Back to the Future with Platform Security

Enrique Nissim
Krzysztof Okupski
Joseph Tartaro

IOActive®

#whoami

@kiqueNissim

@exminium

@droogie1xp

Embedded Security Consultants at IOActive

- Low-level code review
- Reverse engineering
- Specialized tooling development

Outline

- SMM Protections
- SPI Flash Configuration
  - Protections and misconfigurations
- Vulnerable SMI Handlers
  - Writing to the flash
  - Leaking and corrupting SMRAM
- AMD ROM Armor
- AMD Platform Secure Boot
  - Misconfigurations
  - Installing a persistent firmware implant in ring -2
- AMD SMM Supervisor
  - SMM Isolation
  - OEM Policy Analysis
- Conclusions

How this research started

- Acquired Asus Ryzen gaming laptop
  - Fuzzed SMI handlers and the system immediately hung
  - CHIPSEC failed dumping the SPI flash (minimal AMD support)
- Questions
  - What are the registers used to protect SMRAM?
  - What about the SPI flash?
  - What about hardware secure boot?
- Scarce documentation and research on AMD platform security

Lack of Documentation

- Specifications from hell
  - Documentation available only for some families
  - AMD stopped publishing BKDG documents for new models
  - Some information is missing from the Processor Programming References
  - Inconsistencies across Programming Manuals and PPRs
- Many interesting documents are under NDA
  - AMD Platform Secure Boot
  - AMD PSP ROM Armor

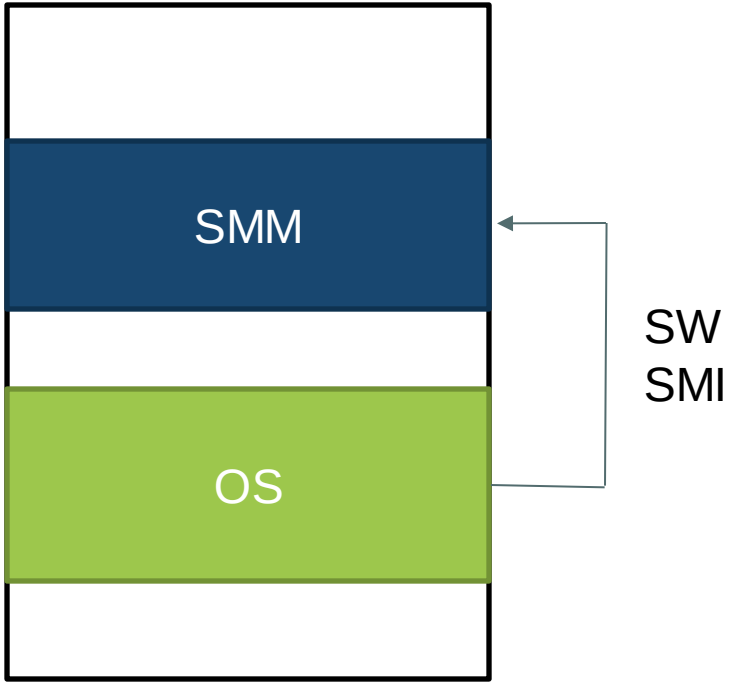
Disclaimer

All the discussed information here has been obtained through reverse engineering and black-box testing without access to restricted documentation. As such, there may be inaccuracies in our conclusions.

SMM Configuration

SMM in a Nutshell

- OS runs in ring 0
- SMM runs in ring –2
- Triggered via System Management Interrupt (SMI)
- SMRAM must be protected from OS



Triggering SMIs

The IO mapped base address of this register block is defined by PMx6A.

SmiCmdBlkx00 SmiCmdPort

Bits	Description
7:0	**SmiCmdPort**. Read-write. Reset: 0. When SMI command port is enabled, a write to this port generates SMI, NMI or IRQ3 depending on the setting of SMIxB0[23:22]. A read from this port returns the previously written value but does not generate SMI.

PMx6A AcpiSmiCmd

Bits	Description
15:0	**AcpiSmiCmd**. Read-write. ColdReset: 00B0h. These bits define the 16-bit IO base address[15:0] of the ACPI SMI command block defined in 3.26.15.7 [SmiCmdBlk]. The address is required to be WORD-aligned (Addr[0]==0).

SMRAM Protection

- Possible SMRAM Locations
  - ASEG (at A0000h–BFFFFh) (like Intel CSEG)
  - TSEG

- Registers
  - MSRC001_0111 (SMM_BASE used for SMM base address)
  - MSRC001_0112 (SMM TSeg Base Address (SMMAddr))
  - MSRC001_0113 (SMM TSeg Mask (SMMMask))
  - MSRC001_0015[SmmLock] (HWCR used for locking the config)

Unlocked TSEG on Acer Swift 3

```
PS C:\Users\minium\Desktop\Platbox> .\compiled\platbox_cli cli
>>> chipset
Detected chipset:
=> Family: 17
=> Model: 60

[...]

MSR C001_0112 SMM TSeg Base Address (SMMAddr)
 => Base : ae000000
 => Limit: aeffffff

MSR C001_0113 SMM TSeg Mask (SMMMask)
 => Value: 0000fffff006603
   -> TSegMask: 000000ffff000000
   -> TMTypeDram: 6
   -> AMTypeDram: 6
   -> TMTypeIoWc: 0
   -> AMTypeIoWc: 0
   -> TClose: 0
   -> AClose: 0
   -> TValid: 1
   -> AValid: 1

MSR C001_0015 Hardware Configuration (HWCR)
 => Value: 9000010
   -> SMMLock: 0 - FAILURE
```

- The MSRC001_0015[SmmLock] was not set
- We can directly disable ASEG and TSEG by changing MSRC001_0113[AValid,TValid] and manipulate SMRAM

Source: https://labs.ioactive.com/2022/11/exploring-security-configuration-of-amd.html

SMM Unlock Key?

15.32 SMM-Lock

The SMM-Lock feature allows platform firmware to prevent System Management Interrupts (SMI) from being intercepted in SVM. The SmmLock bit is located in the HWCR MSR register.

15.32.1 SmmLock Bit — HWCR[0]

The SmmLock bit (bit 0) is located in the HWCR MSR (C001_0015h). When SmmLock is clear, it can be set to one. Once set, the bit cannot be cleared by software and writes to it are ignored. SmmLock can only be cleared using the SMM_KEY MSR (see Section 15.32.2), or by a processor reset. This bit is not affected by INIT or SKINIT. When SmmLock is set, other SMM configuration registers cannot be written. For complete information on the HWCR register, see the *BIOS and Kernel Developer's Guide (BKDG)* or *Processor Programming Reference Manual* applicable to your product.

Only mentioned in the *AMD Programming Manual Vol. 2*

SMM Key Backdoor

15.32.2 SMM_KEY MSR (C001_0119h)

The write-only SMM_KEY MSR is used to create a password-protected mechanism to clear SmmLock.

When SmmLock is zero, writes to SMM_KEY MSR set the 64-bit SMM Key value.

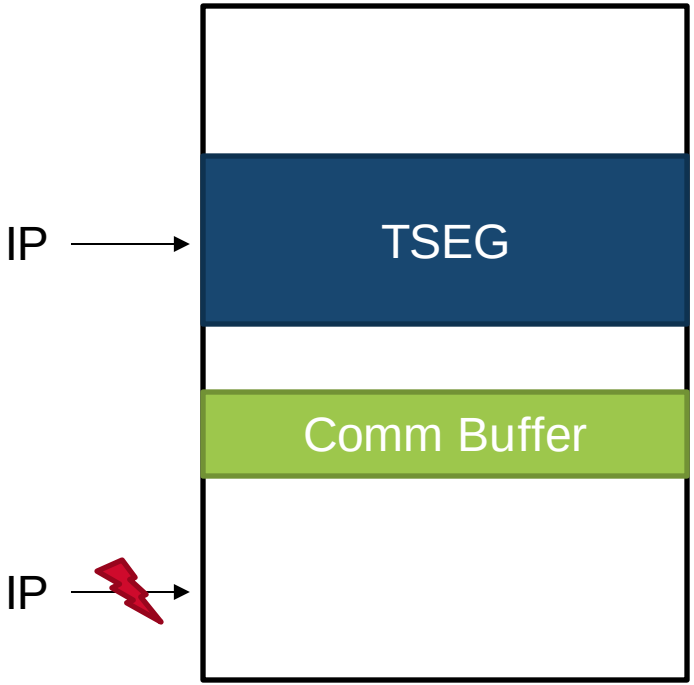
When SmmLock is one, writes to SMM_KEY MSR compare the written value to the SMM Key value; if the values match and are non-zero, the SmmLock bit is cleared. If the values mismatch or the SMM Key value is zero, the write to SMM_KEY is ignored, and SmmLock is unmodified. Software should read SmmLock after writing SMM_KEY to determine whether the unlock succeeded.

If SMM_Key MSR is equal to zero when SmmLock is one, SmmLock can only be cleared by a processor reset.

To preserve the security of the SMM key, reading SMM_KEY MSR always returns zero.

SMM Callout Protections

- Callouts happen when code running in SMM mode ends up executing code outside of SMRAM



SMM Callout Protections

- Intel CPUs configure MSR_SMM_FEATURE_CONTROL to prevent instruction fetching from outside of SMRAM while running in SMM mode
- For AMD there is no MSR or register providing a similar protection
- This means that a simple SMM callout issue that would not be straightforward to exploit on Intel, could be easily exploited on AMD (depending on paging)

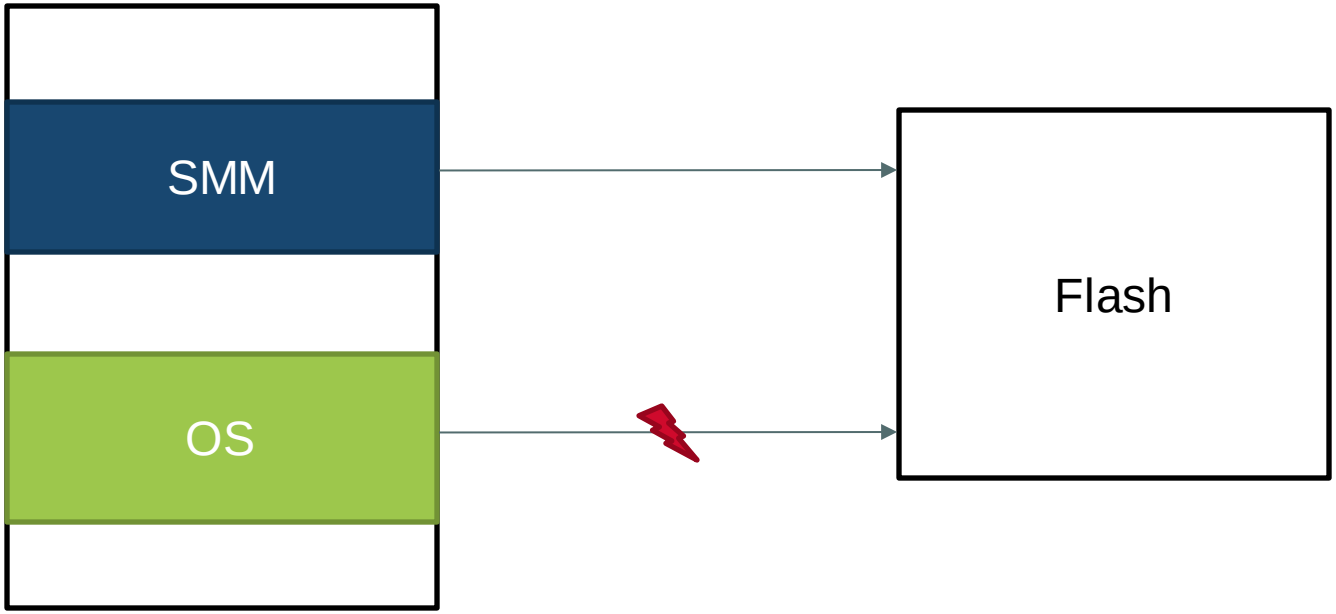
Extra thoughts on SMM Base

- Even if AMD mitigates this, there is still the approach of Bruno Pujos (Synacktiv)
- Requires knowing the value of SMBASE, which was retrieved by using a “leak” of the *PiSmmCpuDxeSMM* driver implementation that comes with EDK2
- This was needed on Intel because the MSR 0x9E (IA32_SMBASE), which holds the SMM base address, is only readable while in SMM mode
- This is not the case for AMD as the MSR 0xC0010111 (SMM_BASE) can be read from the OS.
  - Trapped by the hypervisor in Win11
- An important thing to note is that SMM_BASE is locked when SmmLock is set. This prevents attacks consisting in relocating SMRAM outside of TSEG.

Source: https://www.synacktiv.com/en/publications/through-the-smm-class-and-a-vulnerability-found-there.html

SPI Flash Configuration

SPI Flash in a Nutshell



SPI Flash Basics

- Access modes:
  - “Direct” via memory-mapped IO
  - “Indexed” by programming the SPI controller
- Protections:
  - ROM Protected Range
  - SPI Restricted Commands

ROM Protected Range

- Applies to “direct” access via memory-mapped IO
- Defines up to 4 ranges of up to 16MB each
  - Write-once registers, thus locked automatically
  - Enable/disable bit that must be locked to prevent OS access
  - But… it can be reprogrammed from SMM
- Quirks:
  - Cannot write even with protections switched off; neither from the OS nor SMM
  - Despite the enable-bit not being set, the protection is enforced? Tested with read-access

ROM Protected Range

```
PS C:\Users\minium\Desktop\Platbox> .\compiled\platbox_cli cli
>>> chipset
Detected chipset:
=> Family: 17
=> Model: 60

=== SPI Range Protections ===
Rom Protect 0: ff73a539
  - Base: ff73a000
  - RangeUnit: 1
  - Range: 00000039
  - Protected size: 00390000
  - WriteProtected: 1
  - ReadProtected: 0
  - Total range [ff73a000, ffad9fff]
  - STATUS: OK - Range in use
Rom Protect 1: fff204df
  - Base: fff20000
  - RangeUnit: 0
  - Range: 000000df
  - Protected size: 000df000
  - WriteProtected: 1
  - ReadProtected: 0
  - Total range [fff20000, ffffffff]
  - STATUS: OK - Range in use
Rom Protect 2: 00000000
  - STATUS: Warning - Unused ROM Range Protection
Rom Protect 3: 00000000
  - STATUS: Warning - Unused ROM Range Protection
```

SPI Restricted Commands

- Applies to “indexed” access via SPI registers
- Defines SPI operations that are blocked
  - Up to 8 SPI operations
  - Locked to prevent OS access
  - But… it can be reprogrammed from SMM
- Typically the SPI write enable command is blocked

SPI Controller Programming

Program ROM protected ranges	D14F3x[050,0x54,0x58,0x5C] FCH::ITF::LPC::RomProtect
Program SPI restricted cmds	SPIx04 FCH::ITF::SPI::SPIRestrictedCmd SPIx08 FCH::ITF::SPI::SPIRestrictedCmd2
Lock SPI restricted cmd registers	SPIx00 FCH::ITF::SPI::SPICntrl0[SpiAccessRomEn] = 0 SPIx00 FCH::ITF::SPI::SPICntrl0[SpiHostAccessRomEn] = 0
Enable ROM protected ranges	SPIx1D FCH::ITF::SPI::AltSPICS[SpiProtectEn0] = 1 SPIx1D FCH::ITF::SPI::AltSPICS[SpiProtectEn1] = 1
Lock ROM protected range registers	SPIx1D FCH::ITF::SPI::AltSPICS[SpiProtectLock] = 1

CVE-2023-20579

```
PS C:\Users\minium\Desktop\Platbox> .\compiled\platbox_cli cli
>>> chipset
Detected chipset:
=> Family: 19
=> Model: 44

SPI BASE: fec11000
SPIx00 - SPI_Cntrl0: 0fc00000
  -  SpiAccessMacRomEn: 1 - FAILED
  - SpiHostAccessRomEn: 1 - FAILED

 RestrictedCmd: 00 00 00 00
RestrictedCmd2: 00 00 00 00

SPIx1D - Alt_SPI_CS
  - lock_spi_cs: 0 - FAILED
  - SpiProtectEn0: 1 - OK
  - SpiProtectEn1: 1 - OK
  - SpiProtectLock: 0 - FAILED
  - AltSpiCsEn: 0
```

Model: Lenovo ThinkPad P16s

SPI Read Demo

Program opcode
SPIBAR[0x45]

↓

TX Byte Count
SPIBAR[0x48]

↓

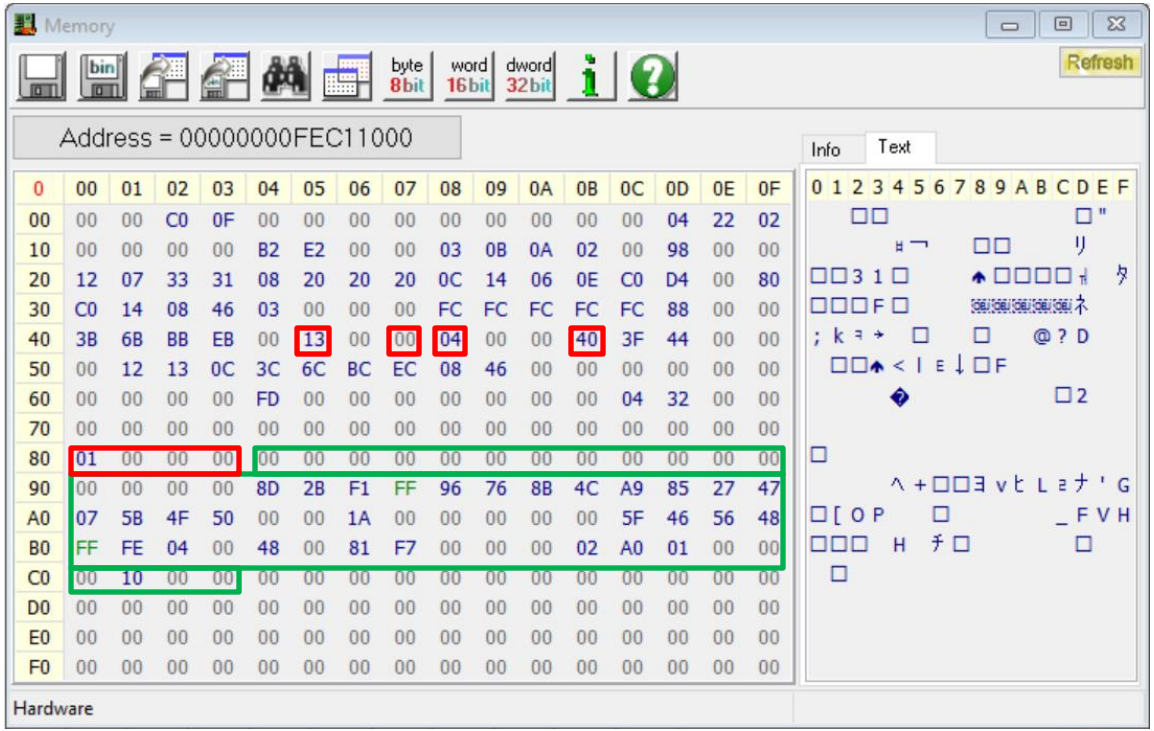
RX Byte Count
SPIBAR[0x4B]

↓

SPI Address
SPIBAR[0x80]

↓

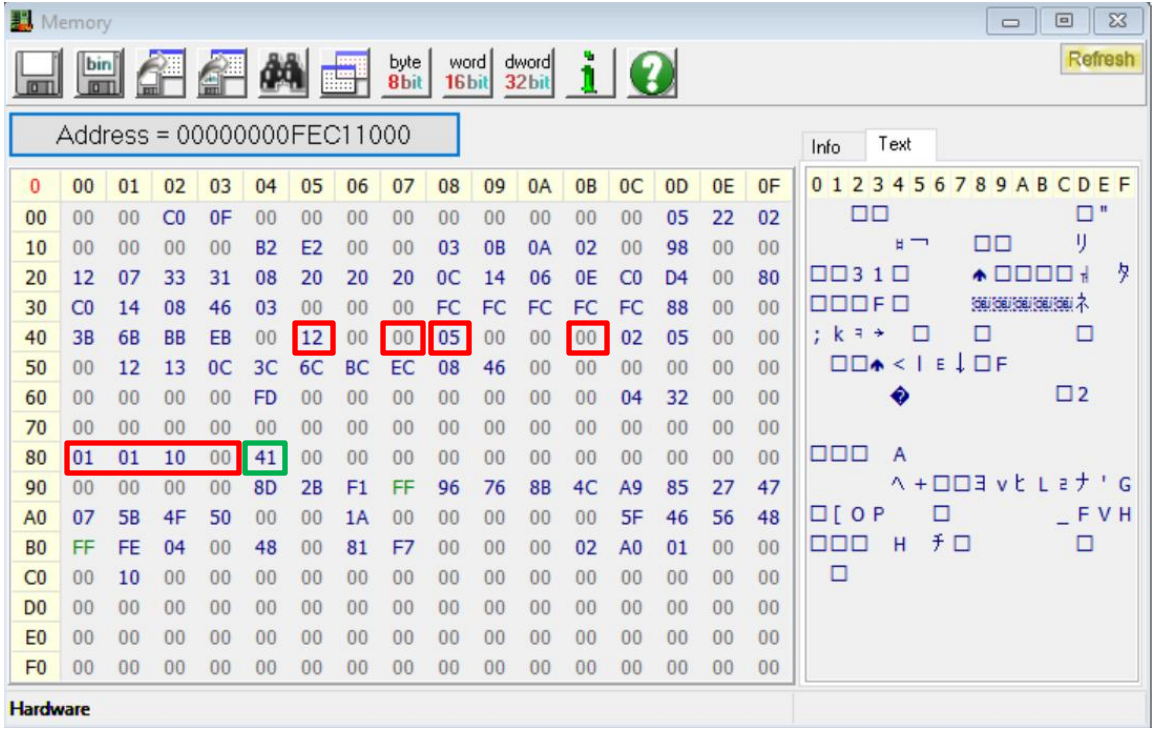
Trigger
SPIBAR[0x47]



0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	00	C0	0F	00	00	00	00	00	00	00	00	00	04	22	02
10	00	00	00	00	B2	E2	00	00	03	0B	0A	02	00	98	00	00
20	12	07	33	31	08	20	20	20	0C	14	06	0E	C0	D4	00	80
30	C0	14	08	46	03	00	00	00	FC	FC	FC	FC	FC	88	00	00
40	3B	6B	BB	EB	00	13	00	00	04	00	00	40	3F	44	00	00
50	00	12	13	0C	3C	6C	BC	EC	08	46	00	00	00	00	00	00
60	00	00	00	00	FD	00	00	00	00	00	00	00	04	32	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	00	00	00	8D	2B	F1	FF	96	76	8B	4C	A9	85	27	47
A0	07	5B	4F	50	00	00	1A	00	00	00	00	00	5F	46	56	48
B0	FF	FE	04	00	48	00	81	F7	00	00	00	02	A0	01	00	00
C0	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

IOActive

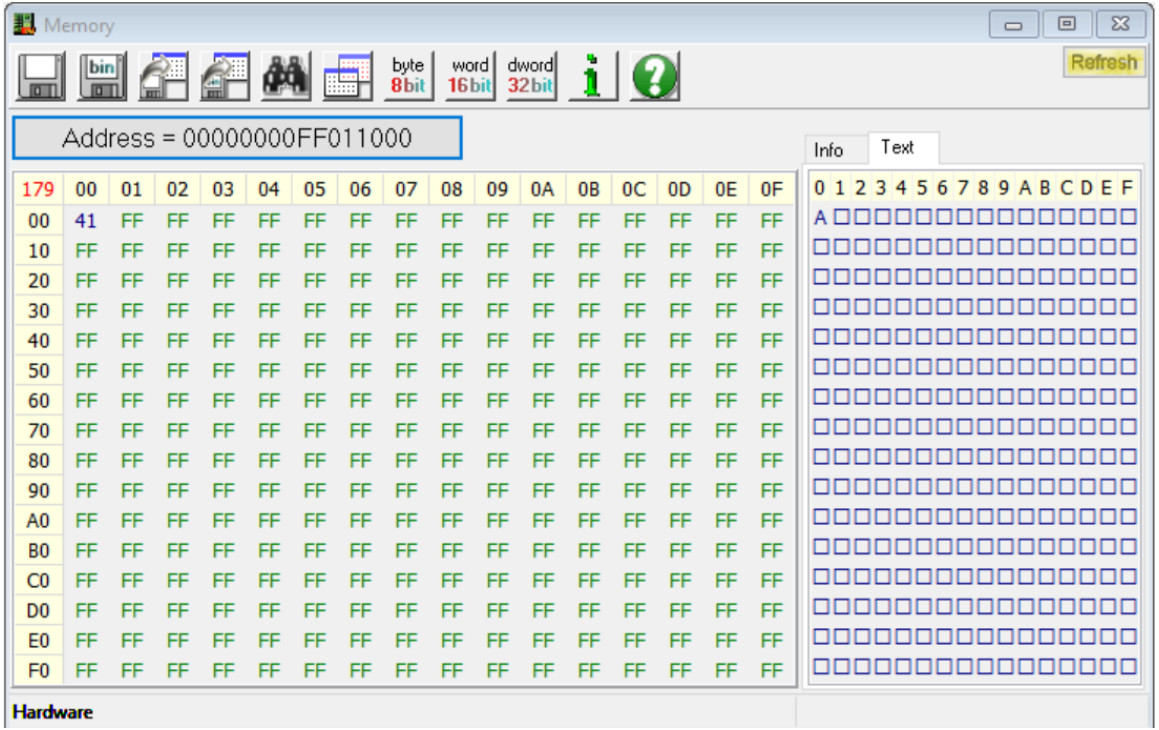
SPI Write Demo



0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	00	C0	0F	00	00	00	00	00	00	00	00	00	05	22	02
10	00	00	00	00	B2	E2	00	00	03	0B	0A	02	00	98	00	00
20	12	07	33	31	08	20	20	20	0C	14	06	0E	C0	D4	00	80
30	C0	14	08	46	03	00	00	00	FC	FC	FC	FC	FC	88	00	00
40	3B	6B	BB	EB	00	12	00	00	05	00	00	00	02	05	00	00
50	00	12	13	0C	3C	6C	BC	EC	08	46	00	00	00	00	00	00
60	00	00	00	00	FD	00	00	00	00	00	00	00	04	32	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	01	01	10	00	41	00	00	00	00	00	00	00	00	00	00	00
90	00	00	00	00	8D	2B	F1	FF	96	76	8B	4C	A9	85	27	47
A0	07	5B	4F	50	00	00	1A	00	00	00	00	00	5F	46	56	48
B0	FF	FE	04	00	48	00	81	F7	00	00	00	02	A0	01	00	00
C0	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

IOActive

SPI Write Demo



IOActive

OBS 29.1.1 - Profile: Untitled - Scenes: Untitled

File Edit

Select Administrator: Command Prompt

```
Host Name:                 SECURE-CORE
OS Name:                   Microsoft Windows 11 Pro
OS Version:                10.0.22621 N/A Build 22621
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Workstation
OS Build Type:             Multiprocessor Free
Registered Owner:          ioactive
Registered Organization:   N/A
Product ID:                00355-60783-34288-AAOEM
Original Install Date:     5/6/2023, 4:07:07 PM
System Boot Time:          5/17/2023, 3:40:37 AM
System Manufacturer:       Acer
System Model:              TravelMate P414-41
System Type:               x64-based PC
Processor(s):              1 Processor(s) Installed.
                           [01]: AMD64 Family 25 Model 68 Stepping 1 AuthenticAMD ~2486 Mhz
BIOS Version:              INSYDE Corp. V1.08, 2/16/2023
Windows Directory:         C:\Windows
System Directory:          C:\Windows\system32
Boot Device:               \Device\HarddiskVolume1
System Locale:             en-us;English (United States)
Input Locale:              en-us;English (United States)
Time Zone:                 (UTC-08:00) Pacific Time (US & Canada)
Total Physical Memory:     15,557 MB
Available Physical Memory: 8,507 MB
Virtual Memory: Max Size:  16,581 MB
```

Displ

Scenes

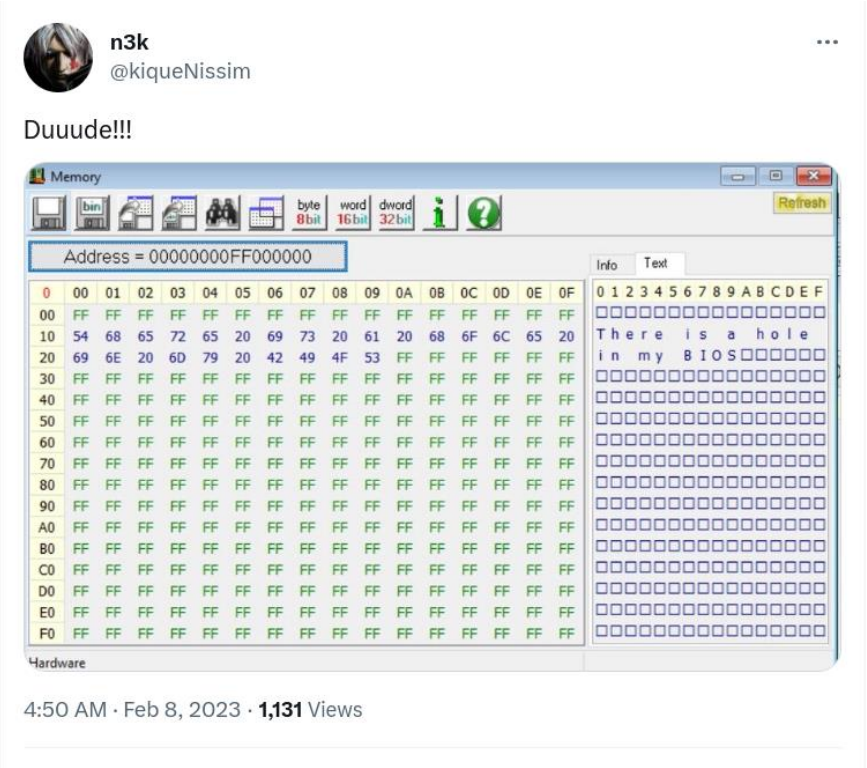
Scene

Studio Mode

Settings

Exit

The Case of Asus Strix G513QR.330



- Same issue on Asus
- They informed AMI
- Still unfixed

Writing to the SPI Flash through SMI Handlers

Vulnerable SMM Modules

- We discovered two SMM modules that expose SPI ops:
  - SystemFirmwareDeviceSmm (CVE-2023-31100)
  - XXXRuntimeDxe (CVE pending)
- We will focus on the second one

The Story of a Dangerous SMI Handler

- First stumbled into *FwBlockServiceSmm* SMM module
- Developed by Insyde, used by various vendors

- Exposes three operations via SMI handler:
  - SPI Read
  - SPI Write
  - SPI Erase

The Interesting Code

```
1  __int64 __fastcall FwBlockServiceSpiRead(
2          FW_BLOCK_SERVICE_PROTOCOL2 *a_This,
3          __int64 a_Offset,
4          __int64 a_BaseAddr,
5          unsigned __int64 *a_Size,
6          char *a_DestAddr)
7  {
8    char *l_SrcAddr; // rdx
9    __int64 v6; // rbx
10
11   l_SrcAddr = (char *)(a_BaseAddr + a_Offset);
12   if ( *(_DWORD *)g_mFlashDevice_0 == 3 )
13     return SpiRead_0(g_mFlashDevice_0, a_DestAddr, l_SrcAddr, *a_Size);
14   if ( *(_DWORD *)g_mFlashDevice_0 > 1u )
15     return 0x8000000000000003ui64;
16   v6 = 0i64;
17   if ( *a_Size && a_DestAddr != l_SrcAddr )
18     MemCpy_1(a_DestAddr, l_SrcAddr, *a_Size);
19   return v6;
20 }
```

The Disabled SMI Interface

```
 7 EFI_STATUS __fastcall FwBlockServiceSmmSmiHandler(
 8         EFI_HANDLE DispatchHandle,
 9         const void *Context,
10         char *CommBuffer,
11         UINTN *CommBufferSize)
12 {
13   __int64 l_Ret; // rax
14
15   if ( CommBuffer
16     && CommBufferSize
17     && !g_DisableFwBlockServiceSmiHandlerFlag
18     && *CommBufferSize == qword_253438 - 24
19     && CommBuffer == qword_253420 + 24
20     && *(CommBuffer + 3) <= 0x1000ui64 )
21   {
22     switch ( *CommBuffer )
23     {
24       case 2i64:
25         l_Ret = FwBlockServiceSpiRead(0i64, *(CommBuffer + 2), 0i64, CommBuffer + 3, CommBuffer + 32);
26         goto LABEL_13;
27       case 3i64:
28         l_Ret = FwBlockServiceSpiWrite(0i64, *(CommBuffer + 2), CommBuffer + 3, CommBuffer + 32);
29         goto LABEL_13;
30       case 4i64:
31         l_Ret = FwBlockServiceSpiErase(0i64, *(CommBuffer + 2), CommBuffer + 3);
32 LABEL_13:
33         *(CommBuffer + 1) = l_Ret;
34         break;
35     }
36   }
37   return 0i64;
38 }
```

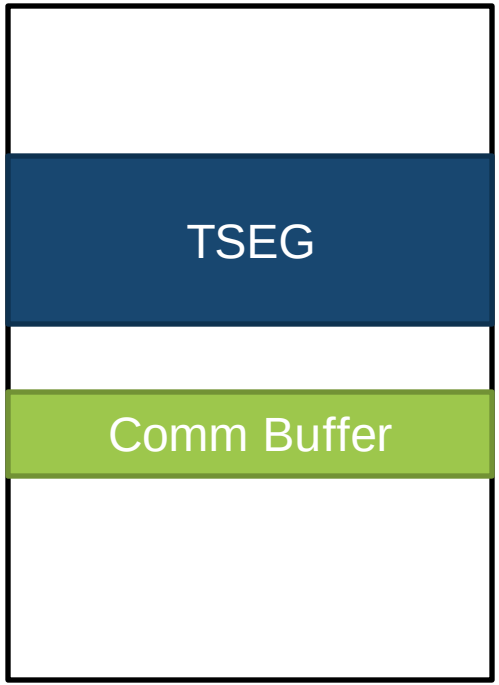
Proxying Through *XXXRuntimeDxe*

```
1  __int64 [redacted]SpiRead()
2  {
3    __int64 l_RBX; // rbx
4    __int64 l_RSI; // rdi
5    __int64 result; // rax
6
7    l_RBX = ReadSaveStateRegister(EFI_SMM_SAVE_STATE_REGISTER_RBX);
8    l_RSI = ReadSaveStateRegister(EFI_SMM_SAVE_STATE_REGISTER_RSI);
9    if ( l_RBX != FetchSomeValFromFlash(&UNK_GUID) + 40 || *(l_RSI + 8) != 'AFMS' )
10     return 0x8000000000000003ui64;
11   result = (g_[redacted]Instance->FwBlockServiceProtocol->SpiRead)(
12               g_[redacted]Instance->FwBlockServiceProtocol,
13               *(l_RSI + 16),
14               0i64,
15               l_RSI + 32,
16               l_RSI + 40);
17   *l_RSI = result;
18   return result;
19 }
```

Exploiting the Bug

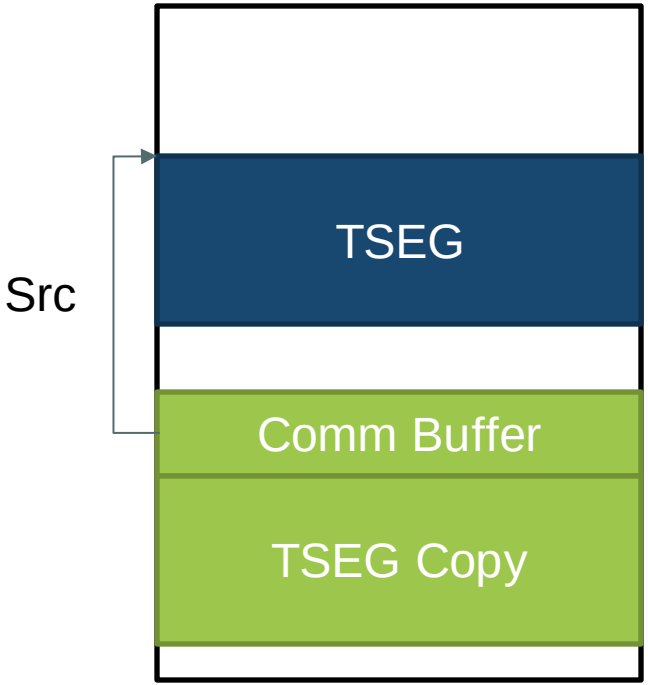
Comm Buffer:

- 0x00: Return status
- 0x08: Signature
- 0x10: *Src address*
- 0x18: Unk
- 0x20: *Size*
- 0x28: *Dst buffer*



Exploiting the Bug

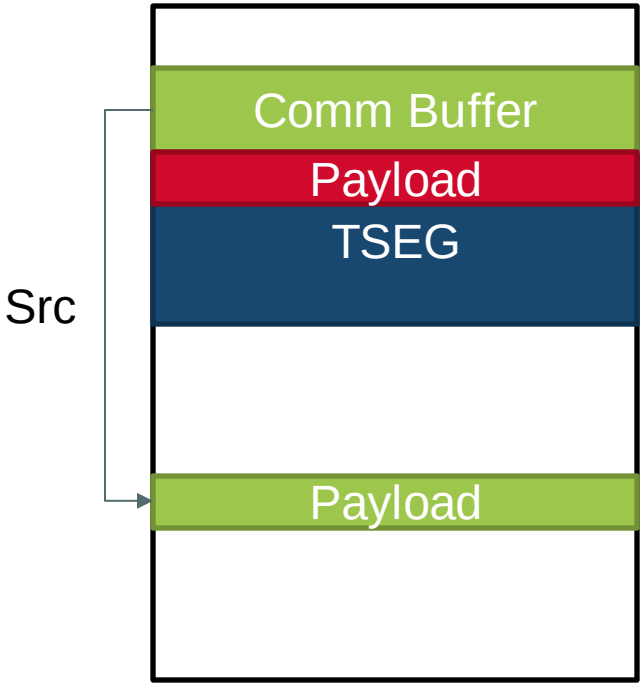
We get an arbitrary read like this…



Exploiting the Bug

… or a write to TSEG like that

… using the previous TSEG leak!



Select C:\Windows\System32\cmd.exe

```
C:\Windows\system32>cd C:\Users\IOActive\Desktop\Platbox\compiled

C:\Users\IOActive\Desktop\Platbox\compiled>.\platbox_cli.exe start
-> service has already been started!

C:\Users\IOActive\Desktop\Platbox\compiled>cd pocs

C:\Users\IOActive\Desktop\Platbox\compiled\pocs>.\Acer-DumpStartTSEG.exe
0000000000000000 | 00 00 00 00 00 00 00 00 53 4d 46 41 00 00 00 00 | ........SMFA....
0000000000000010 | 00 00 00 ae 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0000000000000020 | 00 08 00 00 00 00 00 00 53 4d 4d 53 33 5f 36 34 | ........SMMS3_64
0000000000000030 | a8 4d f6 ae 00 00 00 00 00 b0 cf ae 00 00 00 00 | .M..............
0000000000000040 | 00 80 00 00 00 00 00 00 13 00 01 80 00 00 00 00 | ................
0000000000000050 | 00 40 cf ae 00 00 00 00 28 06 00 00 00 00 00 00 | .@......(.......
0000000000000060 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0000000000000070 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0000000000000080 | 00 00 40 93 ff ae 00 00 00 00 06 d7 e1 8a 46 db | ..@...........F.
0000000000000090 | 45 18 36 ec 41 18 76 e0 67 af 3e ef 63 af 7e e3 | E.6.A.v.g.>.c.~.
00000000000000A0 | c7 3d 0e d4 c3 3d 4e d8 c8 81 a3 69 cc 81 e3 65 | .=...=N....i...e
00000000000000B0 | 68 13 93 52 6c 13 d3 5e 4a a4 9b 51 4e a4 db 5d | h..Rl..^J..QN..]
00000000000000C0 | ea 36 ab 6a ee 36 eb 66 24 88 a4 f5 20 88 e4 f9 | .6.j.6.f$.......
00000000000000D0 | 84 1a 94 ce 80 1a d4 c2 a6 ad 9c cd a2 ad dc c1 | ................
00000000000000E0 | 06 3f ac f6 02 3f ec fa 88 81 a3 71 8c 81 e3 7d | .?...?.....q...}
00000000000000F0 | 28 13 93 4a 2c 13 d3 46 0a a4 9b 49 0e a4 db 45 | (..J,..F...I...E
```

SMRAM Analysis

- Dmytro Oleksiuk (Cr4sh) published the first SMRAM analysis tool
- Some of the magics on which it relies for extracting information are absent in the case of AMD
- We reverse engineered the *FchSmmDispatcher* implementing EFI_SMM_SW_DISPATCH2_PROTOCOL and gave it an update

- https://github.com/IOActive/Platbox/tree/main/tools/amd_smram_parser

```
Registered SMI Handlers:

9C28BE0C-EE32-43D8-A223E7C1614EF7CA
 - SMIH - Handler at ae9e5148 - [HddPassword]
54C03D2D-5903-4DFB-88B7FA7636BE03D1
 - SMIH - Handler at aea1a6ac - [IdeBusDxe]
3A9DB872-7A03-4B99-A9CDB853012DBD5C
 - SMIH - Handler at aeafb494 - [PnpSmm]
EFI_FIRMWARE_PERFORMANCE_GUID
 - SMIH - Handler at aec0e500 - [FirmwarePerformanceSmm]
4B52E4DA-60EB-4EC2-A80CF9FD0AB85E97
 - SMIH - Handler at aec29274 - [NvmExpressLegacySmm]
EFI_ATA_PASS_THRU_PROTOCOL_GUID
 - SMIH - Handler at aec333b0 - [StorageSecurityCommandDxe]
2970687C-618C-4DE5-B8F96C7576DCA83D
 - SMIH - Handler at aec3c6d0 - [SaveSpdToRomDxe]
52C78312-8EDC-4233-98F21A1AA5E388A5
 - SMIH - Handler at aec7ee40 - [NvmExpressDxe]
56947330-585C-4470-A95DC55C529FEB47
 - SMIH - Handler at aec8ddc0 - [AhciBusDxe]
EFI_SMM_LOCK_BOX_COMMUNICATION_GUID
 - SMIH - Handler at aef7a4e4 - [SmmLockBox]
EDKII_SMM_END_OF_S3_RESUME_PROTOCOL_GUID
 - SMIH - Handler at aefeb560 - [PiSmmCore]
EDKII_S3_SMM_INIT_DONE_GUID
 - SMIH - Handler at aefeb518 - [PiSmmCore]
00000000-0000-0000-000000000000000
 - SMIH - Handler at aec00584 - [FchSmmDispatcher]
```

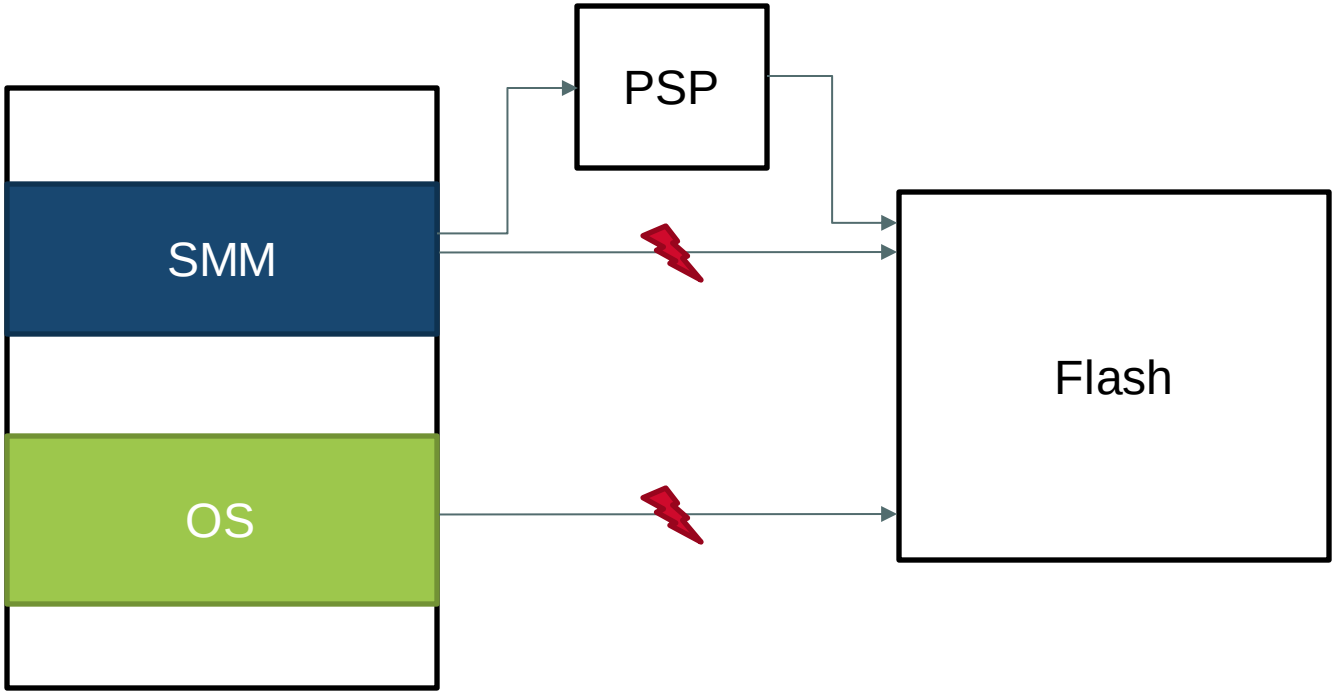
AMD PSP ROM Armor

General

- ROM protect ranges / SPI restricted commands don't stop SMM-attacks
- AMD's PSP ROM Armor ≈ Intel's Protected Range Registers

- When enabled the Host is no longer allowed to make SPI write transactions
- Every time the Host requires to talk to the SPI it must go through the PSP

ROM Armor in a Nutshell



The Whitelisted SPI Regions

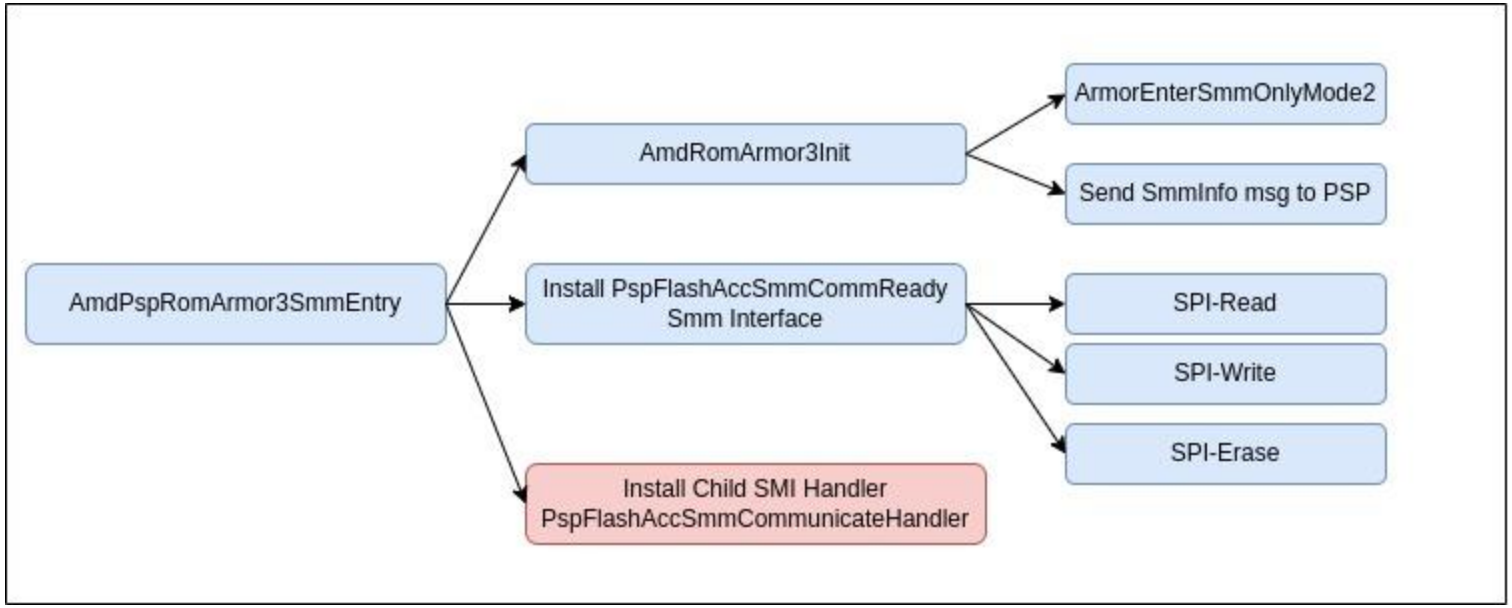
- The OEM provides a policy that dictates writable SPI regions
- This policy is embedded at BIOS build time in the PSP directory
- The writable regions should only include the NVRAM region

- We created a script to extract this information:

```
droogie@vesper tools git:(master) ✗ $ python rom_armor_psp_policy.py ../../amd/uefi_smm_research/Secu
re-Core-Acer/bios-1.08.bin
[6d] - Size:[00025000] ( 148-KB) - |00| S_ADDR:[4000000001f8b000] D_ADDR:[ffffffffffffffff]
[6d] - Size:[00001000] (   4-KB) - |10| S_ADDR:[4000000001fb0000] D_ADDR:[ffffffffffffffff]
[6d] - Size:[00026000] ( 152-KB) - |20| S_ADDR:[4000000001fb1000] D_ADDR:[ffffffffffffffff]
[6d] - Size:[00001000] (   4-KB) - |30| S_ADDR:[4000000001f8a000] D_ADDR:[ffffffffffffffff]
[6d] - Size:[00048000] ( 288-KB) - |40| S_ADDR:[4000000001b20000] D_ADDR:[ffffffffffffffff]
```


IOActive

AmdPspRomArmor3Smm Flow



PspFlashAccSmmCommReady Interface

- SPIRead: Simply reads the memory-mapped SPI flash
- SPIWrite: Sends a PSP cmd 0x51 *ArmorSpiTransaction* with trn_type 2
- SPIErase: Sends a PSP cmd 0x51 *ArmorSpiTransaction* with trn_type 3

```
#pragma pack (push, 1)
typedef struct _ArmorSpiTransactionRecord { // 0x20
    /* 0x00 */ UINT32 record_size;
    /* 0x04 */ UINT32 unused;
    /* 0x08 */ UINT32 trn_type; // 2 Write | 3 Erase
    /* 0x0C */ UINT64 aux_memory_page;
    /* 0x14 */ UINT32 flash_address;
    /* 0x18 */ UINT32 trn_length; // For erase must be 0x1000
    /* 0x1C */ UINT32 unused2;
} ArmorSpiTransactionRecord;
#pragma pack (pop)
```

Host <=> PSP Communication

- Communication with the PSP occurs via a mailbox and doorbell mechanism
- What prevents the Host OS from sending transactions to the PSP directly?
- The trick:
  - Host sends PSP command to enforce SMM-only state (cmd 0x50)
  - The Host sets SmmFlag that lives in SMRAM on every transaction
  - The PSP verifies the SmmFlag when the transaction is triggered

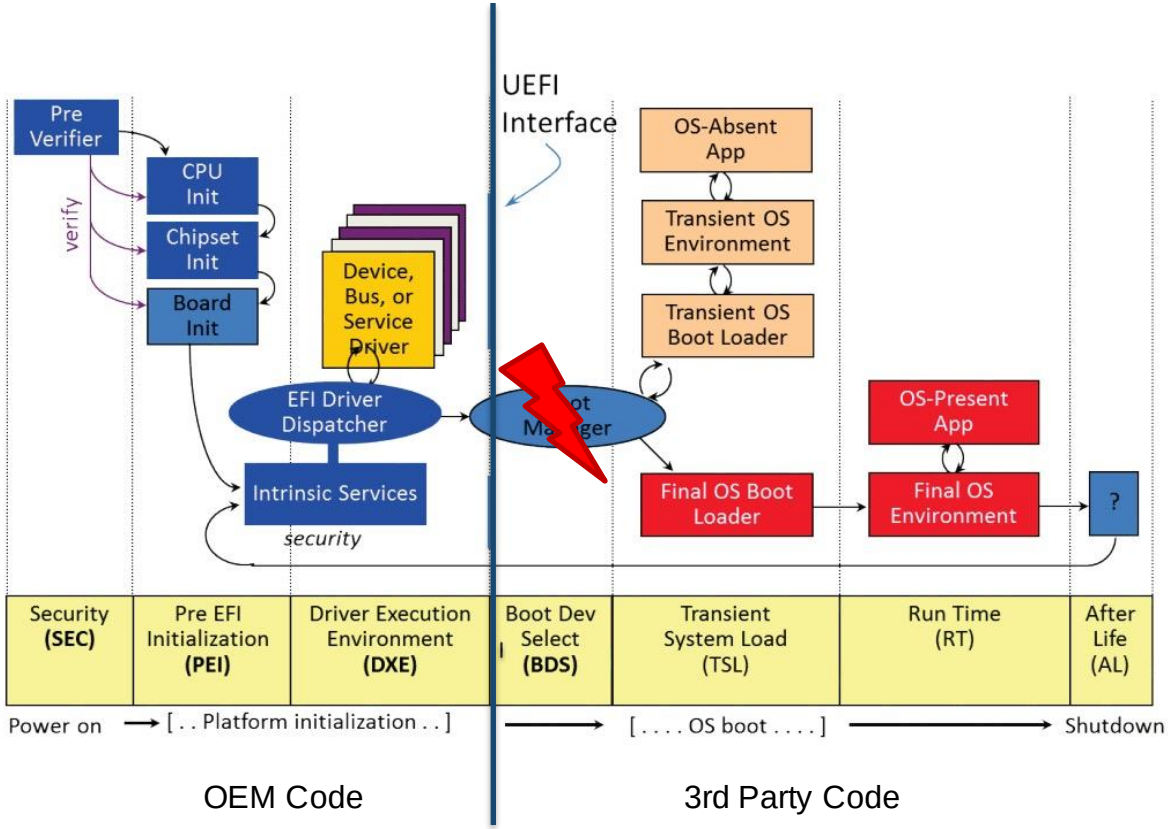
PspFlashAccSmmCommunicateHandler

- This SMI handler is installed for other DXE modules at boot-time
- It exposes various functions:
  - Get the flash block size
  - SPI read
  - SPI write
  - SPI erase
  - Deregister the SMI handler
- Marked it because it has various issues (CVE-2023-20576)

Bad Patterns in Abundance

```
signed __int64 __fastcall PspFlashAccSmmCommunicateHandler(EFI_HANDLE DispatchHandle, void *Context, void *CommBuffer, UINTN *CommBufferSize)
{
  AMD_ROM_ARMOR3_COMMBUFFER *_commBuff; // rbx@1
  signed __int64 v5; // rdi@1
  UINTN v7; // er10@3
  __int64 v8; // rax@10

  _commBuff = (AMD_ROM_ARMOR3_COMMBUFFER *)CommBuffer;
  v5 = 0x8000000000000003i64;
  if ( !(unsigned __int8 IsSmmBufferOutsideValid(CommBuffer, *(_QWORD *)CommBufferSize) )
    return 0x8000000000000002i64;
  DebugPrint(0x80000000, aPspflashaccsmm, LODWORD(_commBuff->Function));
  if ( LODWORD(_commBuff->Function) == 1 )
  {
    _commBuff->FlashAddress = 4096i64;
    return 0i64;
  }
  if ( LODWORD(_commBuff->Function) == 2 )
  {
    LODWORD(v8) = SPIReadData(
                    gPspFlashAccSmmCommReady_INterface,
                    LODWORD(_commBuff->FlashAddress),
                    LODWORD(_commBuff->Length),
                    &_commBuff[1]);
    return v8;
  }
  if ( LODWORD(_commBuff->Function) == 3 )
  {
    LODWORD(v8) = SPIWriteData(
                    gPspFlashAccSmmCommReady_INterface,
                    LODWORD(_commBuff->FlashAddress),
                    LODWORD(_commBuff->Length),
                    &_commBuff[1]);
    return v8;
  }
  if ( LODWORD(_commBuff->Function) == 4 )
  {
    DebugPrint(v7, aSpiEraseFlas_0, _commBuff->FlashAddress, _commBuff->Length);
    LODWORD(v8) = SPIErase(
                    gPspFlashAccSmmCommReady_INterface,
                    LODWORD(_commBuff->FlashAddress),
                    LODWORD(_commBuff->Length) >> 12);
```

Late Deregistration



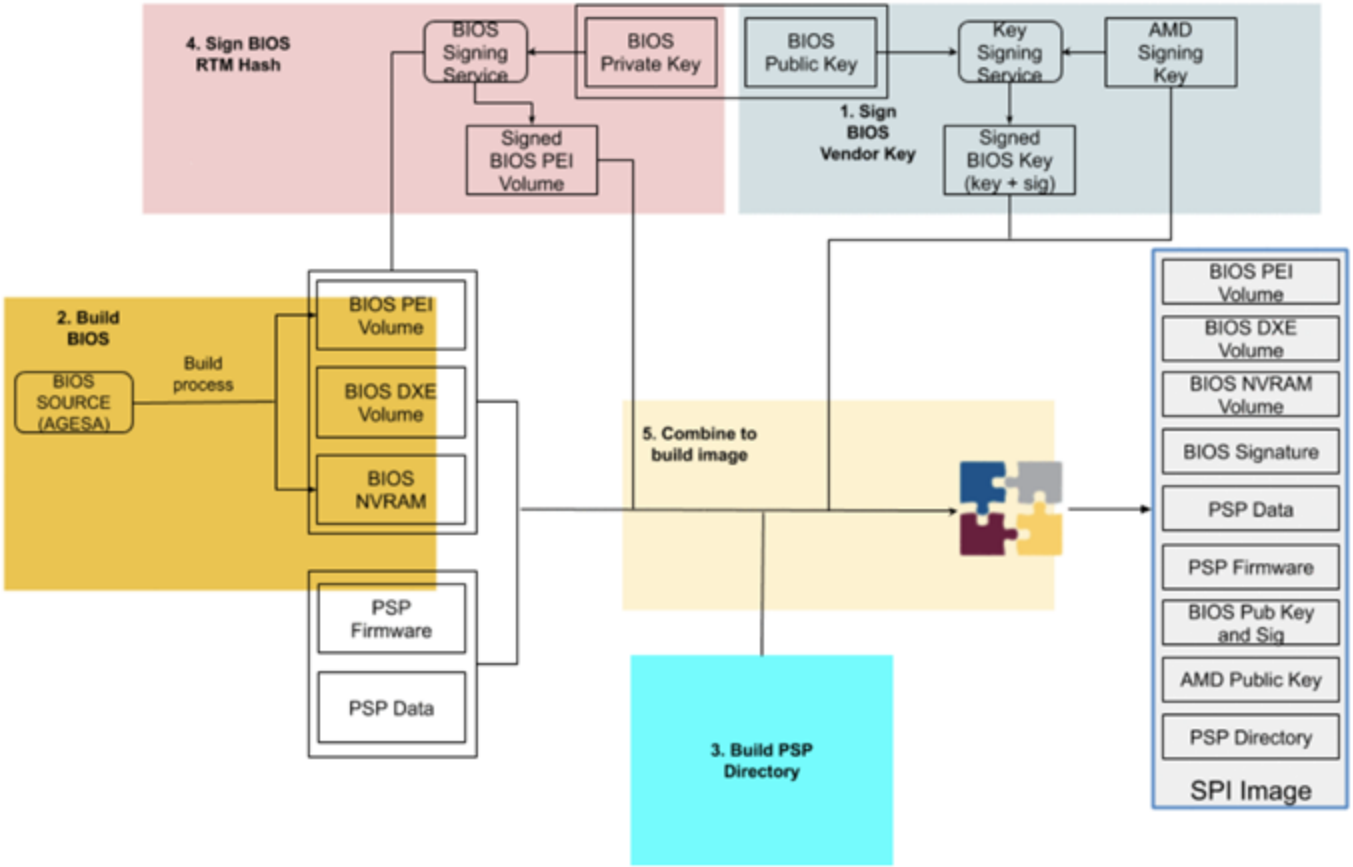
The deregistration is happening at *EXIT_BOOT_SERVICES* instead of *END_OF_DXE*

AMD Platform Secure Boot

General

- Hardware root-of-trust
- Provided by the PSP
- Verifies SEC+PEI phase
- Custom code for verifying DXE

- Here we only focus on configuration issues



Source: https://blog.cloudflare.com/anchoring-trust-a-hardware-secure-boot-story/

Finding the PSB Configuration

- Fetching the PSP MMIO base address
  - SMU index/data pair is used at 0xB8/0xBC of B00D00F00
  - Write:
    - 0x13E102E0 for fam 17h, model 30h+70h or fam 19h, model 20h
    - 0x13B102E0 for all other models

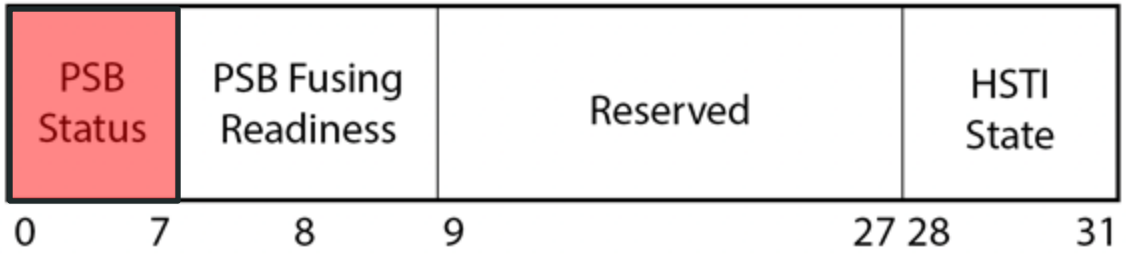
- Navigating to PSP registers:
  - PSB Fuse Register (at offset 0x10994)
  - PSB Status Register (at offset 0x10998)

PSB Fuse Register

Platform Vendor ID	Platform Model ID	BIOS Key Revision
0 – 7	8 – 11	12 – 15

Root Key Select	Res	PSB Enable	Anti Rollback	Disable AMD Key	Disable Secure Debug	Customer Key Lock	Res
16 – 19	20 – 23	24	25	26	27	28	29 – 31

PSB Status Register



Example of a good configuration

```
PS C:\Users\minium\Desktop\Platbox> .\compiled\platbox_cli cli
>>> chipset
Detected chipset:
=> Family: 19
=> Model: 44

[...]

PSP Config at +10994h: 11000290
 - Platform vendor ID: 90
 - Platform model ID: 02
 - BIOS key revision: 00
 - Root key select: 00
 - Platform Secure Boot Enable: 1
 - Disable BIOS key anti-rollback:  0
 - Disable AMD key usage:  0
 - Disable secure debug unlock:  0
 - Customer key lock:  1

PSP Config at +10998h: 70001000
 - PSB status: 00
 - PSB fusing readiness: 0
 - HSTI state: 07
```

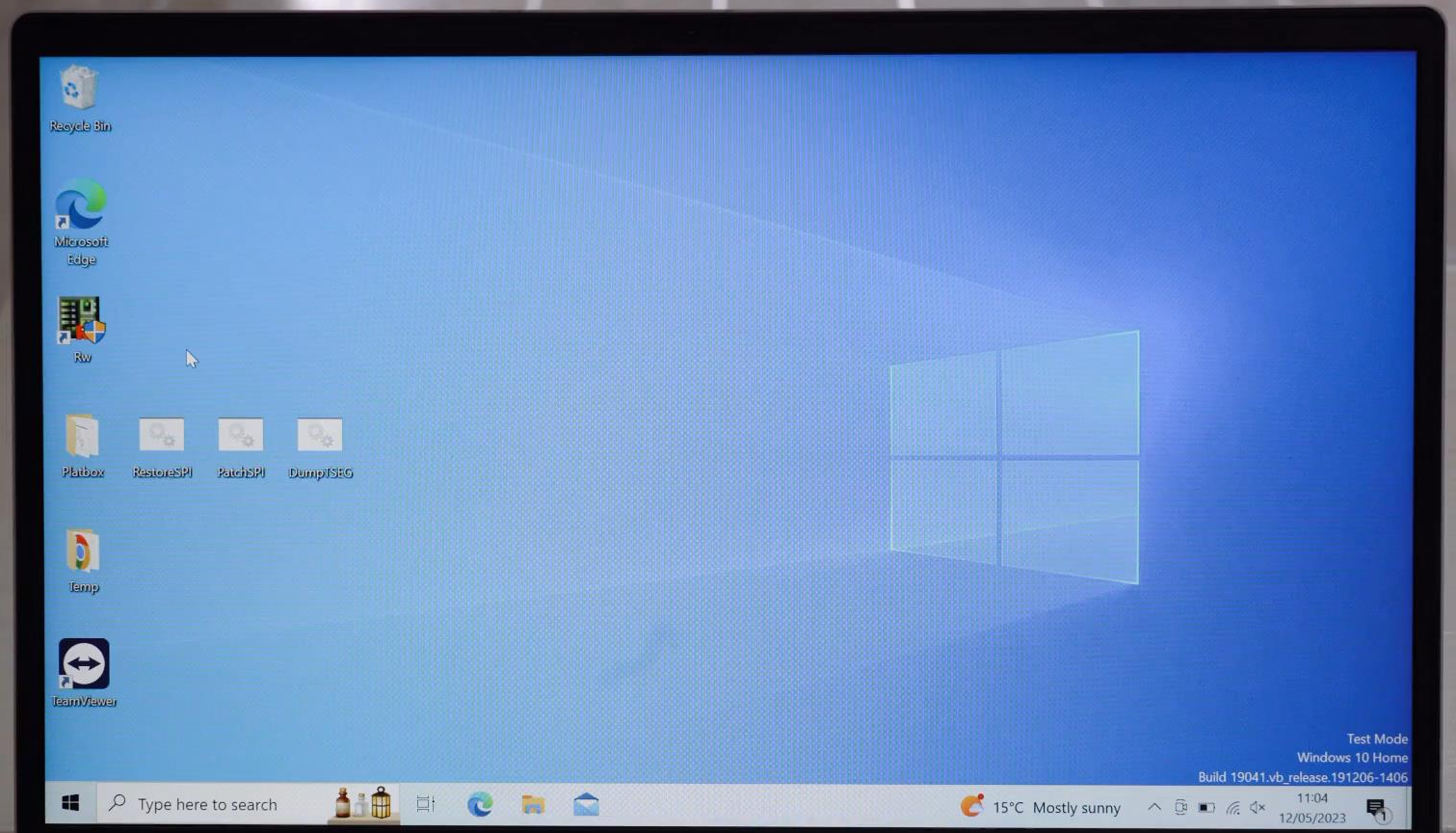
Model: Lenovo ThinkPad P16s

Example of a bad configuration

```
PS C:\Users\minium\Desktop\Platbox> .\compiled\platbox_cli cli
>>> chipset
Detected chipset:
=> Family: 17
=> Model: 60

[...]

PSP Config at +10994h: 00000000
 - Platform vendor ID: 00
 - Platform model ID: 00
 - BIOS key revision: 00
 - Root key select: 00
 - Platform Secure Boot Enable: 0
 - Disable BIOS key anti-rollback:  0
 - Disable AMD key usage:  0
 - Disable secure debug unlock:  0
 - Customer key lock:  0

PSP Config at +10998h: 5000c001
 - PSB status: 01
 - PSB fusing readiness: 0
 - HSTI state: 05
```

Model: Acer Swift 3 SF314-42

DEMO


Recycle Bin
Microsoft Edge
Rw
Platbox
RestoreSPI
PatchSPI
DumpTSEG
Temp
TeamViewer
Test Mode
Windows 10 Home
Build 19041.vb_release.191206-1406
Type here to search
15°C Mostly sunny
11:04
12/05/2023

acer

S W I F T

AMD SMM Supervisor

What is an SMM Supervisor?

- The System Transfer Monitor (STM) is a failed technology
  - Everyone waited years for a reference implementation from Intel. Maybe not enough market demand yet?
  - The technology was too complex
- SMM is too powerful and without an STM, the security guarantees of DRTM cannot be held
- The market went another direction
  - Intel: Intel Hardware Shield
  - AMD: AMD Supervisor
  - Microsoft: Project Mu
- Check Ilja van Sprundel presentation: Assessing the security of an SMM supervisor

ReadMe.rst

Project Mu MM Supervisor Repository

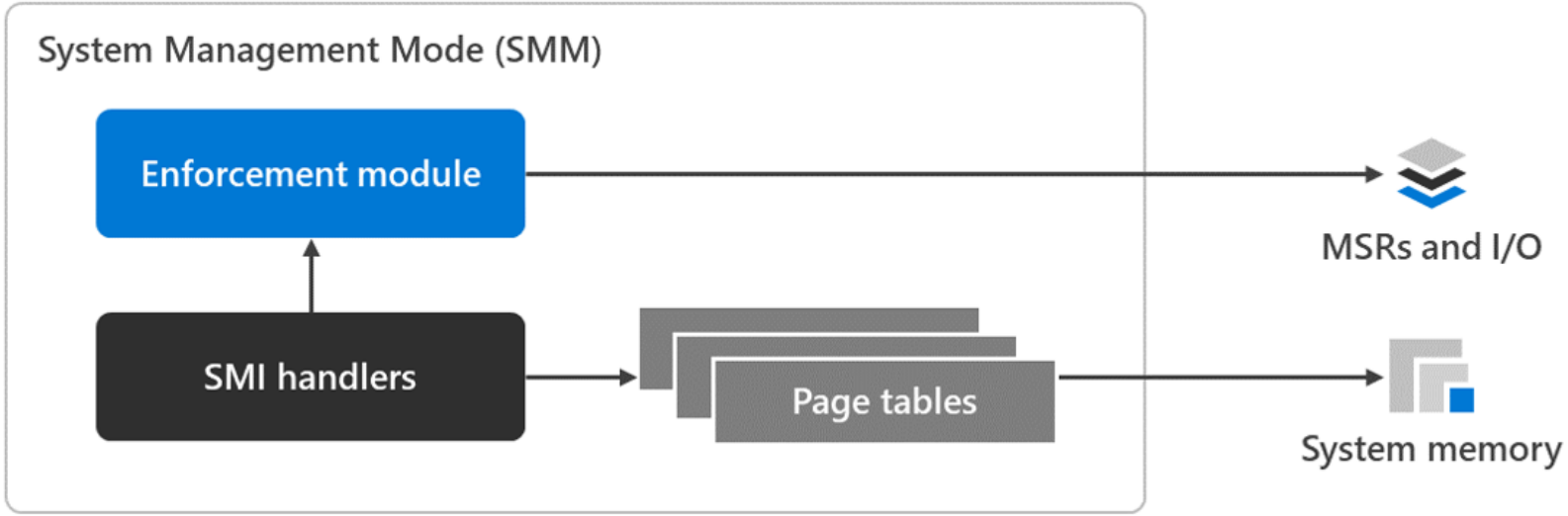
Host Type & Toolchain	Build Status	Test Status	Code Coverage
Windows_VS	Azure Pipelines succeeded	tests 19 passed	coverage coming soon
Ubuntu_GCC5	Azure Pipelines succeeded	tests 19 passed	coverage coming soon

This repository is part of Project Mu. Please see Project Mu for details https://microsoft.github.io/mu

This MM Supervisor feature repo contains the supervisor module under Standalone MM environment for X64 architecture, refactored from TianoCore common modules and public portion of AMD SMM supervisor module. The repo intends to support operating Standalone MM modules in a secure manner. Other peripheral libraries are also included to accomodate user module operations.

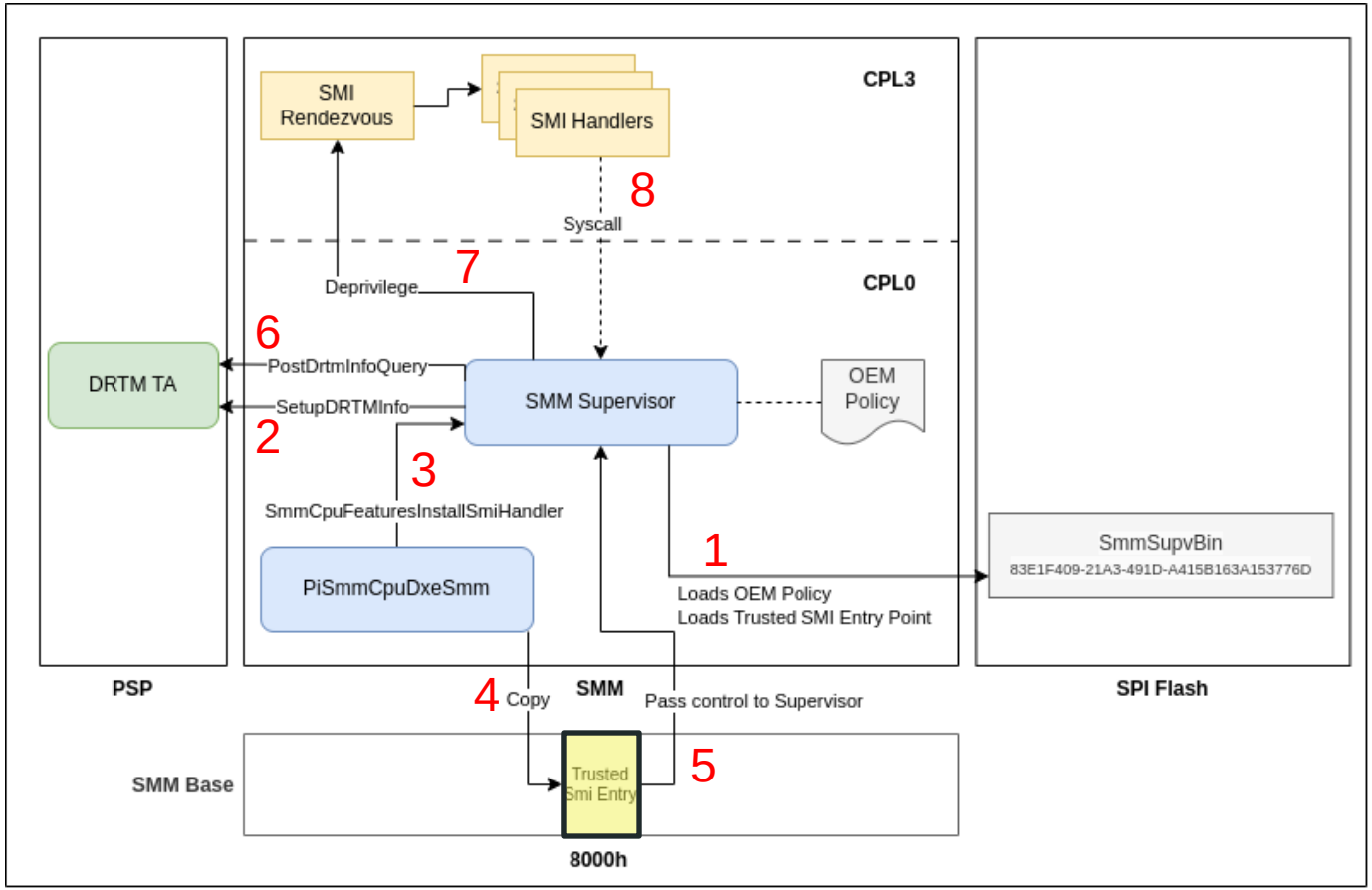
Source: https://github.com/microsoft/mu_feature_mm_supv

Resources Protected by Supervisor



SmmSupervisorBinRelease

- The SMM Supervisor implementation lives in the SmmSupervisorBinRelease SMM module
  - {D11C8E2A-3CD1-443C-AC09-63526DE7E170}
- There is also a Freeform Blob called SmmSupvBin
  - {83E1F409-21A3-491D-A415B163A153776D}
  - contains the OEM policy and other important assets



Example of a Demoted SMI Handler

```
do_rdmsr proc near
push    rbx
sub     rsp, 20h
mov     ebx, ecx
call    check_cpl
and     al, 3
cmp     al, 3
jnz     short loc_887F
```

```
mov     edx, ebx
xor     r9d, r9d
xor     r8d, r8d
xor     ecx, ecx
add     rsp, 20h
pop     rbx
jmp     do_syscall
```

```
loc_887F:
mov     ecx, ebx
rdmsr
shl     rdx, 20h
or      rax, rdx
add     rsp, 20h
pop     rbx
retn
do_rdmsr endp
```

Legacy Supervisor Syscalls

Syscall Num	Description	Project Mu	AMD
0	RDMSR (Read MSR)	X	X
1	WRMSR (Write MSR)	X	X
2	CLI (Clear Interrupt Flag)	X	X
3	IO_READ	X	X
4	IO_WRITE	X	X
5	WBINVD (Write back and invalidate cache)	X	X
6	HLT (CPU Halt)	X	X
7	SVST_READ (Reads Save State)	X	X
8	IHV_SUPV_READ (PSP Read Register) [0x10500-0x10b00]		X
9	IHV_SUPV_WRITE (PSP Write Register) [0x10500-0x10b00] [0x10a20-0x10aac]		X

Protection via Paging

- The supervisor sets the following structures and regions to be ring 0 only:
  - GDT
  - IDT
  - All Supervisor Heap Memory
  - Save-State Area
  - SMI Entry
  - IOMMU
  - Some PSP Registers
- At the end it sets HWCR::SmmPageLock ⇒ Locks CR3

Attack Surface

- Interrupt Handlers
- Syscalls
- PSP messages
- OEM Policy

SmmSupvBin Format

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	24	50	53	50	0B	62	8D	E0	05	00	00	00	10	04	00	20	$PSP.b.à.....
0010h:	C0	00	00	00	00	02	00	00	00	04	00	00	00	00	00	00	À.....
0020h:	C1	00	00	00	A4	02	00	00	00	06	00	00	00	00	00	00	Á...¤.....
0030h:	C2	00	00	00	00	02	00	00	00	09	00	00	00	00	00	00	Â.....
0040h:	C3	00	00	00	40	04	00	00	00	0B	00	00	00	00	00	00	Ã...@.....
0050h:	C4	00	00	00	48	01	00	00	00	10	00	00	00	00	00	00	Ä...H.....
0060h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

SmmSupvBin Format

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000h: 24 50 53 50 0B 62 8D E0 05 00 00 00 10 04 00 20 $PSP.b.à........
0010h: C0 00 00 00 00 02 00 00 00 04 00 00 00 00 00 00 À...............
0020h: C1 00 00 00 A4 02 00 00 00 06 00 00 00 00 00 00 Á...¤...........
0030h: C2 00 00 00 00 02 00 00 00 09 00 00 00 00 00 00 Â...............
0040h: C3 00 00 00 40 04 00 00 00 0B 00 00 00 00 00 00 Ã...@...........
0050h: C4 00 00 00 48 01 00 00 00 10 00 00 00 00 00 00 Ä...H...........
0060h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Magic: $PSP

SmmSupvBin Format

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	24	50	53	50	0B	62	8D	E0	05	00	00	00	10	04	00	20	$PSP.b.à.....
0010h:	C0	00	00	00	00	02	00	00	00	04	00	00	00	00	00	00	À.....
0020h:	C1	00	00	00	A4	02	00	00	00	06	00	00	00	00	00	00	Á...¤.....
0030h:	C2	00	00	00	00	02	00	00	00	09	00	00	00	00	00	00	Â.....
0040h:	C3	00	00	00	40	04	00	00	00	0B	00	00	00	00	00	00	Ã...@.....
0050h:	C4	00	00	00	48	01	00	00	00	10	00	00	00	00	00	00	Ä...H.....
0060h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Table Count: 0x0005

SmmSupvBin Format

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	24	50	53	50	0B	62	8D	E0	05	00	00	00	10	04	00	20	$PSP.b.à.....
0010h:	C0	00	00	00	00	02	00	00	00	04	00	00	00	00	00	00	À.....
0020h:	C1	00	00	00	A4	02	00	00	00	06	00	00	00	00	00	00	Á...¤.....
0030h:	C2	00	00	00	00	02	00	00	00	09	00	00	00	00	00	00	Â.....
0040h:	C3	00	00	00	40	04	00	00	00	0B	00	00	00	00	00	00	Ã...@.....
0050h:	C4	00	00	00	48	01	00	00	00	10	00	00	00	00	00	00	Ä...H.....
0060h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Type: 0xC0

SmmSupvBin Format

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	24	50	53	50	0B	62	8D	E0	05	00	00	00	10	04	00	20	$PSP.b.à.....
0010h:	C0	00	00	00	00	02	00	00	00	04	00	00	00	00	00	00	À.....
0020h:	C1	00	00	00	A4	02	00	00	00	06	00	00	00	00	00	00	Á...¤.....
0030h:	C2	00	00	00	00	02	00	00	00	09	00	00	00	00	00	00	Â.....
0040h:	C3	00	00	00	40	04	00	00	00	0B	00	00	00	00	00	00	Ã...@.....
0050h:	C4	00	00	00	48	01	00	00	00	10	00	00	00	00	00	00	Ä...H.....
0060h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Length: 0x200

SmmSupvBin Format

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	24	50	53	50	0B	62	8D	E0	05	00	00	00	10	04	00	20	$PSP.b.à.....
0010h:	C0	00	00	00	00	02	00	00	00	04	00	00	00	00	00	00	À.....
0020h:	C1	00	00	00	A4	02	00	00	00	06	00	00	00	00	00	00	Á...¤.....
0030h:	C2	00	00	00	00	02	00	00	00	09	00	00	00	00	00	00	Â.....
0040h:	C3	00	00	00	40	04	00	00	00	0B	00	00	00	00	00	00	Ã...@.....
0050h:	C4	00	00	00	48	01	00	00	00	10	00	00	00	00	00	00	Ä...H.....
0060h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Offset: 0x400

IOActive

SmmSupvBin Format

- 0xC0
  - Unknown Signature (512 bytes)
- 0xC1
  - Trusted SMM Entry Code
- 0xC2
  - Unkown Signature (512 bytes)
- 0xC3
  - Unkown (Short header, padding, then 512 bytes)
- 0xC4
  - OEM Supervisor Policy

Project Mu is a fork()

- Identified functions that are near the same as Project Mu:
  - Related to memory Policy Entries
    - GenMemPolicyAndShadowPageTable(..)
    - UpdateMemoryDesc(..)
    - PopulateMemoryPolicyEntries(..)
  - Validation
    - SecurityPolicyCheck(..)

SMM Supervisor Policy Analysis

```
1000h: 00 00 01 00 48 01 00 00 00 00 00 00 00 00 00 00 ....H...........
1010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1020h: 28 00 00 00 05 00 00 00 01 00 00 00 18 00 00 00 (...............
1030h: 02 00 00 00 A0 00 00 00 02 00 00 00 01 00 00 00 .... ...........
1040h: 01 00 00 00 18 00 00 00 03 00 00 00 B0 00 00 00 ............°...
1050h: 0C 00 00 00 01 00 00 00 01 00 00 00 18 00 00 00 ................
1060h: 04 00 00 00 10 01 00 00 03 00 00 00 00 00 00 00 ................
1070h: 01 00 00 00 18 00 00 00 05 00 00 00 28 01 00 00 ............(...
1080h: 02 00 00 00 00 00 00 00 01 00 00 00 18 00 00 00 ................
1090h: 01 00 00 00 48 01 00 00 00 00 00 00 01 00 00 00 ....H...........
10A0h: F8 0C 04 00 0A 00 00 00 FC 0C 04 00 02 00 00 00 ø.......ü.......
10B0h: 80 00 00 C0 01 00 07 00 81 00 00 C0 04 00 07 00 €..À.......À....
10C0h: 10 00 01 C0 01 00 07 00 11 01 01 C0 01 00 07 00 ...À.......À....
10D0h: D9 01 00 00 01 00 07 00 A0 0D 00 00 01 00 07 00 Ù....... .......
10E0h: A0 06 00 00 01 00 07 00 A2 06 00 00 01 00 07 00  .......¢.......
10F0h: A4 06 00 00 05 00 07 00 10 10 01 C0 01 00 07 00 ¤..........À....
1100h: 24 10 01 C0 01 00 07 00 18 10 01 C0 01 00 07 00 $..À.......À....
1110h: 00 00 04 00 00 00 00 00 01 00 04 00 00 00 00 00 ................
1120h: 02 00 04 00 00 00 00 00 00 00 00 00 10 00 00 00 ................
1130h: 02 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00 ................
1140h: 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF ........ÿÿÿÿÿÿÿÿ
```


IOActive

Project Mu to the Rescue

mu_feature_mm_supv / MmSupervisorPkg / Include / **SmmSecurePolicy.h**

Code | Blame | 320 lines (292 loc) · 17.1 KB

```c

// Index of SMM save state access conditions
typedef enum {
  SECURE_POLICY_SVST_UNCONDITIONAL   = 0,
  SECURE_POLICY_SVST_CONDITION_IO_RD = 1,
  SECURE_POLICY_SVST_CONDITION_IO_WR = 2,
  // Do not append after COUNT entry
  SECURE_POLICY_SVST_CONDITION_COUNT = 3
} SECURE_POLICY_SVST_CONDITION;

#pragma pack (push, 1)
typedef struct  {
  UINT32    Version;        // The version of this descriptor. Current Version is 1.
  UINT32    Type;           // The Type of this Parameter.
  UINT32    DescriptorSize; // The size of the descriptor in bytes including the header.
} SMM_SUPV_SECURE_POLICY_DESCRIPTOR_V1;

// SMM Supervisor Secure policy memory descriptor
typedef struct {
  SMM_SUPV_SECURE_POLICY_DESCRIPTOR_V1    Header;        // SMM_SUPV_SECURE_POLICY_DESCRIPTOR_TYPE_MEM
  UINT64                                  BaseAddress;   // Base address of memory
  UINT64                                  Size;          // Size of memory
  UINT32                                  MemAttributes; // Attributes of memory
} SMM_SUPV_SECURE_POLICY_MEM_DESCRIPTOR;
```


IOActive

SMM Supervisor Policy Analysis

```
Supervisor Policy Object
  Version: 65536
  Size: 248
  Policy Roots: 4
    Policy Root
      Version: 0x0000000000000001
      Type: IO
      Count: 0x0000000000000002
      Access Attribute: DENY
        Io Policy Entry
          IoAddress: 0x0CF8
          Size: 0x0004
          Attributes: WRITE | STRICT WIDTH
        Io Policy Entry
          IoAddress: 0x0CFC
          Size: 0x0004
          Attributes: WRITE
```

- Restrict I/O Write
  - 0x0CF8
  - 0x0CFC

SMM Supervisor Policy Analysis

```
Policy Root
  Version: 0x0000000000000001
  Type: MSR
  Count: 0x0000000000000009
  Access Attribute: DENY
    MSR Policy Entry
      MsrAddress: C0000080
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: C0000081
      Size: 0004
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: C0010010
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: C0010111
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: 000001D9
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: 00000DA0
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: 000006A0
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: 000006A2
      Size: 0001
      Attributes: READ | WRITE | EXECUTE
    MSR Policy Entry
      MsrAddress: 000006A4
      Size: 0005
      Attributes: READ | WRITE | EXECUTE
```

- Restrict MSR RWX
  - 0xC0000080 (Extended_Feature_Enable_EFER)
  - 0xC0000081 (SYSCALL_Target_Address_STAR)
  - 0xC0010010 (SYSCFG)
  - 0xC0010111 (SMM Base Addr)
  - 0x000001D9 (DBG_CTL)
  - 0x000006A0 (PERF_CTL)
  - 0x000006A2 (PERF_CTR0)
  - 0x000006A4 (PERF_CTR2)

SMM Supervisor Policy Analysis

```
Policy Root
  Version: 0x0000000000000001
  Type: INSTRUCTION
  Count: 0x0000000000000003
  Access Attribute: ALLOW
    Instruction Policy Entry
      Instruction: CLI
      Attributes: EXECUTE
    Instruction Policy Entry
      Instruction: WBINVD
      Attributes: EXECUTE
    Instruction Policy Entry
      Instruction: HLT
      Attributes: EXECUTE
Policy Root
  Version: 0x0000000000000001
  Type: MEMORY
  Count: 0x0000000000000000
  Access Attribute: DENY
```

- Allow Instructions
  - CLI
  - WBINVD
  - HLT
- Deny Memory Access
  - N/A

SMM Supervisor Policy Analysis

- Comparisons between multiple vendor policies
  - MSR
    - 0xC0011010
    - 0xC0011024
    - 0xC0011018

SMM Supervisor Policy Analysis

- MSR Concerns
  - FS_BASE
  - GS_BASE
  - KernelGSBase
- Other concerns
  - Access to PSP mailbox is unrestricted
  - Could this be used for attacking the supervisor?

Conclusions (1/3)

- We have reported 20+ vulnerabilities
  - AMD modules affected
  - IBV modules affected
    - AMI
    - Insyde
    - Phoenix
  - OEM specific
    - Acer
    - Asus
    - Dell
    - Huawei
    - Lenovo

Conclusions (2/3)

- OEMs are regularly failing to provide a secure configuration for their AMD platforms
  - AMD provides the features to correctly lock down the platform
    - Most issues reported could be fixed with trivial configuration change
  - Security researches appear to primarily focus only on Intel based platforms
  - AMD implementing NDA on platform documentation is likely what caused this pattern

Conclusions (3/3)

- The industry is moving towards isolating SMM and deprivileging the OEMs SMI Handlers
- We feel technologies such as Microsoft System Guard and Secure Launch likely to be the default soon

Call to Action

- Test your own system with Platbox
- Share the results if you see anything interesting

- Platbox is available at:
  https://github.com/IOActive/Platbox

Questions?