MAINTRACK

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

Alex Matrosov

Hola!

Alex Matrosov

@matrosov

- Founder and CEO at BINARLY

- 20+ years doing all shades of cyber
- Break a few times CPU’s and GPU’s
- Dedicating all my free time to surfing

Binarly REsearch Team

Alex Matrosov

@matrosov

Yegor Vasilenko

@yeggorv

Alex Ermolov

@flothrone

Sam Thomas

@xorpse

⇛ 12 CVEs

⇛ 7 CVEs

⇛ 3 CVEs

= 22 CVEs

2022 ⇛ 22

Presentation Agenda

- Intro
- Pre-EFI (PEI) Attack Surface
- Intel PPAM Attack Surface
- Firmware Repeatable Failures
  - 3 new industry-wide vulns

Repeatable Failures

As code complexity increases, memory corruptions remain forever

Vulnerabillity Category	Count	Average Impact
PEI Memory Corruption	3	CVSS: 8.0 (High)
SMM Memory Corruption	56	CVSS: 8.0 (High)
DXE Memory Corruption	7	CVSS: 7.7 (High)
Mitigation Failures	2	CVSS: 6.0 (HighMedium)

https://www.binarly.io/advisories

Vulnerabilities in the Insyde (industry-wide impact)

black hat USA 2022

Firmware Repeatable Failures

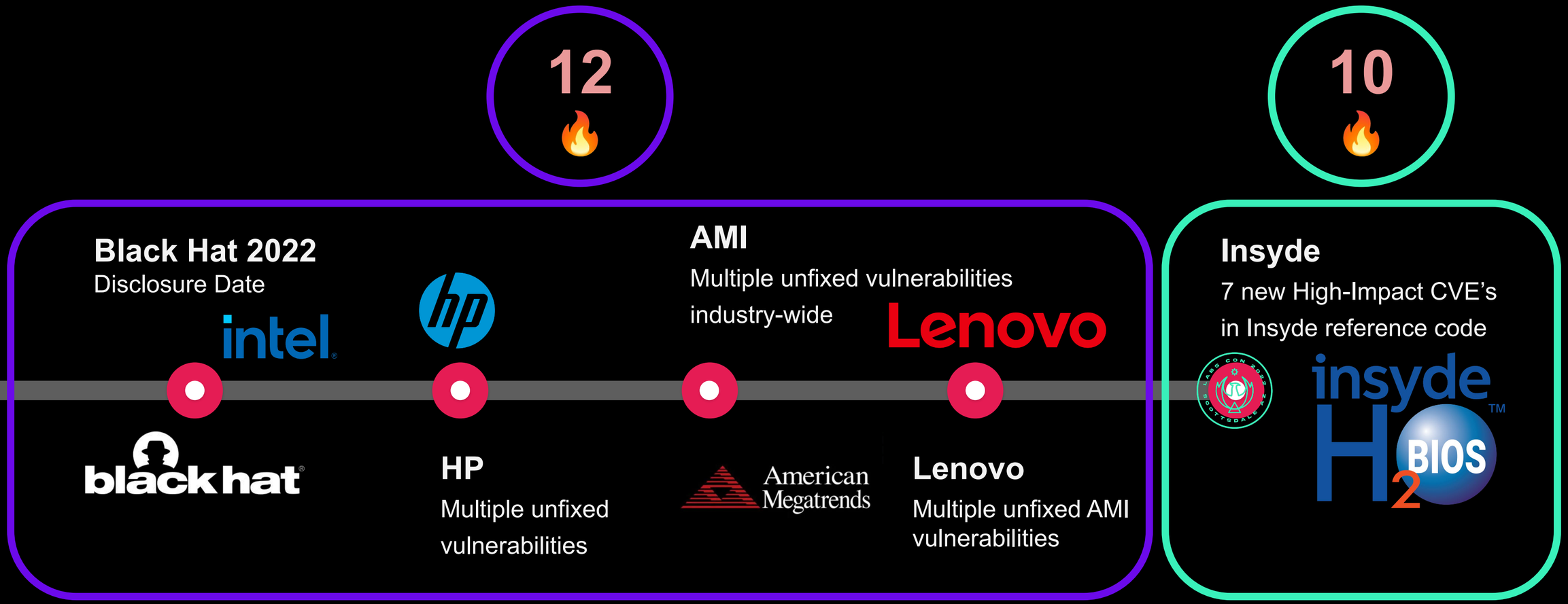
BINARLY

Vendor	Vulnerabilities	Number of Issues	BINARLY ID	CVE ID	CVSS score
intel ami	PEI Memory Corruption (Arbitrary Code Execution)	3	BRLY-2022-027 BRLY-2022-009 BRLY-2022-014	CVE-2022-28858 CVE-2022-36372 CVE-2022-32579	8.2 High 8.2 High 7.2 High
	DXE Arbitrary Code Execution	1	BRLY-2022-015	CVE-2022-34345	7.2 High
	SMM Memory Corruption (Arbitrary Code Execution)	2	BRLY-2022-003 BRLY-2022-016	CVE-2022-27493 CVE-2022-33209	7.5 High 8.2 High
hp	SMM Memory Corruption (Arbitrary Code Execution)	6	BRLY-2022-010 BRLY-2022-011 BRLY-2022-012 BRLY-2022-013 BRLY-2021-046 BRLY-2021-047	CVE-2022-23930 CVE-2022-31644 CVE-2022-31645 CVE-2022-31646 CVE-2022-31640 CVE-2022-31641	8.2 High 7.5 High 8.2 High 8.2 High 7.5 High 7.5 High

Repeatable Failures: last week in the news

https://www.theregister.com/2022/09/13/firmware_bugs_hp

Firmware Security Repeatable Failures



https://www.insyde.com/security-pledge#InsydeSept212022

https://www.binarly.io/posts/Binarly_Discovers_Multiple_High_Severity_Vulnerabilities_in_AMI_based_Devices

https://www.binarly.io/posts/Binarly_Finds_Six_High_Severity_Firmware_Vulnerabilities_in_HP_Enterprise_Devices

Vulnerabilities in the Insyde (industry-wide impact)

BRLY	CVE	CVSS v3	Description
BRLY-2022-017	CVE-2022-36338	7.5 High	SMM callout vulnerability in SMM driver (SMM arbitrary code execution)
BRLY-2022-018	CVE-2022-35894	6.0 Medium	SMM memory leak vulnerability in SMM driver (SMRAM read)
BRLY-2022-019	CVE-2022-36337	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-020	CVE-2022-35407	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-021	CVE-2022-35897	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-022	CVE-2022-35408	7.5 High	SMM callout vulnerability in SMM driver (SMM arbitrary code execution)
BRLY-2022-023	CVE-2022-36448	8.2 High	SMM memory corruption vulnerability in Software SMI handler
BRLY-2022-024	CVE-2022-35895	8.2 High	SMM memory corruption vulnerability in SMM driver (SMRAM write)
BRLY-2022-025	CVE-2022-35896	6.0 Medium	SMM memory leak vulnerability in SMM driver (SMRAM read)
BRLY-2022-026	CVE-2022-35893	8.2 High	SMM memory corruption vulnerability in SMM driver (SMRAM write)

https://www.binarly.io/advisories

Pre-EFI(PEI) ⇒ SMM threat model

Attacker Model:

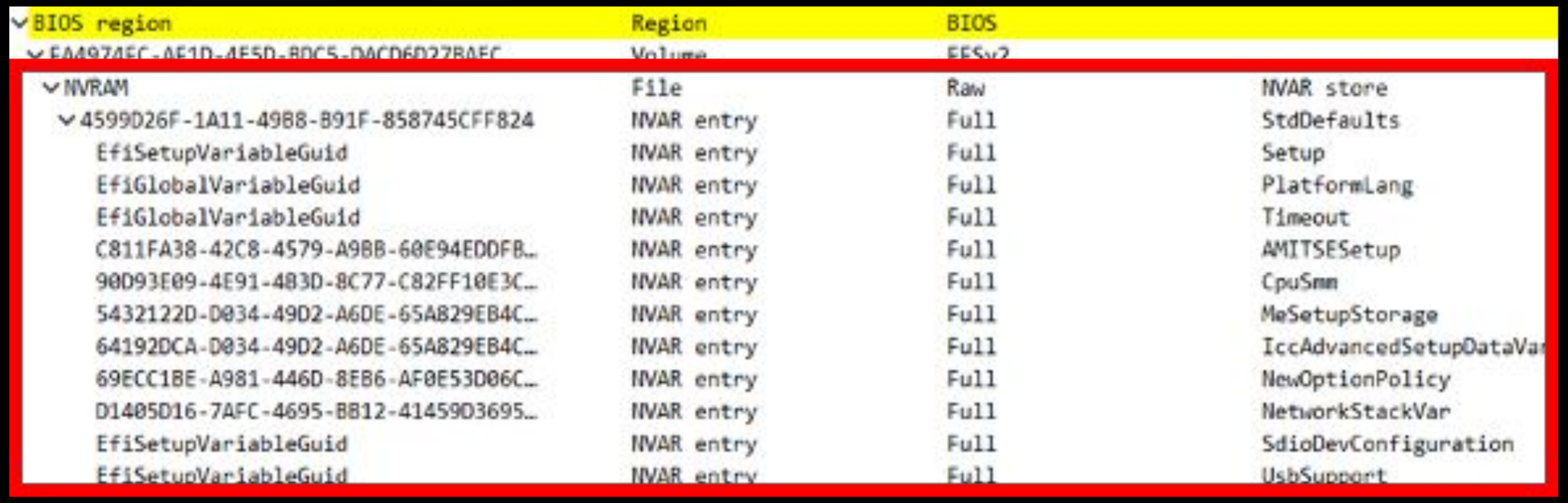
The local attacker uses privileged host OS access to trigger the vulnerability gaining PEI or DXE stage code execution in System Management Mode (SMM).

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).

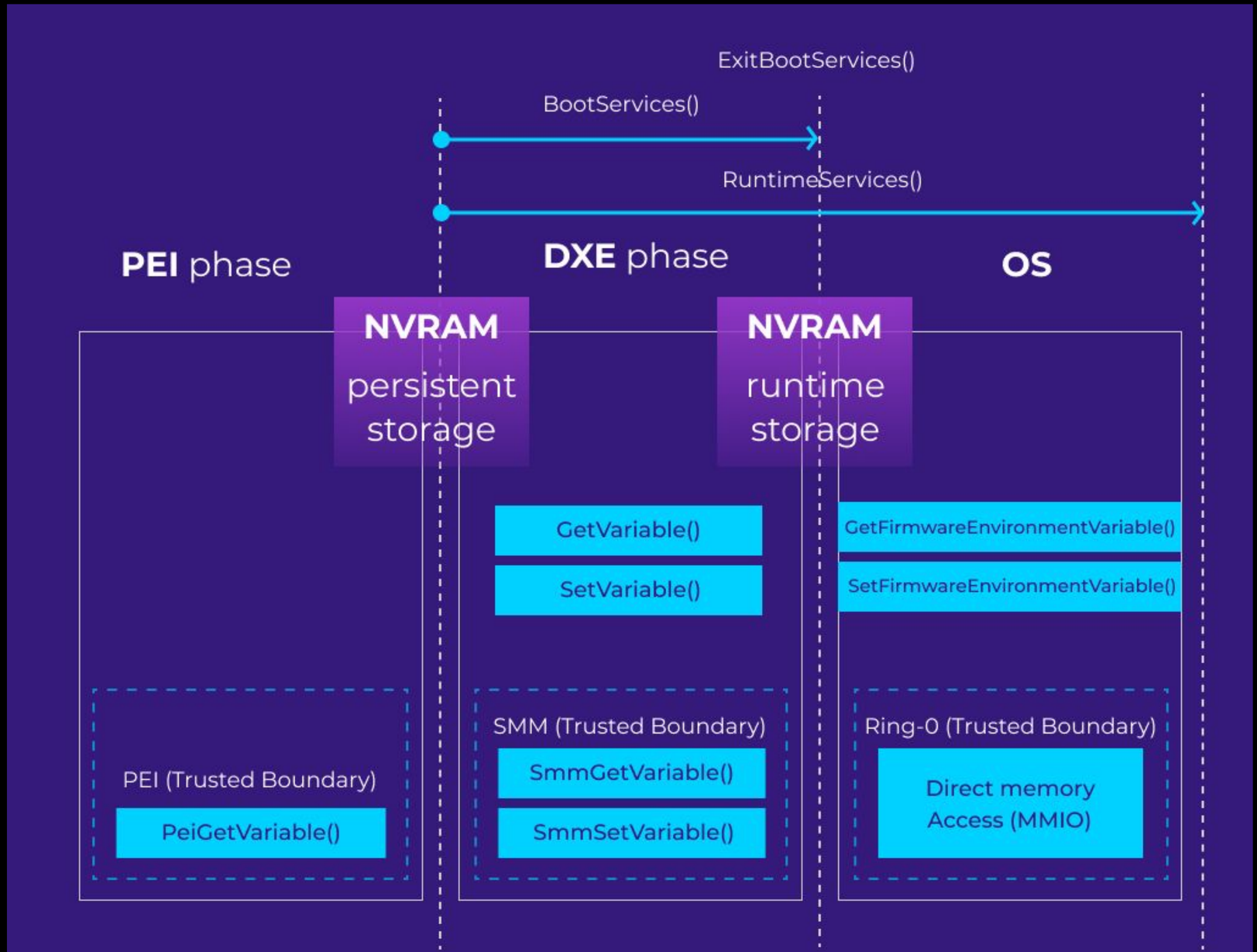


BIOS region	Region	BIOS	
EA4974EC-AE1D-4E5D-BDC5-DACD6D27BAEC	Volume	FFSv2	
NVRAM	File	Raw	NVAR store
4599D26F-1A11-4988-B91F-858745CFF824	NVAR entry	Full	StdDefaults
EfiSetupVariableGuid	NVAR entry	Full	Setup
EfiGlobalVariableGuid	NVAR entry	Full	PlatformLang
EfiGlobalVariableGuid	NVAR entry	Full	Timeout
C811FA38-42C8-4579-A9BB-60E94EDDFB…	NVAR entry	Full	AMITSESetup
90D93E09-4E91-483D-8C77-C82FF10E3C…	NVAR entry	Full	CpuSmm
5432122D-D034-49D2-A6DE-65A829EB4C…	NVAR entry	Full	MeSetupStorage
64192DCA-D034-49D2-A6DE-65A829EB4C…	NVAR entry	Full	IccAdvancedSetupDataVa
69ECC1BE-A981-446D-8EB6-AF0E53D06C…	NVAR entry	Full	NewOptionPolicy
D1405D16-7AFC-4695-BB12-41459D3695…	NVAR entry	Full	NetworkStackVar
EfiSetupVariableGuid	NVAR entry	Full	SdioDevConfiguration
EfiSetupVariableGuid	NVAR entry	Full	UsbSupport

- 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



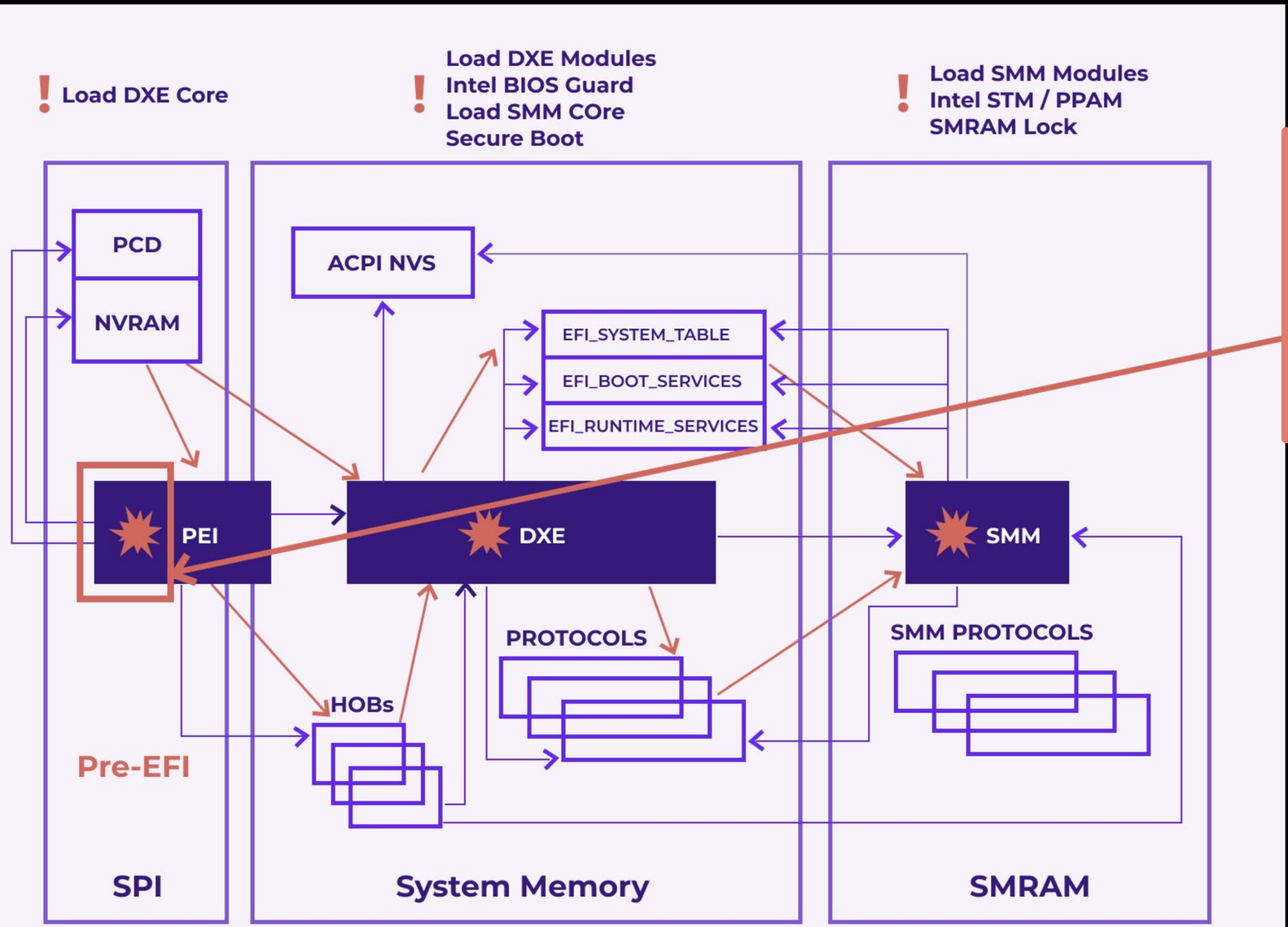
Important Reminder

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

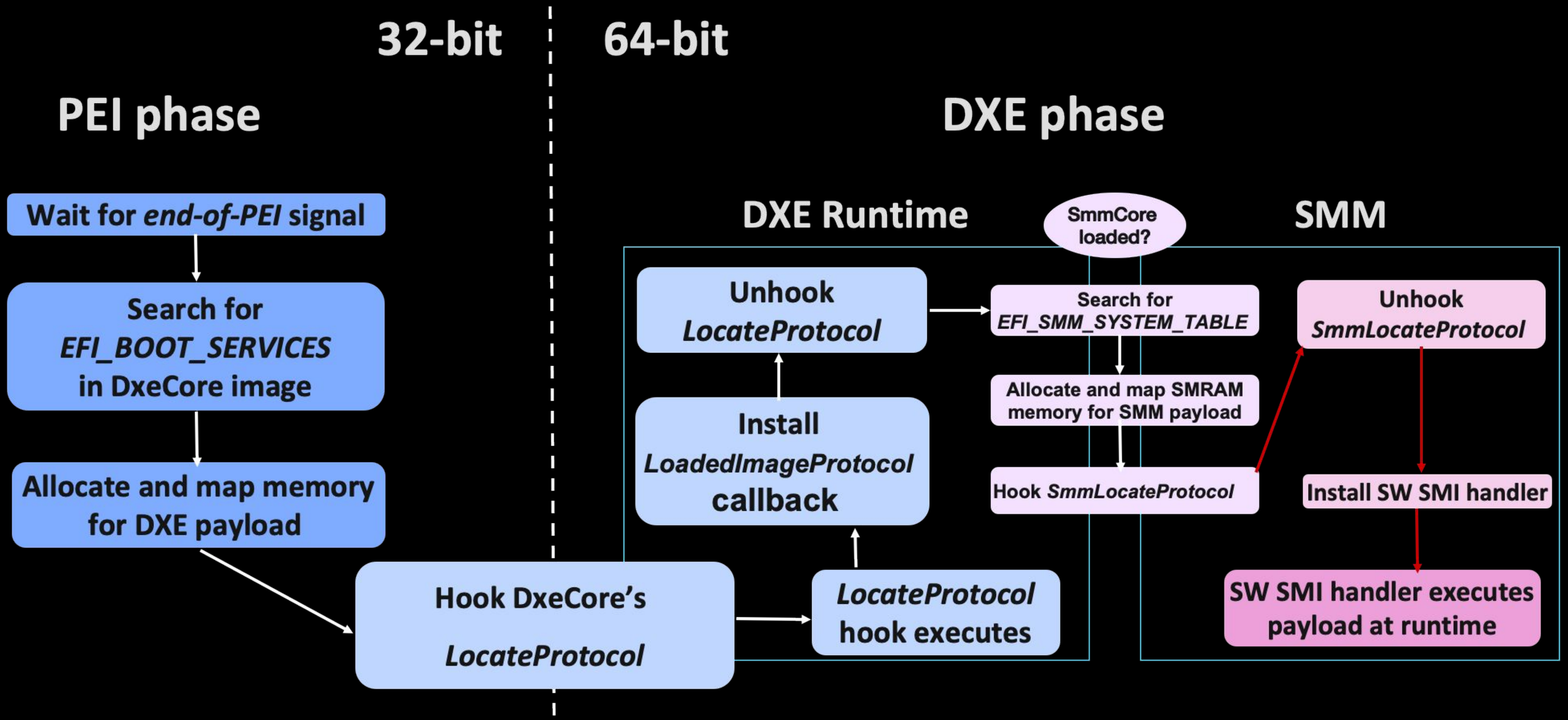
Pre-EFI Attack Surface



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.

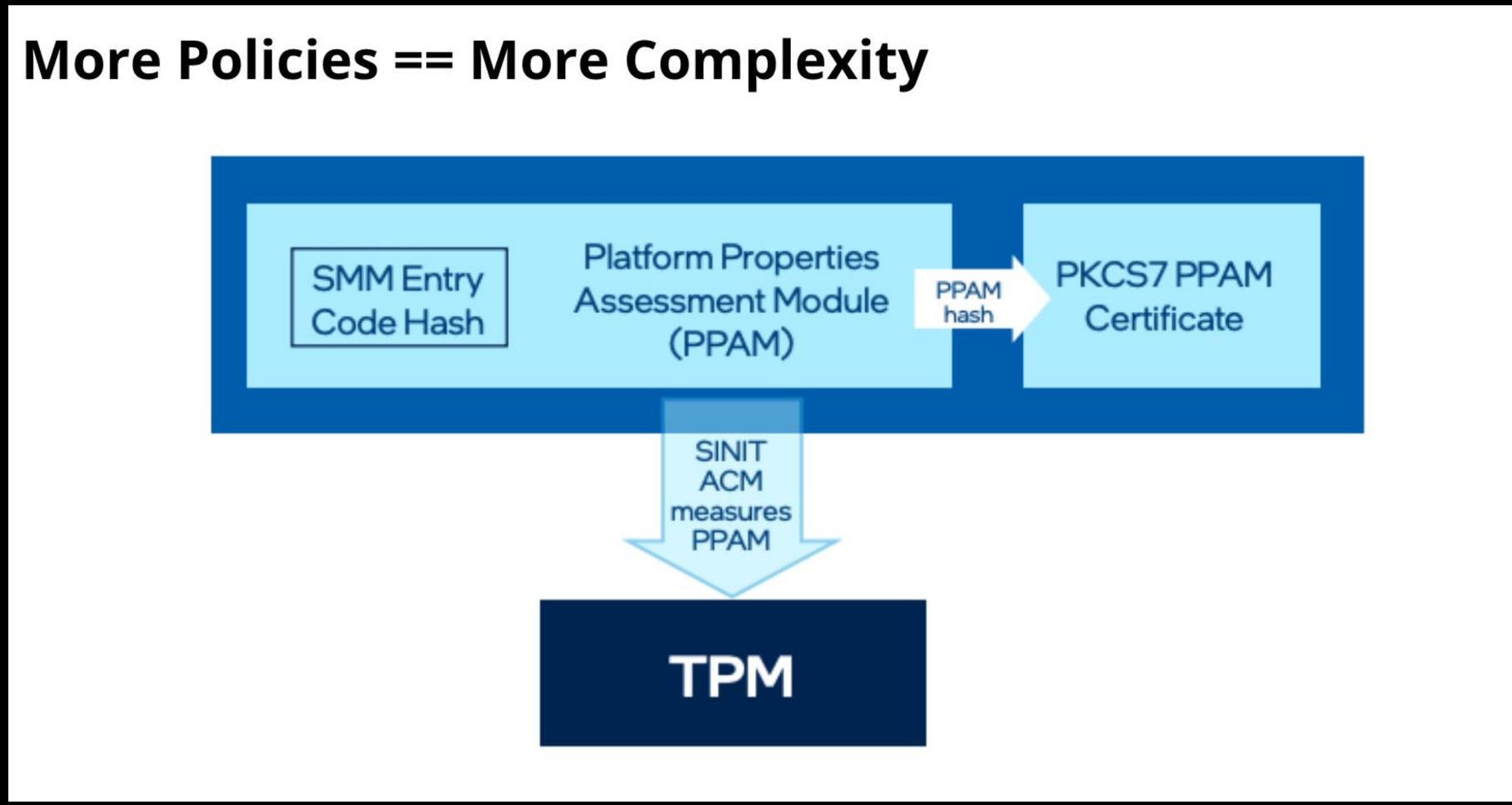
Payload Delivery: Escalate from PEI to SMM



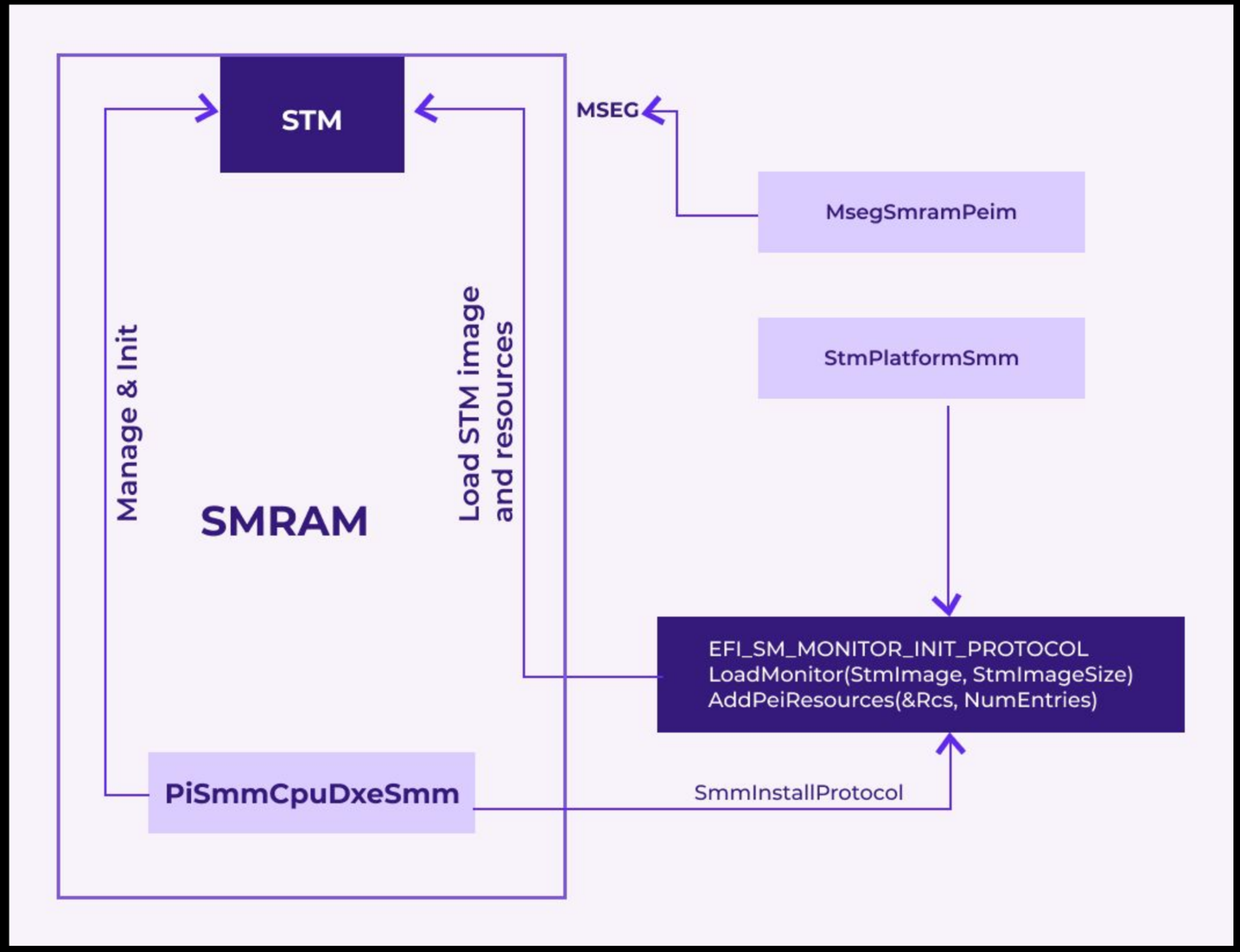
Intel PPAM Attack Surface

Platform Properties Assessment Module (PPAM)

As code complexity increases, design issues remain forever

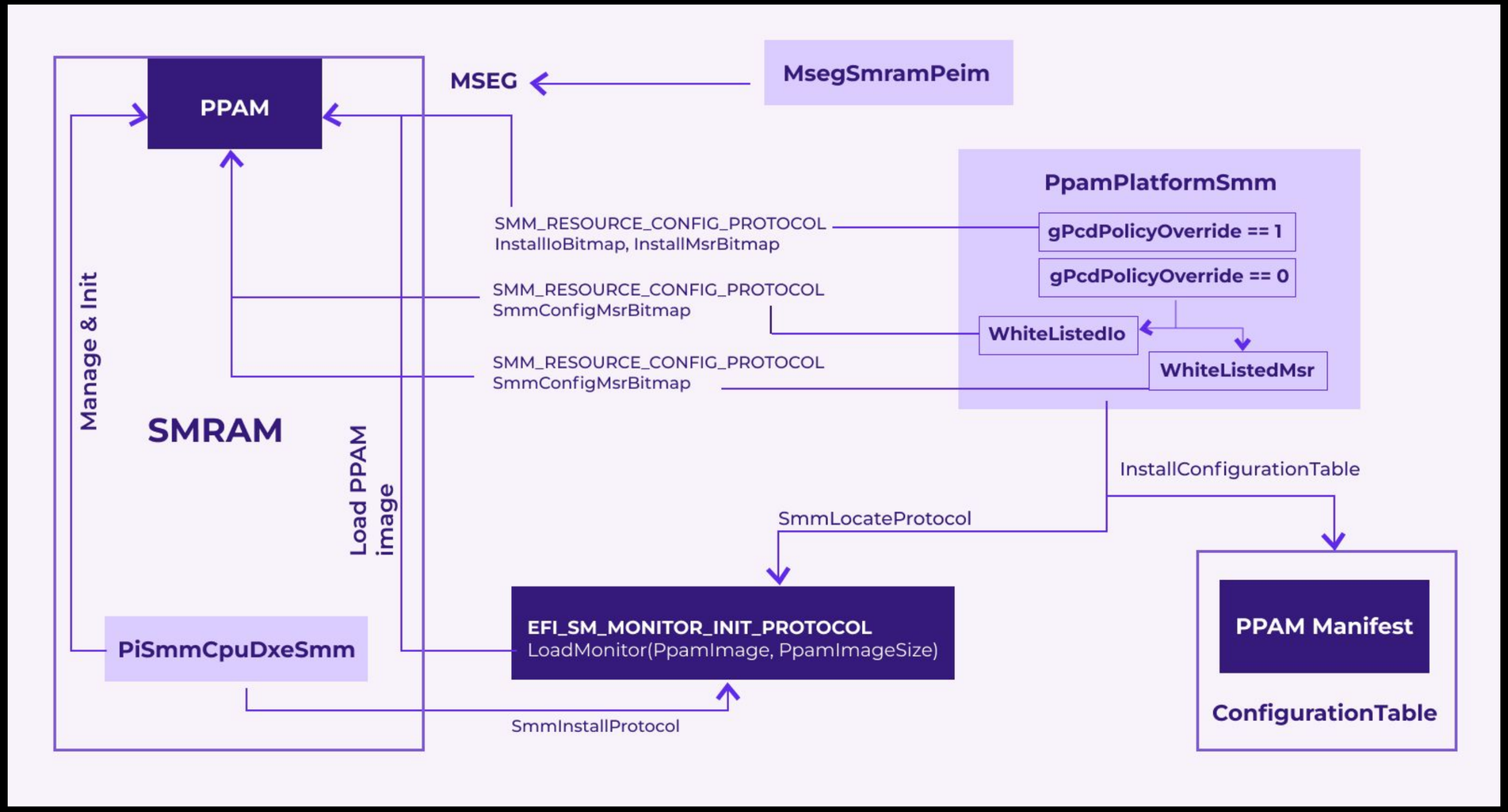


Initializing an SMI Transfer Monitor (STM) in UEFI



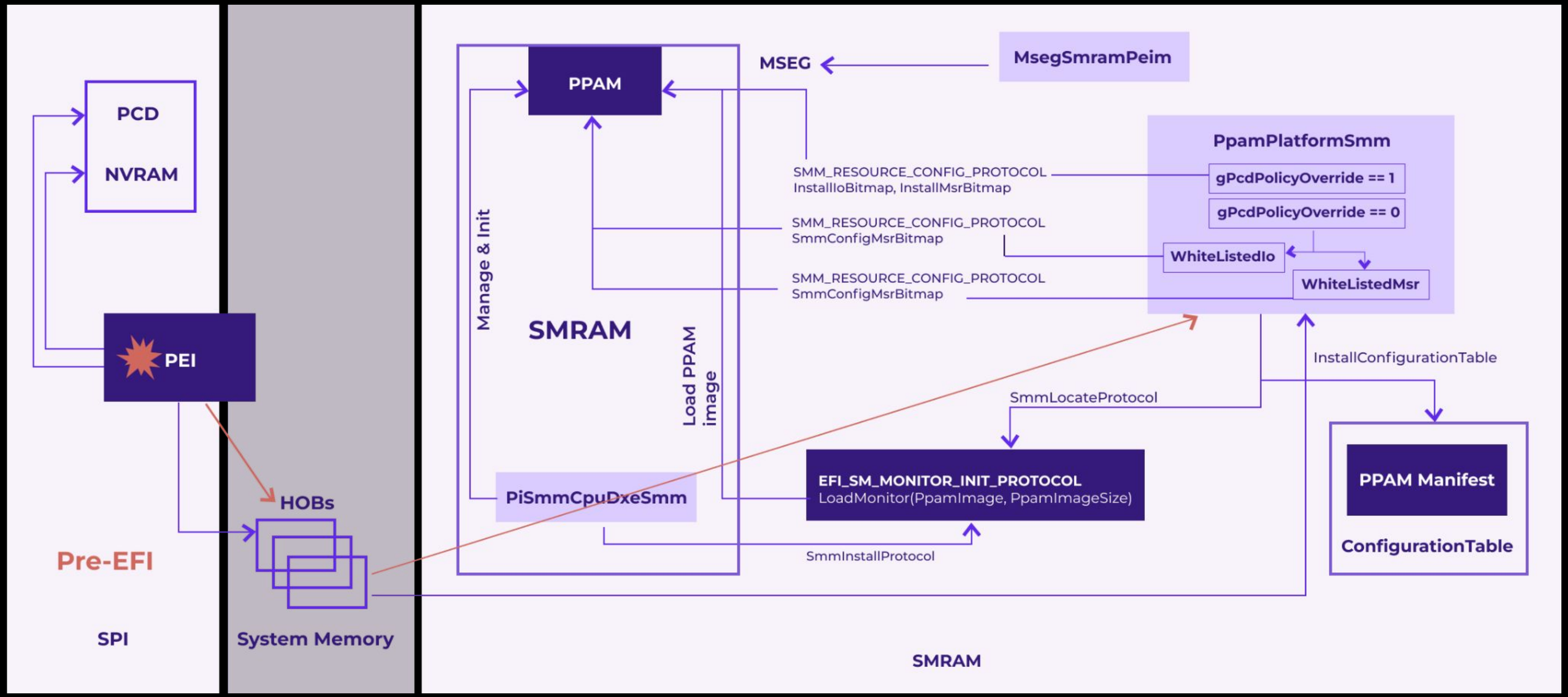
https://www.intel.com/content/dam/develop/external/us/en/documents/a-tour-beyond-bios-launching-stm-to-monitor-smm-in-efi-developer-kit-ii-819978.pdf

Initializing a PPAM in UEFI



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

Compromising the Initialization Process



https://www.binarly.io/posts/Black_Hat_2022_The_Intel_PPAM_attack_story

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();
  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) & 0x80
    || !gEfiSmMonitorInitProtocol )
  {
    return EFI_UNSUPPORTED;
  }
  if ( (LoadPpamImage(PpamSupport) & 0x8000000000000000ui64) == 0i64 )
    LoadPpamManifest(PpamSupport);
  if ( PpamSupport >= 11u )
  {
    if ( (gSmst->SmmLocateProtocol(&EFI_SMM_RESOURCE_CONFIG_PROTOCOL_GUID, 0i64, &gSmmResourceConfigProtocol)
      || !gSmmResourceConfigProtocol )
    {
      return EFI_UNSUPPORTED;
    }
    // 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(&gHobGuid);
  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 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  ................
00000070: 01 00 00 00 D5 91 52 FF                          ......R.
```

https://www.binarly.io/posts/Black_Hat_2022_The_Intel_PPAM_attack_story

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

https://www.binarly.io/posts/Black_Hat_2022_The_Intel_PPAM_attack_story

Intel 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  ._..bfg......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 0d06 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

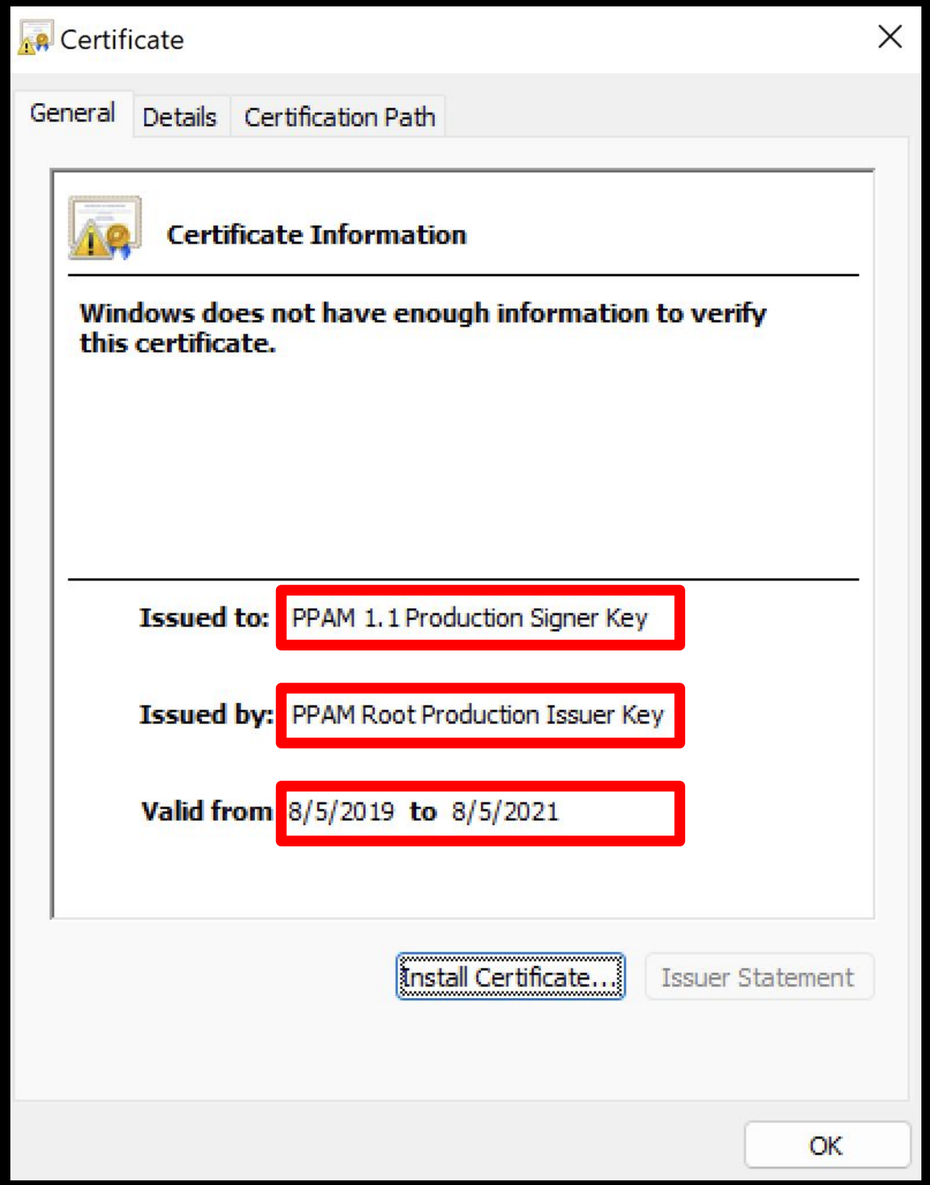
Validity

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

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

https://www.binarly.io/posts/Black_Hat_2022_The_Intel_PPAM_attack_story

Intel PPAM Manifest



https://www.binarly.io/posts/Black_Hat_2022_The_Intel_PPAM_attack_story

Intel PPAM Manifest

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

https://www.binarly.io/posts/Black_Hat_2022_The_Intel_PPAM_attack_story

Firmware Repeatable Failures

same problems, new discoveries

Intel Alder Lake BIOS source code leaked

Top news

BleepingComputer

Intel confirms leaked Alder Lake BIOS Source Code is authentic

12 days ago

TECHSPOT

Source code for Alder Lake BIOS was posted to GitHub

13 days ago

DARKReading

Intel Processor UEFI Source Code Leaked

10 days ago

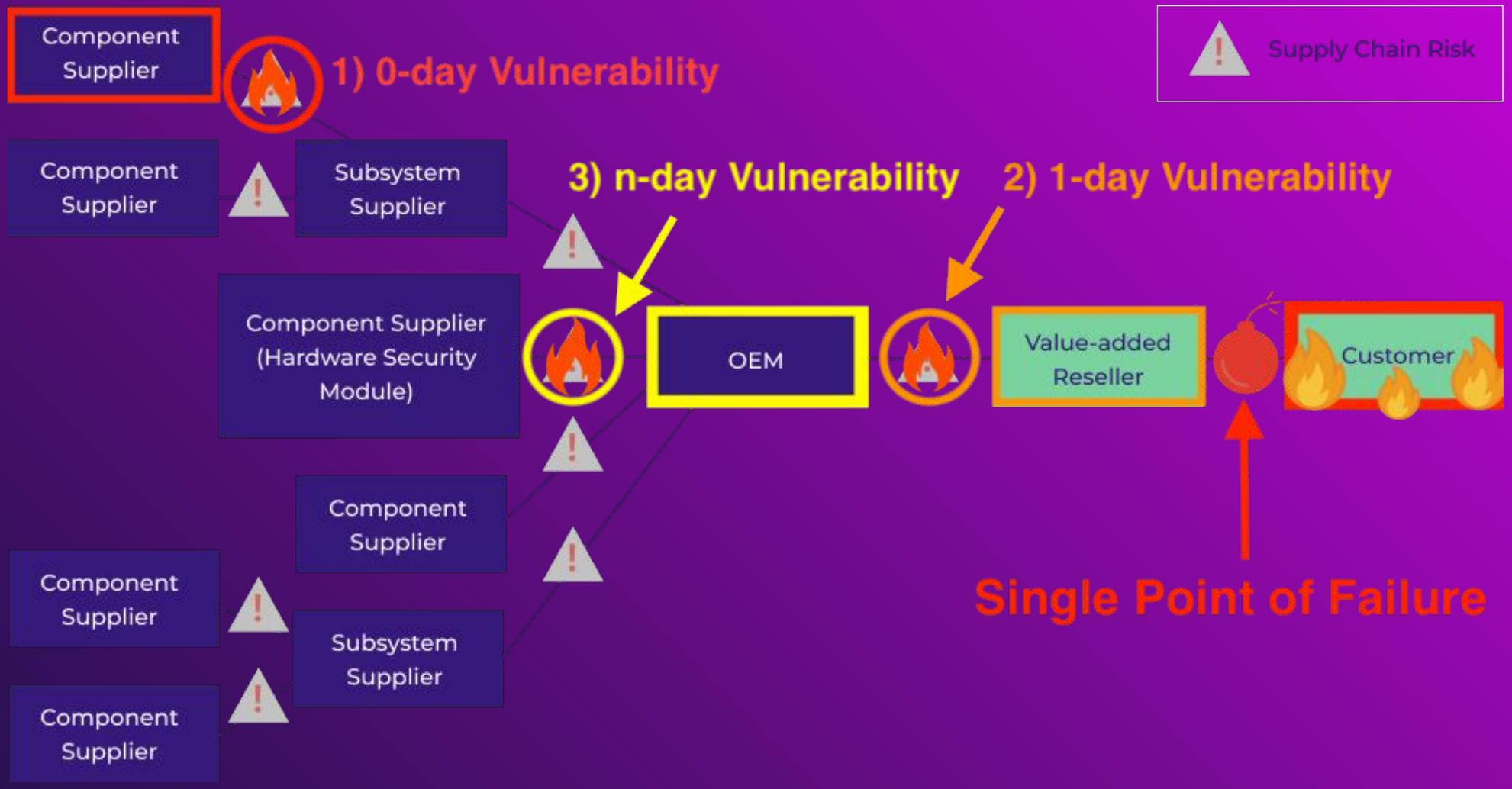
Infosecurity Magazine

Intel Confirms Source Code Leak

11 days ago

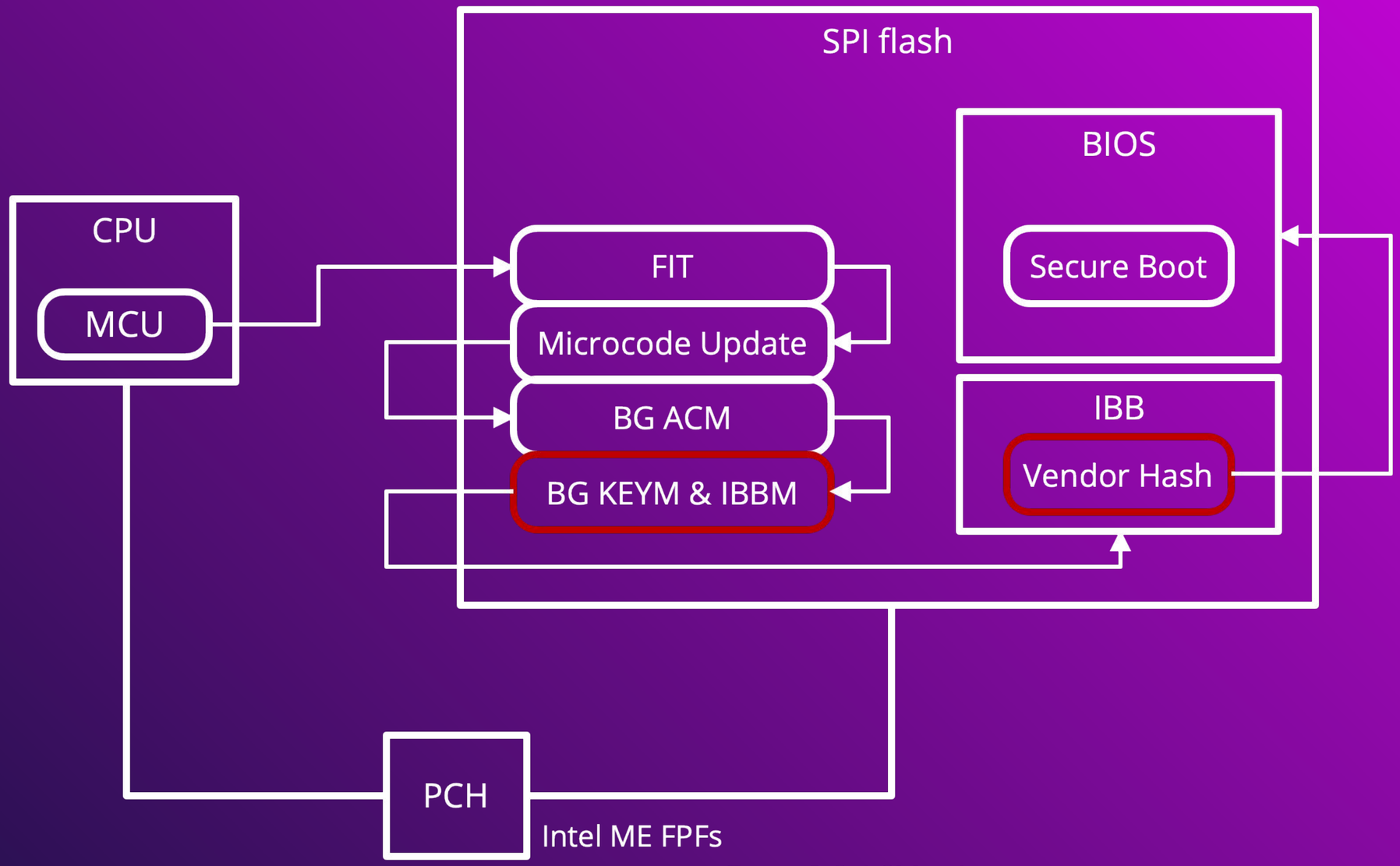
All coverage

https://www.binarly.io/posts/The_Firmware_Supply_Chain_Security_is_broken_Can_we_fix_it



https://www.binarly.io/posts/The_Firmware_Supply_Chain_Security_is_broken_Can_we_fix_it

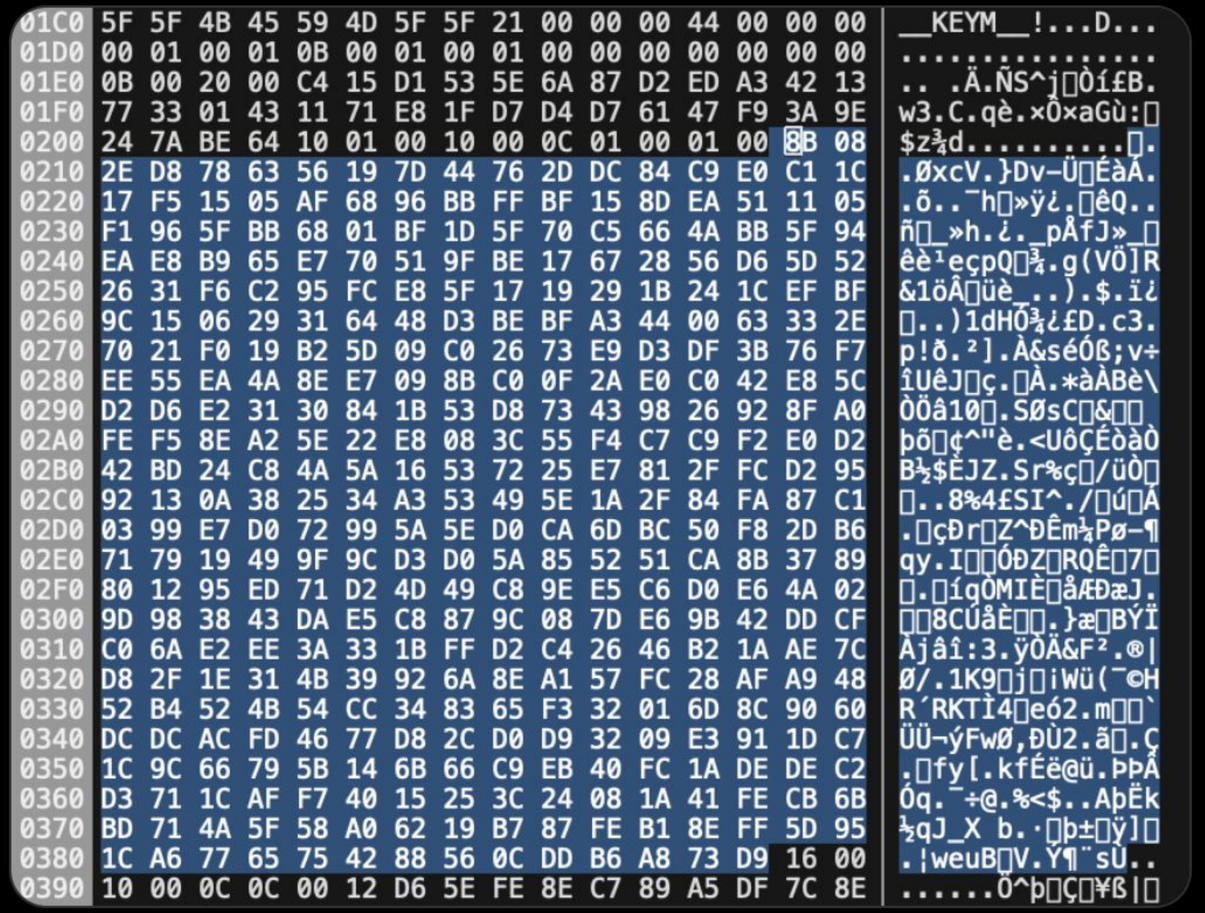

Mark Ermolov
@_markel___
A very bad thing happened: now, the Intel Boot Guard on the vendor's platforms can no longer be trusted...
yprivkey.pem -
8:38 AM · Oct 8, 2022 · Twitter Web App


SPI flash
BIOS
Secure Boot
CPU
MCU
FIT
Microcode Update
BG ACM
BG KEYM & IBBM
IBB
Vendor Hash
PCH
Intel ME FPFs
Force BG ACM = 1
Enforcement Policy = 3

BINARLY @binarly_io · 12 окт.

Lenovo recently leaked Boot Guard Platform Key does not affect Intel Adler Lake-based Yoga and Thinkpad laptops (it doesn't match most of the models in the field).

Kudos to @phretor for developing the #FwHunt rule to check FW images for leaked keys. github.com/binarly-io/FwH...



```
01C0 5F 5F 4B 45 59 4D 5F 5F 21 00 00 00 44 00 00 00  __KEYM__!...D...
01D0 00 01 00 01 0B 00 01 00 01 00 00 00 00 00 00 00  ................
01E0 0B 00 20 00 C4 15 D1 53 5E 6A 87 D2 ED A3 42 13  .. .Ä.ÑS^j□Òí£B.
01F0 77 33 01 43 11 71 E8 1F D7 D4 D7 61 47 F9 3A 9E  w3.C.qè.×Ô×aGù:□
0200 24 7A BE 64 10 01 00 10 00 0C 01 00 01 00 8B 08  $z¾d..........□.
0210 2E D8 78 63 56 19 7D 44 76 2D DC 84 C9 E0 C1 1C  .ØxcV.}Dv-Ü□ÉàÁ.
0220 17 F5 15 05 AF 68 96 BB FF BF 15 8D EA 51 11 05  .õ..¯h□»ÿ¿.□êQ..
0230 F1 96 5F BB 68 01 BF 1D 5F 70 C5 66 4A BB 5F 94  ñ□_»h.¿._pÅfJ»_□
0240 EA E8 B9 65 E7 70 51 9F BE 17 67 28 56 D6 5D 52  êè¹eçpQ□¾.g(VÖ]R
0250 26 31 F6 C2 95 FC E8 5F 17 19 29 1B 24 1C EF BF  &1öÂ□üè_..).$.ï¿
0260 9C 15 06 29 31 64 48 D3 BE BF A3 44 00 63 33 2E  □..)1dHÓ¾¿£D.c3.
0270 70 21 F0 19 B2 5D 09 C0 26 73 E9 D3 DF 3B 76 F7  p!ð.²].À&séÓß;v÷
0280 EE 55 EA 4A 8E E7 09 8B C0 0F 2A E0 C0 42 E8 5C  îUêJ□ç.□À.*àÀBè\
0290 D2 D6 E2 31 30 84 1B 53 D8 73 43 98 26 92 8F A0  ÒÖâ10□.SØsC□&□□
02A0 FE F5 8E A2 5E 22 E8 08 3C 55 F4 C7 C9 F2 E0 D2  þõ□¢^"è.<UôÇÉòàÒ
02B0 42 BD 24 C8 4A 5A 16 53 72 25 E7 81 2F FC D2 95  B½$ÈJZ.Sr%ç□/üÒ□
02C0 92 13 0A 38 25 34 A3 53 49 5E 1A 2F 84 FA 87 C1  □..8%4£SI^./□ú□Á
02D0 03 99 E7 D0 72 99 5A 5E D0 CA 6D BC 50 F8 2D B6  .□çÐr□Z^ÐÊm¼Pø-¶
02E0 71 79 19 49 9F 9C D3 D0 5A 85 52 51 CA 8B 37 89  qy.I□□ÓÐZ□RQÊ□7□
02F0 80 12 95 ED 71 D2 4D 49 C8 9E E5 C6 D0 E6 4A 02  □.□íqÒMIÈ□åÆÐæJ.
0300 9D 98 38 43 DA E5 C8 87 9C 08 7D E6 9B 42 DD CF  □□8CÚåÈ□□.}æ□BÝÏ
0310 C0 6A E2 EE 3A 33 1B FF D2 C4 26 46 B2 1A AE 7C  Àjâî:3.ÿÒÄ&F².®|
0320 D8 2F 1E 31 4B 39 92 6A 8E A1 57 FC 28 AF A9 48  Ø/.1K9□j□¡Wü(¯©H
0330 52 B4 52 4B 54 CC 34 83 65 F3 32 01 6D 8C 90 60  R´RKTÌ4□eó2.m□□`
0340 DC DC AC FD 46 77 D8 2C D0 D9 32 09 E3 91 1D C7  ÜÜ¬ýFwØ,ÐÙ2.ã□.Ç
0350 1C 9C 66 79 5B 14 6B 66 C9 EB 40 FC 1A DE DE C2  .□fy[.kfÉë@ü.ÞÞÂ
0360 D3 71 1C AF F7 40 15 25 3C 24 08 1A 41 FE CB 6B  Óq.¯÷@.%<$..AþËk
0370 BD 71 4A 5F 58 A0 62 19 B7 87 FE B1 8E FF 5D 95  ½qJ_X b.·□þ±□ÿ]□
0380 1C A6 77 65 75 42 88 56 0C DD B6 A8 73 D9 16 00  .¦weuB□V.Ý¶¨sÙ..
0390 10 00 0C 0C 00 12 D6 5E FE 8E C7 89 A5 DF 7C 8E  ......Ö^þ□Ç□¥ß|□
```

https://github.com/binarly-io/FwHunt/tree/main/rules/SupplyChain

Firmware Repeatable Failures

same problems, new discoveries

Vulnerabilities in the Insyde (industry-wide impact)

BRLY	CVE	CVSS v3	Description
BRLY-2022-017	CVE-2022-36338	7.5 High	SMM callout vulnerability in SMM driver (SMM arbitrary code execution)
BRLY-2022-018	CVE-2022-35894	6.0 Medium	SMM memory leak vulnerability in SMM driver (SMRAM read)
BRLY-2022-019	CVE-2022-36337	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-020	CVE-2022-35407	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-021	CVE-2022-35897	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-022	CVE-2022-35408	7.5 High	SMM callout vulnerability in SMM driver (SMM arbitrary code execution)
BRLY-2022-023	CVE-2022-36448	8.2 High	SMM memory corruption vulnerability in Software SMI handler
BRLY-2022-024	CVE-2022-35895	8.2 High	SMM memory corruption vulnerability in SMM driver (SMRAM write)
BRLY-2022-025	CVE-2022-35896	6.0 Medium	SMM memory leak vulnerability in SMM driver (SMRAM read)
BRLY-2022-026	CVE-2022-35893	8.2 High	SMM memory corruption vulnerability in SMM driver (SMRAM write)

https://www.binarly.io/advisories

BRLY-2022-017 and BRLY-2022-022

UsbLegacyControlSmm

```
EFI_STATUS __fastcall UsbSmiHandler(
        EFI_HANDLE DispatchHandle,
        const void *Context,
        void *CommBuffer,
        UINTN *CommBufferSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  Status = 0i64;
  if ( (MEMORY[0xC00F8094] & 0xF00) != 0 )
    Status = (MEMORY[0xC00F8094] >> 8) & 0xF;
  v5 = UsbLegacyProtocolFunc1;
  if ( UsbLegacyProtocolFunc1 )
    Status = UsbLegacyProtocolFunc1(Status, UsbLegacyProtocolFunc2);
  if ( !gNotTheFirstTime )
  {
    OldTpl = (gBS->RaiseTPL)(31i64, Context, v5, CommBufferSize);
    (gBS->RestoreTPL)(OldTpl);
    if ( OldTpl == 31 )
      (gBS->RestoreTPL)(8i64);
    gNotTheFirstTime = 1;
    if ( (gBS->LocateHandleBuffer(ByProtocol, &ProprietaryProtocol, 0i64, &NoHandles
    {
      v7 = 0i64;
      if ( NoHandles )
      {
        while ( (gBS->HandleProtocol(Buffer[v7], &ProprietaryProtocol, &Interface) &
             || *(Interface + 24) )
        {
          if ( ++v7 >= NoHandles )
            goto _Exit;
        }
        UsbLegacyProtocolFunc1 = *(Interface + 1);
        UsbLegacyProtocolFunc2 = *(Interface + 2);
      }
    }
_Exit:
    Status = gBS->FreePool(Buffer);
```

FwBlockServiceSmm

```
EFI_STATUS __fastcall ChildSwSmiHandler(
        EFI_HANDLE DispatchHandle,
        const void *Context,
        struct_CommBuffer *CommBuffer,
        UINTN *CommBufferSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( CommBuffer && CommBufferSize && !gExitBootServices )
  {
    Interface = gProtocolInterface;
    if ( !gProtocolInterface )
    {
      if ( gBS->LocateProtocol(&ProprietaryProtocol, 0i64, &gProtocolInterface) & 0x8000000000000000ui64) != 0i64 )
        return 0i64;
      Interface = gProtocolInterface;
    }
```

- **BRLY-2022-017** – SMM callout in **FwBlockServiceSmm** module
- **BRLY-2022-022** – SMM callout in **UsbLegacyControlSmm** module
- The vulnerable code patterns in SMM can be triggered during boot
- An attacker capable of executing code in DXE phase could exploit this vulnerability to escalate privileges to SMM (ring -2)

BRLY-2022-018 and BRLY-2022-025

```
__int64 __fastcall SmfbFunc1(__int64 This, __int64 addr, __int64 offset, UINTN *size_ptr, void *dst)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  src = (offset + addr);
  size = *size_ptr;
  if ( *gValueInitializedByUnknownProtocol == 3 )
    return sub_31CC(size, dst, src);            // SMM memory read from a controllable address
  if ( *gValueInitializedByUnknownProtocol > 1u )
    return EFI_UNSUPPORTED;
  res = 0i64;
  if ( size && dst != src )
    CopyMem(dst, src, *size_ptr);               // SMM memory read from a controllable address
  return res;
}
```

- SMM memory leak vulnerabilities
- The same code pattern used in different drivers: **FwBlockServiceSmm**, **FvbServicesRuntimeDxe**
- *addr* – main pointer for the operation, retrieved as `*((_QWORD *)CommBuffer + 2))`
- *size* – size of data for the operation, located at `(_QWORD *)CommBuffer + 3)`
- *dest* – destination buffer address for the operation (if applicable), the rest of Communication Buffer located at `(__int64)(CommBuffer + 0x20)`

BRLY-2022-023

```
EFI_STATUS __fastcall SwSmiHandler(
        EFI_HANDLE DispatchHandle,
        const void *Context,
        void *CommBuffer,
        UINTN *CommBufferSize)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( ToGetIhisiParamBufferVar(&IhisiParamBufferValue, &Val, CommBuffer) >= EFI_SUCCESS )
  {
    Ptr = IhisiParamBufferValue;
    if ( *(IhisiParamBufferValue + 16) == 'BSI$' )
    {
      Index = *(*(IhisiParamBufferValue + 0x28) + 6i64);
      if ( (Index & 0xFFF8) == 80 )             // Index: 80, 81, 82, 83, 84, 85, 86, 87
        ResValue = (gFuncsTable[Index & 7])(*(IhisiParamBufferValue + 0x28));
      else
        ResValue = 0x82;
      *(Ptr + 8) = ResValue;
    }
  }
  return 0i64;
}
```

```
; __int64 __fastcall GetIhisiParamBufferVar()
GetIhisiParamBufferVar proc near        ; CODE XREF: ToGetIhisiParamBufferVar+34↓p

arg_20          = qword ptr  28h

                mov     rax, cs:gEfiSmmVariableProtocol
                test    rax, rax
                jnz     short loc_4CF4
                cmp     cs:gSmst_3, rax
                jz      short loc_4CCC
                mov     rax, EFI_UNSUPPORTED
                retn
; ---------------------------------------------------------------------------

loc_4CCC:                               ; CODE XREF: GetIhisiParamBufferVar+13↑j
                mov     rax, cs:gRT_2
                lea     rcx, gIhisiParamBufferValue ; Ptr
                mov     [rsp+arg_20], rcx
                lea     rdx, gVariableGuid
                lea     rcx, aIhisiparambuff ; "IhisiParamBuffer"
                xor     r8d, r8d
                jmp     qword ptr [rax+48h] ; gRT->GetVariable
                                        ; VariableName = IhisiParamBuffer
                                        ; VariableGuid = 92e59835-5f42-4e0b-9a84-47c7810ea806
                                        ; Attributes = 0
                                        ; DataSize = 8 (from parent function)
                                        ; Data = Ptr
```

- SMM memory corruption vulnerability
- The `Ptr` can be controlled via the NVRAM variable `IhisiParamBuffer {92e59835-5f42-4e0b-9a84-47c7810ea806}`
- A predictable value can be written by address `(Ptr + 8)`

BRLY-2022-024 and BRLY-2022-026

```
EFI_STATUS __fastcall sub_27F8(__int64 val, char *addr, char *dest, unsigned __int64 *size_ptr)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  // ChildSwSmiHandler -> SmfbFunc2 -> sub_27f8
  if ( *gValueInitializedByUnknownProtocol )
    return EFI_UNSUPPORTED;
  v8 = *(gValueInitializedByUnknownProtocol + 11);
  v9 = 0i64;
  v10 = 0i64;
  v11 = *(v8 + 16);
  v12 = *(v8 + 8);
  v16 = v11;
  if ( !*size_ptr )
    return v9;
  while ( 1 )
  {
    if ( addr == dest || !sub_1060(addr, dest, 1i64) )
      goto LABEL_12;
    SetMem(v12, 1ui64, 0xAAu);
    SetMem(v11, 1ui64, 0x55u);
    SetMem(v12, 1ui64, 0xA0u);
    // Write controllable byte to a controllable address
    SetMem(addr, 1ui64, *dest);
    sub_3D5C();
```

- SMM memory corruption vulnerability
- the same code pattern used in different drivers: **FwBlockServiceSmm**, **FvbServicesRuntimeDxe**
- *`addr`* and *`dest`* are controlled by the attacker through the contents of the `CommBuffer`
- In the same handler there are **multiple memory corruptions with the pattern** `SetMem(addr, 1, const_byte)`

BRLY-2022-019, BRLY-2022-020 and BRLY-2022-021

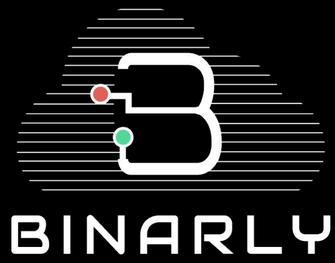
```
(gRT->GetVariable)(L"SecureBootEnforce", &EFI_GENERIC_VARIABLE_GUID, &Attributes, &DataSize, &SecureBootEnforceData);
gRT->GetVariable(L"SecureBoot", &EFI_SIMPLE_BOOT_FLAG_VARIABLE_GUID, 0i64, &DataSize, &SecureBootData);
Res = SecureBootEnforceData & SecureBootData;
Status = 0i64;
if ( !gRT->GetVariable(L"RestoreBootSettings", &gVendorGuid, 0i64, &DataSize, &RestoreBootSettingsData)
```

```
DataSize = 17i64;
gRT->GetVariable(L"MeSetupStorage", &gVariableGuid, 0i64, &DataSize, MeSetupStorageValue);
gRT->GetVariable(L"MeBackupStorage", &gVariableGuid, 0i64, &DataSize, MeBackupStorageValue);
```

```
DataSizea = 0i64;
Status = gRT->GetVariable(L"MebxData", &VendorGuid, 0i64, &DataSizea, &MebxDataValue);
if ( Status == EFI_BUFFER_TOO_SMALL )
  Status = gRT->GetVariable(L"MebxData", &VendorGuid, 0i64, &DataSizea, &MebxDataValue);
```

- A first service call `gRT->GetVariable()` can be used to set the stack variable `DataSize`
- A second `gRT->GetVariable()` service call can be used to overflow the stack buffer and execute the arbitrary code

- This vulnerable pattern is regularly found in the firmware of different manufacturers:
  - https://binarly.io/posts/Repeatable_Firmware_Security_Failures_16_High_Impact_Vulnerabilities_Discovered_in_HP_Devices/index.html
  - https://twitter.com/esetresearch/status/1547166334651334657

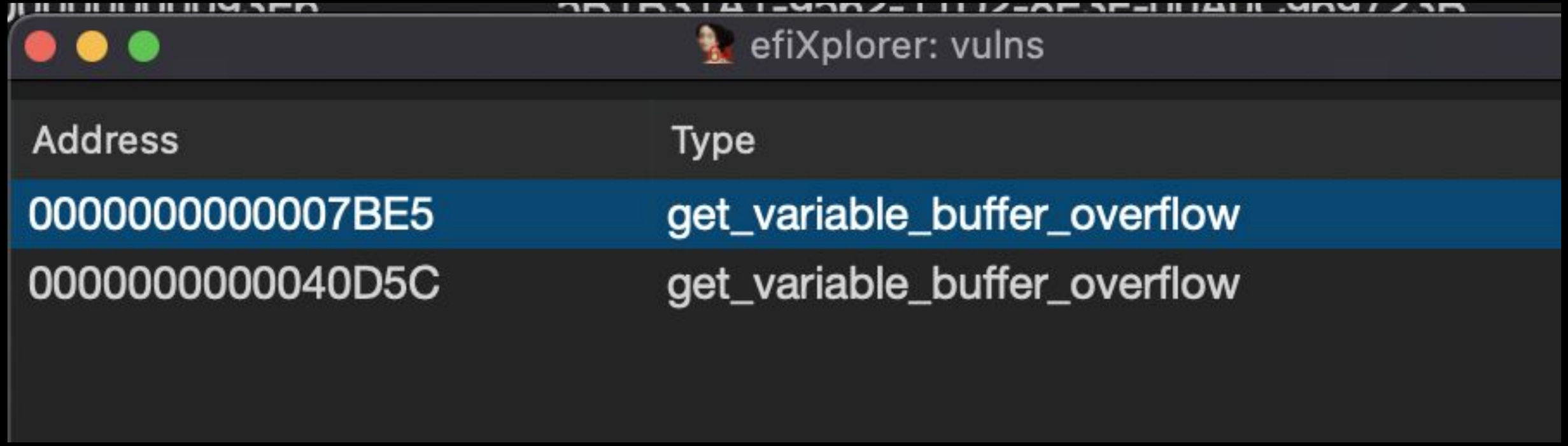


BRLY-2022-019:
https://www.insyde.com/security-pledge/SA-2022039

BRLY-2022-020:
https://www.insyde.com/security-pledge/SA-2022040

BRLY-2022-021:
https://www.insyde.com/security-pledge/SA-2022041

BRLY-2022-019, BRLY-2022-020 and BRLY-2022-021



Finding Repeatable Failures

same problems, new discoveries

efiXplorer: analysis of Aarch64 modules

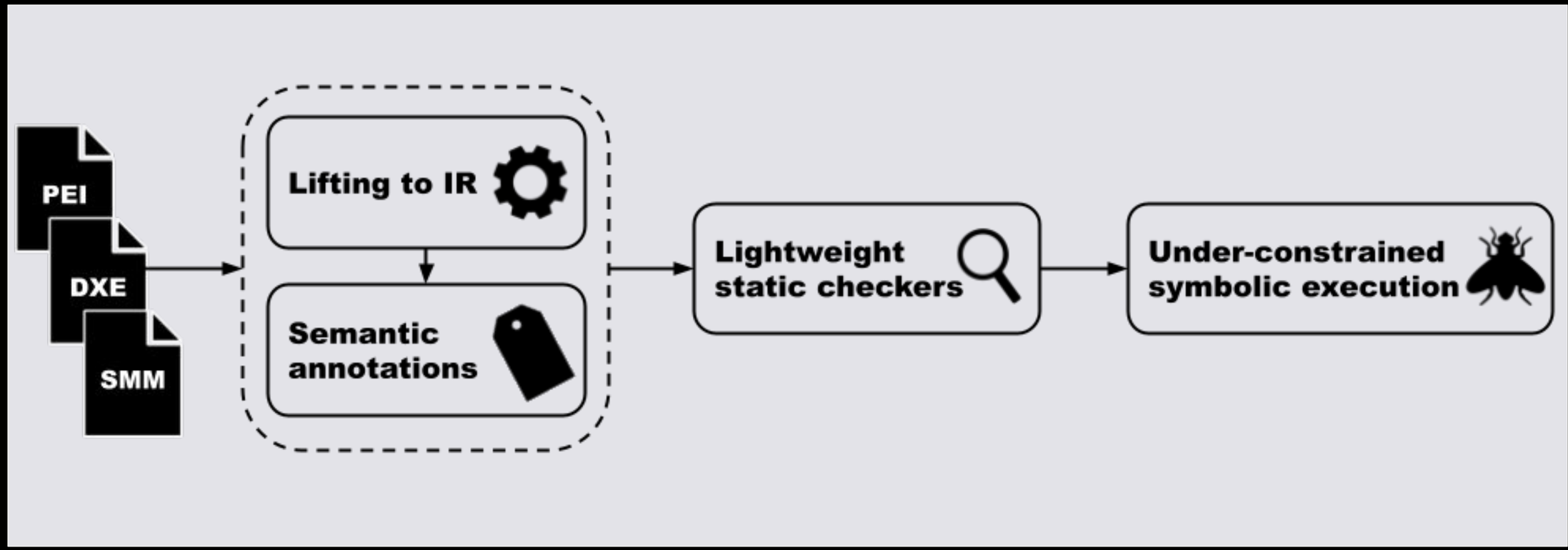
```
EFI_STATUS __cdecl ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  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 == 0x80000000000000Eui64 )
    {
      v11 = (gBS->LocateProtocol)(&EDKII_VARIABLE_POLICY_PROTOCOL_GUID, 0i64, &gEdkiiVariablePolicyProtocol
      if ( (v11 & 0x8000000000000000ui64) != 0 )
      {
        gEdkiiVariablePolicyProtocol = 0i64;
        if ( v11 == 0x800000000000000Eui64
          && ((gBS->LocateProtocol)(&EDKII_VARIABLE_LOCK_PROTOCOL_GUID, 0i64, &gEdkiiVariableLockProtocol)
        {
          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

https://github.com/binarly-io/efiXplorer

Binarly Internal Analysis Pipeline



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

https://www.binarly.io/posts/Using_Symbolic_Execution_to_Detect_UEFI_Firmware_Vulnerabilities

```
(base)
┌──(sam@binarly)-
└─[~/Projects/binarly-symbolic]─  ./target/release/smiscan -v -d data ./SmmSmbiosElog-8e61fd6b-7a8b-404f-b83f-aa90a47cabdf.smm
```

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

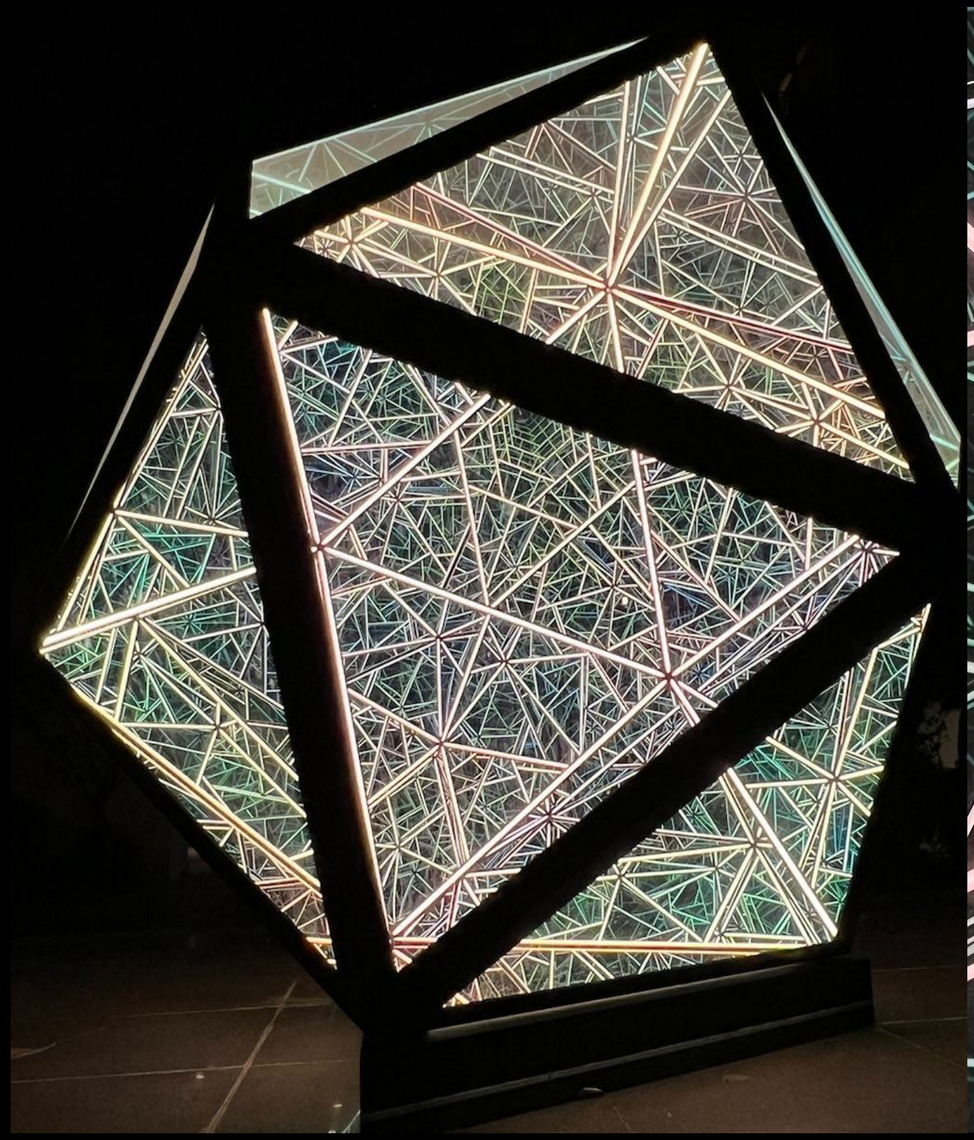
FwHunt detection rules: https://github.com/binarly-io/FwHunt/tree/main/rules

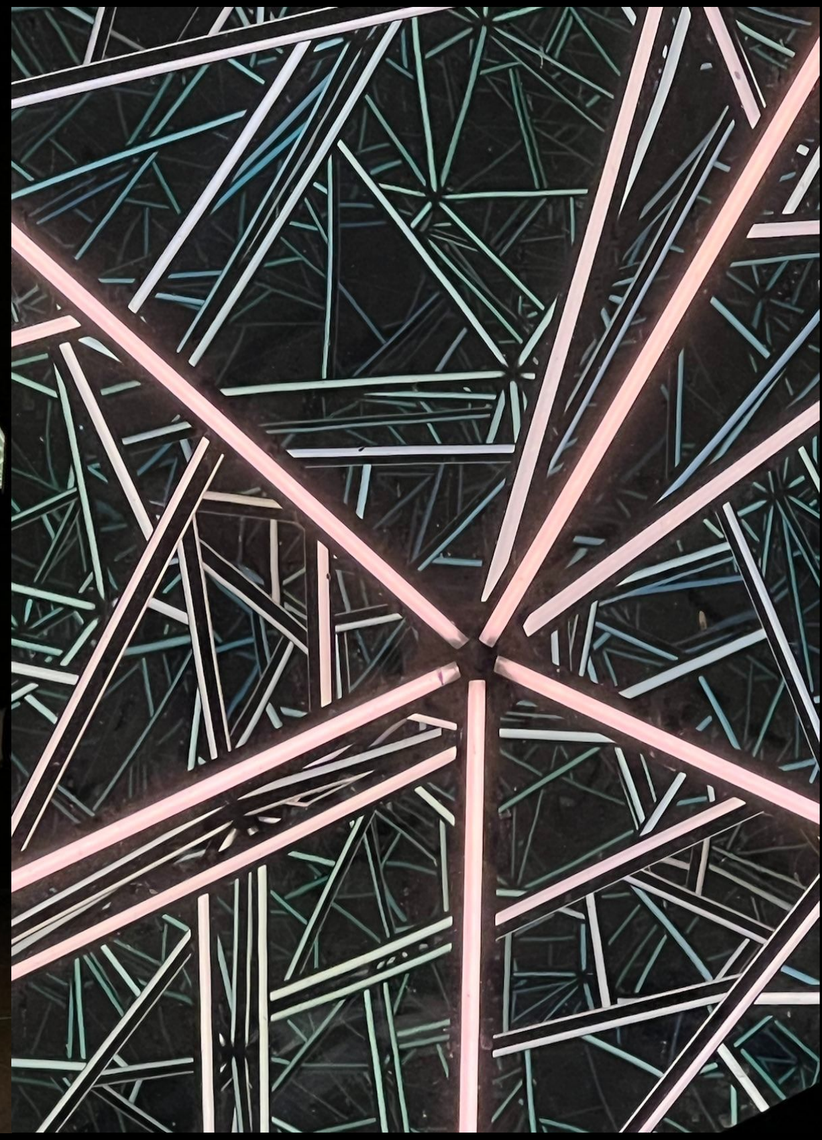
FwHunt Scan Results

- not present - not detected - detected

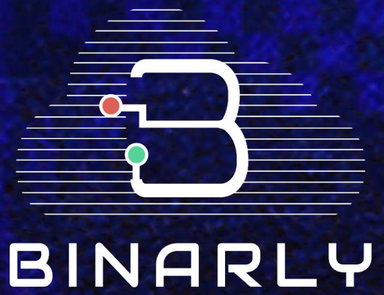
Status	Type	Description
Detected	Vulnerability	BRLY-2022-016 : Stack overflow vulnerability in SMI handler
Detected	Vulnerability	BRLY-2022-027 : The stack buffer overflow vulnerability leads to arbitrary code execution during PEI phase
Detected	Vulnerability	BRLY-2022-003 : SMM memory corruption vulnerability in OverClockSmiHandler SMM driver
Detected	MitigationFailures	BRLY-2022-028-RsbStuffing : Check if StuffRsb used before RSM
Detected	Vulnerability	BRLY-2022-014 : The arbitrary write vulnerability leads to arbitrary code execution during PEI phase
Not detected	Vulnerability	BRLY-2021-045 : SMM arbitrary code execution in USBRT SMM driver on Dell devices
Not detected	Vulnerability	BRLY-2022-004 : SMM arbitrary code execution in USBRT SMM driver on Dell devices

Complexity is the Enemy of Security





What the next?



THANKS! GRACIAS!

@matrosov

fwhunt@binarly.io