Tackling Security Through the Supply Chain

UEFI 2022 Virtual Summit
May 17, 2022
Presented by Alex Matrosov, CEO, Binarly
and Tim Lewis, CTO, Insyde Software
Meet the Presenters

Tim Lewis
CTO, Insyde Software

Alex Matrosov
CEO, Binarly
Agenda

• Firmware Supply Chain and Firmware Security
• Case Studies In Supply Chain Failures
• Bridging The Gaps In The Supply Chain
• Next Steps
“Firmware presents a large and ever-expanding attack surface, as the population of electronic devices grows. Securing the firmware layer is often overlooked, but it is a single point of failure in devices and is one of the stealthiest methods in which an attacker can compromise devices at scale. Over the past few years, hackers have increasingly targeted firmware to launch devastating attacks.”

– U.S. Dept. of Commerce and Homeland Security, 24 Feb 2022
Firmware Security Supply Chain
How Does Security Get To The End-User Today?

• How does each stage know that what they have received has not been tampered with?

• How does each stage know if a security disclosure applies to what they have received?
Firmware Security Supply Chain
The Critical Role of ODMs and OEMs

• ODMs/OEMs create production firmware binaries and distribute them.

• Last stage to see firmware ingredients.
Firmware Security Supply Chain
How Does Security Get To The End-User Today?

• Production firmware consists of many ingredient providers through several stages.

• Firmware ingredients are received from the previous stage, possibly modified and combined, and then passed to the next stage.
Firmware Security Supply Chain
The Critical Role of IT and End Users

• IT/End User responsible for updating firmware.

• End users are the least likely to read published reports and have the least visibility into firmware ingredients.
Supply Chain Complexity is Growing

1) 0-day Vulnerability
Component Supplier

Subsystem Supplier

Component Supplier (Hardware Security Module)

Component Supplier

Subsystem Supplier

2) 1-day Vulnerability

OEM

Value-added Reseller

Customer

3) n-day Vulnerability

Supply Chain Risk

www.uefi.org
## Binarly Vulnerability Disclosures Statistics

<table>
<thead>
<tr>
<th>Vulnerability Category</th>
<th>Count</th>
<th>Average Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMM Privilege Escalation</td>
<td>15</td>
<td>CVSS: 8.2</td>
</tr>
<tr>
<td>SMM Memory Corruption</td>
<td>22</td>
<td>CVSS: 8.2</td>
</tr>
<tr>
<td>DXE Memory Corruption</td>
<td>5</td>
<td>CVSS: 7.7</td>
</tr>
</tbody>
</table>

* Based on Binarly disclosures: [https://www.binarly.io/advisories](https://www.binarly.io/advisories)
<table>
<thead>
<tr>
<th>LEADER BOARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND ONE</td>
</tr>
<tr>
<td><strong>Join us in congratulating...</strong></td>
</tr>
<tr>
<td>First to Submit:</td>
</tr>
<tr>
<td>Highest Single Payout:</td>
</tr>
<tr>
<td>Most Eligible Reports: Mickey (@HackingThings)</td>
</tr>
<tr>
<td>Widest Impact:</td>
</tr>
<tr>
<td>Helping Others:</td>
</tr>
<tr>
<td>Just in Time:</td>
</tr>
</tbody>
</table>

* Intel Project Circuit Breaker: [https://www.projectcircuitbreaker.com/](https://www.projectcircuitbreaker.com/)
Where Firmware Supply Chain Has Failed

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel BSSA DFT</td>
<td>Silicon Vendor Reference Code</td>
</tr>
<tr>
<td>AMD CVE-2021-39298</td>
<td>Silicon Vendor Reference Code</td>
</tr>
<tr>
<td>Insyde IdeBusDxe</td>
<td>IBV Reference Code</td>
</tr>
<tr>
<td>AMI UsbRt</td>
<td>IBV Reference Code</td>
</tr>
<tr>
<td>HP CVE-2022-23932</td>
<td>ODM/OEM Firmware Code</td>
</tr>
<tr>
<td>Dell CVE-2022-24419</td>
<td>ODM/OEM Firmware Code</td>
</tr>
<tr>
<td>Lenovo CVE-2021-3971</td>
<td>ODM/OEM Firmware Code</td>
</tr>
</tbody>
</table>

* Based on Binarly disclosures: [https://www.binarly.io/advisories](https://www.binarly.io/advisories) www.uefi.org
```c
char isPhysicalPresenceEstablished()
{
    return 1;
}
```

```c
(*PeiServices)->LocatePpi(PeiServices, &gReadOnlyVariable2Guid, 0, 0, &ReadOnlyPpi);
ZeroMem(syscg_stack, 2048);
ReadOnlyPpi->GetVariable(ReadOnlyPpi, L"syscg", &gSsaBiosVariablesGuid, 0, &DataSize, syscg_stack);
sycg = AllocatePool(DataSize);
memcpy_0(syscg, syscg_stack, DataSize);
TotalConfigs = *(syscg + 0x10);
EvLoadTool(host, syscg, &ConfigIndex, &ImageBase);
```

https://www.binarly.io/posts/Attacking_(pre)EFI_Ecosystem
https://www.binarly.io/posts/Detecting_Firmware_vulnerabilities_at_scale_Intel_BSSA_DFT_case_study
IdeBusDxe (VU#796611)

Based on Binarly disclosures: [https://www.binarly.io/advisories](https://www.binarly.io/advisories)
UsbRt - Intel & Binarly Disclosures

* Based on Binarly disclosures: [https://www.binarly.io/advisories](https://www.binarly.io/advisories)
```c
int64 __fastcall SwSmiHandler(__int64 a1)
{
    __int64 result; // rax
    UINT32 v3; // [rsp+30h] [rbp-48h] BYREF
    UINTN v4; // [rsp+38h] [rbp-40h] BYREF
    EFI_SMM_SW_DISPATCH2_PROTOCOL *EfiSmmSwDispatch2Protocol; // [rsp+40h] [rbp-38h] BYREF
    __int64 v6[6]; // [rsp+48h] [rbp-30h] BYREF

    v3 = 6;
    v4 = 32164;

    if ((gRT->GetVariable(L"AmdMemContextData", &VendorGuid, &v3, &v4, v6) & 0x8000000000000000u64) == 0164) //
        // overwrite GetVariable service address in the EFI_RUNTIME_SERVICES table with the shellcode address

    sub_15FC(v6);
    result = gSmst->SmmLocateProtocol(&EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID, 0164, &EfiSmmSwDispatch2Protocol);
    if ( result != 0 )
        return EfiSmmSwDispatch2Protocol->UnRegister(EfiSmmSwDispatch2Protocol, a1);
    return result;
}
```

https://www.binarly.io/advisories/BRLY-2021-004
Static Code Analysis Tools Limitations

// 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)(0x230) == 1 )
                HandlerUnregister();
        }
    }
    ...
}

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

```c
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)
```
Compilers-Generated Artifacts

SmmIsBufferOutsideSmmValid() - SMM input pointer validation routine
1 - normal version
2 - compiler-optimized version (hardcoded size)

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

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

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

    if (ptr >= *(QWORD *)(v9 + 8) && ptr + size <= *(QWORD *)(v9 + 8) + (*(QWORD *)(v9 + 0x18) << 12 ))
    if (ptr >= *(QWORD *)(v8 + 8) && ptr + 0x20 <= *(QWORD *)(v8 + 8) + (*(QWORD *)(v8 + 0x18) << 12 ))
    ...
```
Firmware Security Supply Chain
Bridging Supply Chain Gaps Securely

• What firmware ingredients does each product have?

• Which firmware ingredients have a known vulnerability?

• Have firmware ingredients been tampered with?

• Do users know to update their product?
# What Firmware Security Tools Do We Have?

<table>
<thead>
<tr>
<th>Delivery</th>
<th>Supply Chain</th>
<th>Verify pedigree and tamper status of source code/pre-built binaries</th>
<th>SPDX, CycloneDX, SWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Time</td>
<td>Static Analysis</td>
<td>Analyze the source code and binaries for common security mistakes</td>
<td>Klocwork, Coverity, Visual Studio code analysis, FwHunt, binary analysis</td>
</tr>
<tr>
<td>Code Review</td>
<td></td>
<td>Review the source code based on threat model in high-risk technology areas</td>
<td>EDK2 security review guidelines, MISRA C</td>
</tr>
<tr>
<td>Testing</td>
<td>Security Testing</td>
<td>Use traditional testing and fuzzing for data that crosses trust boundaries</td>
<td>Defensics, CHIPSEC, FWTS</td>
</tr>
<tr>
<td>Runtime</td>
<td>Tamper Protection</td>
<td>Measure firmware code/data, compare measurements, log measurements</td>
<td>Intel BootGuard, TCG event log, golden measurements</td>
</tr>
<tr>
<td></td>
<td>Compiler Runtime Protections</td>
<td>Use the compiler to inject checking code for common security failures</td>
<td>Integer overflow, Uninitialized variables, Local stack corruption</td>
</tr>
<tr>
<td></td>
<td>Access Control</td>
<td>Protect against module accessing resources that are not permitted</td>
<td>Intel System Resource Defense (ISRD), Heap Guard, Stack Guard, NULL Pointer, PE/COFF</td>
</tr>
<tr>
<td></td>
<td>Kernel Protections</td>
<td>Detect unsafe usage patterns and data corruption in BIOS kernel</td>
<td>Heap and pool corruption, TPL inversion, critical data structure checks, ASLR, panic</td>
</tr>
<tr>
<td></td>
<td>IT Intervention</td>
<td>Alert users/IT, provide response options (lock, shutdown, reflash, etc.)</td>
<td>HIRS ACA, BMC, firmware security monitoring</td>
</tr>
</tbody>
</table>
Firmware Security Tools We Have

SPDX SBOMs

- SPDX SBOMs focus on transmission of source ingredients before creating the production firmware binaries.

- SPDX SBOMs help OEMs/ODMs to:
  - Track the origin and licenses of ingredient source code and binaries.
  - Identify whether ingredients have been modified from stage to stage.
  - Know if products contain ingredients with reported known vulnerabilities.

- Support:
  - Tianocore tags all files with “SPDX-License-Identifier” to help automation.
  - Tools readily available: https://github.com/spdx/tools-python
Firmware Security Tools We Have

SWID SBOMs

- SWID SBOMs focus on identity of production firmware binaries.

- SWID SBOMs can help IT/End-Users to
  - Inventory firmware on the platform (executables and blobs).
  - Check for security disclosures reported to affect that firmware.

- UEFI firmware can:
  - Retrieve attached firmware information using DMTF’s SPDM.
  - Validate firmware measurements against golden values.
  - Record measurements and identifiers in the TCG event log.

- See Traceable Firmware Bill of Materials Overview – UEFI 2021 Virtual Plugfest, python-uswid and LVFS.
Firmware Security Tools We Have

• Intel’s CHIPSEC (github.com/chipsec)
  – Checks running configuration for vulnerable settings.

• Binarly’s efiXplorer (github.com/binarly-io/efiXplorer)
  – Help investigate vulnerabilities in BIOS binaries.

• Binarly’s FwHunt (github.com/binarly-io/FwHunt)
  – Checks BIOS binaries for known-bad code patterns (code semantic-based approach).
<table>
<thead>
<tr>
<th>Address</th>
<th>Variable name</th>
<th>Variable GUID</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000F9A38</td>
<td>db</td>
<td>D719B2CB-3D3A-4596-A3BC-DA0D0E67656F</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000F9AAB</td>
<td>PK</td>
<td>8BE4DF4D-9B3A-11D2-A00D-00E09803288C</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000F9B9C</td>
<td>SecureBootEnable</td>
<td>F0A38BC7-4F08-4556-99C4-001009C93A44</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000A15D99</td>
<td>TCG2_CONFIGURATION</td>
<td>6339D4B7-26B8-42B8-9A5D-687E25D74B8C</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000A71482</td>
<td>LenovoTpmFwUpdate</td>
<td>38243F72-E87F-468F-B19C-4785EF8AC63F</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000A715CB</td>
<td>LenovoSecurityConfig</td>
<td>A2C1008F-0D4F-4CC9-A619-D1641D39D49</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000A7168F</td>
<td>LenovoTpmFwUpdate</td>
<td>38243F72-E87F-468F-B19C-4785EF8AC63F</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000A72070</td>
<td>TCG2_CONFIGURATION</td>
<td>6339D4B7-26B8-42B8-9A5D-687E25D74B8C</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC5351</td>
<td>ESRTPLATFORMENTRY</td>
<td>6706A37A-64B4-C6F7-B421-6FF116DE0BE</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC5374</td>
<td>ESRTPLATFORMENTRY</td>
<td>D1C3FF88-B539-7DCC-A04A-C2466A3217AF</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC648D</td>
<td>db</td>
<td>C076EC0C-7A28-4A99-A072-71EE5C44889F</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC6585</td>
<td>db</td>
<td>D719B2CB-3D3A-4596-A3BC-DA0D0E67656F</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC656A</td>
<td>db</td>
<td>D719B2CB-3D3A-4596-A3BC-DA0D0E67656F</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC65CC</td>
<td>KEK</td>
<td>3D80DD74-0001-0000-A072-500160600000</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC656D</td>
<td>PK</td>
<td>3D80DD74-0001-0000-A072-500160600000</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC656C</td>
<td>SecureBootEnable</td>
<td>F0A38BC7-4F08-4556-99C4-001009C93A44</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC746D</td>
<td>LenovoSecurityConfig</td>
<td>A2C1008F-0D4F-4CC9-A619-D1641D39D49</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AC754A</td>
<td>TCG2_CONFIGURATION</td>
<td>6339D4B7-26B8-42B8-9A5D-687E25D74B8C</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AB43E</td>
<td>LenovoScratchData</td>
<td>6C73286E-4EFC-49F6-9729-0760B84109A7</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AB536</td>
<td>CpuSetup</td>
<td>D80BF7FF-EE68-4193-A997-5E9E900ADB32</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AB55A</td>
<td>EPCSW</td>
<td>D6A279B5-5B58-45D1-A148-711B9BB4B521</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AFF352</td>
<td>OemVariable</td>
<td>F0393D2C-784A-4BB9-AF08-2932CA00C11E</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000AFF410</td>
<td>OemVariable</td>
<td>F0393D2C-784A-4BB9-AF08-2932CA00C11E</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000B013A0</td>
<td>RstOptaneConfig</td>
<td>4DA4F952-2516-4D06-8975-65036403ABC7</td>
<td>SetVariable</td>
</tr>
<tr>
<td>0000000000B01430</td>
<td>RstOptaneConfig</td>
<td>4DA4F952-2516-4D06-8975-65036403ABC7</td>
<td>SetVariable</td>
</tr>
</tbody>
</table>
BRLY-2022-004:

meta:
  author: Binarly (https://github.com/binarly-io/FwHunt)
  license: CC0-1.0
  name: BRLY-2022-004
  namespace: vulnerabilities
  CVE number: CVE-2022-24419
  description: SMM arbitrary code execution in USBRT SMM driver on Dell devices
  volume guids:
    - 04EAAA1-29A1-11D7-8838-00500473D4EB

code:
  and:
    - pattern: 488b05........488b88....00004885c9....483a0....000000...0fb704250e04000c1e00405040100008b084885c9
    place: sw_smi_handlers
SecureBackDoor-CVE-2021-3971:

meta:
  author: Binarly
  name: SecureBackDoor-CVE-2021-3971
  namespace: vulnerabilities
  CVE number: CVE-2021-3971
  vendor id: LEN-73440
  advisory: https://github.com/eset/vulnerability-disclosures/blob/master/CVE-2021-3971/CVE-2021-3971.md
  description: Disabled SPI flash firmware storage protections
  volume guids:
    - 16F41157-1DE8-484E-B316-DDB77CB4080C

wide_strings:
  and:
    - utf16le: cE!

hex_strings:
  and:
    - 5de6cc6a35da394bb64b5ed927a7dc7e
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 guides:
    - 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

fwhunt@binarly.io
github.com/binarily-io/FwHunt

demo$ ./target/release/fwhunt --data data/ --rules /tmp/fwhunt-rules/ -g tests/image-bios.bin

fwhunt@binarily.io
Next Steps

• For OEMs/ODMs, find the ingredients that make up your firmware.

• For IT/end-users, find out how to be notified of vulnerabilities in your firmware and its ingredients.

• At any stage of the supply chain, use tools to simplify vulnerability tracking on your platforms.
Questions?
Thanks for attending the UEFI 2022 Virtual Summit

For more information on UEFI Forum and UEFI Specifications, visit https://www.uefi.org
Reference – SBOMs, Tools, Gov’t

• Standards
  – SPDX
  – SWID

• Tools
  – CHIPSEC
  – FwHunt
  – efiXplorer

• Government
  – ASSESSMENT OF THE CRITICAL SUPPLY CHAINS SUPPORTING THE U.S. INFORMATION AND COMMUNICATIONS TECHNOLOGY INDUSTRY
Reference – Binary SBOMs

• Traceable Firmware Bill of Materials Overview – UEFI 2021 Virtual Plugfest
• General Supply Chain Guidelines
  – ISO/IEC 28000:2007 - Specification for security management systems for the supply chain
  – UK NCSC – Supply Chain Security Guideline
• Standards / Guidelines
  – NIST SP800-155 (draft) – BIOS Integrity Measurement Guideline
  – TCG PC Client Platform Firmware Profile (PFP)
  – TCG PC Client Firmware Integrity Measurement (FIM)
  – TCG PC Client Reference Integrity Manifest (RIM)
  – TCG Platform Certificate Profile
  – TCG DICE Attestation Architecture
  – TCG DICE Layering Architecture
  – TCG DICE Certificate Profile
  – IETF RATS Remote Attestation Architecture
  – IETF SACM Concise SWID
  – IETF RATS Concise RIM
  – DMTF Secure Protocol and Data Model

www.uefi.org