

# DATA-ONLY ATTACKS AGAINST UEFI BIOS

Alex Ermolov



# Binarly REsearch Team

Alex Metrosov



@matrosov

Yegor Vasilenko



@yeggorv

Alex Ermolov



@Flothrone

Sam Thomas



@xorpsse

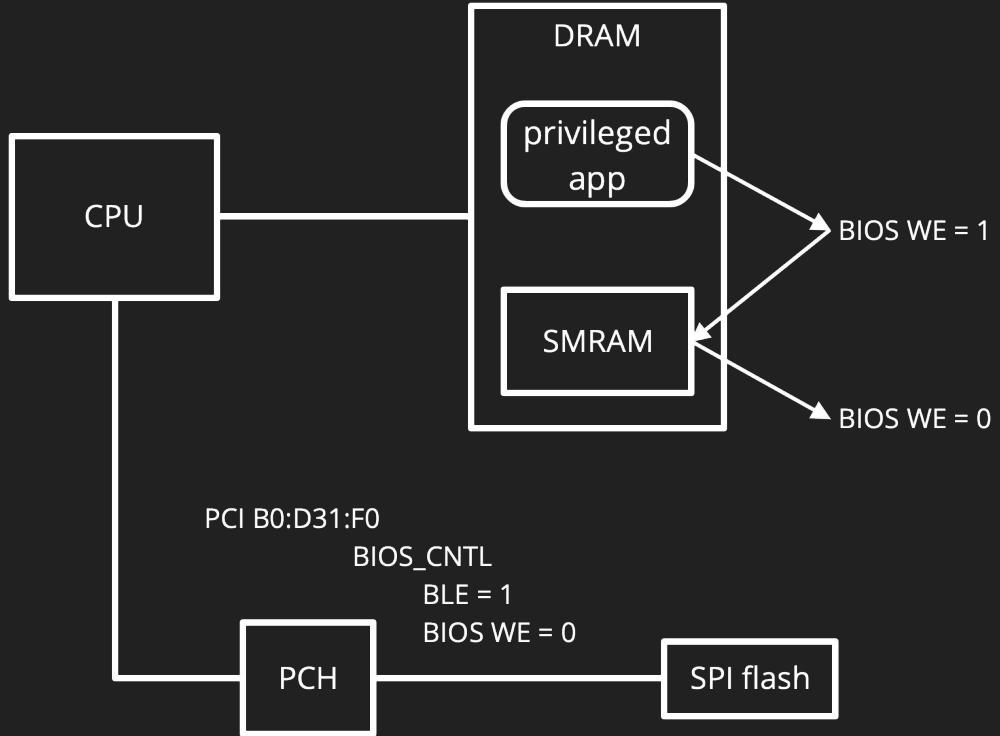
- UEFI BIOS Security
- SMM Mitigations
- Is that enough?
- Vulns
- NVRAM stories
- Breaking Firmware Trust from Pre-EFI (2nd talk)

# **UEFI BIOS SECURITY**

# UEFI BIOS common security

## BIOS Lock Enable (BLE)

If BLE = 1, any attempt to set BIOS WE = 1 (required before starting flash operation cycles in SPIBAR) will trigger hardware SMI handler to set BIOS WE = 0.



[OpenSecurityTraining by Xeno Kovah & Corey Kallenberg, LegbaCore LLC](#)

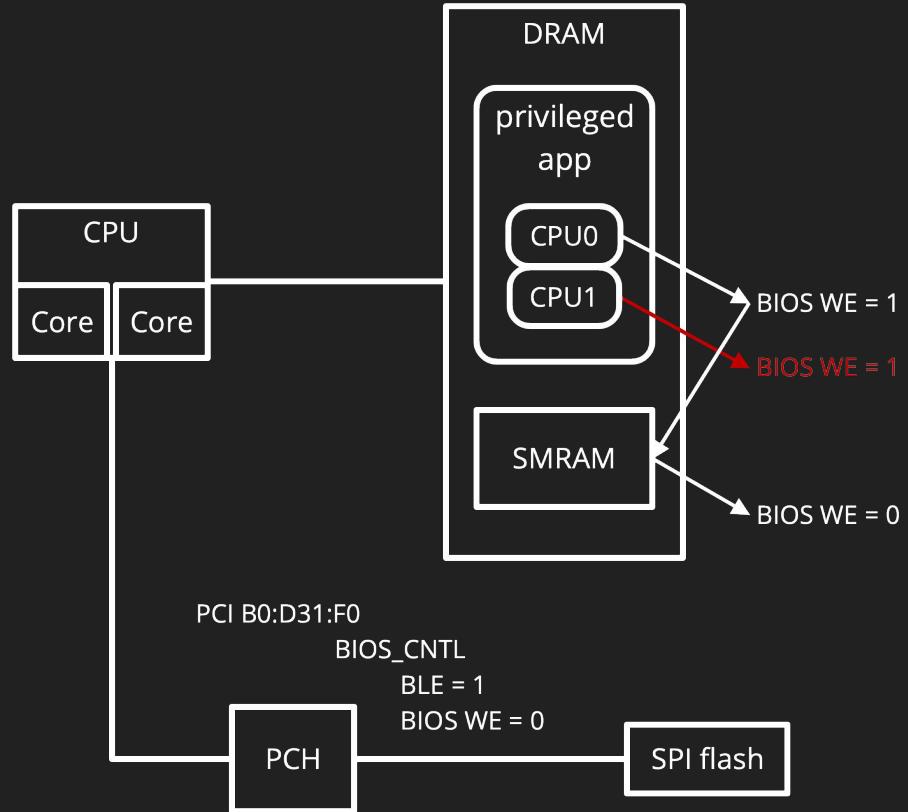
**BTW check this out!** [OpenSecurityTraining2 by Xeno Kovah](#)

# UEFI BIOS common security

## BIOS Lock Enable (BLE)

On multicore systems this protection is vulnerable to a race condition attack (aka Speed Racer).

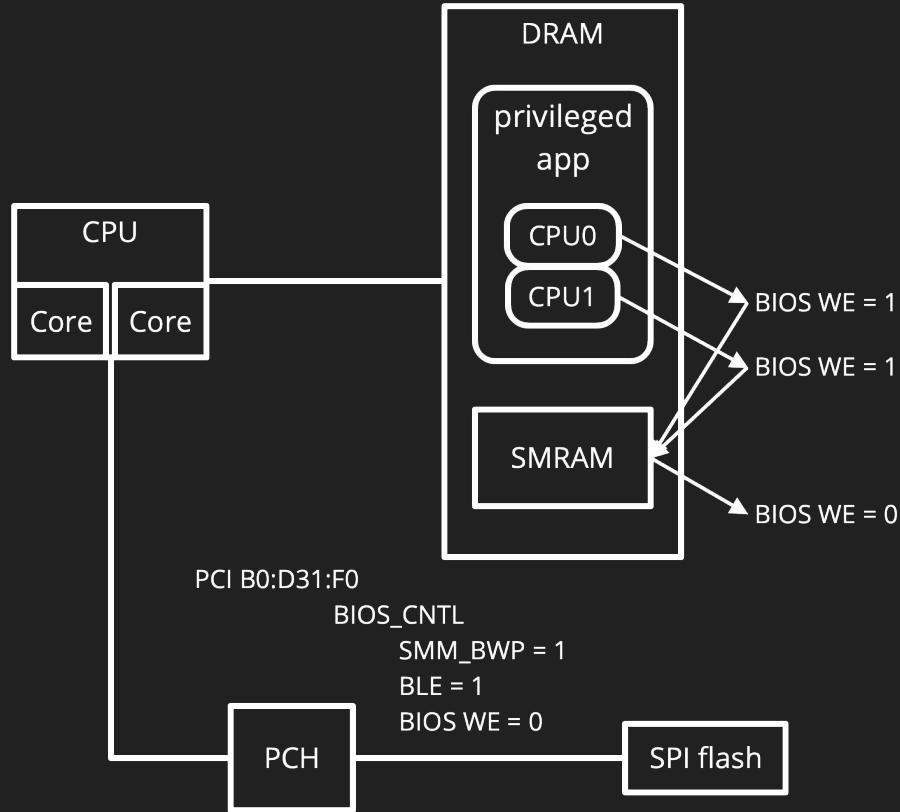
While one CPU Core is in SMM and the other still not, there is a small time window to write to BIOS region before it will be switched to SMM.



# UEFI BIOS common security

## SMM BIOS Write Protect (SMM\_BWP)

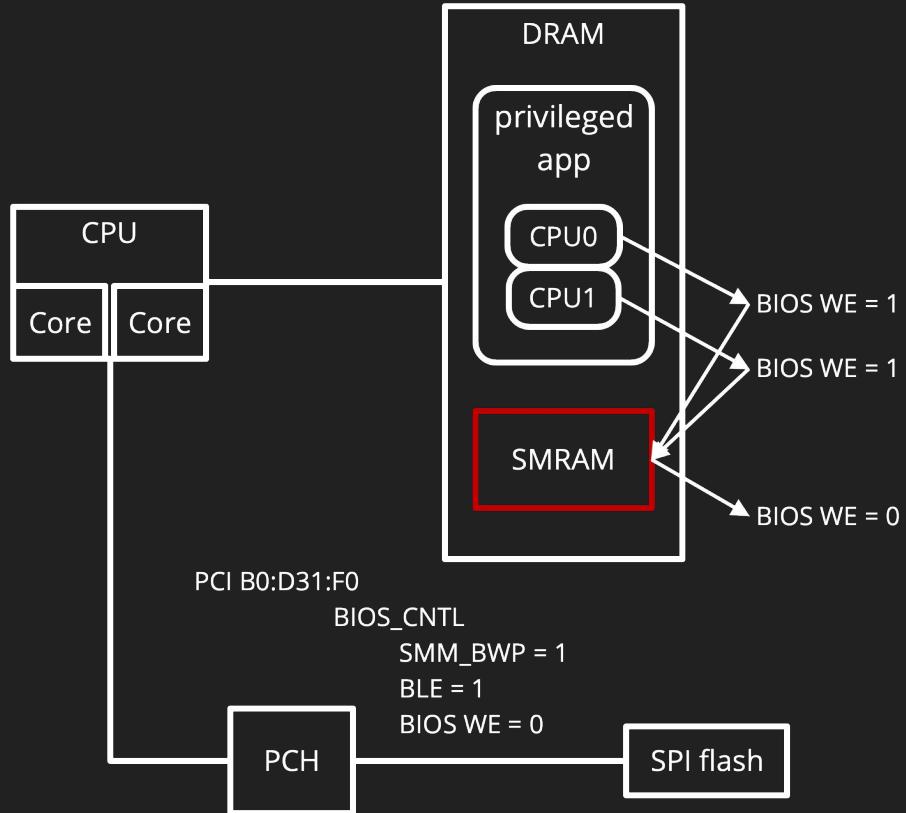
If SMM\_BWP = 1, BIOS is not writable unless all processors are in SMM.



# UEFI BIOS common security

## BLE & SMM\_BWP

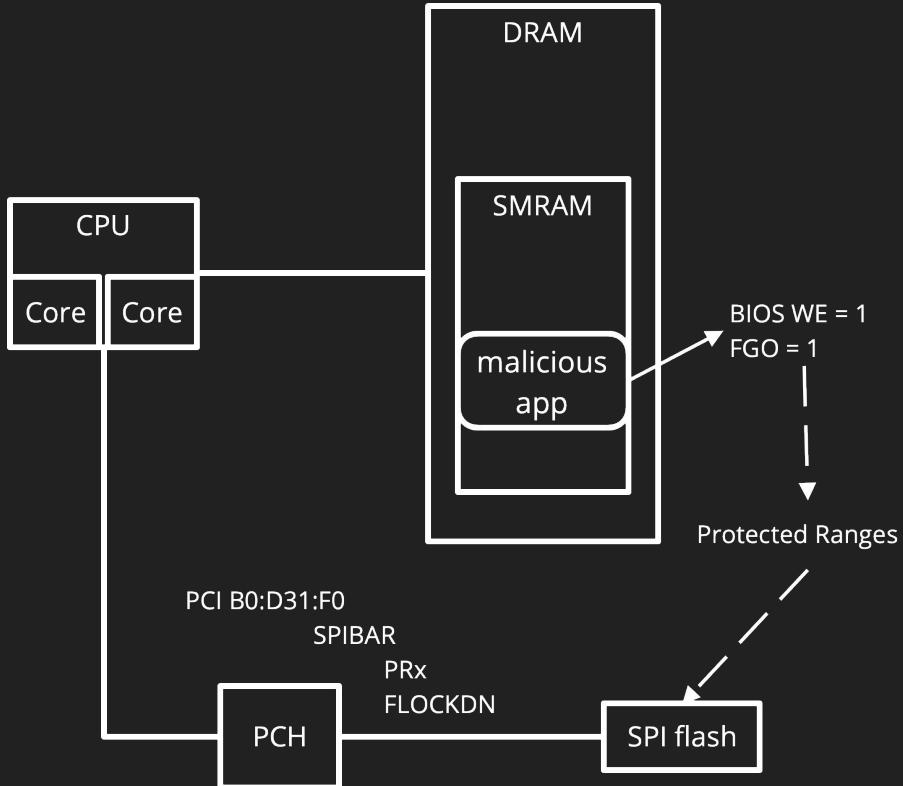
SMM-based protections makes SMM code as a main target, gaining arbitrary code execution will bypass the protections.



# UEFI BIOS common security

## Protected Ranges

PRx registers allows to mark certain BIOS regions as read-only. This configuration is locked by Flash Configuration Lock Down (FLOCKDN) bit.

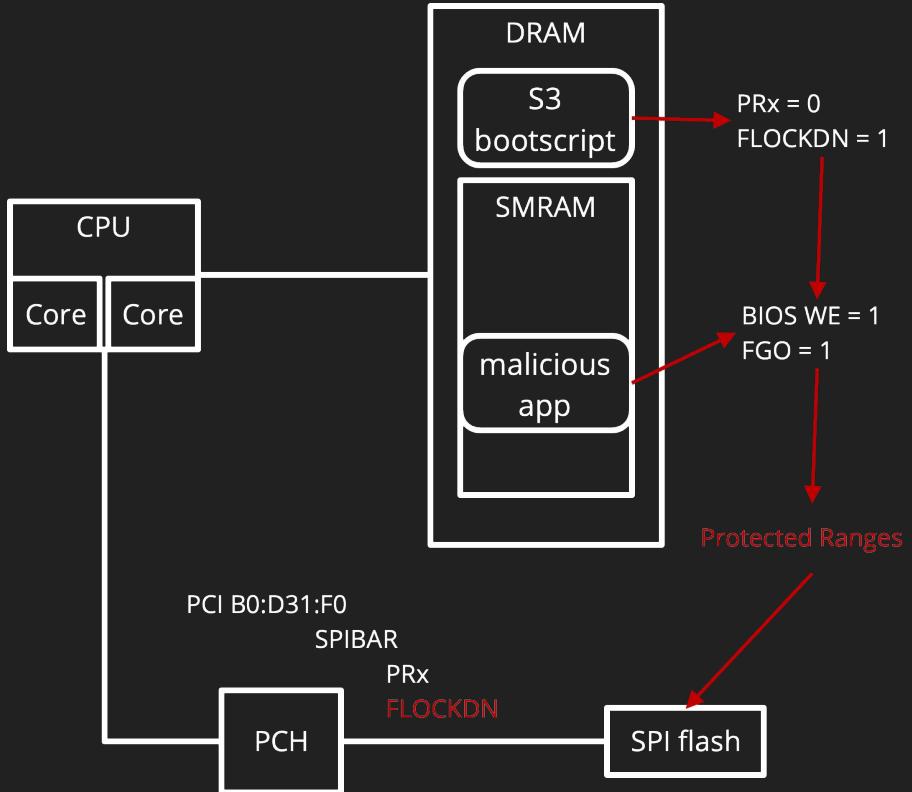


# UEFI BIOS common security

## Protected Ranges

Potential bypasses:

- Misconfigurations
  - FLOCKDN not set
  - Not all code-containing BIOS regions are marked
- S3 bootscript attacks
- S3 boot path attacks (SMM)
- BIOS update procedure



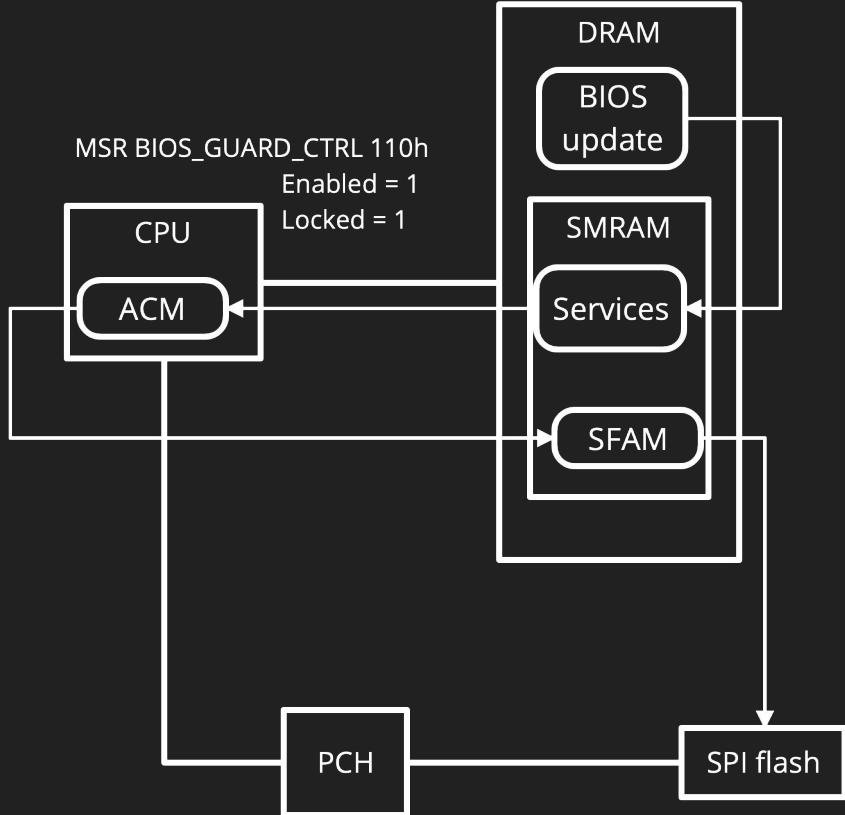
# UEFI BIOS hardened security

## Intel BIOS Guard

Aka Platform Flash Armoring Technology (PFAT), small trusted component granted for write operations onto SPI.

It is called Intel BIOS Guard Authenticated Code Module (ACM).

Signed Flash Address Map (SFAM) structure describes what regions are marked as Signed – means, a valid certificate is required to update them.



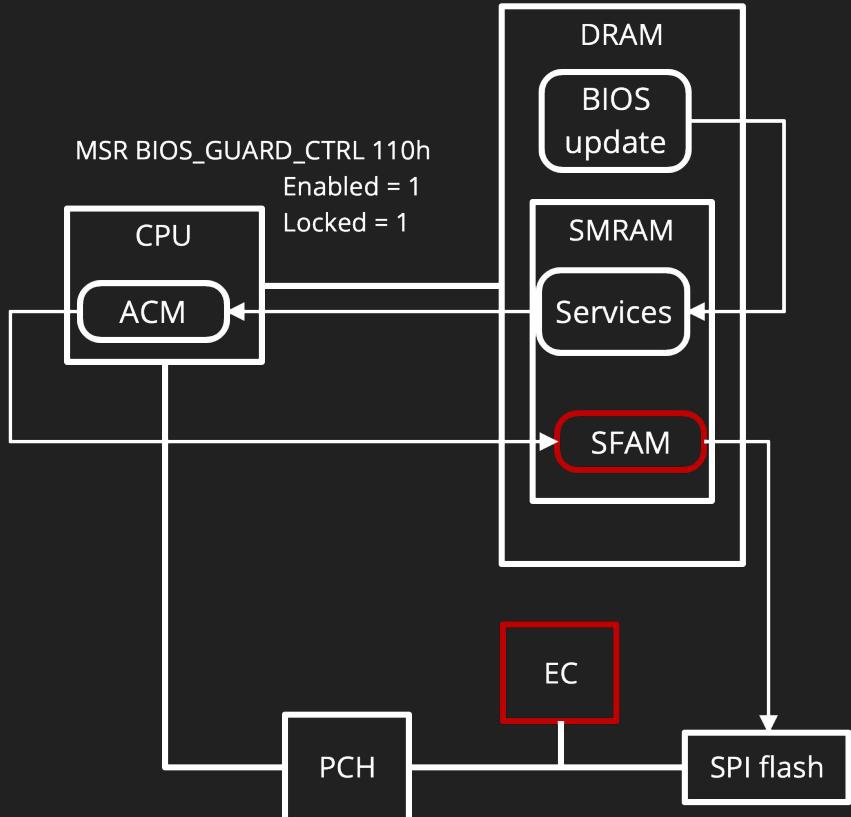
[Untrusted Roots: Exploiting Vulnerabilities in Intel ACMs by Alexander Ermolov & Dmitriy Frolov](#)

# UEFI BIOS hardened security

## Intel BIOS Guard (BiG)

Potential bypasses:

- SFAM misconfigurations
- ACPI Embedded Controller (EC) with access to SPI
- Downgrade ACM to vulnerable version (1day)
- Breaking logic of BiG initialization procedure



[Breaking Through Another Side. Bypassing Firmware Security Boundaries from Embedded Controller](#)  
by Alex Matrosov & Alexandre Gazet

# UEFI BIOS hardened security

## Intel BIOS Guard disable

PlatformInitPreMem EEEE611D-F78F-4FB9-B868-55907F169280:

```
void sub_FFFA4C4F(int a1):
{
    *(_DWORD *)(*(_DWORD *) (a1 + 0x1C) + 4) = CallocPool(0x14);

    ...

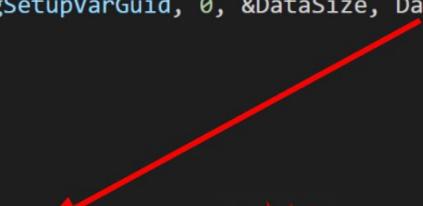
    GetPpi(gReadOnlyVarPpiGuid, &ReadOnlyVarPpi);

    DataSize = 0xFFD;
    result = (*ReadOnlyVarPpi)(ReadOnlyVarPpi, L"CpuSetup", gSetupVarGuid, 0, &DataSize, Data);

    ...

    v4 = *(_DWORD *) (v1 + 0xC);

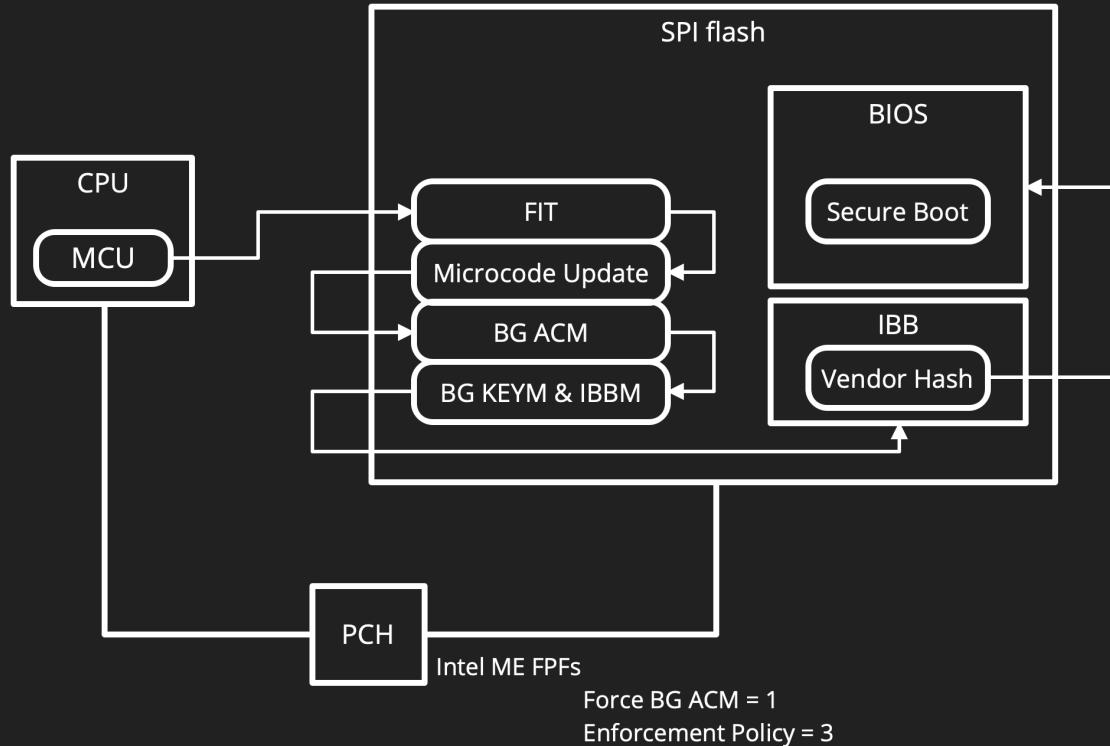
    if ( v4 )
        *(_DWORD *) (v4 + 0x50) ^= (*(_DWORD *) (v4 + 0x50) ^ Data[0x167]) & 1;
}
```



# UEFI BIOS hardened security

## Intel Boot Guard

Hardware-supported BIOS verification  
mechanism.



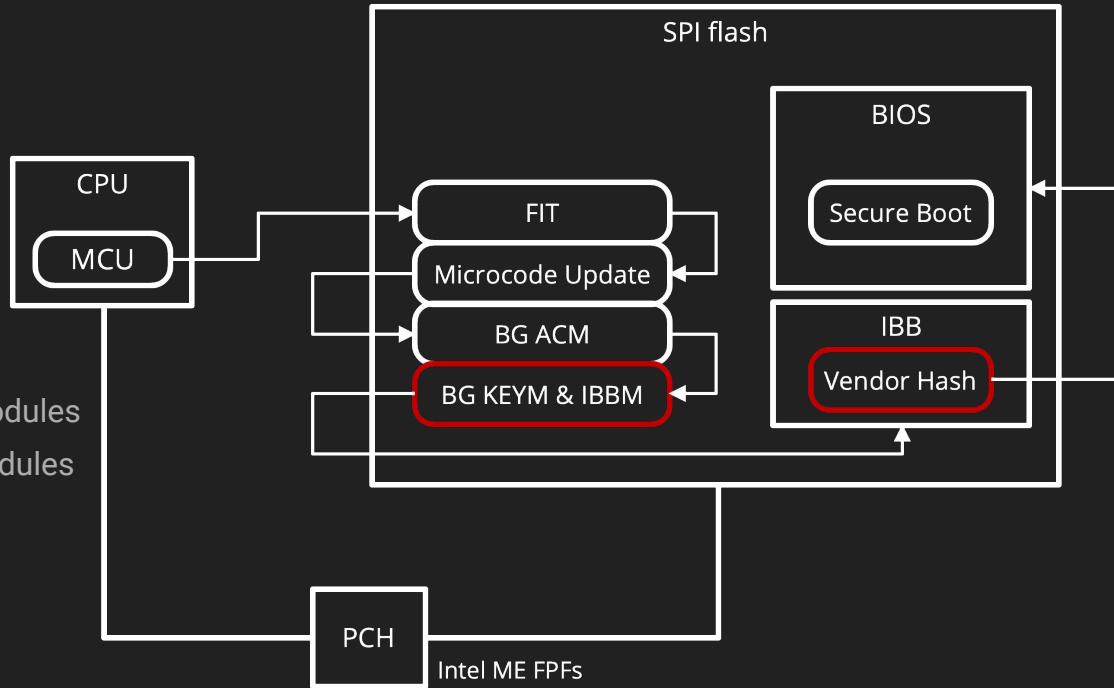
[Betraying the BIOS: where the guardians of the BIOS are failing by Alex Matrosov](#)

# UEFI BIOS hardened security

## Intel Boot Guard

Potential bypasses:

- Misconfigurations
  - Fuses are not set
  - IBB hash is not covering all SEC/PEI modules
  - Vendor hash is not covering all DXE modules
  - Downgrade MCU to vulnerable version



[Betraying the BIOS: where the guardians of the BIOS are failing by Alex Matrosov](#)

[Bypassing Intel Boot Guard by Alexander Ermolov](#)

[TOCTOU Attacks Against BootGuard by Peter Bosch & Trammell Hudson](#)

[\[BRLY-2021-002\] LENOVO SYSTEM FIRMWARE HAS MISSING COVERAGE WITH BOOT GUARD PROTECTED RANGES \(IBB\) FOR UEFI MODULES](#)

Force BG ACM = 1

Enforcement Policy = 3

# SMM MITIGATIONS

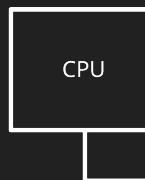
# UEFI BIOS mitigated security

## SMM Nx-bit & Sanity Checks

The most known and widely-used mitigation, protects from SMM call-outs to non-SMRAM.

- ValidateMemoryBuffer()
- ValidateMmioBuffer()
- ValidateSmramBuffer()

```
MSR SMM_FEATURE_CONTROL    4E0h
                  SMM_Code_Chk_En = 1
MSR SMM_MCA_CAP          17Dh
                  SMM_Code_Access_Chk = 1
MSR IA32_SMRR_PHYSBASE   1F2h
MSR IA32_SMRR_PHYSMASK   1F3h
```

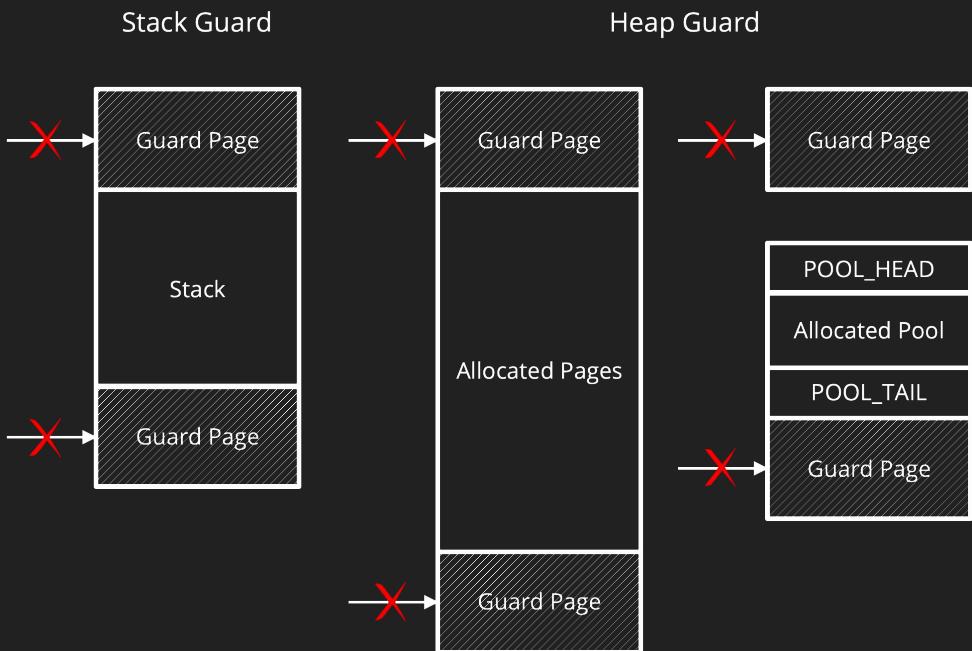


# UEFI BIOS mitigated security

## IA32 (SMM) Stack/Heap Guard

Guard Page is created before and after the allocation, accessing them will cause **Page Fault**.

- For Stack it is a production-feature  
Disabled by default
- For Heap it is a debug-feature  
Two different builds required for Pool to get  
Guard page close to POOL\_HEAD and POOL\_TAIL.

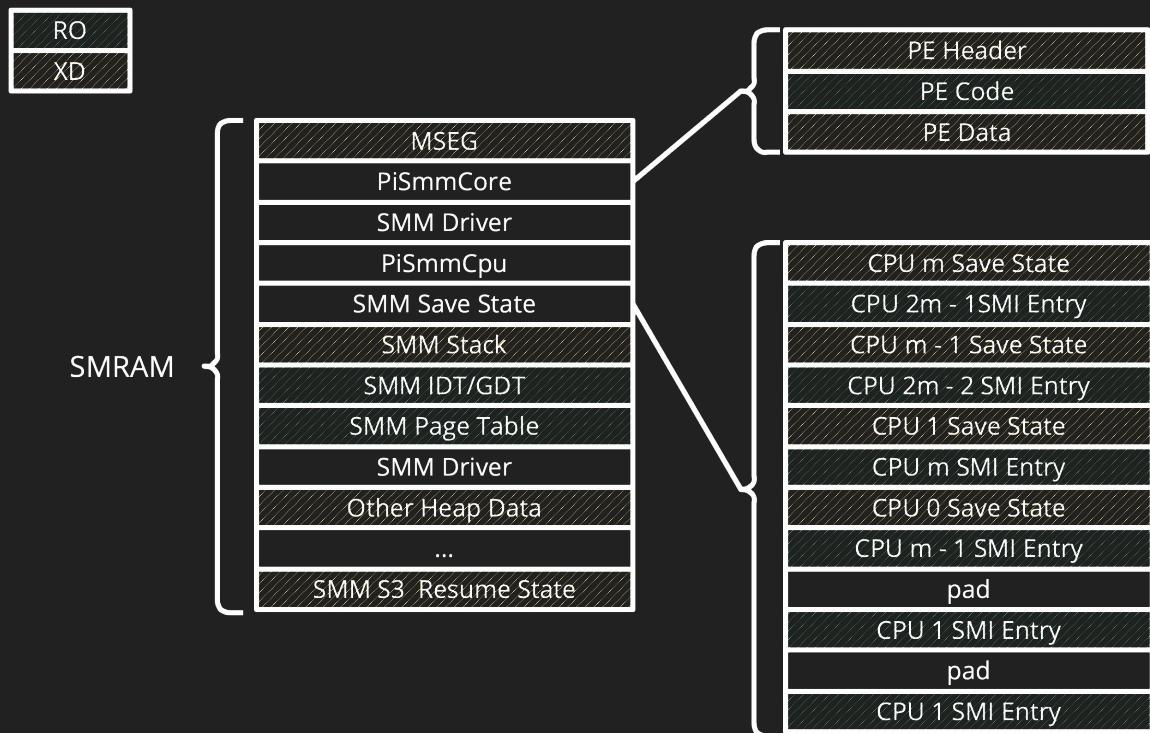


# UEFI BIOS mitigated security

## SMRAM memory protection (RO/XD)

Violating Read-Only / Execute-Disable  
attributes will cause **Segmentation Fault**.

- Page Table is dynamic by default
- Disabled by default
- ROP-attackable

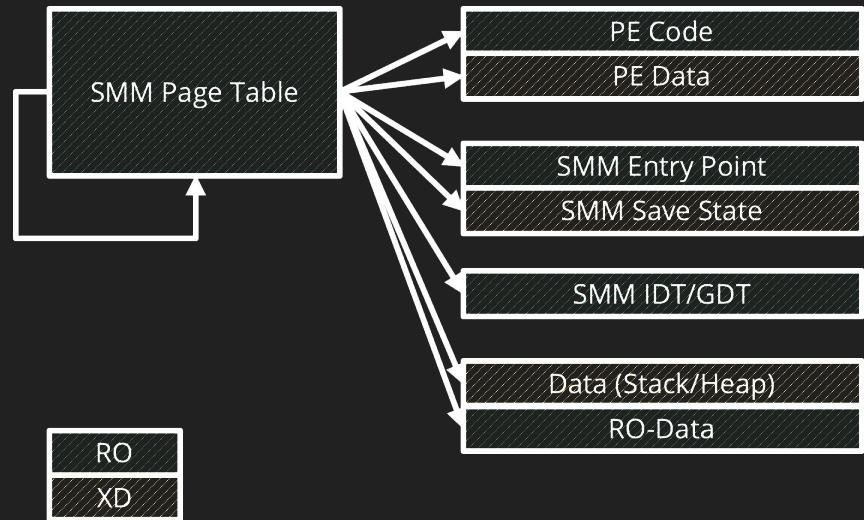


# UEFI BIOS mitigated security

## SMM static page table (RO)

SMRAM Protection + Static Page Table.

- Not compatible with Heap Guard
- Disabled by default
- ROP-attackable



- Address Space Layout Randomization (ASLR)

## ASLR requirement in UEFI firmware

The current EDK II code does not support address space randomization. The memory allocation algorithm is top-down. In order to support the randomization in the pre-boot environment, we define below requirement:

- Intel CET (Control-flow Enforcement Technology)

exists only here:

[A Tour Beyond BIOS - Security Enhancement to Mitigate Buffer Overflow in UEFI by Jiewen Yao, Vincent J. Zimmer, Jian Wang](#)

[A Technical Look at Intel's Control-flow Enforcement Technology by Baiju V Patel](#)

# UEFI BIOS mitigated security

## Windows SMM Security Mitigation Table (WSMT)

This is a static ACPI table configured by vendor and stored in system memory.

<https://twitter.com/matrosov/status/1401656449965596673?s=20>



Alex Matrosov  
@matrosov

WSMT (Windows SMM Security Mitigation Table) has a static nature by design which leads the results like that. The mitigation is enabled doesn't mean this mitigation is used or configured correctly.

[Перевести твит](#)

Assaf Carlsbad @assaf\_carlsbad · 6 июн.

UEFI firmware: All my SMI handlers will check pointers nested in the communication buffer to make sure they don't overlap with SMRAM.  
SMI handler: Just pass me a pointer in the communication buffer and I'll take care of writing there.

The WSMT is a lie.

```
[*] Administrator: po6-gt - chips x + 
C:\Users\carlsbad\Code\chipsec [wsmt_module *]> python \chipsec_main.py --no
WARNING: ****
WARNING: Chipsec should only be used on test systems!
WARNING: It should not be installed/deployed on production end-user systems.
WARNING: See https://github.com/0x49/Chipsec_Live_Easy for more information.
WARNING: ****

[CHIPSEC] API mode: using CHIPSEC kernel module API
[*] loaded chipsec.modules.common.wsmt
[*] running loaded modules ...
[*] running module: chipsec.modules.common.wsmt
[*] module: WSMT Configuration
[*] unmounting modules ...

WINDOWS SMM Mitigations Table (WSMT) Contents
FIXED_COMM_BUFFERS : True
COMM_BUFFER_NESTED_PTR_PROTECTION : True
SYSTEM_RESOURCE_PROTECTION : True

[*] PASSED: WSMT table is present and reports all supported mitigations

[CHIPSEC] ***** SUMMARY *****
[CHIPSEC] Time elapsed 0.222
[CHIPSEC] Modules passed 0:
[CHIPSEC] Modules failed 0:
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules not implemented 0:
[CHIPSEC] Modules not applicable 0:
[CHIPSEC] *****

C:\Users\carlsbad\Code\chipsec [wsmt_module *]>
```

**IS THAT ENOUGH?**

# Updated attack surface

## NVRAM

Definitely not enough about secure operations with it.

- R/W area
- Not verified
- Not measured

	Image	Intel
Descriptor region	Region	Descriptor
GbE region	Region	GbE
ME region	Region	ME
▼BIOS region	Region	BIOS
➤FA4974FC- <b>AF1D-4E5D-BDC5-DACD6D27BAEC</b>	Volume	FFSv2
➤FA4974FC- <b>AF1D-4E5D-BDC5-DACD6D27BAEC</b>	Volume	FFSv2
Padding	Padding	Empty (0xFF)
➤4F1C52D3-D824-4D2A-A2F0-EC40C23C5916	Volume	FFSv2
➤AFDD39F1-19D7-4501-A730-CE5A27E1154B	Volume	FFSv2
Pad-file	File	Pad
Pad-file	File	Raw
➤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		

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	01	01	00	..xÿ	.....
0020	00	00	80	D8	FF	00	00	00	00	00	00	00	00	00	01	01	00	..Eÿ	.....
0030	00	00	DA	FF	00	00	00	00	00	00	00	00	00	00	01	01	00	..Uÿ	.....
0040	00	00	F1	FF	00	00	00	00	00	00	00	00	00	00	01	02	00	..ñÿ	.....
0050	00	00	80	DC	FF	00	00	00	00	00	00	00	00	00	01	0B	00	..Üÿ	.....
0060	00	00	CC	FC	FF	00	00	00	00	00	00	00	00	00	01	0C	00	.Iüÿ	.....

Updated attack surface



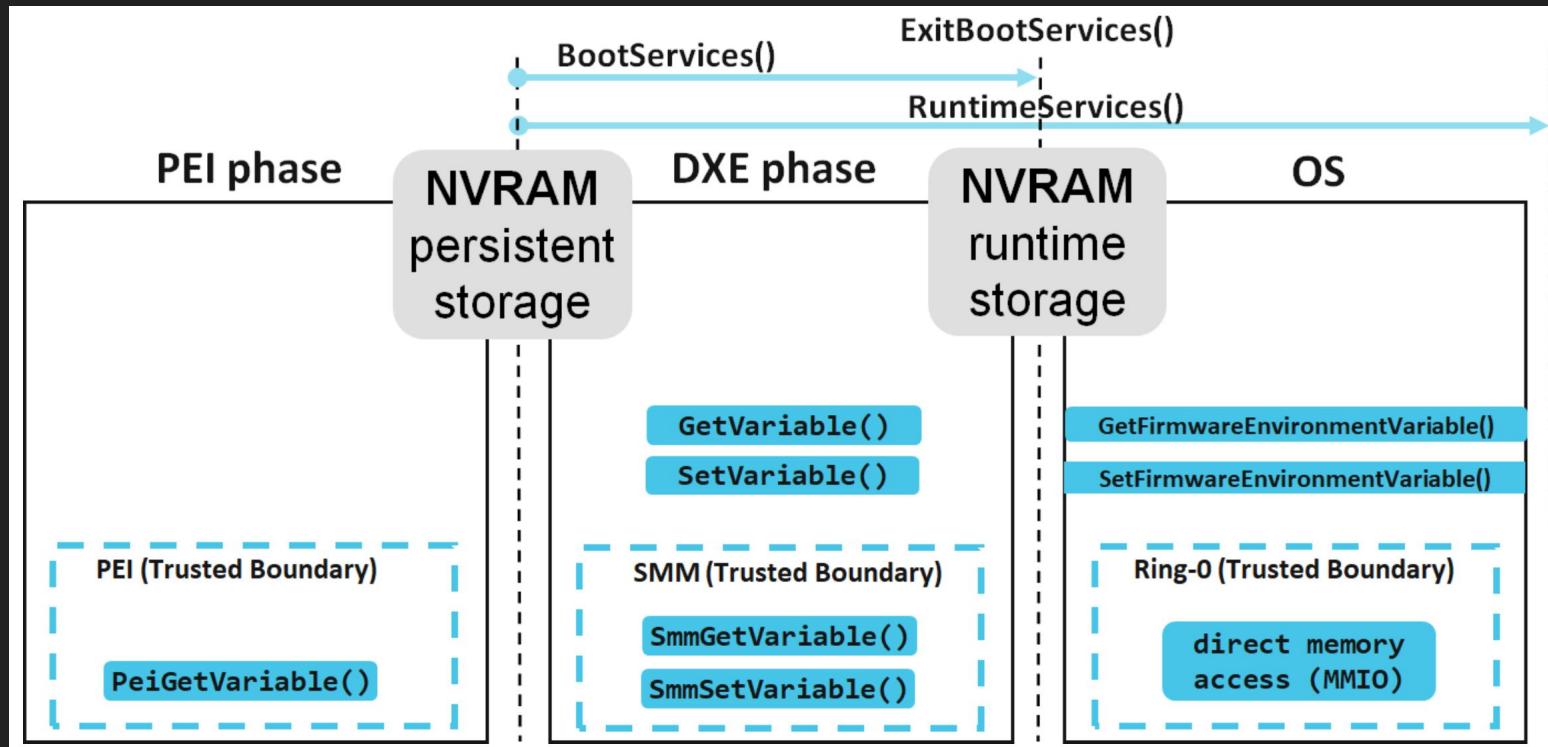
# Safeguarding UEFI Ecosystem: Firmware Supply Chain is Hard(coded)

Alex Tereshkin, Alex Matrosov, Adam 'pi3' Zabrocki

#BHUSA @BlackHatEvents

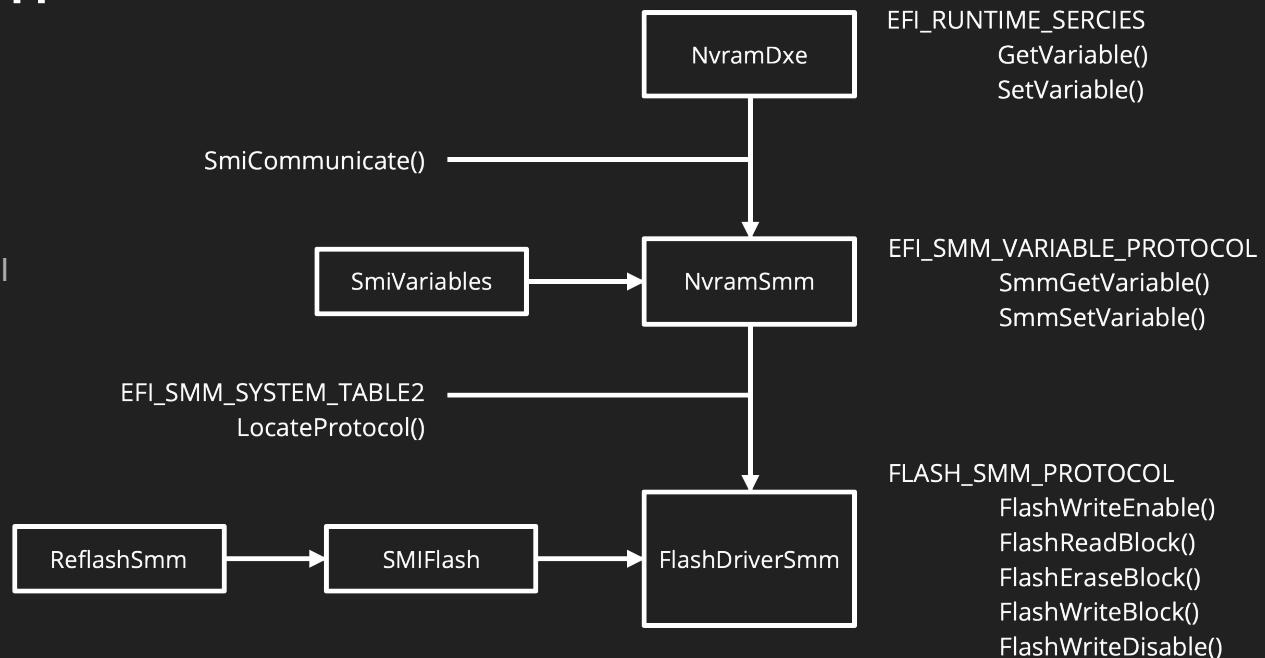
[Safeguarding UEFI Ecosystem: Firmware Supply Chain is Hard\(coded\) by Alex Tereshkin, Alex Matrosov, Adam 'pi3' Zabrocki](#)

# Updated Attack Surface



# AMI NVRAM driven experience

This what happens when you call EFI NVAR R/W procedures.



**INTEL-SA-00343**

**CVE-2020-0530**

**CVSS 7.8 High**

<https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00343.html>

# PEI module S3Resume2Pei {89E549B0-7CFE-449D-9BA3-10D8B2312D71}

```
// Retrieves 'FPDT_Variable_NV' EFI variable upon booting from S3 (Sleep) mode
DataSize = 4;
FDPT_ptr = 0;

if ( (*ReadOnlyVar)(ReadOnlyVar, L"FPDT_Variable_NV", &gFPDT_Variable_NV_Guid, 0, &DataSize, &FDPT_ptr) >= 0 ) {

    ptr = (_DWORD *) *FDPT_ptr;

    // Obviously, 'FPDT_Variable_NV' variable points to some structure, where the first field should be pointing to 'TP3S' signature
    if ( *(_DWORD *) *FDPT_ptr == 'TP3S' ) {
        if ( *((_QWORD *) FDPT_ptr + 2) ) {
            if ( !*((_WORD *) ptr + 4) ) {

                ...

                // After some basic validation a calculated values are written to the structure
                ptr[3] = v11;
                ptr[6] = result;
                ptr[7] = v12;
                ptr[4] = v7;
                ptr[5] = v9;
```

# PEI module SmbiosPeim

## {968C1D9F-80C4-43B7-8CAE-668AA56C4E71}

```
// Retrieves 'WakeUpType' EFI variable upon booting from S3 (Sleep) mode
result = (*ReadOnlyVar)(ReadOnlyVar, L"WakeUpType", &gWakeUpTypeVarGuid, 0, &DataSize, &Data);

if ( !result )
{
    if ( (unsigned int) Data <= 0xF0000 )
    {

        ...

    }
    // The value is written at address pointed by 'WakeUpType' EFI variable
    else
    {
        v4 = sub_FFFAD0F2(v3);
        LOBYTE(result) = v4[1](6);

        if ( (unsigned __int8)result >= 9u )
            LOBYTE(result) = 6;

        *Data = result;
    }
}
```

- Craft the 'FPDT\_Variable\_NV' or 'WakeUpType' EFI Variable
- Write onto NVRAM (via SMM if hardened security or RT if common security)
- Restart or Sleep/Wake Up the system

The vulnerability will be triggered upon every boot.

However we need to craft the variable (bypass all watermark checks) very carefully to gain something.  
Complicated, let's look into another vulnerability.

**INTEL-SA-00343**

**CVE-2020-0526**

**CVSS 7.7 High**

<https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00343.html>

# SMM module NvramSmm

## {447A1B58-8F3E-4658-ABAA-9E7B2280B90A}

```
// Retrieves NvramMailBox EFI variable

UINTN Data[3];
UINTN DataSize = 0x18;

result = gEfiSystemTable_0->RuntimeServices->GetVariable(L"NvramMailBox", &NVRAM_MAILBOX_VARIABLE_GUID, 0, &DataSize, Data);

if ( result >= 0 )
{
    ...

    // Processes the first and last QWORD from NvramMailBox

    ...

    // Deletes NvramMailBox variable
    gEfiSystemTable_0->RuntimeServices->SetVariable(L"NvramMailBox", &NVRAM_MAILBOX_VARIABLE_GUID, 0, 0, 0);

    result = gSmst_1->SmmAllocatePool(EfiRuntimeServicesData, 0x10000ui64, &Buffer);
    if ( result >= 0 )
    {
        ...

        result = gSmst_1->SmiHandlerRegister(SmiHandler, &NVRAM_MAILBOX_VARIABLE_GUID, &DispatchHandle);
        if ( result >= 0 )
        {
            // Calls a function pointed by the second QWORD from NvramMailBox!
            result = ((__int64 (*)(void)) Data[1])();
        }
    }
}
```

It is very simple to craft this variable so that it would lead to gaining code execution in SMM.

However at the moment of running NvramSmm this variable is already created in DXE module NvramDxe {1807040D-5934-41A2-A088-8E0F777F71AB}:

```
Data[0] = (_int64) &dword_8000E620;  
Data[1] = (_int64) sub_80001C60;  
Data[2] = (_int64) &byte_8000E924;  
  
result = SetVariable_local(L"NvramMailBox", &NvramMailBoxVarGuid, 2, 0x18, Data);
```

But since the variable is deleted in NvramSmm (with all its attributes):

```
// Deletes NvramMailBox variable  
gEfiSystemTable_0->RuntimeServices->SetVariable(L"NvramMailBox", &NVRAM_MAILBOX_VARIABLE_GUID, 0, 0, 0);
```

We can create the new one! Using just EFI\_RUNTIME\_SERVICES->SetVariable() with NON\_VOLATILE | RUNTUME\_ACCESS | AUTH\_ACCESS attributes to lock it for NvramDxe.

- Build payload
- Write onto NVRAM (via SMM if hardened security or RT if common security)
- Craft the 'NvramMailBox' EFI Variable so its second QWORD will point to the payload inside NVRAM and with attributes to lock it for boot phase
- Restart the system

The payload will be executed upon every boot

Tested on Intel® NUC Kit NUC7i7BNH

- This vulnerability bypasses all known BIOS protection mechanisms since:
  - NVRAM is a R/W area by design not measured/verified
  - Lack of anti-exploit mitigations (no ASLR, no Memory Guards, no RO/XD etc.)
- Ring 3 admin no SMM required API
- Good point is:

**Given different patch cycles across vendors, these vulnerabilities can stay unpatched on endpoint devices for 6-9 months (sometimes even longer).**

[Firmware Supply Chain is Hard\(Coded\)](#) by Binarly team

- Even if BIOS stored in Apple T2 security chip (you can still exploit memory corruptions in NVRAM)

# NVRAM STORIES

## with efiXplorer

Thanks to Yegor Vasilenko [@veggory](#) for helping with this part

<https://github.com/binary-io/efiXplorer>

# Load & Scan

Testing efiXloader / efiXplorer on some old buggy firmware.

## Results:

- a few SMM callouts
- a few GetVariable() buffer overflows

The screenshot shows the efiXplorer interface with several windows open:

- Functions** pane: A list of function names, many of which end in a numeric suffix (e.g., \_0, \_1, \_2, etc.).
  - ACPI\_entry\_B880782D
  - ACPI\_entry\_B880782D\_0
  - AHCI\_entry\_2BCF3A28
  - AINT13\_entry\_9402E944
  - AINT13\_entry\_9402E944\_0
  - AMITSE\_entry\_23777467
  - AMITSE\_entry\_23777467\_0
  - AMITSE\_entry\_23777467\_1
  - AMITSE\_entry\_23777467\_2
  - AMITSE\_entry\_23777467\_3
  - AMITSE\_entry\_23777467\_4
  - Acoustic\_entry\_6FF71A32
  - Acoustic\_entry\_6FF71A32\_0
  - AcpiModeEnable\_entry\_204C8CEF
  - AcpiModeEnable\_entry\_204C8CEF\_0
  - AcpiPlatformSmi\_entry\_1C114519
  - AcpiPlatformSmi\_entry\_1C114519\_0
  - AcpiPlatform\_entry\_CB8F5E15
  - AcpiPlatform\_entry\_CB8F5E15\_0
- IDA View-A** pane: Shows assembly code for the selected function.
- Pseudocode-A** pane: Shows pseudocode for the selected function.
- efiXplorer: vulns** pane: Shows a list of vulnerabilities, with the first one highlighted in blue:

Address	Type
0000000000429DE8	smm_callout
0000000000429E06	smm_callout
0000000000429E2F	smm_callout
0000000000429E44	smm_callout
000000000081C2DB	smm_callout
000000000064373D	get_variable_buffer_overflow
000000000064B633	get_variable_buffer_overflow
000000000064F633	get_variable_buffer_overflow
00000000006FE46E	get_variable_buffer_overflow
- Graph overview** pane: A graph visualization showing the flow of control between functions.

# GetVariable() buffer overflow

GetVariable() used to retrieve the size of the variable, which further is ignored.

The screenshot shows the IDA Pro interface with several windows open: IDA View-A, Pseudocode-A, efiXplorer: vulns, Hex View-1, Structures, Enums, and Registers. The assembly code in the main window is as follows:

```
_int64 __fastcall sub_64F590(unsigned __int16 a1, __int64 a2)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL- "+" TO EXPAND]

    DataSize = 0i64;
    v9 = 1280i64;
    gBS_6526B0->LocateProtocol(&EFI_HII_STRING_PROTOCOL_GUID_651C78, 0i64, &Interface);
    v4 = gRT_6526B8->GetVariable(L"PlatformLang", &EFI_SIMPLE_BOOT_FLAG_VARIABLE_GUID_651BB0, 0i64, &DataSize, Data);
    if ( v4 == EFI_BUFFER_TOO_SMALL )
        v4 = gRT_6526B8->GetVariable(L"PlatformLang", &EFI_SIMPLE_BOOT_FLAG_VARIABLE_GUID_651BB0, 0i64, &DataSize, Data);
    if ( v4 >= 0 )
        return (Interface->GetString)(Interface, Data, qword_652728, a1, a2, &v9, 0i64);
    else
        return EFI_NOT_FOUND;
}
```

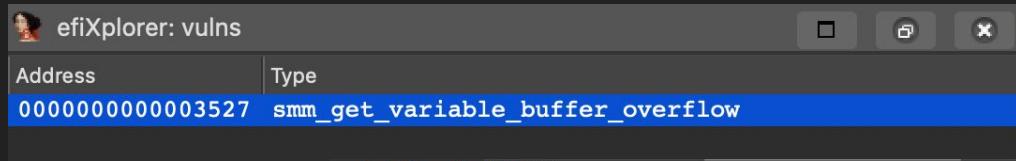
# SMM call-outs

Usage of  
EFI\_RUNTIME\_SERVICES  
in SMI Handler.

```
× IDA View-A × Pseudocode-A × efiXplorer: vulns × Hex View-1 × Structures ×
EFI_STATUS sub_429DAC()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL- "+" TO EXPAND]

    v0 = qword_42AE08;
    if (*qword_42AE08 != 'RAVN')
        return EFI_DEVICE_ERROR;
    switch (*qword_42AE08 + 40)
    {
        case 1:
            result = gRT_42AE28->GetVariable(
                (qword_42AE08 + 72),
                (qword_42AE08 + 8),
                (qword_42AE08 + 4),
                (qword_42AE08 + 24),
                (qword_42AE08 + 72));
            break;
        case 2:
            result = gRT_42AE28->SetVariable(
                (*(qword_42AE08 + 24) + qword_42AE08 + 72),
                (qword_42AE08 + 8),
                *(qword_42AE08 + 4),
                *(qword_42AE08 + 24),
                (qword_42AE08 + 72));
            break;
        case 3:
            result = gRT_42AE28->GetNextVariableName((qword_42AE08 + 24), (qword_42AE08 + 72), (qword_42AE08 + 8));
            break;
        case 4:
            result = gRT_42AE28->QueryVariableInfo(
                *(qword_42AE08 + 4),
                (qword_42AE08 + 48),
                (qword_42AE08 + 56),
                (qword_42AE08 + 64));
            break;
        default:
            return EFI_DEVICE_ERROR;
    }
    *(v0 + 32) = result;
    *(v0 + 40) = 0;
    *v0 = 0xADBEA9B1;
    return result;
}
```

# SmmGetVariable() buffer overflow



```
efiXplorer: vulns
Address          Type
0000000000003527 smm_get_variable_buffer_overflow

int64 sub_34B8()
{
    int64 result; // rax
    char Data; // [rsp+48h] [rbp+18h] BYREF
    UINT32 Attributes; // [rsp+50h] [rbp+20h] BYREF
    UINTN DataSize; // [rsp+58h] [rbp+28h] BYREF

    DataSize = 0i64;
    result = gSmmVar_5B40->SmmGetVariable(gVariableName1, &gVendorGuid, &Attributes, &DataSize, 0i64);
    if ( result == EFI_BUFFER_TOO_SMALL )
    {
        result = gSmmVar_5B40->SmmGetVariable(gVariableName1, &gVendorGuid, &Attributes, &DataSize, &Data);
        if ( result >= 0 )
        {
            DataSize = 0i64;
            (gSmmVar_5B40->SmmSetVariable)(gVariableName1, &gVendorGuid, Attributes, 0i64, 0i64);
            DataSize = 1i64;
            return (gSmmVar_5B40->SmmSetVariable)(gVariableName2, &gVendorGuid, Attributes);
        }
    }
    else if ( result == 0x8000000000000000Eui64 )
    {
        return 0i64;
    }
    return result;
}
```

- NVRAM Data handling Vector to bypass all known protections
- UEFI BIOS attack surface keeps extending:
  - More NVRAM format & EFI variable parsing code
  - More UEFI API for the runtime (Redfish etc.)
- Even a signed firmware can't be trusted (c) Binarly

THANK YOU



*Fwhunt@binarly.io*