



# Bypassing a Hardware-Based Trusted Boot Through x86 CPU Microcode Downgrade

---

Alexander Ermolov  
[@flothrone](https://twitter.com/flothrone)



# #WhoAmI

---

- Former team member at [Digital Security](#) and [Embedi](#)
- Intel ME
  - [Intel AMT. Stealth Breakthrough](#)
- Intel Boot Guard
  - [Safeguarding rootkits: Intel Boot Guard](#)
  - [Bypassing Intel Boot Guard](#)
- UEFI BIOS
  - [UEFI BIOS holes: So Much Magic, Don't Come Inside](#)
  - [NUClear exploitation](#)



# #Agenda

---

- CPU microcode basics
- Downgrading microcode
- Discovering impact
- Mitigations & takeaways



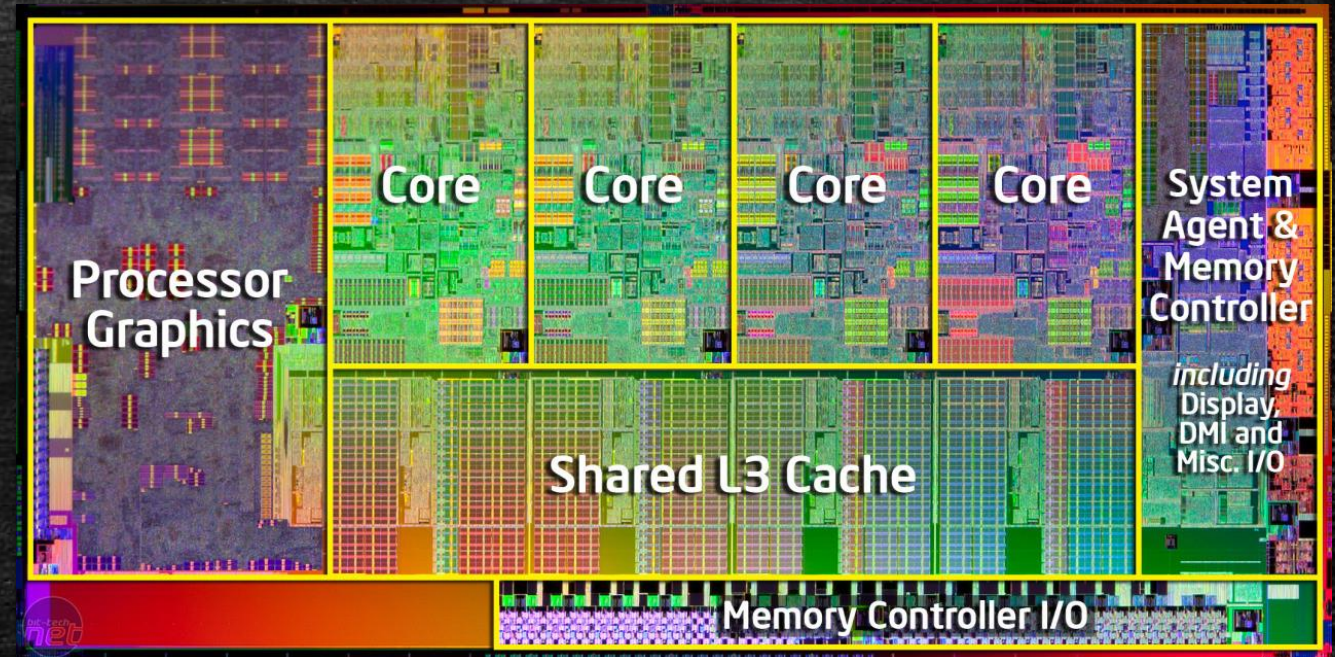
# CPU microcode basics

---



# Inside Intel CPU

- Processor cores
  - BSP (Bootstrap Processor)
  - APs (Application Processors)
- Graphics core
- IMC (Integrated Memory Controller)
- L3 cache
- I/O logic

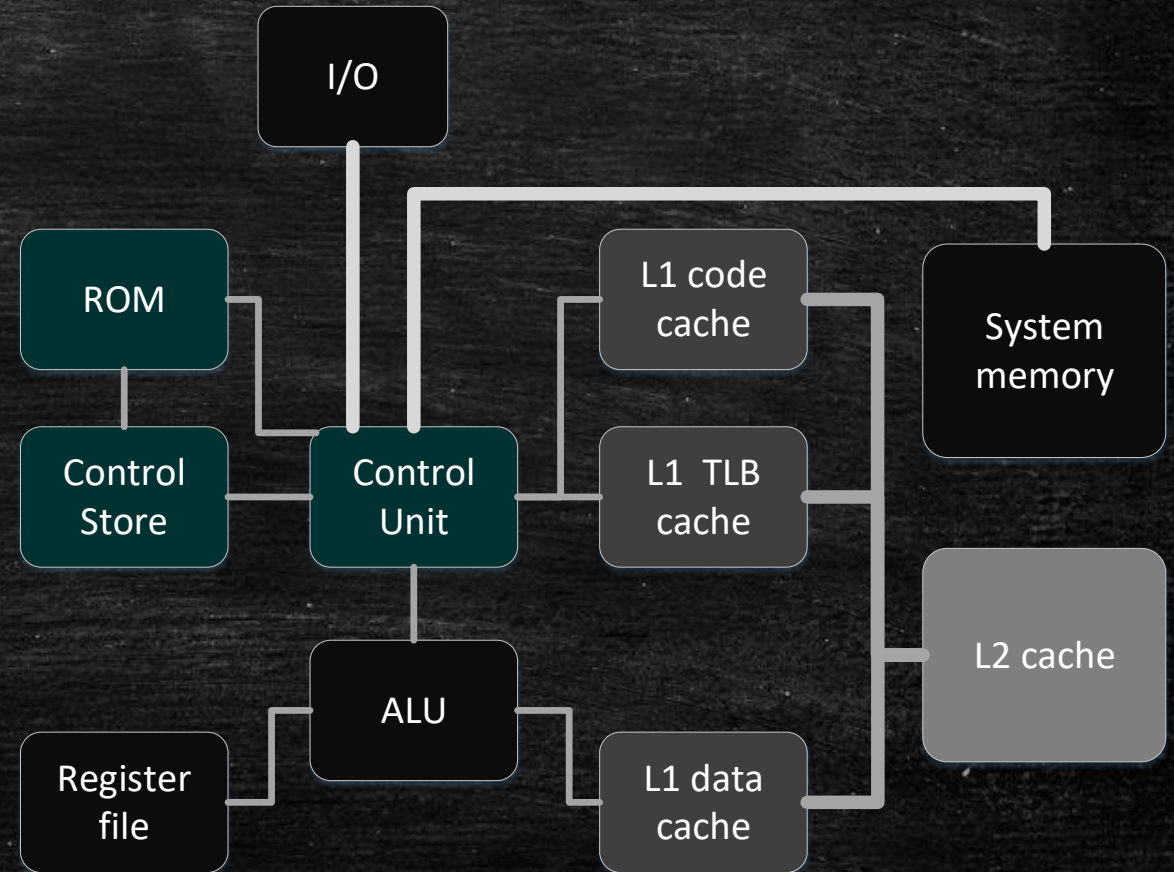




# Inside Intel CPU

Each core has its own:

- Control (execution) unit to decode instructions
- ALU to perform arithmetic, load/store, ... actions
- Register file
- L1 and L2 cache





# Microcode

Control Unit has Microcode ROM that contains the CPU microcode - a program written in a hardware-level instructions to implement a higher-level instructions

For example, MOVS instruction implementation:

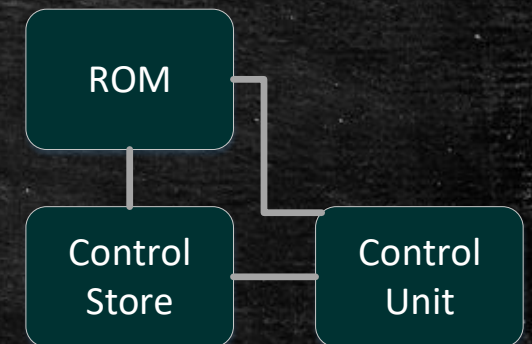
```
LLDF      ; load direction flag to latch in functional unit
OR ecx, ecx ; test if ECX is zero
JZ end    ; terminate string move if ECX is zero
```

loop:

```
MOVFM tmp0, [esi] ; move the data to tmp data from source and inc/dec ESI
MOVIM [edi], tmp0 ; move the data to destination and inc/dec EDI
EDECXJNZ loop    ; dec ECX and repeat until zero
```

end:

```
EXIT
```



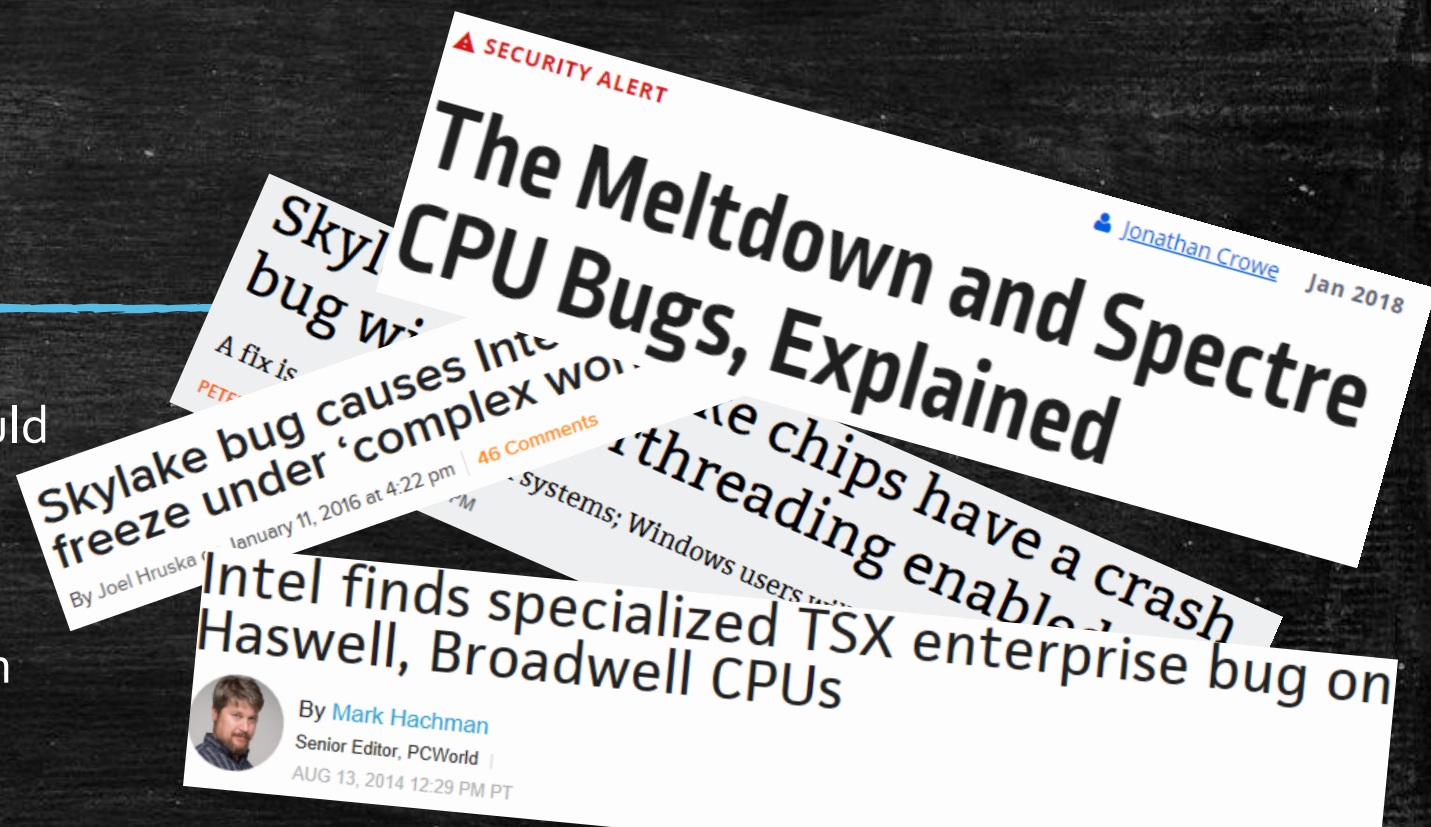
Security Analysis of x86 Processor Microcode  
[https://www.dcddcc.com/docs/2014\\_paper\\_microcode.pdf](https://www.dcddcc.com/docs/2014_paper_microcode.pdf)



# Microcode update

Microcode can have bugs, so it should be updatable

The updated microcode has to be loaded into Control Store upon each CPU power on

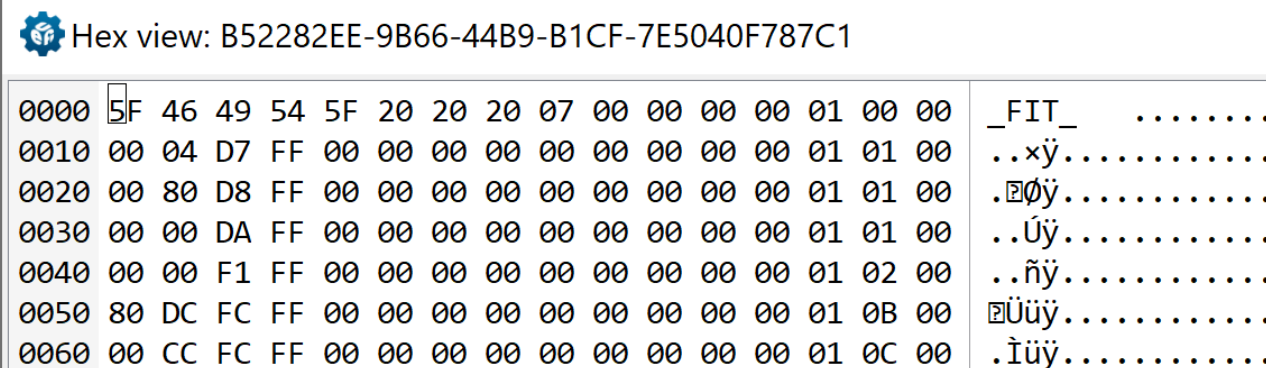


	Address	Size	Version	Checksum	Type	
1	_FIT_	00000070h	0100h	00h	FIT Header	
2	00000000FFD70400h	00017C00h	0100h	00h	Microcode	CPUID: 000906EAh, Revision: 00000096h, Date: 02.05.2018
3	00000000FFD88000h	00018000h	0100h	00h	Microcode	CPUID: 000906EBh, Revision: 0000008Eh, Date: 24.03.2018
4	00000000FFDA0000h	00017800h	0100h	00h	Microcode	CPUID: 000906ECh, Revision: 00000084h, Date: 19.02.2018
5	00000000FFF10000h	00008000h	0100h	00h	BIOS ACM	LocalOffset: 00000018h, EntryPoint: 00003BD1h, ACM SVN: 0000h, Date: 09.02.2017
6	00000000FFFCDC80h	00000000h	0100h	00h	BootGuard Key Manifest	
7	00000000FFFC0000h	00000000h	0100h	00h	BootGuard Boot Policy	



# Firmware Interface Table (FIT)

✓Intel image		Image	Intel
Descriptor region		Region	Descriptor
GbE region		Region	GbE
ME region		Region	ME
✓BIOS region		Region	BIOS
>FA4974FC-AF1D-4E5D-BDC5-DACD6D27BAEC	NVRAM	Volume	FFSv2
>FA4974FC-AF1D-4E5D-BDC5-DACD6D27BAEC		Volume	FFSv2
Padding		Padding	Empty (0xFF)
>4F1C52D3-D824-4D2A-A2F0-EC40C23C5916	DXE	Volume	FFSv2
✓>AFDD39F1-19D7-4501-A730-CE5A27E1154B	FVDATA	Volume	FFSv2
Pad-file		File	Pad
B52282EE-9B66-44B9-B1CF-7E5040F787C1		File	Raw
Pad-file			
>Microcode			
Pad-file			
BiosAc			
Volume free space			
>14E428FA-1A12-4875-B637-8B3CC87FDF07	PEI		
>61C0F511-A691-4F54-974F-B9A42172CE53	PEI + SEC		
OxFFFFF0C0			





# Firmware Interface Table (FIT)

- Is a required element for Intel 64 architecture since introduction of Boot Guard technology
- Can point to microcode update (MCU) binaries
- CPU can load microcode updates from FIT prior to execution of BIOS and before starting Intel Boot Guard

5	00000000FFDD6EB0h	00012C00h	0100h	00h	Microcode	CPUID: 000506E2h, Revision: 0000002Ch, Date: 01.07.2015
6	00000000FFDE9AB0h	00017800h	0100h	00h	Microcode	CPUID: 000806E9h, Revision: 00000042h, Date: 02.10.2016
7	00000000FFE012B0h	00017800h	0100h	00h	Microcode	CPUID: 000906E9h, Revision: 00000042h, Date: 02.10.2016
8	00000000FFE18AB0h	00017800h	0100h	00h	Microcode	CPUID: 000506E8h, Revision: 00000034h, Date: 10.07.2016
9	00000000FFFE0000h	00008000h	0100h	00h	BIOS ACM	LocalOffset: 00000018h, EntryPoint: 00003BB1h, ACM SVN: 0002h, Date: 07.02.2016
10	00000000FFFD4C80h	00000241h	0100h	00h	BootGuard Key Manifest	LocalOffset: 00000018h, KM Version: 10h, KM SVN: 00h, KM ID: 0Fh
11	00000000FFFD3C00h	000002DFh	0100h	00h	BootGuard Boot Policy	LocalOffset: 00000018h, BP SVN: 00h, ACM SVN: 02h



# Microcode Update binary main header

Microcode Update binary starts with the main header followed by an extended header and update data

```
typedef struct MICROCODE_UPDATE_HEADER {
    unsigned long header_version;        // 1
    unsigned long update_revision;
    unsigned long date;                  // BCD format
    unsigned long processor_signature;    // CPUID
    unsigned long checksum;
    unsigned long loader_revision;
    unsigned long processor_flags;
    unsigned long data_size;             // in bytes
    unsigned long total_size;            // in bytes
    unsigned char reserved[0x0C];
};
```

0000h:	01 00 00 00	A6 00 00 00	16 20 21 08	E3 06 05 00	....!....!.ã...
0010h:	1F 67 51 E9	01 00 00 00	36 00 00 00	D0 7B 01 00	.gQé.....6...Đ{...
0020h:	00 7C 01 00	00 00 00 00	00 00 00 00	00 00 00 00	. .....
0030h:	00 00 00 00	A1 00 00 00	01 00 02 00	A6 00 00 00	....i..... ...
0040h:	06 00 00 00	51 5E 00 00	21 08 16 20	E1 17 00 00	....Q^...!... á...
0050h:	01 00 00 00	E3 06 05 00	00 00 00 00	00 00 00 00	....ã.....
0060h:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
0070h:	00 00 00 00	36 00 00 00	04 00 00 00	00 00 00 00	....6.....
0080h:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
0090h:	C2 F3 14 67	67 C0 32 94	6F 7C DF 6D	49 3C EB C3	Âó.ggÀ2"o BmI<ëÃ
00A0h:	DC 5B 0D 20	E5 63 FA 8C	58 7A 7E A4	A2 6F 80 D4	Ů[. âcúEXz~HcoœÔ
00B0h:	A7 A0 92 D1	11 94 5F 4F	6B 9C AF E4	62 CE 00 7A	\$ 'Ñ."_Okæ-abî.z
00C0h:	A4 42 6D 81	7B 27 3E 07	03 63 AE B7	67 0D 63 9B	Hm.{ ' > ..c@·g.c>
00D0h:	C3 5F 0E C2	40 31 58 20	37 39 FC 12	9C B6 B6 C6	Ã_.Ã@1X 79ü.œqgÆ
00E0h:	6C 17 2B 8D	AC 09 2F E3	FC EC 36 2C	64 32 A8 A9	l.+.-./ãüi6,d2"©
00F0h:	99 B6 35 1F	AA E5 B8 14	CA 21 12 7F	B0 0B 41 F5	mq5. *ð.Ê!...°Að



# Microcode Update binary extended header

```
typedef struct MICROCODE_UPDATE_EXTENDED_HEADER {
    unsigned short module_type;           // 0
    unsigned short module_subtype;        // 0
    unsigned long header_size;             // in dwords
    unsigned long header_version;         // 0x20001
    unsigned long update_revision;
    unsigned long unknown[2];
    unsigned long date;                   // BCD format
    unsigned long update_size;            // in dwords
    unsigned long svn;
    unsigned long processor_signature;
    unsigned long unknown2[0x0E];
    unsigned char update_hash[0x20];      // SHA256 hash of the decrypted update data
    unsigned char rsa_mod[0x100];         // RSA 2048 public key modulus
    unsigned long rsa_exp;                 // RSA 2048 public key exponent
    unsigned char signature[0x100];       // RSA 2048 signature of the header
};
```

0000h:	01 00 00 00	A6 00 00 00	16 20 21 08	E3 06 05 00	.... ....!.ă...
0010h:	1F 67 51 E9	01 00 00 00	36 00 00 00	D0 7B 01 00	.gQé....6...Đ{..
0020h:	00 7C 01 00	00 00 00 00	00 00 00 00	00 00 00 00	. .....
0030h:	00 00 00 00	A1 00 00 00	01 00 02 00	A6 00 00 00	....i..... ...
0040h:	06 00 00 00	51 5E 00 00	21 08 16 20	E1 17 00 00	....Q^...!.. á...
0050h:	01 00 00 00	E3 06 05 00	00 00 00 00	00 00 00 00	....ă.....
0060h:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
0070h:	00 00 00 00	36 00 00 00	04 00 00 00	00 00 00 00	....6.....
0080h:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
0090h:	C2 F3 14 67	67 C0 32 94	6F 7C DF 6D	49 3C EB C3	Âó.ggÀ2"o BmI<ěĂ
00A0h:	DC 5B 0D 20	E5 63 FA 8C	58 7A 7E A4	A2 6F 80 D4	Ü[. âcúEXz~ꝛcoEO
00B0h:	A7 A0 92 D1	11 94 5F 4F	6B 9C AF E4	62 CE 00 7A	\$ 'Ñ."_Okœābî.z
00C0h:	A4 42 6D 81	7B 27 3E 07	03 63 AE B7	67 0D 63 9B	ꝛBm.{'>..c@·g.c>
00D0h:	C3 5F 0E C2	40 31 58 20	37 39 FC 12	9C B6 B6 C6	Ă_.Ă@1X 79ü.œŧŧE
00E0h:	6C 17 2B 8D	AC 09 2F E3	FC EC 36 2C	64 32 A8 A9	1.+.-./ăüi6,d2"©
00F0h:	99 B6 35 1F	AA F5 B8 14	CA 21 12 7F	B0 0B 41 F5	ꝛŧ5.âö..Ê!...°.Aö



# Microcode Update binary data

---

- The main part in MCU binary is Data (encrypted, the decryption key is hardcoded into CPU)
- Hash of RSA public key to authenticate the MCU is also hardcoded into CPU
- So no one knows exactly what Microcode is capable of



# Known facts about Microcode

---

- Implements instructions
- Configures the execution logic on the line (that's how side-channels are fixed)
- Implements some startup behavior (like FIT parsing)
- Loads MCU from FIT
- Loads and executes Intel Authenticated Code Modules (ACMs) (from FIT or not)



# Authenticated Code Modules (ACMs)

---

- Signed and sometimes encrypted Intel code modules
  - Loaded and executed from L3 cache (sometimes called AC RAM)
  - Serve as a Root-of-Trusts and a core of implementation for technologies:
    - Intel Boot Guard  
type/subtype: 2.3, **not encrypted**, signed with **KEY<sub>2</sub>**
    - Intel Trusted Execution Technology (TXT)  
type/subtype: 2.0, **not encrypted**, signed with **KEY<sub>3</sub>**
    - Intel BIOS Guard (PFAT)  
type/subtype: 1.1, **encrypted**, signed with **KEY<sub>1</sub>**
- FYI: microcode update binary  
type/subtype: 1.0, **encrypted**, signed with **KEY<sub>1</sub>**



# Useful links to start digging

---

- Docs:
  - [Intel 64 Software Developer's manual](#)
  - leaked Intel confidential documentation
- Papers:
  - [Security Analysis of x86 Processor Microcode](#) by Daming D. Chen and Gail-Joon Ahn
  - [Reverse Engineering x86 Processor Microcode](#) by Benjamin Kollenda and Philipp Koppe, Ruhr
- Tools
  - [UEFItool](#) by CodeRush
  - [MCExtractor](#) by platomav



# Downgrading microcode

---



# Updating Microcode in UEFI BIOS

---

- Updates are to improve stability, performance and apply security fixes
- Updates should be loaded each time CPU is powered on, this means after S<sub>3</sub> (Sleep) / S<sub>4</sub> (Hibernation) / S<sub>5</sub> (Shutdown) modes
- Far not always updates can be loaded by CPU from FIT
- Updates that requires something special (like initialized DRAM) has to be loaded by the BIOS as early as possible from the moment conditions are satisfied
- Updates should be loaded on each CPU core separately



# Microcode Update loading process

---

## update\_microcode:

```
mov rcx, 79h      ; IA32_BIOS_UPDATE_TRIGGER in RCX
xor rax, rax      ; clear RAX
xor rbx, rbx      ; clear RBX
mov rax, MicrocodeUpdate ; Linear address of the microcode update
add rax, 48h      ; Offset of Update Data in the Update
xor rdx, rdx      ; Zero RDX
wrmsr             ; trigger the microcode update
```

## check\_update\_revision:

```
mov rcx, 08bh     ; IA32_BIOS_SIGN_ID
rdmsr             ; read MSR, Update Revision will be in RDX
```



# Normal Boot. Step 1. CpuPei

---

```
// Find the appropriate MCU in FIT
MicrocodeAddr = FindMCUinFIT ();

if (MicrocodeAddr != NULL) {
    MicrocodeSize = ((MICROCODE_UPDATE_HEADER *) MicrocodeAddr)->TotalSize;

    // Copy the MCU from the mapped SPI flash memory into RAM
    Status = (*PeiServices)->AllocatePages ( ... , EFI_SIZE_TO_PAGES (MicrocodeSize), &MicrocodeBuffer);
    if (!EFI_ERROR (Status)) {
        (*PeiServices)->CopyMem (MicrocodeBuffer, MicrocodeAddr, MicrocodeSize);

        // Save this pointer into a HOB
        Status = (*PeiServices)->CreateHob ( ... , &UcodeHob);
        if (!EFI_ERROR (Status)) {
            AmiUcodeHobGuid = EFI_GUID ("94567C6F-F7A9-4229-1330-FE11CCAB3A11");
            memcpy (&UcodeHob->EfiHobGuidType.Name, &AmiUcodeHobGuid, sizeof(EFI_GUID));

            UcodeHob->UcodeAddr = MicrocodeBuffer;
        }
    }
}
```



## Normal Boot. Step 2. PlatformInit

---

- Later the microcode update loader finds this HOB
- Retrieves the MCU buffer address
- Updates CPU microcode with it



# Normal Boot. Step 3. CpuSpSmi

---

```
// Find the MCU HOB and retrieve a saved MCU address
UcodeHob = (AMI_UCODE_HOB *) GetEfiConfigurationTable (pSystemTable, &HobListGuid);

if (UcodeHob != NULL) {
    Status = FindNextHobByGuid (&gAmiUcodeHobGuid, &UcodeHob);

    if (Status == EFI_SUCCESS && UcodeHob->UcodeAddr != NULL && UcodeHob->UcodeAddr != 0xFFFF) {
        gMicrocodeStart = UcodeHob->UcodeAddr;
    }
}

...

// Copy the applied MCU into SMRAM (to protect it from being replaced by OS)
if (gMicrocodeStart != NULL && ((MICROCODE_UPDATE_HEADER *) gMicrocodeStart)->HeaderVersion == 1) {
    UcodeSize = ((MICROCODE_UPDATE_HEADER *) gMicrocodeStart)->TotalSize;

    Status = pSmst->SmmAllocatePages ( ... , EFI_SIZE_TO_PAGES (UcodeSize), &SmramUcodeAddr);
    if (!EFI_ERROR (Status)) {
        memcpy (SmramUcodeAddr, gMicrocodeStart, UcodeSize);
    }
}
```



# Normal Boot. Step 3. CpuSpSmi

---

```
gIntUcodeVarGuid = EFI_GUID ("eda41d22-7729-5b91-b3ee-ba619921cefa");
```

```
...
```

```
// Save its address into the 'IntUcode' EFI variable
```

```
IntUcodeVarData.Version = 1;
```

```
IntUcodeVarData.UcodeAddr = SmmUcodeAddr;
```

```
IntUcodeVarData.Unknown = 0;
```

```
IntUcodeVarData.Unknown2 = 0;
```

```
Status = pRuntimeServices->SetVariable (L"IntUcode", &gIntUcodeVarGuid,  
                                         EFI_VARIABLE_NON_VOLATILE |  
                                         EFI_VARIABLE_BOOTSERVICE_ACCESS |  
                                         EFI_VARIABLE_RUNTIME_ACCESS,  
                                         sizeof(IntUcodeVarData), &IntUcodeVarData);
```



# Waking from S3. Step 1. CpuPei

---

```
// Instead of searching for the MCU again, get the pointer from 'IntUcode' EFI variable
Status = ReadOnlyVariable2->GetVariable ( ... , "IntUcode", & gIntUcodeVarGuid, NULL,
                                          &VarSize, &IntUcodeVarData);

if (!EFI_ERROR (Status)) {
    MicrocodeAddr = IntUcodeVarData.UcodeAddr;

    Status = (*PeiServices)->CreateHob ( ... , &UcodeHob);
    if (!EFI_ERROR (Status)) {
        AmiUcodeHobGuid = EFI_GUID ("94567C6F-F7A9-4229-1330-FE11CCAB3A11");
        memcpy (&UcodeHob->EfiHobGuidType.Name, &AmiUcodeHobGuid, sizeof(EFI_GUID));

        UcodeHob->uCodeAddr = MicrocodeAddr;
    }
}
```



## Waking from S3. Step 2. PlatformInit

---

- Later the microcode update loader finds this HOB
- Retrieves the MCU buffer address
- Updates CPU microcode with it



# Microcode Downgrade

---

This specific allows an attacker:

- to load an old microcode update capsule into memory
- make the 'IntUcode' EFI variable to point to it
- perform Sleep/Wake-up cycle

The system will be booted with the attacker-provided microcode (if it was valid and passed the integrity check, of course)



# Microcode Downgrade

---

- 2019 version of MCU of CPU ID 0x806EA

CPU ID	N/A	000806EA
PATCH ID	N/A	00000096

- Downgraded to 2018 version

CPU ID	N/A	000806EA
PATCH ID	N/A	00000084



# Discovering impact

---



# Side channel attacks

---

- Get rid of fixes (side channel attacks)
- Most of these attacks – extremely hard to apply in the wild
- Have never been spotted, however there's not much of detection tools:
  - [SCADET](#) by Majid Sabbagh
- [Introduction to software-based microarchitectural side-channel attacks](#) by Alexander Romyantsev
- [A New Memory Type Against Speculative Side-Channel Attacks](#) by [@IntelSTORM](#)



# Debug capabilities

---

- Unlock debug capabilities
- Get rid of [INTEL-SA-00073](#) fix (CVE-2017-5684)
- [Intel DCI Secrets](#) by Maxim Goryachy and Mark Ermolov



# Downgrading ACMs

- The ACM authentication is performed by a Microcode
- Older Microcode versions load older ACM (with reduced SVN)
- Downgraded ACM has exploitable 1days which makes vulnerable the technology they support

<https://twitter.com/matrosov/status/1139491430110584832>

**Alex Matrosov**  
@matrosov

Follow

Intel microcode downgrade is a huge supply-chain problem. Even after the patch problem still exists in many platforms. Btw ACM's downgrade is also possible (a bit more tricky but downgrade both Microcode + ACM is a key to success).  
Great job @flothrone and the team!

**Alexander Ermolov** @flothrone  
Our team (@ttbr0 , @undermarble and me) walks through UEFI BIOS again, as a result:  
- 6 Escalation of Privileges to SMM  
- microcode downgrade vulnerability, allowing to bypass hardware root-of-trusts.  
Details coming soon!  
[Show this thread](#)

4:15 AM - 14 Jun 2019



# Downgrading ACMs. Intel Boot Guard

---

- Not encrypted, binary diffing is applicable to find 1 days
- Executed only on startup (prior to BIOS) upon CPU is powered on and released from the RESET state
- ACM does not verify BIOS when waking from S<sub>3</sub> (performance optimizations) except each 12 boot

The implementation of vendor part of trusted boot is a target here.  
Plenty of techniques are already in public



# Downgrading ACMs. Intel BIOS Guard

---

- Encrypted, extremely hard to find a fixed issue
- Triggered to run SPI flash operations via CPU MSRs from SMM
- Downgrade is possible if SPI flash write access is gained (at which point further attack is unnecessary)

First bypass is already in public:

[Breaking Through Another Side: Bypassing Firmware Security Boundaries from Embedded Controller](#) by Alex Matrsov



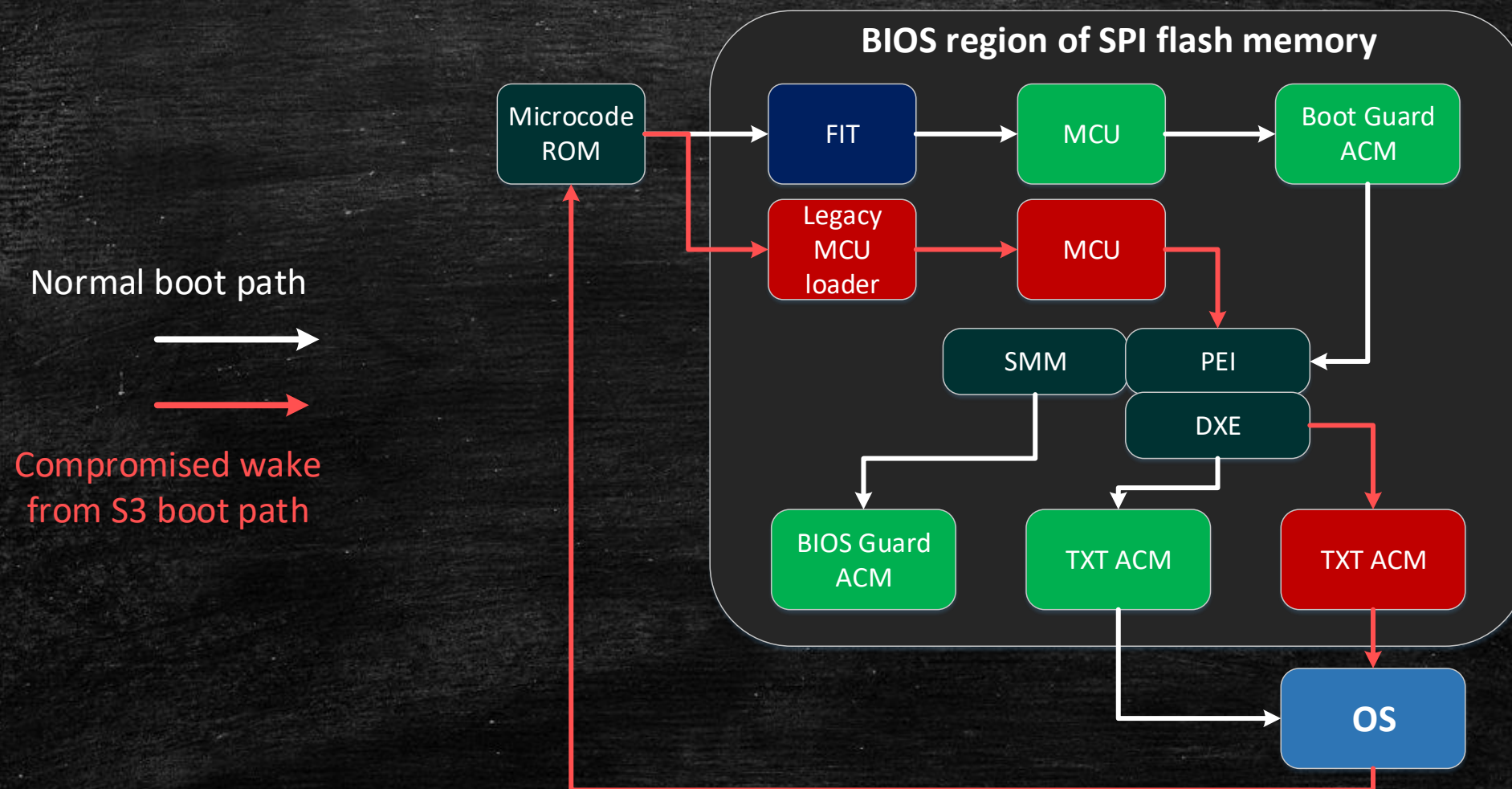
# Downgrading ACMs. Intel TXT

---

- Not encrypted, binary diffing is applicable to find a 1 days
- SINIT ACM is a target
- Triggered via GETSEC instruction from BIOS / OS to measure boot chain components
- Address of this ACM is specified in EBX register
- Address doesn't change from boot to boot, so downgrade is possible just by replacing this ACM in memory!
- [INTEL-SA-00035](#)



# Downgrading ACMs. Intel TXT





# #Report and Reaction

---

- Reported to Intel on 3<sup>rd</sup> July 2018
- Confirmed as a valid issue on 28 August 2018
- [INTEL-SA-00264](#) on 11 June 2019
- Affected: AMI-based UEFI BIOS for Intel hardware (since ~2014)

Would like to thank Intel PSIRT and AMI for resolving this issue



# #Mitigations

---

- Intel SGX
  - does not check MCU SVN when leaving S3
- Protect 'IntUcode' EFI variable (mark as read-only and close from runtime access)
  - Could be bypassed if an attacker manages to run arbitrary code in SMM
- Make an OS to update the Microcode to the latest version
  - Process could be already compromised at the moment of validating the update version
- Supply only the updates which could be loaded from FIT



# #Takeaways

---

- Supply chain problem
- The problem in a basic component compromises all technologies it serves as a Root-of-Trust
- The full impact is yet to discover



Thank you

---