigation – Measure Function Pointers (2)


```
-extern struct tpm_if tpm_12_if;
-extern struct tpm_if tpm_20_if;
-extern struct tpm_if *g_tpm;
+extern struct tpm_if_data tpm_if_data;
+extern const struct tpm_if_fp tpm_12_if_fp;
+extern const struct tpm_if_fp tpm_20_if_fp;
+extern uint8_t g_tpm_ver;
 extern uint8_t g_tpm_family;

 extern bool tpm_validate_locality(uint32_t locality);
@@ -501,6 +510,8 @@
 extern bool tpm_relinquish_locality_crb(uint32_t
 extern bool txt_is_launched(void);
 extern bool tpm_workaround_crb(void);
+extern struct tpm_if *get_tpm(void);
+extern const struct tpm_if_fp *get_tpm_fp(void);
```

Extract Data from TPM Interfaces

```
--- a/tboot/common/tpm_12.c
+++ b/tboot/common/tpm_12.c
@@ -1914,8 +1914,7 @@

     return ( re
 }
-
-struct tpm_if tpm_12_if = {
+const struct tpm_if_fp tpm_12_if_fp = {
     .init = tpm12_init,
                    12_pcr_read,
                    pm12_pcr_extend,
                    @@
                    pm12_save_state,
                    12_cap_pcrs,
                    check,

                    t_a = TIMEOUT_A,
                    t_b = TIMEOUT_B,
                    t_c = TIMEOUT_C,
                    t_d = TIMEOUT_D,
```

Move Interfaces to Read-Only Area

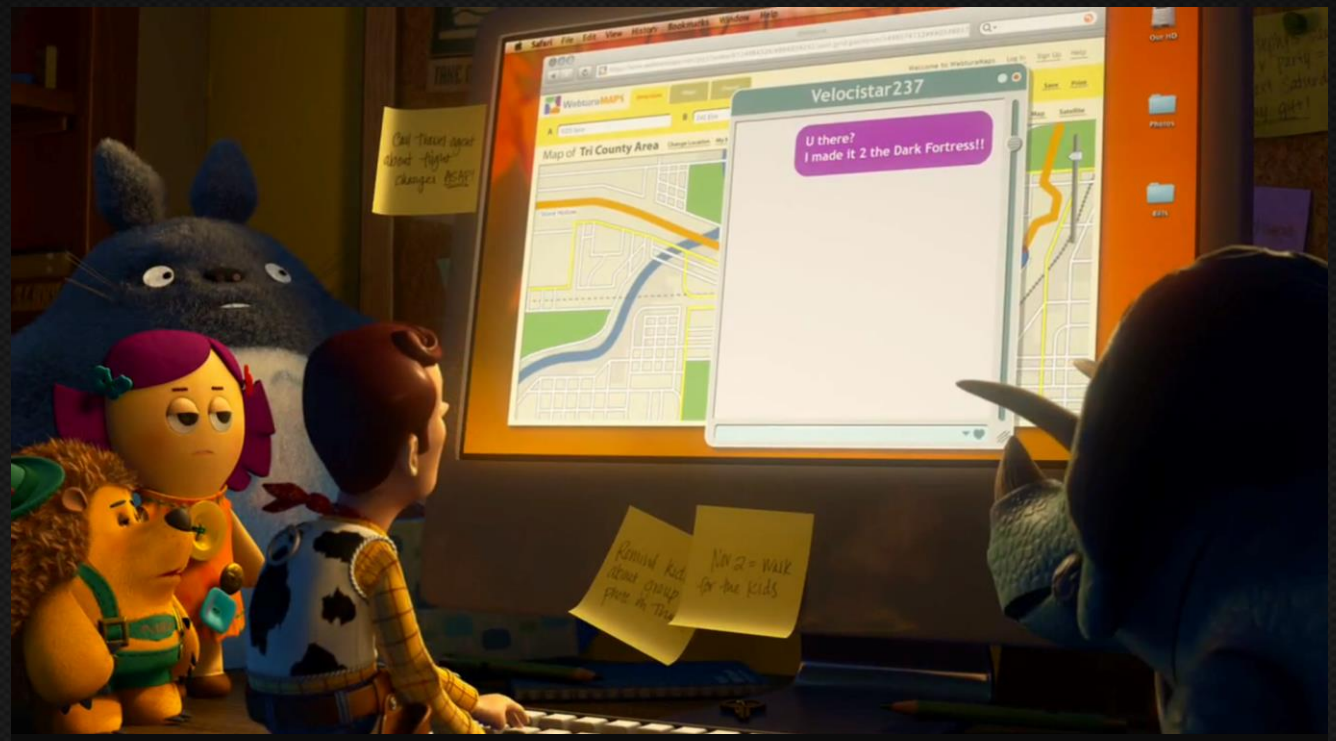
```
--- a/tboot/common/tpm_20.c
+++ b/tboot/common/tpm_20.c
@@ -2615,7 +2615,7 @@
     return true;
 }

-struct tpm_if tpm_20_if = {
+const struct tpm_if_fp tpm_20_if_fp = {
     .init = tpm20_init,
     .pcr_read = tpm20_pcr_read,
     .pcr_extend = tpm20_pcr_extend,
```

Conclusion and Black Hat Sound Bytes

- **ACPI S3 sleeping states turn off the CPU and the TPM**
  - When the system wakes up, it should turn on security function of CPU and recover PCRs of TPM
- **tBoot does not measure all function pointers!**
  - “Lost Pointer” vulnerability can be used to forge the PCR values while system sleeps and wakes up
  - tBoot should measure all data related to the control flow
- **Update your tBoot to the latest version!**
  - Or disable the sleep feature in your BIOS!

Questions ?



EMAIL!

Contact: hanseunghun@nsr.re.kr, @kkamagui1
parkparkqw@nsr.re.kr, @davepark312

Reference

- 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.
- Wojtczuk, Rafal, and Joanna Rutkowska. “Attacking intel trusted execution technology.” Black Hat DC. 2009.
- Wojtczuk, Rafal, Joanna Rutkowska, and Alexander Tereshkin. “Another way to circumvent Intel trusted execution technology.” Invisible Things Lab. 2009.
- Wojtczuk, Rafal, and Joanna Rutkowska. “Attacking Intel TXT via SINIT code execution hijacking.” Invisible Things Lab. 2011.
- Sharkey, Joseph. “Breaking hardware-enforced security with hypervisors.” Black Hat USA. 2016.