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

Alexander Ermolov
@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 explosion
# 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, MOV$S$ 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
```
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.
**Firmware Interface Table (FIT)**

<table>
<thead>
<tr>
<th>Intel Image</th>
<th>Image</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptor region</td>
<td>Region</td>
<td>Descriptor</td>
</tr>
<tr>
<td>GBE region</td>
<td>Region</td>
<td>GBE</td>
</tr>
<tr>
<td>ME region</td>
<td>Region</td>
<td>ME</td>
</tr>
<tr>
<td>BIOS region</td>
<td>Region</td>
<td>BIOS</td>
</tr>
<tr>
<td>NVRAM region</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
<tr>
<td>Padding</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
<tr>
<td>DXE</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
<tr>
<td>FVDATA</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
<tr>
<td>Pad-file</td>
<td>File</td>
<td>Pad</td>
</tr>
<tr>
<td>B52282EE-9B66-44B9-B1CF-7E5040F787C1</td>
<td></td>
<td>Raw</td>
</tr>
<tr>
<td>Pad-file</td>
<td>Microcode</td>
<td></td>
</tr>
<tr>
<td>Pad-file</td>
<td>Pad-file</td>
<td></td>
</tr>
<tr>
<td>BIOS</td>
<td>BIOS</td>
<td></td>
</tr>
<tr>
<td>Volume free space</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
<tr>
<td>14E428FA-1A12-4B85-4B637-8B3CC87FDF07</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
<tr>
<td>61C0F511-A691-4F54-974F-89A42712CE53</td>
<td>Volume</td>
<td>FFSSv2</td>
</tr>
</tbody>
</table>

**Hex view:** 0xFFFFFC0

<table>
<thead>
<tr>
<th>FIT</th>
<th>xF</th>
<th>yF</th>
<th>zF</th>
<th>aF</th>
<th>bF</th>
<th>cF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>5F 46 49 54 5F 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0010</td>
<td>00 04 07 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0020</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0030</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0040</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0050</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0060</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0070</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Notes:**
- FIT stands for Firmware Interface Table.
- The image shows a section of the FIT table with various fields such as Intel Image, Descriptor region, GBE region, ME region, and BIOS region.
- The table includes fields like NVRAM, DXE, FVDATA, Pad-file, Microcode, Pad-file, BIOS, Volume free space, and FFSSv2 volume.
- The hex view at the bottom provides a detailed representation of the FIT data.
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
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];
};
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
};
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
- Serve as a Root-of-Trust and a core of implementation for technologies:
  - Intel Boot Guard
  - Intel BIOS Guard (PFAT)
  - Intel Trusted Execution Technology (TXT)
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
  – MCEExtractor 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 S3 (Sleep) / S4 (Hibernation) / S5 (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 (SmmUcodeAddr, 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

<table>
<thead>
<tr>
<th>CPU ID</th>
<th>N/A</th>
<th>000806EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATCH ID</td>
<td>N/A</td>
<td>00000096</td>
</tr>
</tbody>
</table>

- Downgraded to 2018 version

<table>
<thead>
<tr>
<th>CPU ID</th>
<th>N/A</th>
<th>000806EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATCH ID</td>
<td>N/A</td>
<td>00000084</td>
</tr>
</tbody>
</table>
Discovering impact
Side channel attacks

- Get rid of fixes (side channel attacks)
- Most of them – 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 Rumyantsev
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 1day which makes vulnerable the technology they support

https://twitter.com/matrosov/status/1139491430110584832
Downgrading ACMs. Intel Boot Guard

- Not encrypted, binary differencing is applicable to find 1-day exploits.
- 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 S3 (performance optimizations) except each 12 boot.

The implementation of vendor-provided 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 BIOS / OS
- 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
#Report and Reaction

- Reported to Intel on 3rd July 2018
- Confirmed as a valid issue on 28 August 2018
- **INTEL-SA-00264** on 11 June 2019

- 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