

Untrusted Roots: exploiting vulnerabilities in Intel ACMs

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

Dmitry Frolov
[@allowitsme](https://twitter.com/allowitsme)

#WhoAmI

- Alexander Ermolov
- [ZeroNights](#) security conference
- Intel Management Engine
 - [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](#)
 - [Microcode downgrade](#)



#WhoAmI

Dmitry Frolov - security researcher at Digital Security



Has a major experience in UEFI firmware researching and vulnerability hunting (over 15 CVEs for Intel systems).

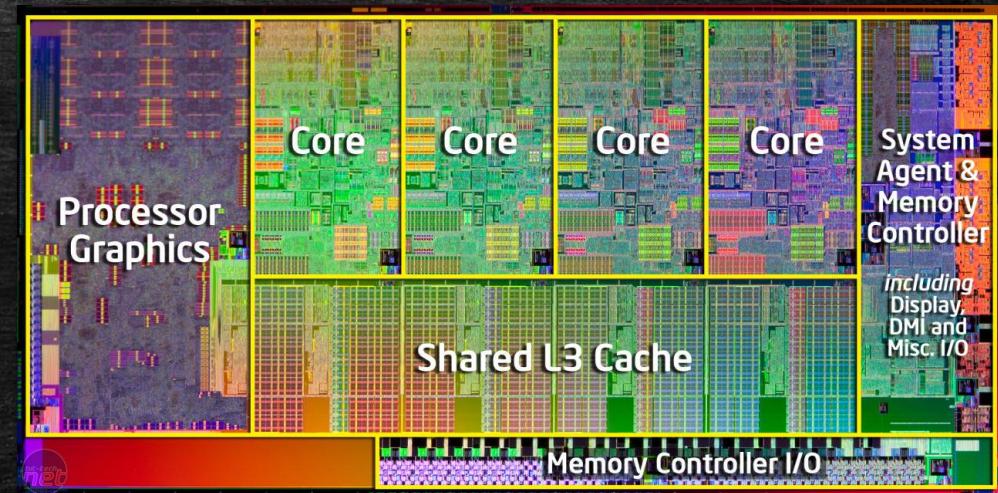
#Agenda

- Intel CPU Roots of Trust overview
 - Intel Microcode
 - FIT
 - Intel ACMS
- Intel BIOS Guard ACM
- Takeaways

Intel CPU Roots of Trust overview

Inside Intel CPU

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



Microcode

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

Microcode

Dumped example of CPUID instruction implementation:

```
4760: 00050003b800      tmp11:= SUB_DSZ32(0x00000000, rax)
4761: 0fef02000000      GENARITHFLAGS(0x00000002)
4762: 100a80000200      MERGEFLGS(0x00000080)
-----2-4e80-00-09-----
4764: 100a40800200      MERGEFLGS(0x00000040)
-----0-4776-00-06-----
4765: 006311031200      tmp1:= READURAM(0x0011, 64)
4766: 0e6510072371      tmp2:= LDPPHYSTICKLE_DSZ64_ASZ64_SC1(tmp1, 0x00000510, mode=01)

4768: 006520031232      tmp1:= SHR_DSZ64(tmp2, 0x00000020)
4769: 00161f03323b      tmp3:= BTR_DSZ32(tmp11, 0x0000001f)
476a: 013e00032cf2      tmp2:= SUB1AND_DSZ32(tmp2, tmp3)
-----3-0000-00-03-----
476c: 003700032c72      tmp2:= CMOVCC_DSZ32_CONDNB(tmp2, tmp1)
476d: 00251f031232      tmp1:= SHR_DSZ32(tmp2, 0x0000001f)
476e: 002401034cc8      tmp4:= SHL_DSZ32(0x00000001, tmp3)
-----3-0000-00-03-----
4770: 000400034cb4      tmp4:= AND_DSZ32(tmp4, tmp2)
4771: 00071f032cc8      tmp2:= NOTAND_DSZ32(0x0000001f, tmp3)
4772: 013e00032cb1      tmp2:= SUB1AND_DSZ32(tmp1, tmp2)
-----3-0000-00-03-----
4774: 013500032d32      tmp2:= CMOVCC_DSZ32_CONDNZ(tmp2, tmp4)
4775: 2929903c0032      CMPUJNZ DIRECT NOTTAKEN(tmp2, 0x00000000, 0x1f90)
```

Undocumented behaviour of CPUID instruction
https://twitter.com/_markel_/status/1324848756915920898?s=20

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 00000000FFFDA000h	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 00000000FFFCCC00h	00000000h	0100h	00h	BootGuard Boot Policy	

A collage of news articles and headlines about CPU bugs. Headlines include "Skylake bug causes Intel chips to freeze under 'complex workloads'", "Intel finds specialized TSX enterprise bug on Haswell, Broadwell CPUs", and "The Meltdown and Spectre CPU Bugs, Explained". The articles are from various sources like PCWorld and Ars Technica, dated from January 2016 to August 2014.

Firmware Interface Table (FIT)

Intel image
Descriptor region
GbE region
ME region
✓BIOS region
➤FA4974FC-AF1D-4E5D-BDC5-DACD6D27BAEC NVRAM
➤FA4974FC-AF1D-4E5D-BDC5-DACD6D27BAEC
Padding
➤4F1C52D3-D824-4D2A-A2F0-EC40C23C5916 DXE
✓AFDD39F1-19D7-4501-A730-CE5A27E1154B FVDATA
Pad-file
B52282EE-9B66-44B9-B1CF-7E5040F787C1
Pad-file
➤Microcode
Pad-file
BiosAc
Volume free space
➤14E428FA-1A12-4875-B637-8B3CC87FDF07 PEI
➤61C0F511-A691-4F54-974F-B9A42172CE53 PEI + SEC
0xFFFFFFF0

Image	Intel
Region	Descriptor
Region	GbE
Region	ME
Region	BIOS
Volume	FFSv2
Volume	FFSv2
Padding	Empty (0xFF)
Volume	FFSv2
Volume	FFSv2
File	Pad
File	Raw

Hex view: B52282EE-9B66-44B9-B1CF-7E5040F787C1

0000	5F 46 49 54 5F 20 20 20 07 00 00 00 00 00 01 00 00	_FIT_
0010	00 00 04 D7 FF 00 00 00 00 00 00 00 00 00 00 01 01 00	..xy..
0020	00 80 D8 FF 00 00 00 00 00 00 00 00 00 00 01 01 00	.x0y..
0030	00 00 DA FF 00 00 00 00 00 00 00 00 00 00 01 01 00	..Uy..
0040	00 00 F1 FF 00 00 00 00 00 00 00 00 00 00 01 02 00	..ny..
0050	80 DC FC FF 00 00 00 00 00 00 00 00 00 01 0B 00	xÜy..
0060	00 CC FC FF 00 00 00 00 00 00 00 00 00 01 0C 00	.Iüy..

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	00000000FFFFE0000h	00008000h	0100h	00h	BIOS ACM	LocalOffset: 00000018h, EntryPoint: 00003BB1h, ACM SVN: 0002h, Date: 07.02.2016
10	00000000FFF4D4C80h	00000241h	0100h	00h	BootGuard Key Manifest	LocalOffset: 00000018h, KM Version: 10h, KM SVN: 00h, KM ID: 0Fh
11	00000000FFF3C00h	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 !.å...
0010h:	1F 67 51 E9	01 00 00 00	36 00 00 00	D0 7B 01 00	.gQé....6...D{...
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 00i.....
0040h:	06 00 00 00	51 5E 00 00	21 08 16 20	E1 17 00 00Q^...!.. á...
0050h:	01 00 00 00	E3 06 05 00	00 00 00 00	00 00 00 00ä.....
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 00 006.....
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	Áó.ggÀ2"o BmI<ëÃ
00A0h:	DC 5B 0D 20	E5 63 FA 8C	58 7A 7E A4	A2 6F 80 D4	Ü[. ácúGXz~¤¤€Ó
00B0h:	A7 A0 92 D1	11 94 5F 4F	6B 9C AF E4	62 CE 00 7A	S 'Ñ._Okœ~äbÍ.z
00C0h:	A4 42 6D 81	7B 27 3E 07	03 63 AE B7	67 0D 63 9B	¤Bm.{'>..c®.g.c}
00D0h:	C3 5F 0E C2	40 31 58 20	37 39 FC 12	9C B6 B6 C6	Ä_.Ä@1X 79ü.¤¶¶¶
00E0h:	6C 17 2B 8D	AC 09 2F E3	FC EC 36 2C	64 32 A8 A9	1.+.-./äüì6,d2"©
00F0h:	99 B6 35 1F	AA E5 B8 14	CA 21 12 7F	B0 0B 41 F5	¶¶¶5. "ö .È! .. ° Äô

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!....!..ä... .gQé.....6...D...
0010h:	1F 67 51 E9	01 00 00 00	36 00 00 00	D0 7B 01 00
0020h:	00 7C 01 00	00 00 00 00	00 00 00 00	00 00 00 00i.....
0030h:	00 00 00 00	A1 00 00 00	01 00 02 00	A6 00 00 00Q^...!..á...ä.....
0040h:	06 00 00 00	51 5E 00 00	21 08 16 20	E1 17 00 006.....
0050h:	01 00 00 00	E3 06 05 00	00 00 00 00	00 00 00 00	Äó.ggÀ2"o ßmI<ëÄ Ü[. ácúEXz~¤co€Ö
0060h:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	\$ 'Ñ."_Okœ~äbí.z #Bm.('>..c@.g.c>
0070h:	00 00 00 00	36 00 00 00	04 00 00 00	00 00 00 00	Ä_.Ä@IX 79ü.œ¶¶E 1.+.-./ääü6,d2"©
0080h:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	“¶¶5. “ö .È!..°.Að
0090h:	C2 F3 14 67	67 C0 32 94	6F 7C DF 6D	49 3C EB C3	
00A0h:	DC 5B 0D 20	E5 63 FA 8C	58 7A 7E A4	A2 6F 80 D4	
00B0h:	A7 A0 92 D1	11 94 5F 4F	6B 9C AF E4	62 CE 00 7A	
00C0h:	A4 42 6D 81	7B 27 3E 07	03 63 AE B7	67 0D 63 9B	
00D0h:	C3 5F 0E C2	40 31 58 20	37 39 FC 12	9C B6 B6 C6	
00E0h:	6C 17 2B 8D	AC 09 2F E3	FC EC 36 2C	64 32 A8 A9	
00F0h:	99 B6 35 1F	AA E5 B8 14	CA 21 12 7F	B0 0B 41 F5	

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
- Recently it was dumped decrypted, however format is yet to be researched
 - <https://github.com/chip-red-pill>
 - by Dmitry Sklyarov (@_Dmit), Mark Ermolov (@_markel____), Maxim Goryachy (@h0t)

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 KEY2
 - Intel Trusted Execution Technology (TXT)
 - type/subtype: 2.0, **not encrypted**, signed with KEY3
 - Intel BIOS Guard (PFAT)
 - type/subtype: 1.1, **encrypted**, signed with KEY1

FYI: microcode update binary

type/subtype: 1.0, **encrypted**, signed with KEY1

Intel ACM header

```
typedef struct ACM_HEADER {
    unsigned short module_type;
    unsigned short module_subtype;
    unsigned long header_size;           // in dwords
    unsigned long header_version;
    unsigned short chipset_id;
    unsigned short flags;
    unsigned long module_vendor;         // 0x8086
    unsigned long date;                 // BCD format
    unsigned long module_size;           // in dwords
    unsigned short acm_svn;
    unsigned short se_svn;

    ...
    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 + data
};

0000 CD A3 13 F0 C7 E2 99 93 25 58 B7 4E
0000 B4 87 4F 4F C7 DC E3 45 24 D8 96 4C
0000 88 AF 63 49 43 98 27 F1 39 24 3F 4B
0100 E2 35 3A 58 37 F0 ED 05 70 1F 05 7D
0110 80 07 F1 A1 AD 52 BA E4 09 64 46 51
0120 63 22 1D C2 FB 5F D2 A6 2D 2A E7 D0
0130 C9 93 F4 90 F5 C7 F4 3E AB C6 B4 5F
0140 86 35 50 D4 8B B3 90 FD 5E BF 45 A0
0150 13 96 12 17 32 CA F1 32 8F 79 CB B0
0160 81 9C DE B2 E0 E5 B1 E8 9A 3B 3B B1
0170 EF 31 F6 63 95 95 64 90 5E DD 6A 71
0180 11 00 00 00 25 08 98 5C 69 BB 4D 70
0190 1D 73 5A 51 0D BF C4 CF AD B0 38 E0
01A0 C9 C5 04 4F 5A 06 09 3A 6C 6F 71 80
01B0 BF 09 7A BD F8 79 71 90 92 A7 8B D0
01C0 D2 35 8A 93 D5 4D 09 C0 1A 7D AE D0
01D0 ED 5E 79 94 C5 1A EB 72 7C 57 16 5A
01E0 AD 4C 9F 4D 91 12 3F 01 B9 65 89 90
```

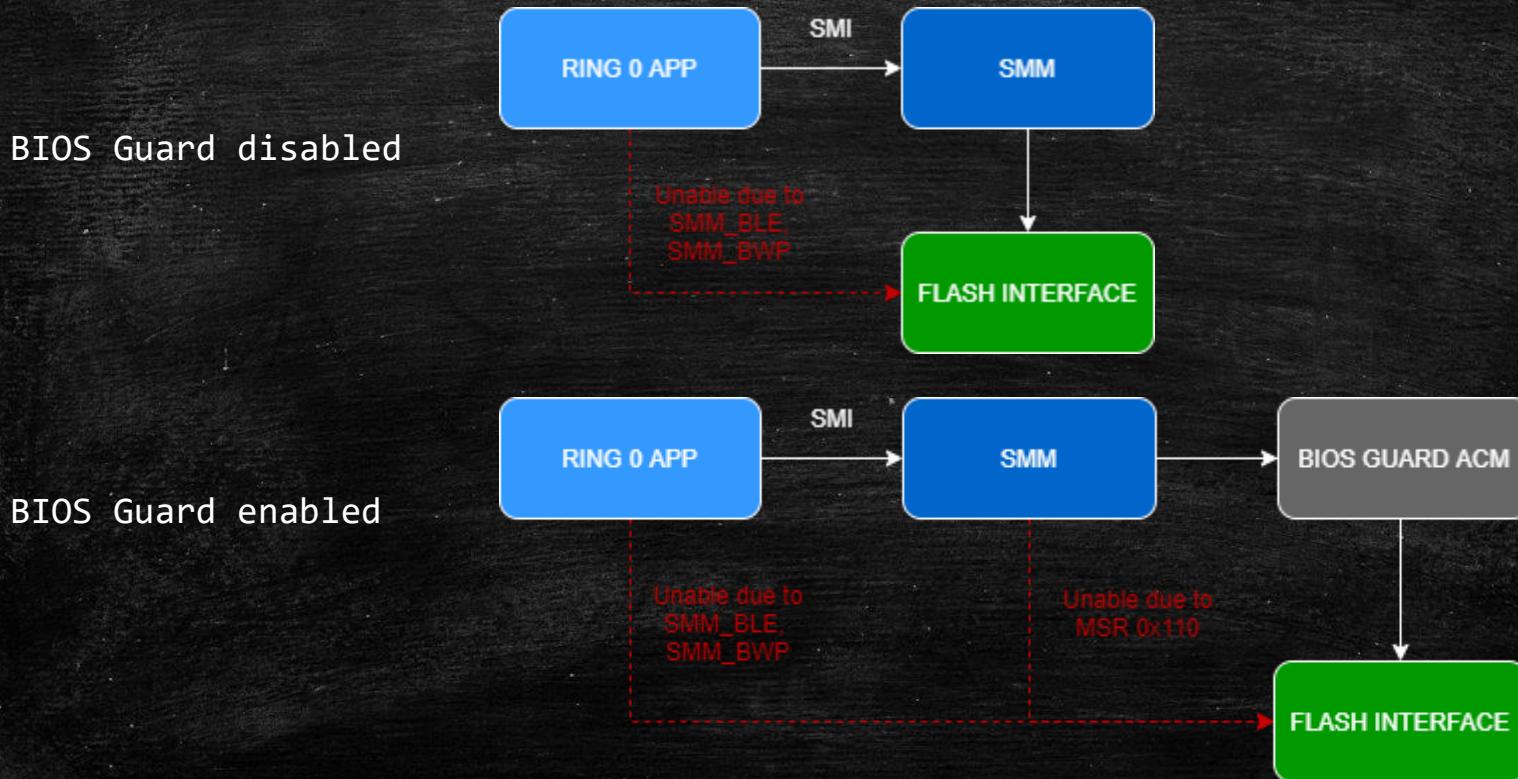
0000	02	00	03	00	A1	00	00	00	00	00	00	00	00	00	00	00
0010	86	80	00	00	30	04	19	20	00	20	00	00	02	00	02	00
0020	00	00	00	00	00	00	00	00	20	00	00	00	98	05	00	00
0030	08	00	00	00	D1	3B	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	40	00	00	00	8F	00	00	00
0080	C7	1A	C1	E2	A4	57	E7	FC	AA	58	55	72	AF	E2	BA	AB
0090	FC	FC	17	BA	FB	C5	EE	D9	71	E1	28	83	A2	68	F7	EA
00A0	6E	2C	97	38	F4	93	D7	F5	97	14	4B	1A	F3	F1	87	15
00B0	68	39	78	3C	50	33	92	C9	20	88	F8	9C	75	BD	BC	43
00C0	0E	9B	A6	3D	E6	89	0C	AC	5F	22	17	79	09	D7	C2	CF
00D0	CD	A3	13	F0	C7	E2	99	93	25	58	B7	40	3B	D1	D2	DF
00E0	B4	87	4F	4F	C7	DC	E3	45	24	D8	96	40	4B	64	FA	1E
00F0	88	AF	63	49	43	98	27	F1	39	24	3F	4B	D6	3A	E2	97
0100	E2	35	3A	58	37	F0	ED	05	70	1F	05	7E	39	B4	F3	BD
0110	80	07	F1	A1	AD	52	BA	E4	09	64	46	5E	1D	04	30	4B
0120	63	22	1D	C2	FB	5F	D2	A6	2D	2A	E7	DB	2F	D4	7F	62
0130	C9	93	F4	90	F5	C7	F4	3E	AB	C6	B4	5D	B2	0E	CC	69
0140	86	35	50	D4	8B	B3	90	FD	5E	BF	45	AF	C3	A7	AF	05
0150	13	96	12	17	32	CA	F1	32	8F	79	CB	BD	0C	96	FC	EE
0160	81	9C	DE	B2	E0	E5	B1	E8	9A	3B	3B	B7	9D	71	67	DD
0170	EF	31	F6	63	95	95	64	90	5E	DD	6A	7F	A7	F7	5A	EE
0180	11	00	00	00	25	0B	98	5C	69	BB	4D	7B	96	9A	88	44
0190	1D	73	5A	51	0D	BF	C4	CF	AD	B0	38	EC	E0	FB	E4	D6
01A0	C9	C5	04	4F	5A	06	09	3A	6C	6F	71	8E	E1	CB	6A	8A
01B0	BF	09	7A	BD	F8	79	71	90	92	A7	8B	DA	64	D9	A1	61
01C0	D2	35	8A	93	D5	4D	09	C0	1A	7D	AE	DB	BC	0E	3A	AE
01D0	ED	5E	79	94	C5	1A	EB	72	7C	57	16	54	6E	C4	43	EB
01E0	AD	4C	9F	4D	91	12	3F	01	B9	65	89	98	A9	51	9A	CB

CPU Roots of Trust inside UEFI BIOS

>4F1C52D3-D824-4D2A-A2F0-EC40C23C5916	Volume	FFSv2	
✓AFDD39F1-19D7-4501-A730-CE5A27E1154B	Volume	FFSv2	
Pad-file	File	Pad	
B52282EE-9B66-44B9-B1CF-7E5040F787C1	File	Raw	FIT
Pad-file	File	Pad	
>Microcode	File	Raw	Microcode Updates
Pad-file	File	Pad	
BiosAc	File	Raw	Intel TXT ACM
Volume free space	Free space		
✓61C0F511-A691-4F54-974F-B9A42172CE53	Volume	FFSv2	
>AprioriPei	File	Freeform	
✓8E295870-D377-4B75-BFDC-9AE2F6DBDE22	File	NET module	PEI apriori file
Pad-Tile	File	Pad	
C30FFF4A-10C6-4C0F-A454-FD319BAF6CE6	File	Raw	
Pad-file	File	Pad	
7C9A98F8-2B2B-4027-8F16-F7D277D58025	File	Raw	
>8E295870-D377-4B75-BFDC-9AE2F6DBDE22	File	Freeform	
7934156D-CFCE-460E-92F5-A07909A59ECA	File	Raw	Intel BIOS Guard ACM
Pad-file	File	Pad	
6520F532-2A27-4195-B331-C0854683E0BA	File	Raw	Intel Boot Guard ACM
>Non-empty pad-file	File	Pad	
TxtPeiAp	File	Raw	
Pad-file	File	Pad	
>Volume Top File	File	SEC core	Volume Top File

Intel BIOS Guard

Why Intel BIOS Guard (BiG)?



Intel BIOS Guard presence indications

- 0xCE PLATFORM_INFO_MSR:
Bit 35: BIOS Guard supported bit

PLATFORM_INFO	0xCE	08080838F1012400
---------------	------	------------------

- 0x110 MSR_PLAT_FRMW_PLOT_CTRL (BIOS_GUARD_CTRL_MSR):
Bit 0: Lock bit
Bit 1: BIOS Guard enable/disable bit

BIOS_GUARD_CTRL	0x110	00000000000000000000000000000001	BIOS_GUARD_CTRL	0x110	00000000000000000000000000000003
-----------------	-------	----------------------------------	-----------------	-------	----------------------------------

This is the only BiG-related MSR accessible from kernel mode. The others are only for SMM, and any attempt to read them from OS will cause MCE (Machine Check Exception).

Intel BIOS Guard presence indications

- The most of BiG support logic implemented in BiosGuardServices {6D4BAA0B-F431-4370-AF19-99D6209239F6}
shared between all IBVs (AMI, Phoenix, Insyde)

BiosGuardServices	File	SMM module	BiosGuardServices
MM dependency section	Section	MM dependency	
PE32 image section	Section	PE32 image	
UI section	Section	UI	
Version section	Section	Version	

- In some old firmwares this module is called PfatServices (can still be seen on Insyde)

Only BIOS Guard Control MSR indicates, if Intel BiG enabled or not!

Intel BIOS Guard Components

- 0 - Encrypted ACM (addr 0x7AE1DD10)
- 1 - BiG Platform Data Table (BGPDT) Empty entry
- 2 - BiG update package (BGUP)
- 3 - BGUP certificate
- 4 - BIOS Guard log

#1: BGPDT		#0: BiG ACM (addr 0x7ADC0000)		#2: BGUP (addr 0x79600000)	
E90	42 47 52 44 00 00 00 00 D0 DF E1 7A 00 00 00 00	BGRD	...Ряб....		
EA0	98 7C E0 7A 00 00 00 00 34 7D E0 7A 00 00 00 00	.jaz....4az....			
EB0	00 7A E0 7A 00 00 00 00 00 00 DC 7A 00 00 00 00	.zaz....bz....			
EC0	10 DD E1 7A 00 00 00 00 01 00 60 79 00 00 00 02	.36z....`y....			
ED0	00 00 00 00 00 00 00 FE 00 00 00 00 00 00 00 FEю....ю....			
EE0	00 00 00 00 00 00 FF 00 00 70 00 00 00 00 00я....я....			
		End marker			

Betraying the BIOS: Where the Guardians of the BIOS are Failing
by Alex Matrosov

Breaking Through Another Side by Alex Matrosov

Intel BIOS Guard ACM

Modifying the external header will not affect BiG ACM execution.

Internal header corrupting will result in a system crash.

External Header (0x80 bytes)	0000 01 00 01 00 00 00 02 00 26 11 17 20 00 00 02 00 0010 04 7A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0030 00 00 00 00 00 00 00 00 00 08 00 40 10 00 00 00 00 0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0080 10 00 00 00 A1 00 00 00 01 00 02 00 00 00 00 02 00 0090 00 00 00 00 00 00 00 00 26 11 17 20 61 1E 00 00 00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00E0 86 9E 3F 65 AC 67 9E CA 01 05 B1 0C D0 3D 50 C6 00F0 72 DF B9 43 DF 4F 3E 15 B7 BD 5D AF A7 5C FF 73 0100 A7 A0 92 D1 11 94 5F 4F 6B 9C AF E4 62 CE 00 7A 0110 A4 42 6D 81 7B 27 3E 07 03 63 AE B7 67 0D 63 9B 0120 C3 5F 0E C2 40 31 58 20 37 39 FC 12 9C B6 B6 C6 0130 6C 17 2B 8D AC 09 2F E3 FC EC 36 2C 64 32 A8 A9 0140 99 B6 35 1F AA F5 B8 14 CA 21 12 7F B0 0B 41 F5 0150 EE AC CC CD 0B CE 8B B0 3B 70 2A 5E 39 E1 46 F1 0160 6A 74 53 7B 5F 47 47 74 82 A3 9A 46 07 AB 06 0F 0170 FC EE 32 E8 8E 87 8C 4E 8F B2 33 A1 FC 8F 8A 70 0180 44 22 63 12 A3 41 33 4B 49 71 F8 3F B1 03 D4 CB 0190 B8 DD 27 81 53 9C F0 78 F4 1A 4C 21 13 5A E8 ED 01A0 FD 5D 32 FE 83 9F 39 2B 70 B2 31 ED 71 08 5E 88 01B0 88 CE 35 D9 3D 64 74 CA 06 D7 23 9B 80 13 B1 DF 01C0 9F 10 8C 1C 86 92 ED 6E E0 60 45 02 7B 4F 83 D0 01D0 9A CB 35 BA 80 49 09 04 F2 3C 97 BD 30 32 07 4C
Internal Header (0x60 bytes)	0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0080 10 00 00 00 A1 00 00 00 01 00 02 00 00 00 00 02 00 0090 00 00 00 00 00 00 00 00 26 11 17 20 61 1E 00 00 00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00E0 86 9E 3F 65 AC 67 9E CA 01 05 B1 0C D0 3D 50 C6 00F0 72 DF B9 43 DF 4F 3E 15 B7 BD 5D AF A7 5C FF 73 0100 A7 A0 92 D1 11 94 5F 4F 6B 9C AF E4 62 CE 00 7A 0110 A4 42 6D 81 7B 27 3E 07 03 63 AE B7 67 0D 63 9B 0120 C3 5F 0E C2 40 31 58 20 37 39 FC 12 9C B6 B6 C6 0130 6C 17 2B 8D AC 09 2F E3 FC EC 36 2C 64 32 A8 A9 0140 99 B6 35 1F AA F5 B8 14 CA 21 12 7F B0 0B 41 F5 0150 EE AC CC CD 0B CE 8B B0 3B 70 2A 5E 39 E1 46 F1 0160 6A 74 53 7B 5F 47 47 74 82 A3 9A 46 07 AB 06 0F 0170 FC EE 32 E8 8E 87 8C 4E 8F B2 33 A1 FC 8F 8A 70 0180 44 22 63 12 A3 41 33 4B 49 71 F8 3F B1 03 D4 CB 0190 B8 DD 27 81 53 9C F0 78 F4 1A 4C 21 13 5A E8 ED 01A0 FD 5D 32 FE 83 9F 39 2B 70 B2 31 ED 71 08 5E 88 01B0 88 CE 35 D9 3D 64 74 CA 06 D7 23 9B 80 13 B1 DF 01C0 9F 10 8C 1C 86 92 ED 6E E0 60 45 02 7B 4F 83 D0 01D0 9A CB 35 BA 80 49 09 04 F2 3C 97 BD 30 32 07 4C

Intel BIOS Guard invoking

- 0x115 MSR - points to BiG components array
- 0x116 MSR - any write will trigger BiG ACM execution

The ACM will run only on BSP, all APs will be put to Sleep mode via invoking this ACM on them the same way.

```
for ( i = 0; i < gSmst->NumberOfCpus; ++i ) {
    if ( i != gSmst->CurrentlyExecutingCpu )
    {
        gSmst->SmmStartupThisAp(
            TriggerBiosGuardAcm,
            i,
            mBiosGuardDirectoryPtr);
    }
}

TriggerBiosGuardAcm(mBiosGuardDirectoryPtr);

...
void TriggerBiosGuardAcm(void *BiosGuardDirectoryPtr)
{
    ...
    AsmWriteMsr64(MSR_PLAT_FRMW_PROT_TRIG_PARAM,
    BiosGuardDirectoryPtr);
    AsmWriteMsr64(MSR_PLAT_FRMW_PROT_TRIGGER, 0);
    ...
}
```

Intel BIOS Guard Verification Flow

Initialization

BGPDT build,
including SFAM
defining

BGPDT's SHA256
put in MSRs
0x111-114

Setting MSR 0x110

RT SMM

BGPDT

BGUP

BGUP certificate

BiG directory pointer
passed to ACM thru
MSR 0x115

BiG ACM

Current BGPDT's
SHA256 compared to
0x111-114 MSR one

Certificate
is present

BGUP certificate gets
validated

Certificate
is absent

Flash writes within SFAM
(defined in BGPDT) are
allowed.

Flash writes within SFAM
(defined in BGPDT) are
not allowed.

AMI BIOS Guard update capsule

AMI BIOS Guard
update capsule
header

First BGUP header

BiG script section
of the BGUP

Data section



0000	EF	00	00	00	40	5B	00	00	5F	41	4D	49	50	46	41	54	n...@[.._AMIPFAT
0010	04	41	4D	49	5F	42	49	4F	53	5F	47	55	41	52	44	5F	.AMI_BIOS_GUARD
0020	46	4C	41	53	48	5F	43	4F	4E	46	49	47	55	52	41	54	FLASH_CONFIGURAT
0030	49	4F	4E	53	0D	0A	31	20	2F	45	43	20	31	20	3B	42	IONS..1 /EC 1 ;B
0040	49	4F	53	5F	46	56	5F	45	43	2E	62	69	6E	0D	0A	31	IOS_FV_EC.bin..1
0050	20	2F	4E	20	31	20	3B	42	49	4F	53	5F	46	56	5F	4E	/N 1 ;BIOS_FV_N
0060	56	52	41	4D	2E	62	69	6E	0D	0A	31	20	2F	47	50	20	VRAM.bin..1 /GP
0070	31	20	3B	42	49	4F	53	5F	46	56	5F	47	50	4E	56	2E	1 ;BIOS_FV_GPNV.
0080	62	69	6E	0D	0A	31	20	2F	50	20	34	20	3B	42	49	4F	bin..1 /P 4 ;BIO
0090	53	5F	46	56	5F	4D	41	49	4E	2E	62	69	6E	0D	0A	31	S_FV_MAIN.bin..1
00A0	20	2F	44	41	54	41	20	31	20	3B	42	49	4F	53	5F	46	/DATA 1 ;BIOS_F
00B0	56	5F	44	41	54	41	2E	62	69	6E	0D	0A	31	20	2F	42	V_DATA.bin..1 /B
00C0	42	41	20	31	20	3B	42	49	4F	53	5F	46	56	5F	42	42	BA 1 ;BIOS_FV_BB
00D0	41	2E	62	69	6E	0D	0A	31	20	2F	42	20	31	20	3B	42	A.bin..1 /B 1 ;B
00E0	49	4F	53	5F	46	56	5F	42	42	2E	62	69	6E	0D	0A	02	IOS_FV_BB.bin...
00F0	00	00	00	4B	41	42	59	4C	41	4B	45	00	00	00	00	00KABYLAKE.....
0100	00	00	00	0D	00	00	00	02	00	00	00	90	01	00	00	00b.....
0110	00	02	00	00	70	05	00	01	00	00	00	00	00	00	00	01p.....
0120	00	00	00	00	00	00	00	55	00	00	00	00	F0	21	00	53U....p!.S
0130	00	00	00	00	F0	01	00	51	00	00	00	00	00	00	00	51p.Q.....Q
0140	00	01	00	00	10	00	00	51	00	02	00	00	00	00	00	51Q.....Q
0150	00	03	00	00	00	00	00	51	00	04	00	03	00	00	00	51Q.....Q
0160	00	05	00	00	00	00	00	51	00	06	00	00	00	00	00	90Q.....b
0170	00	00	00	17	00	00	00	31	00	06	00	01	00	00	00	70l.....p
0180	00	06	04	00	00	00	00	94	00	00	00	2B	00	00	00	90"....+..b
0190	00	00	00	17	00	00	00	31	00	02	00	01	00	00	00	70l.....p
01A0	00	02	04	00	00	00	00	94	00	00	00	2D	00	00	00	90"....-..b
01B0	00	00	00	1D	00	00	00	31	00	03	00	01	00	00	00	70l.....p
01C0	00	03	04	00	00	00	00	94	00	00	00	2F	00	00	00	90"..../.b
01D0	00	00	00	21	00	00	00	12	00	01	00	01	00	00	00	B0!.....!
01E0	00	05	00	00	00	00	00	71	00	05	00	00	00	00	00	92Q.....'

Intel BIOS Guard Script

```
0x0000 start
0x0008 set flash0, 34F000h
0x0010 set buffer0, FF000h
0x0018 set r0, 0
0x0020 set r1, 1000h
0x0028 set r2, 0
0x0030 set r3, 0
0x0038 set r4, 3
0x0040 set r5, 0
0x0048 set r6, 0
0x0050 jmp lb_B8

0x0058 lb_58:
0x0058 add r6, 1
0x0060 cmp r6, r4
0x0068 jge lb_158
0x0070 jmp lb_B8

0x0078 lb_78:
0x0078 add r2, 1
0x0080 cmp r2, r4
0x0088 jge lb_168
0x0090 jmp lb_E8
```

```
0x0098 lb_98:
0x0098 add r3, 1
0x00A0 cmp r3, r4
0x00A8 jge lb_178
0x00B0 jmp lb_108

; check if the data is already flashed
0x00B8 lb_B8:
0x00B8 load buffer1, flash0, r1
0x00C0 rdsts r5
0x00C8 cmp r5, 0
0x00D0 jne lb_58
0x00D8 cmp buffer0, buffer1, r1
0x00E0 je lb_128

; erase necessary flash areas
0x00E8 lb_E8:
0x00E8 eraseblk flash_0
0x00F0 rdsts r5
0x00F8 cmp r5, 0
0x0100 jne lb_78

; store the update to the flash
0x0108 lb_108:
0x0108 store flash0, buffer0, r1
0x0110 rdsts r5
0x0118 cmp r5, 0
0x0120 jne lb_98
```

```
0x0128 lb_128:
0x0128 add r0, r1
0x0130 cmp r0, 100000h
0x0138 jge lb_180
0x0140 sub flash_0, r1
0x0148 sub buff0, r1
0x0150 jmp lb_B8

0x0158 lb_158:
0x0158 set r15, 1
0x0160 jmp lb_180

0x0168 lb_168:
0x0168 set r15, 2
0x0170 jmp lb_180
```

BUT !

How did we manage to disassemble the
Script?

We got a decrypted ACM :)

Intel BIOS Guard ACM decryption attempts

We used to try:

- Relocation of MMIO areas (for example, overlap SPIBAR to different BiG components)
- Dump decrypted ACM from cache and while debugging
- Find and exploit Race Conditions

Finally, when we started to intersect BiG directory items with ACM in physical memory...

Intel BIOS Guard ACM decryption attempts

It failed too! :)

It was really hard to dump the decrypted ACM. Intel did a good job on protecting ACM.

Intel BIOS Guard ACM decryption attempts

Intel investigating breach after 20GB of internal documents leak online

Leak confirmed to be authentic. Many files are marked "confidential" or "restricted secret."



By Catalin Cimpanu for Zero Day | August 6, 2020 -- 19:23 GMT (12:23 PDT) | Topic: Security

MORE FROM CAT



- BIOSGuardModule_10.0
- empty.bridid
- huffman_cnl_dic1

23.04.2020 9:20

23.04.2020 9:20

23.04.2020 9:20

The decrypted version of ACM was inside Alder Lake package for Simics from the recent leak from Intel!

Intel BIOS Guard ACM decryption attempts

- chip-red-pill debugging setup (Apollo Lake architecture)

35:36	RO	0x0 (rst)	RESERVED_2 (RESERVED_2) reserved
37:37	RW	0x0 (rst)	TIMED_MWAIT_ENABLE (Timed MWAIT Enable) If set, this field indicates that the core Timed WAIT feature is supported.
36:36	RO	0x0 (rst)	RESERVED_3 (RESERVED_3) reserved
35:35	RW	0x0 (rst)	PFAT_ENABLE (PFAT_ENABLE) If set, this field indicates that PFAT is enabled

- Find PFAT ACM from the system
- Enable Intel BIOS Guard via debugging with Intel DCI
- Dump & patch ‘wrmsr’ instruction (exit from ACM) with the code to dump the cache into physical memory

Intel BIOS Guard ACM analysis

ACM execution consists of two stages.

First - hardware initialization. Includes some initial I/O, MSR operations and BiG directory parsing + it's security checks.

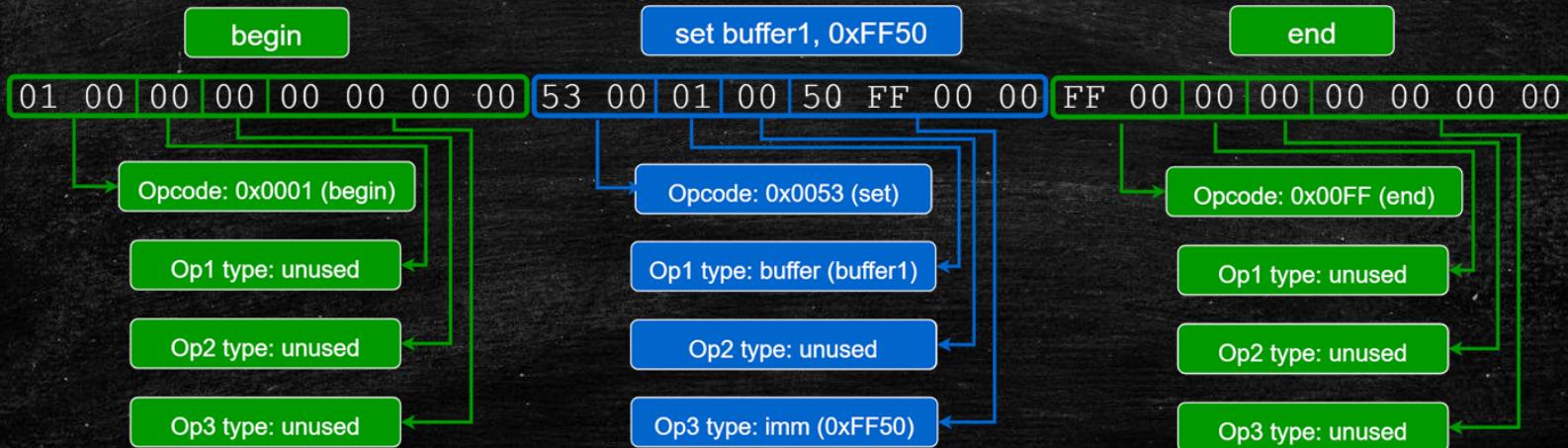
```
    mov    [rcx+0F058h], r12
    mov    [rcx+0F060h], r13
    mov    [rcx+0F068h], r14
    mov    [rcx+0F070h], r15
    mov    rdi, rcx
    mov    r8, rcx
    add    rcx, 10000h
    mov    rsp, rcx
    call   biosguard_main
    lea    rcx, loc_7F80017E

loc_7F80017E:                                ; DATA XREF: start+
    and   rcx, 0xFFFFFFFFFFFFC0000h
    mov   r8, [rcx+0F038h]
```

BIOS Guard script interpreter

Second stage - BIOS Guard Script interpreter, the heart of BiG ACM.

Instructions opcode analysis:



BIOS Guard script interpreter

A few types of operands:

- data buffer
- flash pointer
- register
- imm value

buffer0 points to the
BGUP's data section, not
writeable.

```
1 set r1, 0x1000
2 set flash1, 0xA00000
3 load buffer1, flash1, r1
4 cmp buffer1, buffer0, r1
5 jne _somewhere
```

See it for yourself!

We have created Intel BIOS Guard Script assembler/disassembler tool.

<https://github.com/allowitsme/big-tool>

#Takeaways

- Supply chain problems
 - disadvantage of security through obscurity model
- The problem in a basic component compromises all technologies it serves as a Root-of-Trust
- PC Firmware vendors aimed to protect code integrity, still forgetting about other attacks vectors
 - e.g. data-only attacks

Thank you
