The Firmware Supply-Chain Security Is Broken: Can We Fix It?

Alex Matrosov, Alex Ermolov, Richard Hughes & Kai Michaelis

Alex Matrosov
Binarly

Alex Ermolov
Binarly

Richard Hughes
Red Hat

Kai Michaelis
Immune GmbH

Who are we?

Alex Matrosov (@matrosov)
Founder and CEO @Binarly_io
All shades of security REsearch below the OS

Alex Ermolov (@flothrone)
Principal Security Researcher @Binarly_io
Low-level design, firmware and system software
Fuzzing & testing automation

Richard Hughes (@hughsient)
Principal Engineer at Red Hat
LVFS and fwupd maintainer

Kai Michaelis
CTO @ immune
Previously FW & crypto developer

What is the Firmware and Hardware Supply Chain?

Firmware?

FPGA firmware

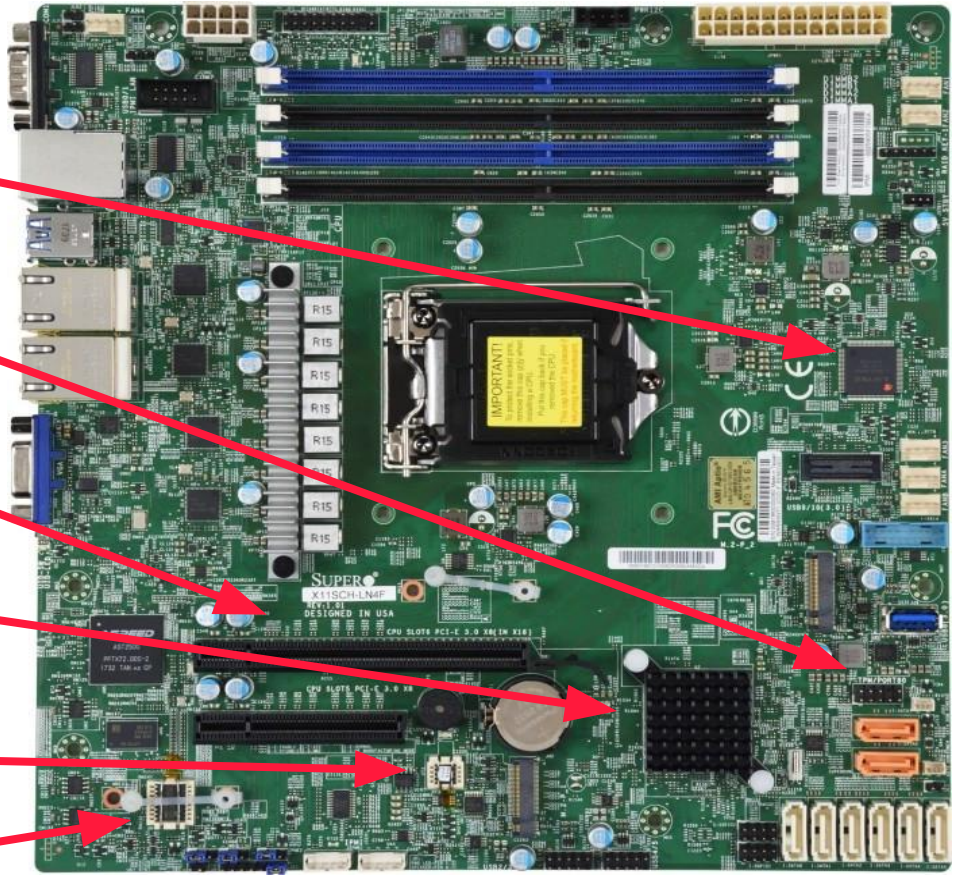
TPM firmware

PCI option ROMs

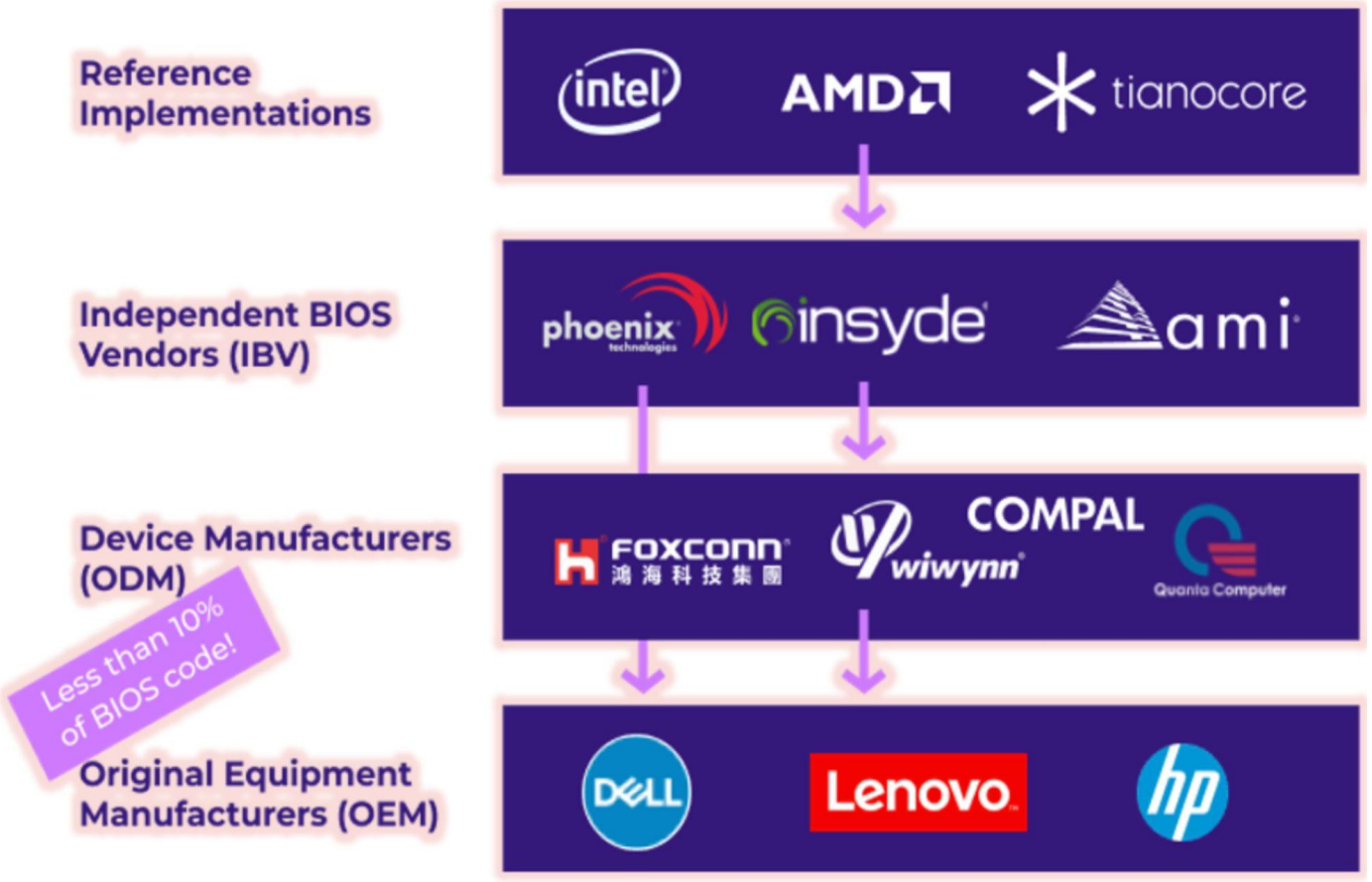
Southbridge firmware

UEFI/BIOS firmware

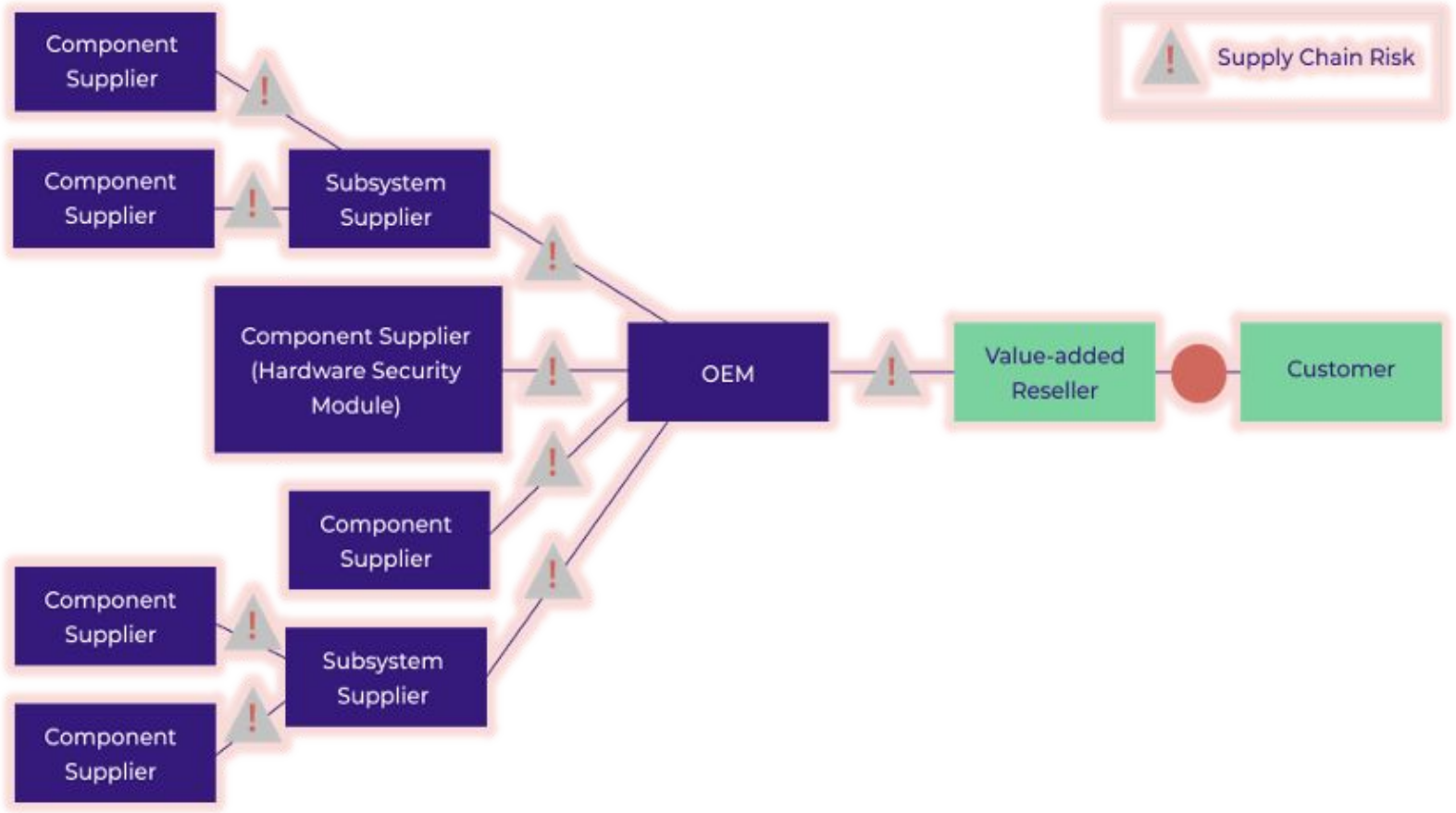
BMC firmware



Firmware Supply Chain Risks



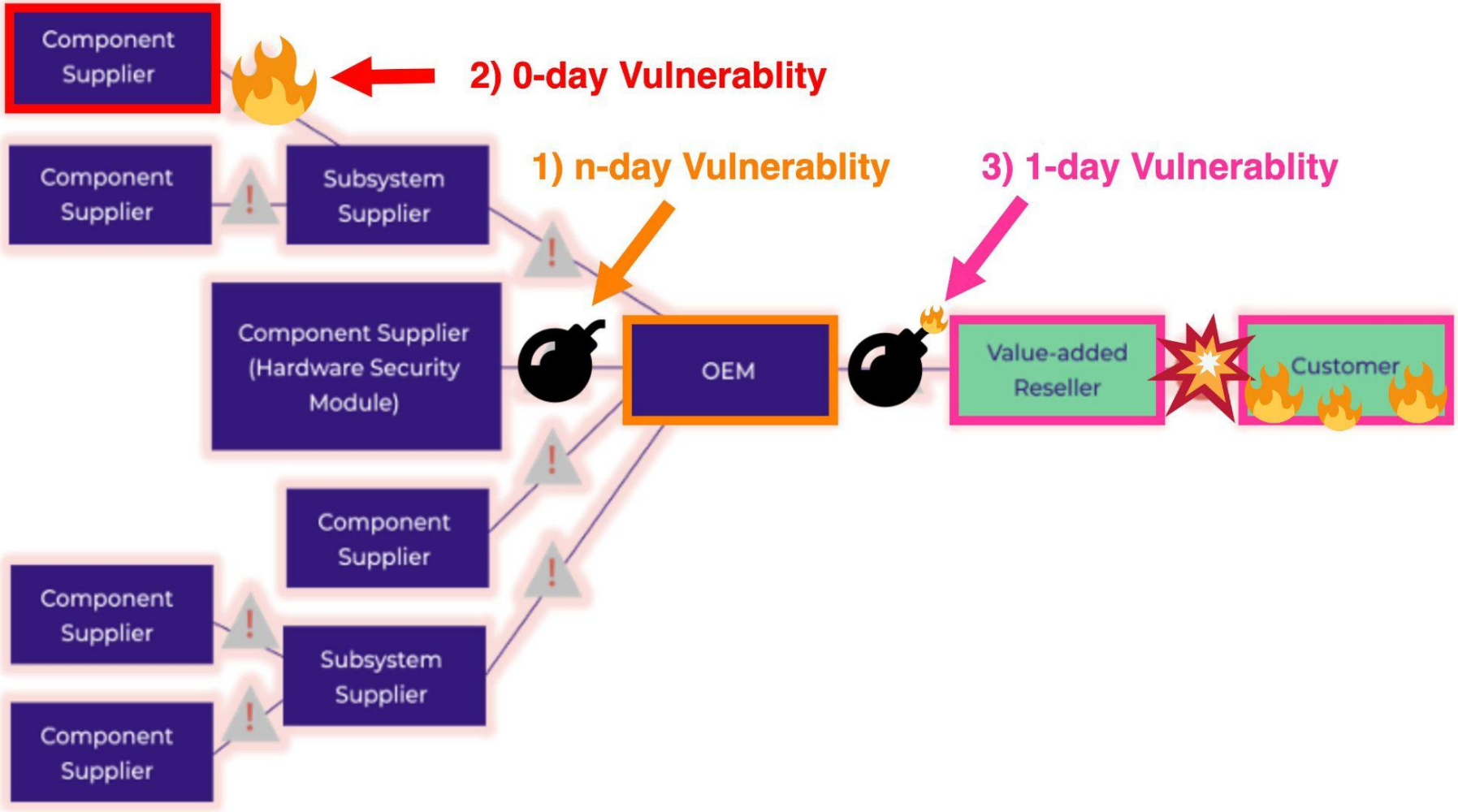
Firmware Supply Chain Risks



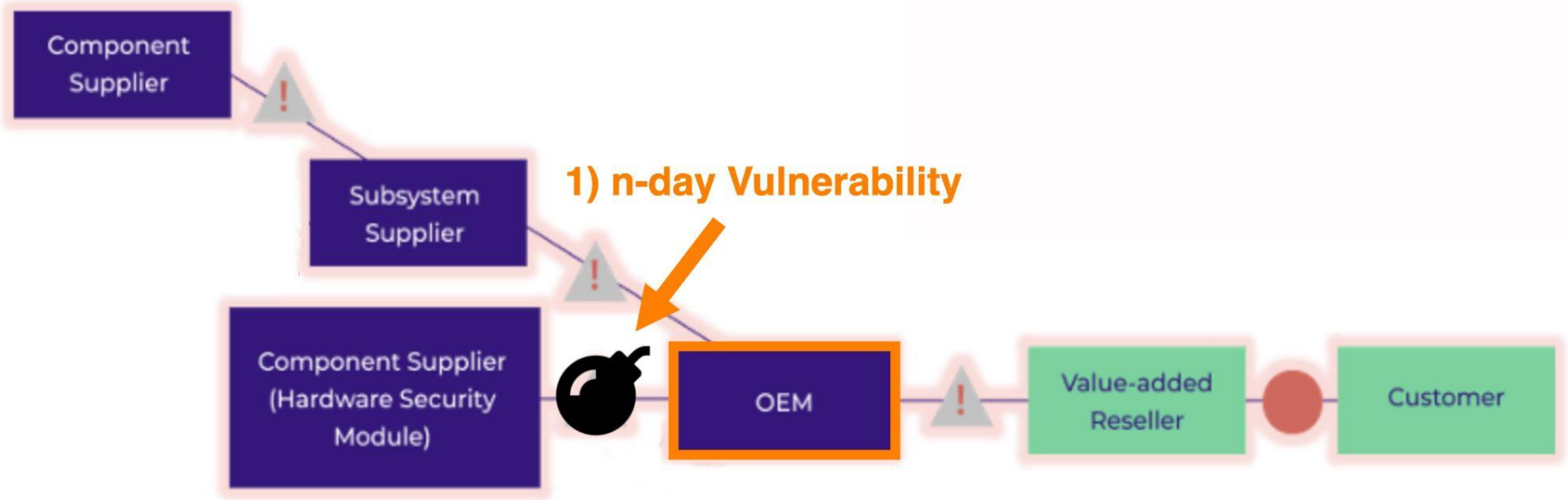
Firmware Supply Chain Risks

- **It's a tree of suppliers and sub-suppliers**
  - Firmware and it's updates flow downstream to the OEM.
  - Little to no coordination.
    - ➤ **More suppliers means more complexity and more risk**

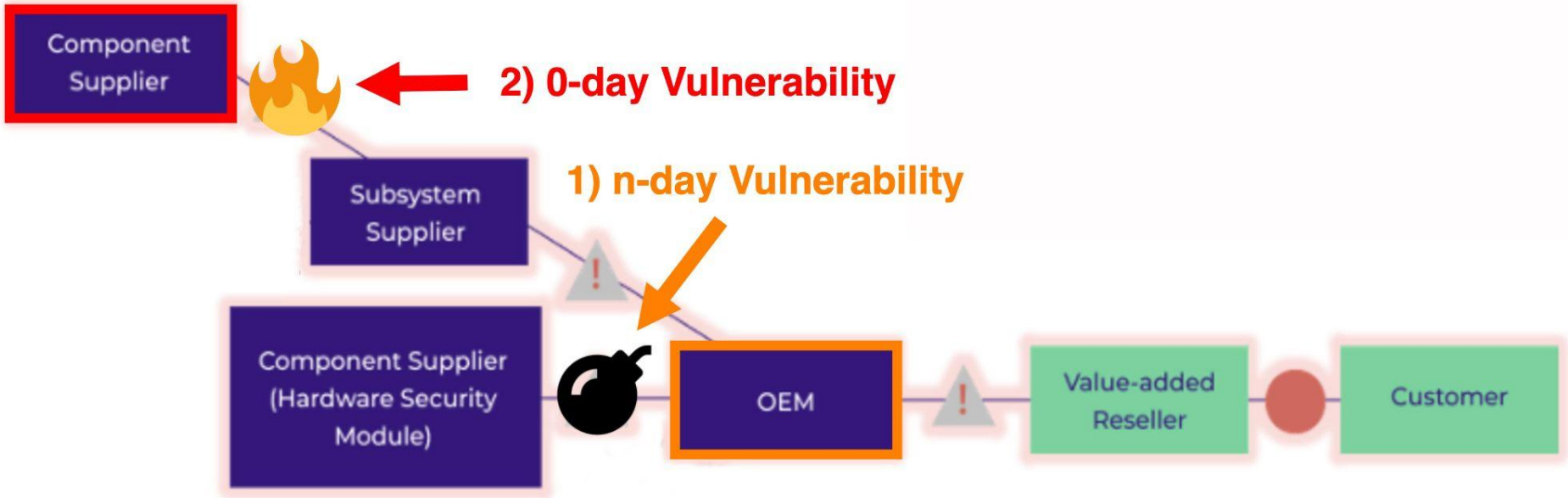
- **OEMs need to package updates**
  - Packaging updates is often a fairly manual process.
    - ➤ **Consumers need to actually install that update**


Component Supplier
2) 0-day Vulnerablity
Component Supplier
Subsystem Supplier
1) n-day Vulnerablity
3) 1-day Vulnerablity
Component Supplier (Hardware Security Module)
OEM
Value-added Reseller
Customer
Component Supplier
Component Supplier
Subsystem Supplier
Component Supplier

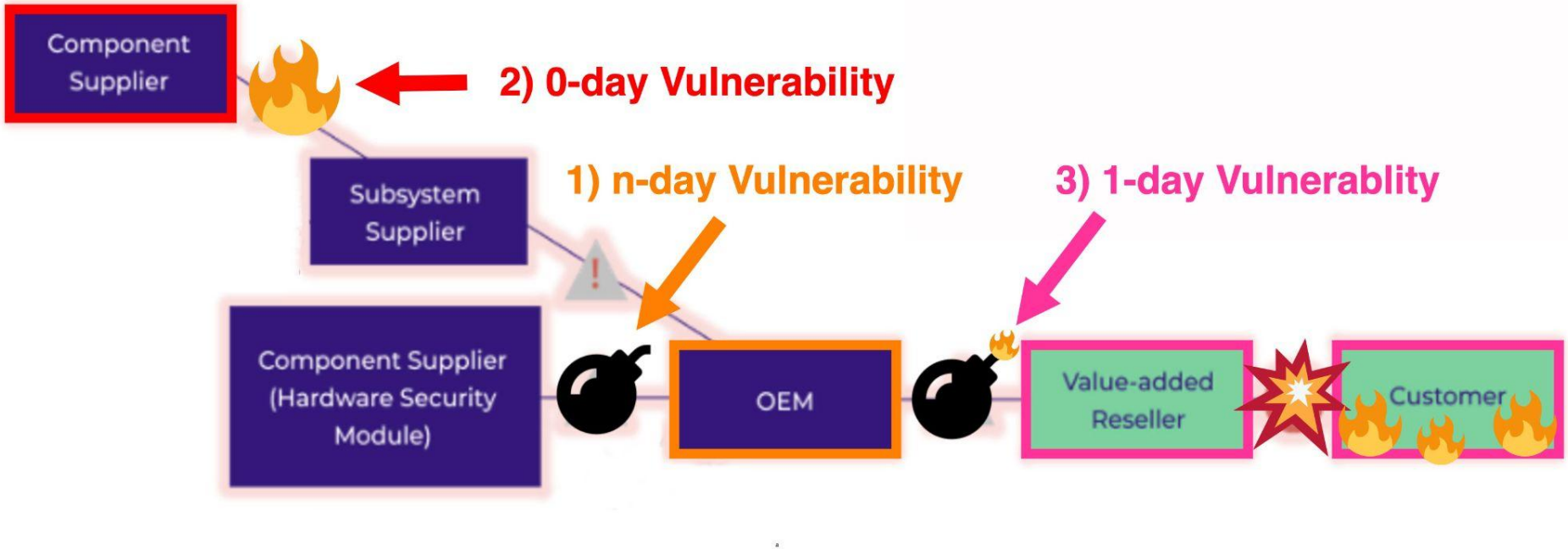
Firmware Supply Chain Risks



Firmware Supply Chain Risks



Firmware Supply Chain Risks



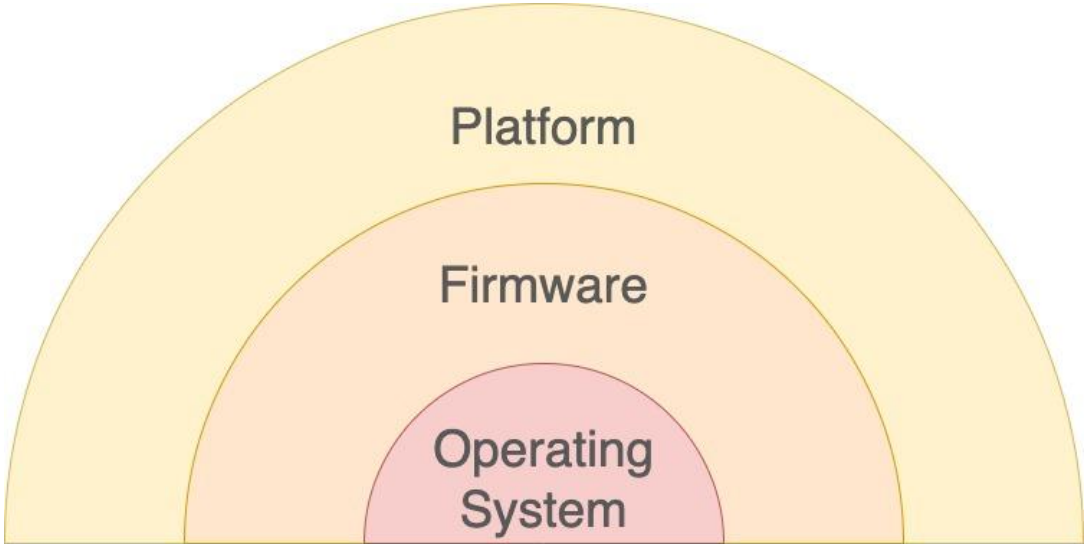
Firmware Supply Chain Risks

- **Vulnerabilities are assessed in isolation**
  - One may look unexploitable and thus not fixed
  - But, if combined with another becomes exploitable
    - ➤ **Benign looking bugs can have devastating impact**

- **Vulnerabilities have long lifetimes**
  - Devices reaching their EOL are no longer updated
  - Lack of coordination with downstream suppliers.
    - ➤ **Possibility of Supply Chain Race Conditions**

Layers of Security

- **Platform protections**
  - Boot Guard
  - TPM measurements
- **Firmware protections**
  - UEFI Secure Boot
  - Capsule signatures
  - SMM code
- **Operating system protections**
  - Signed kernel modules and applications



- **If things break, everything after breaks too**

Can we fix it?

- **Make vendors care more about security**
  - Security update are a necessity, even after EOL.
  - Dramatically decrease turnaround times.
- **Increase Transparency**
  - Improve coordination between vendors.
  - Centralized update repositories.
- **Open Source Firmware**
  - Allow end users to inspect and (reproducibly) build their firmware.
  - Closed source vendors are well entrenched
  - Widespread use of NDAs creates a high barrier of entry
- **Authenticated Hardware**
  - Bunnie's Precursor
  - Inspectable by the end user
  - Low upper bound for complexity

What are Firmware Supply Chain Failures?

Binarly Vulnerability Disclosures Statistics

Vulnerability Category	Count	Average Impact
SMM Privilege Escalation	15	CVSS: 8.2
SMM Memory Corruption	22	CVSS: 8.2
DXE Memory Corruption	5	CVSS: 7.7

* Based on Binarly disclosures: https://www.binarly.io/advisories

LEADER BOARD

ROUND ONE

Join us in congratulating...

First to Submit:
ZwinK

Highest Single Payout:
Mickey (@HackingThings)

Most Eligible Reports:
Alex Matrosov

Widest Impact:
Alex Matrosov

Helping Others:
Mickey (@HackingThings)

Just in Time:
Dan Lutas

ROUND TWO

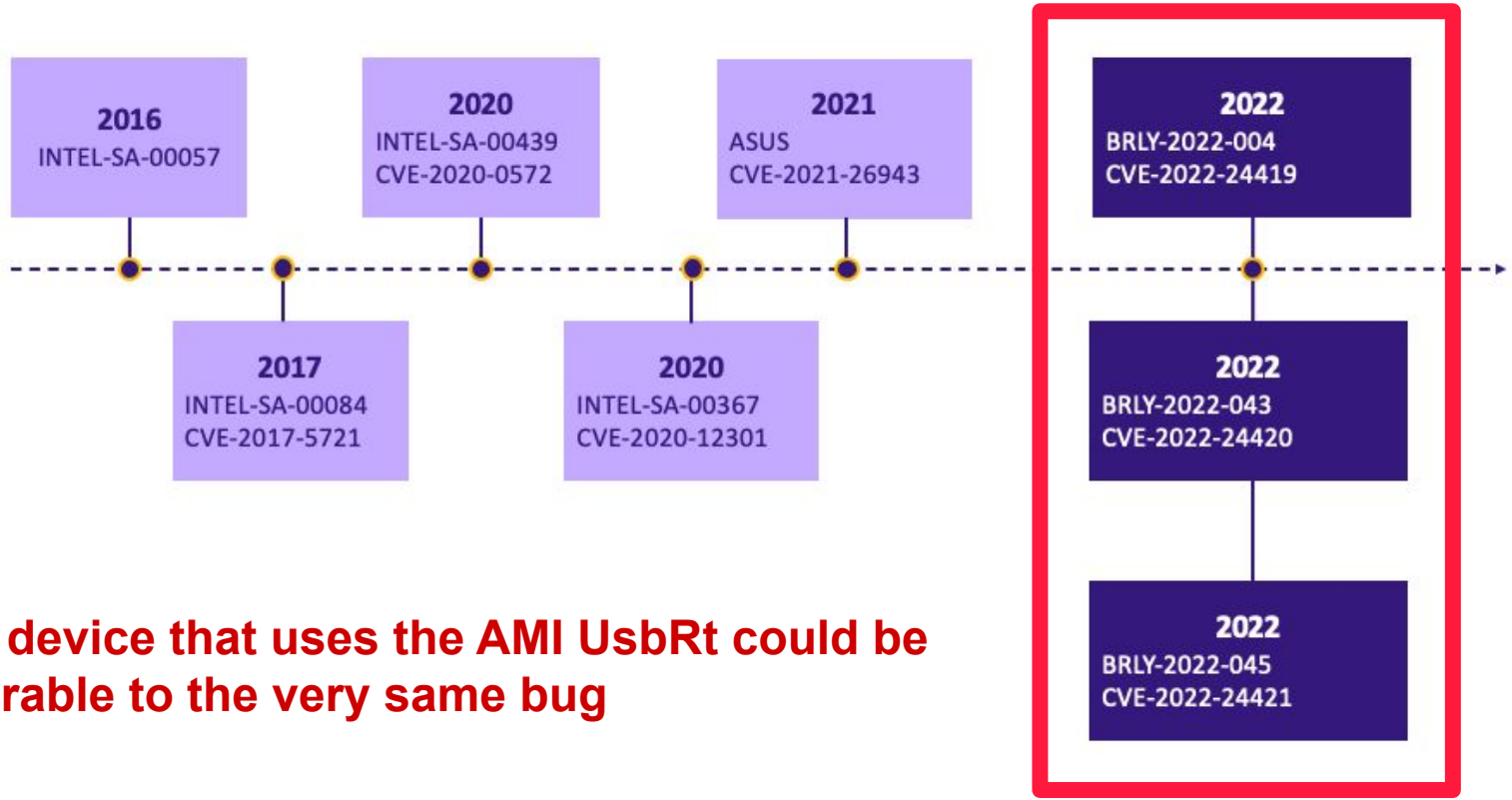
Bounty recipients named mid-April

ROUND THREE

Bounty recipients named mid-May

* **Intel Project Circuit Breaker**: https://www.projectcircuitbreaker.com/

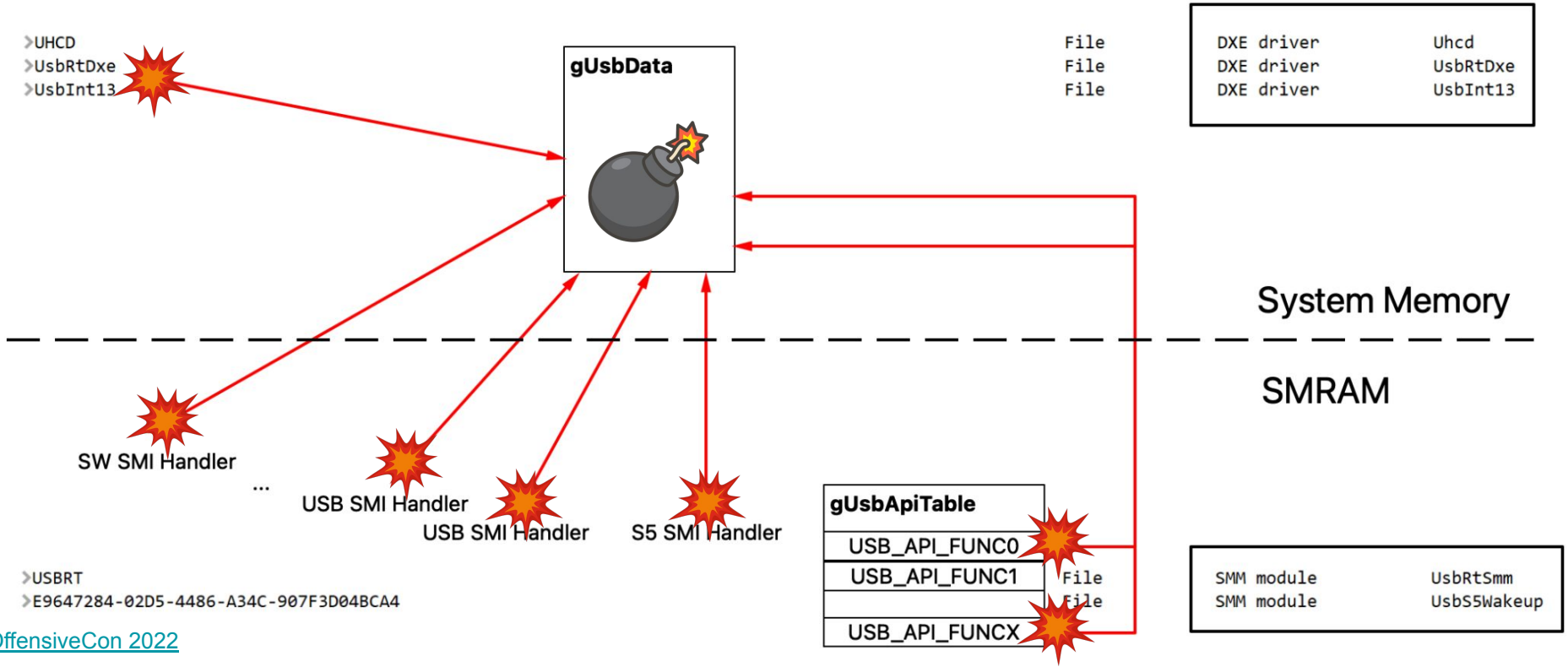
AMI UsbRt vulnerability is a perfect example of supply chain failures and code complexity



! Any device that uses the AMI UsbRt could be vulnerable to the very same bug

AMI UsbRt architecture and class of attacks

Initially complex architecture with a lot of pointers stored inside an object in system memory (Global USB Data) looks like this:



BTW how's UsbRt doing in 2022?

- **CVE-2017-5721** discovered in-the-wild (again)
- Exists on some **devices** - still in disclosure process with the vendor at the moment, the device was **receiving updates in 2021**
- **SMM_Code_Chk_En** not set, so no mitigation to block execution outside SMRAM

First ROP technique against SMM demo in public!

- **CVE-2020-12301** discovered in-the-wild
- Exists on some devices - still in disclosure process with the vendor at the moment, the device was **receiving updates in 2021**
- **SMM_Code_Chk_En is set**, execution of code outside SMRAM is not permitted

UsbRt 2022 ROP PoC

AMI Clarification on UsbRt issues

Feb 11, 2022 | Tech Blog

Recent news and reports in technology media and from leading security researchers indicate continued interest in UEFI security exploits (see: MoonBounce, Insyde vulnerability). Following this trend, security researchers from Binarly made a presentation last week at the OffensiveCon 2022 security event in Berlin titled "UEFI Firmware Vulnerabilities: Past, Present and Future". In the "Past" portion of its presentation, Binarly referred to a firmware vulnerability located in a section of UEFI BIOS firmware reference source code from AMI.

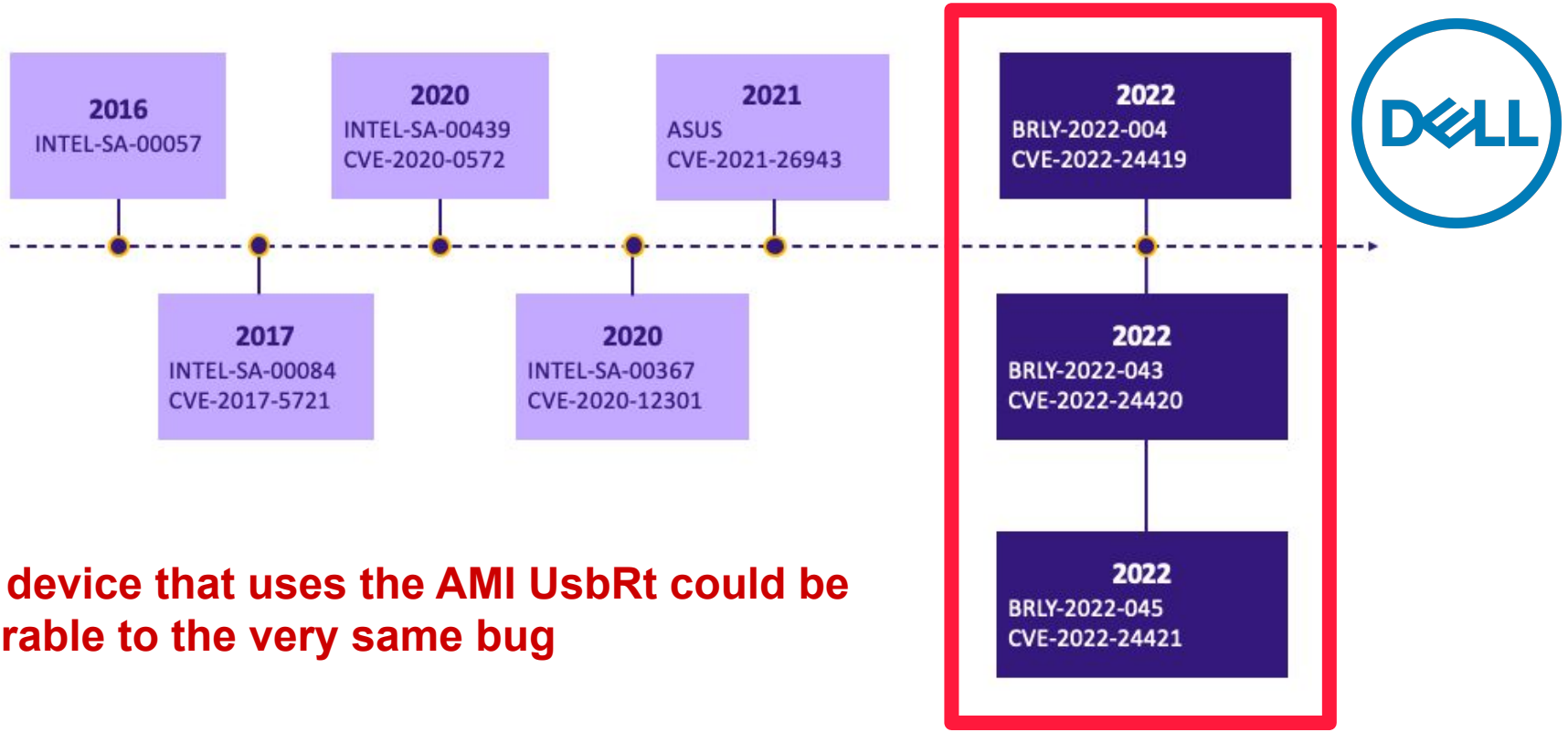
In short, AMI can confidently state that the vulnerability described in the presentation is firmly in the past – as indicated within the presentation. AMI resolved and closed this security issue several years ago. However, to alleviate any potential concern that our partners, customers or end-users may have, AMI can share the following additional details:

What is the compromised code in question?

The Binarly presentation refers to a section of AMI UEFI BIOS reference code in the context of showing historical and current examples of UEFI firmware vulnerabilities. It specifically mentioned the "AMI UsbRT architecture" and described an attack methodology for a vulnerability within it. AMI would like to emphasize that this portion of AMI source code is now approximately seven years old and no longer featured in current AMI UEFI products.

AMI Clarification on UEFI Firmware Vulnerabilities Presentation at OffensiveCon 2022

AMI UsbRt vulnerability is a perfect example of supply chain failures and code complexity



! Any device that uses the AMI UsbRt could be vulnerable to the very same bug

OSFC 2021

Collisions in Supply Chain: Intel M15

OverClockSmiHandler - 4698C2BD-A903-410E-AD1F-5EEF3A1AE422

BRLY-2022-003 - CVE-2022-27493

```
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;
Ptr16 = (RbxValue + 16);
```

- SMM memory corruption - no validation applied for input *Ptr*
- **2019 y. vulnerability** from **AMI codebase** discovered in 2022 firmware
- Fixed: vulnerable module removed

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

Collisions in Supply Chain: Intel M15

PlatformInitAdvancedPreMem - EEEE611D-F78F-4FB9-B868-55907F169280

BRLY-2022-004 - CVE-2022-28858

```
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;
}
```

1. Fixed-size stack buffer
2. DataSize will be rewritten if “SaSetup” size > DataSize
3. CpuSetupData overflowed if “CpuSetup” size > old DataSize

- Stack buffer overflow while EFI variable processing in DXE phase
- Vulnerability from **AMI codebase** discovered in 2022 firmware
- Fixed: size is hardcoded for the both read operations (before each operation)

Collisions in Supply Chain: HP EliteBook x360 1040 G8

017D - C3145BF3-201E-4838-88CD-8F5B7F7759A2

BRLY-2021-052

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

  if ( *(_DWORD *)CommBuffer == 'MCVP' )
    *((_QWORD *)CommBuffer + 3) = sub_1578(*((_QWORD *)CommBuffer + 1), *((_DWORD *)CommBuffer + 4));
  return 0;
}
```

If ACPI Communication Buffer is located 1 byte below SMRAM and CommBufferSize = 1 the validation in PiSmmCommunicationSmm will be passed

- SMM memory corruption - no validation applied for input *CommBuffer*
- Vulnerability in **HP codebase** discovered in 2022 firmware
- Fixed: added size check

https://support.hp.com/us-en/document/ish_5661066-5661090-16/hpsbhf03765

Collisions in Supply Chain: HP EliteBook x360 1040 G8

0614 - 03E0A38B-3FBE-49CB-B311-726611213182

BRLY-2021-053

```
char Buffer; // [rsp+A0h] [rbp+20h] BYREF
UINTN DataSize; // [rsp+A8h] [rbp+28h] BYREF
...

VendorGuid.Data1 = 0xFB3B9ECE;
*&VendorGuid.Data2 = 0x49334ABA;
*VendorGuid.Data4 = 0xD6B49DB4;
*&VendorGuid.Data4[4] = 0x5123897D;
gBS->LocateProtocol(&SA_POLICY_PROTOCOL_GUID, 0, &Interface);
...
ZeroMem(&Buffer, 1);
DataSize = 0;
Status = gRT->GetVariable(L"PciePwrMgmt", &VendorGuid, 0, &DataSize, &Buffer);
if ( Status == EFI_BUFFER_TOO_SMALL )
  Status = gRT->GetVariable(L"PciePwrMgmt", &VendorGuid, 0, &DataSize, &Buffer);
```

1. **Fixed-size stack buffer (1 byte)**
2. **DataSize = 0 and first GetVariable() call used to get the actual size**
3. **Buffer overflowed if “PciePwrMgmt” size > 1 byte**

- Stack buffer overflow while EFI variable processing in DXE phase
- Vulnerability in **HP codebase** discovered in 2022 firmware
- Fixed: removed second GetVariable() call

https://support.hp.com/us-en/document/ish_5661066-5661090-16/hpsbhf03765

Collisions in Supply Chain: HP EliteBook x360 1040 G8

- **BRLY-2021-050**
- **BRLY-2021-051**
- **BRLY-2021-052**
- **BRLY-2021-053**

```
PcdProtocol = LocatePcdProtocol();
if ( (PcdProtocol->Get8)(0x23B) == 1 )
```

```
*((_QWORD *)CommBuffer + 1) = status;
```

Patch without issuing advisory and CVE increases severity of a vulnerability:

- No notification
- Discourages ODMs/OEMs/IT etc. to push security fixes
- Puts endpoint customers at risk

```
Interface = 0;
if ( CommBuffer && CommBufferSize )
{
  Res = gSmst->SmmLocateProtocol(&ProprietaryProtocol_8, 0, &Interface);
  if ( !Res )
    Res = (*Interface)();
  *CommBuffer = Res;
}
```

https://support.hp.com/us-en/document/ish_5661066-5661090-16/hpsbhf03765

Collisions in Supply Chain: UEFI App with SMM code

TrustedDeviceSetupApp - 658D56F0-4364-4721-B70E-732DDC8A2771

BRLY-2021-044 - not exploitable on Intel M15

```
DataSize = GetDataSize(Data);
Buffer = gBuffer;
Size = DataSize;
while ( Buffer != &gBuffer )
{
  if ( !CompareMemWrapper(Buffer + 49, Data, Size) )
  {
    CopyMemWrapper((Buffer + 2), a2, 32);
    return 0;
  }
  Buffer = *Buffer;
}
Mem = AllocateZeroPool(Size + 0x31);          // Callout here (gBS->AllocatePool)
```

- Call-out vulnerability in **SMI handler registered in UEFI Application**
- Code removed from EDKII in 2018
- The pattern discovered in 2022 firmware, linked from another library in SecurityPkg by mistake

Constraints of source code static analysis

```
// BRLY-2021-040 (CVE-2022-23932)
// HP coordinated fix 03/08/2022

if ( CommBuffer->Sig == 'GFCU' )
        {
          if ( CommBuffer->Case == 0x10 )
          {
            if ( !gBufferPtr )
            {
              BufferPtr1 = GetCopy(0x78, &CommBuffer->BufferPtr);
              BufferSize = CommBuffer->BufferSize;
              BufferPtr2 = CommBuffer->BufferPtr;
              gBufferPtr = BufferPtr1;
              sub_2288(BufferPtr2, BufferSize);

              // Vulnerability present below
              PcdProtocol = BsLocatePcdProtocol();
              if ( (PcdProtocol->Get8)(0x2C4) == 1 )
                HandlerUnregister();
            }
          }
          ...
        }
```

```
// BRLY-2021-047 (CVE-2022-XXXXX)
// HP silent fix 03/15/2022

if ( CommBuffer->Sig == 'GFCU' )
        {
          switch ( CommBuffer->Case )
          {
            case 0x10:
              if ( !gBufferPtr )
              {
                BufferPtr1 = GetCopy(0x78, &CommBuffer->BufferPtr1);
                BufferSize = CommBuffer->BufferSize;
                BufferPtr2 = CommBuffer->BufferPtr;
                gBufferPtr = BufferPtr1;
                sub_261C(BufferPtr2, BufferSize);

                // Vulnerability present below
                PcdProtocol = BsLocatePcdProtocol();
                if ( (PcdProtocol->Get8)(0x23B) == 1 )
                  HandlerUnregister();
              }
            }
            ...
        }
```

https://www.binarly.io/advisories/BRLY-2021-040

Compilers-generated artifacts

SmmIsBufferOutsideSmmValid() - SMM input pointer validation routine

1 - normal version

2 - compiler-optimized version (hardcoded size)

1
```
char __fastcall SmmIsBufferOutsideSmmValid(unsigned __int64 ptr, unsigned __int64 size)
{
```
2
```
char __fastcall SmmIsBufferOutsideSmmValid(unsigned __int64 ptr)
{
```
...

1
```
if ( size <= gTopMemoryAddress && ptr <= gTopMemoryAddress )
```
2
```
if ( (unsigned __int64)gTopMemoryAddress >= 0x20 && ptr <= gTopMemoryAddress && ptr <= gTopMemoryAddress - 0x1F )
```
...

1
```
if ( v6 < ptr + size )
  return 0;
```
2
```
if ( v5 < ptr + 0x20 )
  return 0;
```
...

1
```
if ( ptr >= *(_QWORD *)(v9 + 8) && ptr + size <= *(_QWORD *)(v9 + 8) + (*(_QWORD *)(v9 + 0x18) << 12) )
```
2
```
if ( ptr >= *(_QWORD *)(v8 + 8) && ptr + 0x20 <= *(_QWORD *)(v8 + 8) + (*(_QWORD *)(v8 + 0x18) << 12) )
```
...

Compilers-Generated Artifacts

SmmIsBufferOutsideSmmValid() - SMM input pointer validation routine

1 - normal version

2 - compiler-optimized version (hardcoded size)

1
```
char __fastcall SmmIsBufferOutsideSmmValid(unsigned __int64 ptr, unsigned __int64 size)
{
```
2
```
char __fastcall SmmIsBufferOutsideSmmValid(unsigned __int64 ptr)
{
```

...

1
```
if ( size <= gTopMemoryAdd
```
2
```
if ( (unsigned __int64)gTo
```

...

1
```
if ( v6 < ptr + size )
  return 0;
```
2
```
if ( v5 < ptr + 0x20 )
  return 0;
```

...

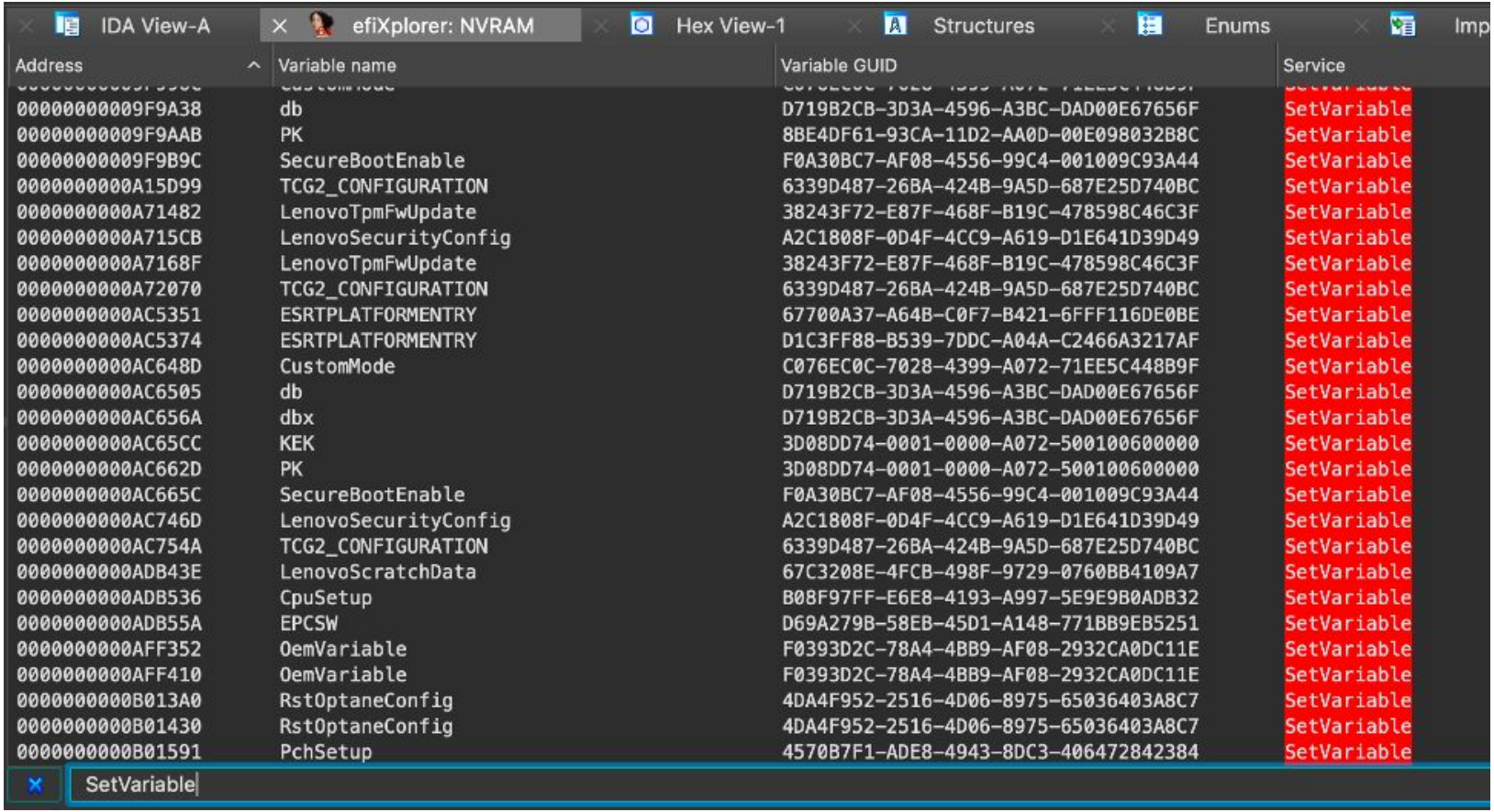
1
```
if ( ptr >= *(_QWORD *)(v9 + 8) && ptr + size <= *(_QWORD *)(v9 + 8) + (*(_QWORD *)(v9 + 0x18) << 12) )
```
2
```
if ( ptr >= *(_QWORD *)(v8 + 8) && ptr + 0x20 <= *(_QWORD *)(v8 + 8) + (*(_QWORD *)(v8 + 0x18) << 12) )
```

...

```
result = sub_800049D0(CommBuffer);
LABEL_81:
  byte_800077B8 = 0;
LABEL_82:
  *((_QWORD *)CommBuffer + 0x28) = result;
```

Out of validation boundaries write operation

github.com/binarly-io/efiXplorer



Address	Variable name	Variable GUID	Service
00000000009F9A38	db	D719B2CB-3D3A-4596-A3BC-DAD00E67656F	SetVariable
00000000009F9AAB	PK	8BE4DF61-93CA-11D2-AA0D-00E098032B8C	SetVariable
00000000009F9B9C	SecureBootEnable	F0A30BC7-AF08-4556-99C4-001009C93A44	SetVariable
0000000000A15D99	TCG2_CONFIGURATION	6339D487-26BA-424B-9A5D-687E25D740BC	SetVariable
0000000000A71482	LenovoTpmFwUpdate	38243F72-E87F-468F-B19C-478598C46C3F	SetVariable
0000000000A715CB	LenovoSecurityConfig	A2C1808F-0D4F-4CC9-A619-D1E641D39D49	SetVariable
0000000000A7168F	LenovoTpmFwUpdate	38243F72-E87F-468F-B19C-478598C46C3F	SetVariable
0000000000A72070	TCG2_CONFIGURATION	6339D487-26BA-424B-9A5D-687E25D740BC	SetVariable
0000000000AC5351	ESRTPLATFORMENTRY	67700A37-A64B-C0F7-B421-6FFF116DE0BE	SetVariable
0000000000AC5374	ESRTPLATFORMENTRY	D1C3FF88-B539-7DDC-A04A-C2466A3217AF	SetVariable
0000000000AC648D	CustomMode	C076EC0C-7028-4399-A072-71EE5C448B9F	SetVariable
0000000000AC6505	db	D719B2CB-3D3A-4596-A3BC-DAD00E67656F	SetVariable
0000000000AC656A	dbx	D719B2CB-3D3A-4596-A3BC-DAD00E67656F	SetVariable
0000000000AC65CC	KEK	3D08DD74-0001-0000-A072-500100600000	SetVariable
0000000000AC662D	PK	3D08DD74-0001-0000-A072-500100600000	SetVariable
0000000000AC665C	SecureBootEnable	F0A30BC7-AF08-4556-99C4-001009C93A44	SetVariable
0000000000AC746D	LenovoSecurityConfig	A2C1808F-0D4F-4CC9-A619-D1E641D39D49	SetVariable
0000000000AC754A	TCG2_CONFIGURATION	6339D487-26BA-424B-9A5D-687E25D740BC	SetVariable
0000000000ADB43E	LenovoScratchData	67C3208E-4FCB-498F-9729-0760BB4109A7	SetVariable
0000000000ADB536	CpuSetup	B08F97FF-E6E8-4193-A997-5E9E9B0ADB32	SetVariable
0000000000ADB55A	EPCSW	D69A279B-58EB-45D1-A148-771BB9EB5251	SetVariable
0000000000AFF352	OemVariable	F0393D2C-78A4-4BB9-AF08-2932CA0DC11E	SetVariable
0000000000AFF410	OemVariable	F0393D2C-78A4-4BB9-AF08-2932CA0DC11E	SetVariable
0000000000B013A0	RstOptaneConfig	4DA4F952-2516-4D06-8975-65036403A8C7	SetVariable
0000000000B01430	RstOptaneConfig	4DA4F952-2516-4D06-8975-65036403A8C7	SetVariable
0000000000B01591	PchSetup	4570B7F1-ADE8-4943-8DC3-406472842384	SetVariable

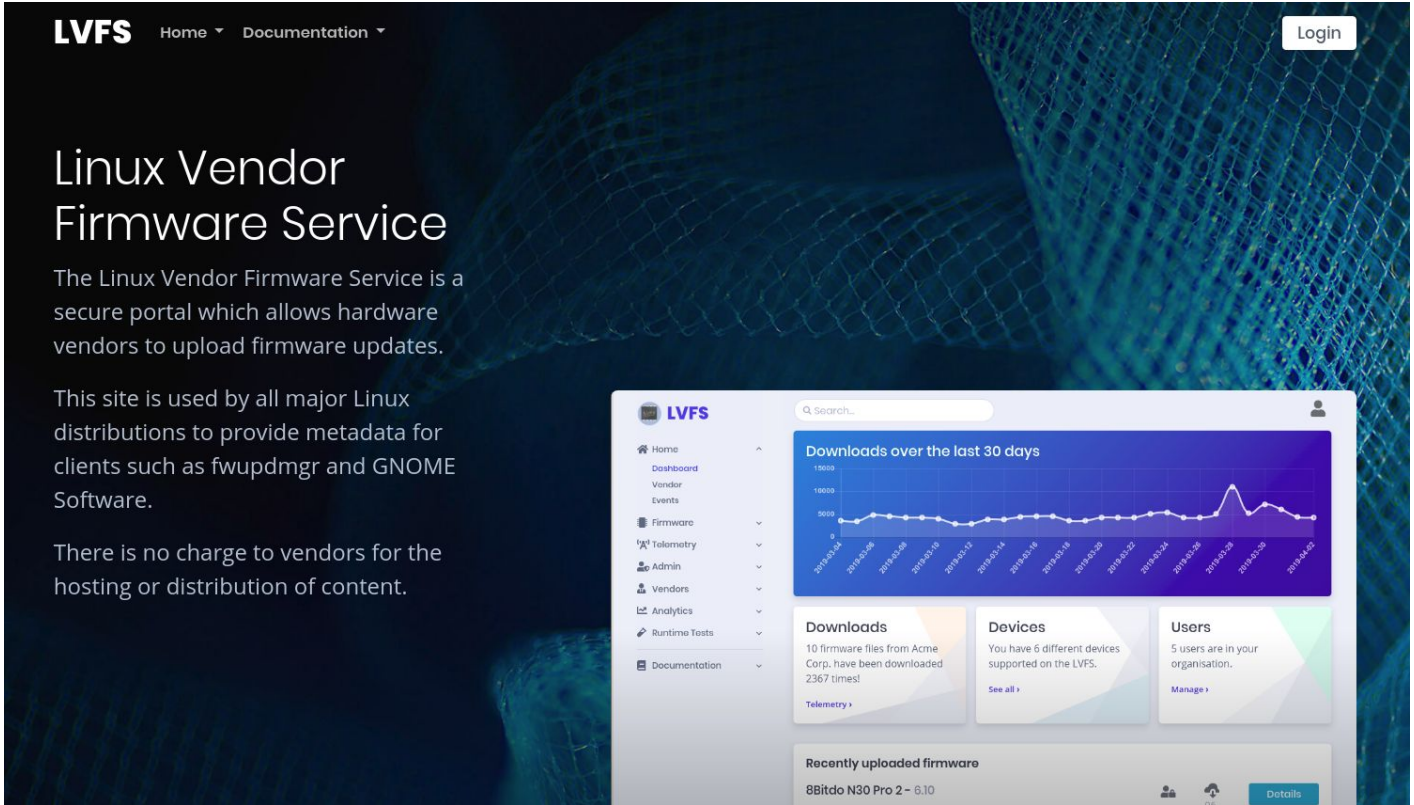
github.com/binarly-io/FwHunt

```yaml
BRLY-2021-011:
  meta:
    author: Binarly (https://github.com/binarly-io/FwHunt)
    license: CC0-1.0
    name: BRLY-2021-011
    namespace: vulnerabilities
    CVE number: CVE-2021-33627
    advisory: https://binarly.io/advisories/BRLY-2021-011/index.html
    description: SMM memory corruption vulnerability in combined DXE/SMM driver (SMRAM write)
    volume guids:
      - 74D936FA-D8BD-4633-B64D-6424BDD23D24
  variants:
    variant1:
      code:
        and:
          - pattern: 488b5310498d48204d8b4018e8....0000
            place: child_sw_smi_handlers
          - pattern: 4981392010000075
            place: child_sw_smi_handlers
    variant2:
      code:
        - pattern: 488b5310498d40204c8bc948894424..4533c033c9e8
          place: child_sw_smi_handlers
```

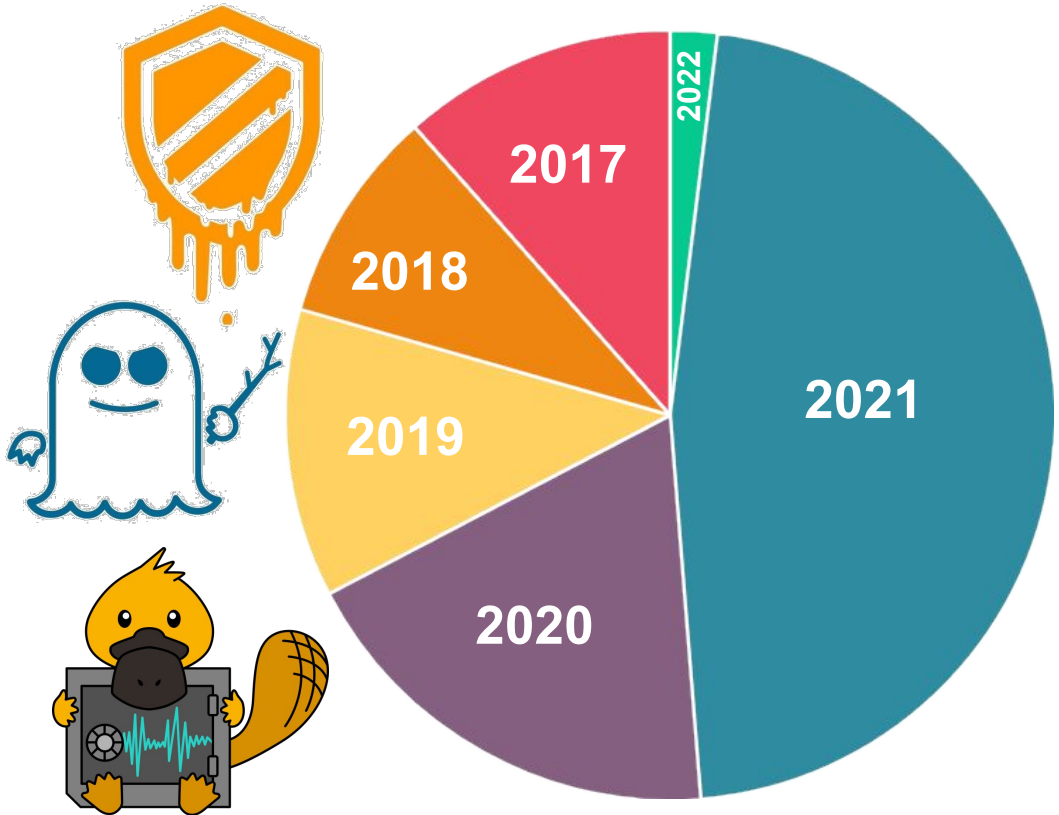
github.com/binarly-io/FwHunt

The ecosystem was broken! How are we fixing it?

Vendors upload firmware to the LVFS for Linux users

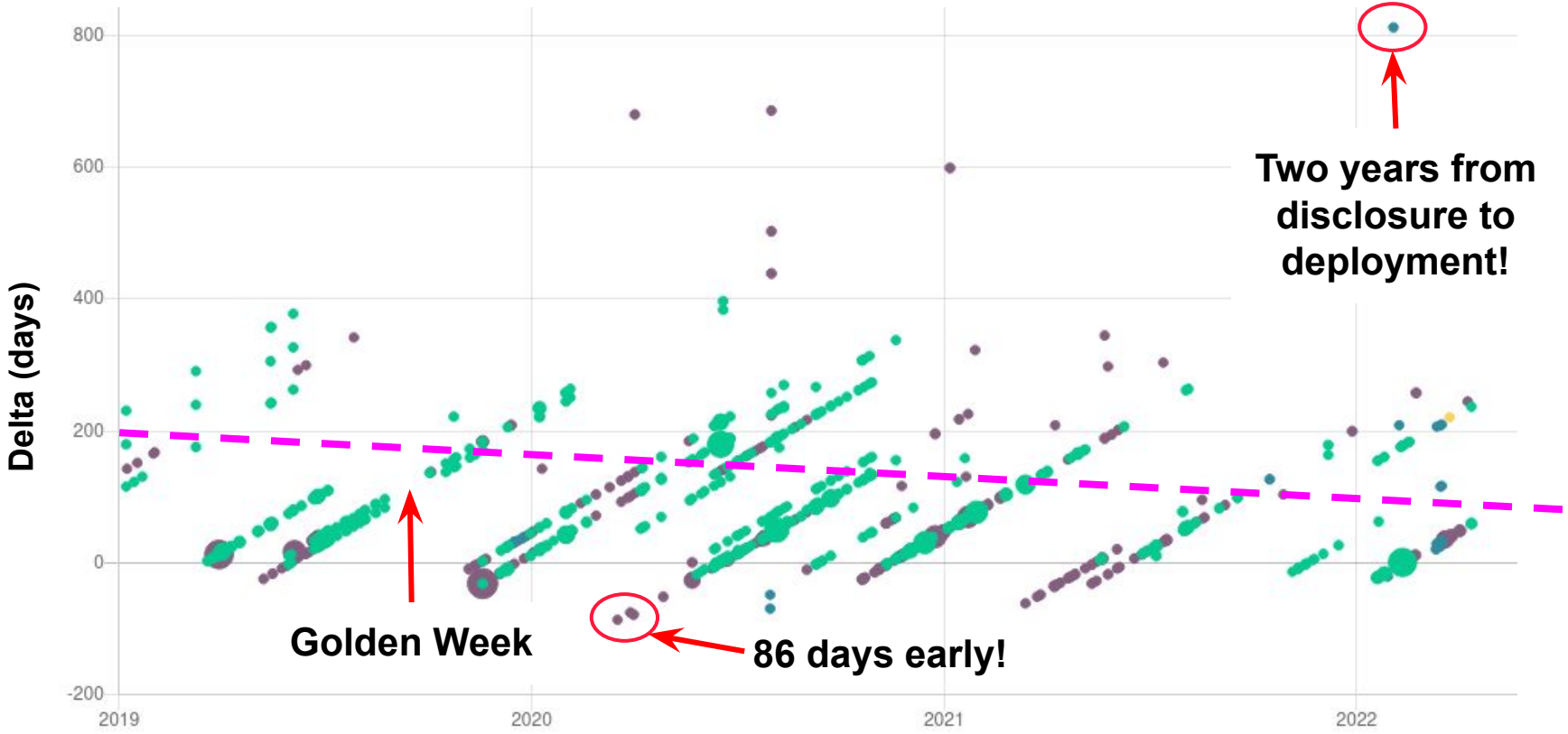


The **newest** versions of Intel Microcode for ~800 devices



CVE-2022-21151
Processor Speculative Cross Store Bypass Advisory

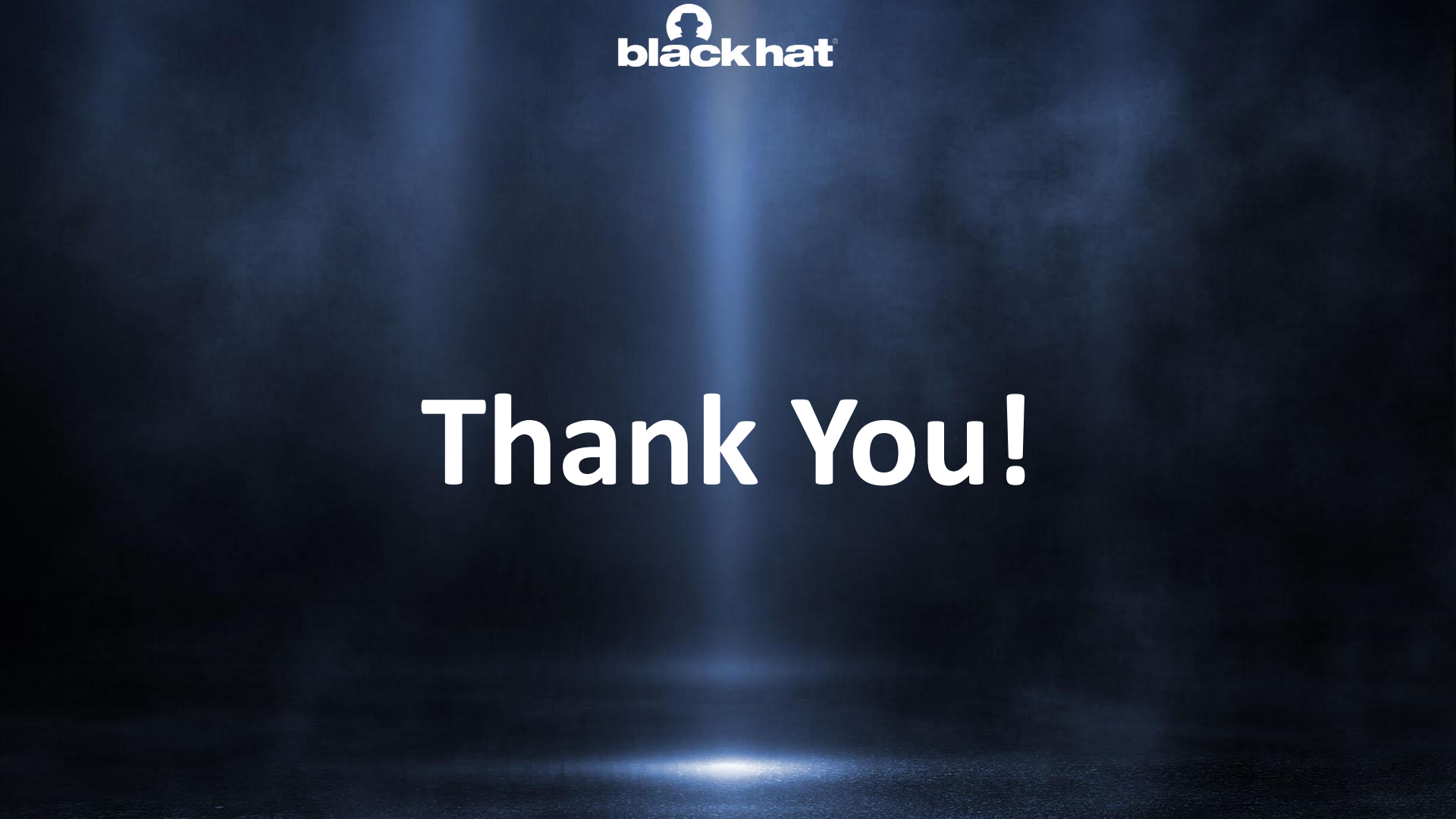
Vendors take a long time to roll out security fixes



Using FwHunt we *remind* vendors about the embargo

```
hex_strings:
  - 56e8........593c01....80be....000000
    # 56                                    push    esi
    # E8 .. .. .. ..                       call    x_BiosSsaEnabled
    # 59                                    pop     ecx
    # 3C 01                                 cmp     al, 1
    # .. ..                                 jnz     short loc_FFDE86FD
    # 80 BE .. .. 00 00 00                  cmp     byte ptr [esi+81h], 0
    # .. ..                                 jz      short loc_FFDE86FD
  - 6a006a0268be00000056e8
    # 6A 00                                 push    0
    # 6A 02                                 push    2
    # 68 BE 00 00 00                        push    0BEh
    # 56                                    push    esi
    # E8 .. .. .. ..                       call    SsaApi
```

Conclusion


black hat
Thank You!