UEFI Ecosystem Investments and Open-Source Contributions

UEFI Fall 2023 Developers Conference & Plugfest
October 9-12, 2023
Presented by: Michael Kubacki (Microsoft)
Agenda

• Introduction
• Security
• Features
• Tests & Tools
• Future Investments
• Questions
Introduction

“Core UEFI”

Microsoft Azure
Microsoft Security

Windows
Microsoft Surface

www.uefi.org
Open-Source Firmware Repos

📖 Documentation
- mu
- mu_devops
- mu_crypto_release

🛠 Build & Release
- mu_basecore
- mu_tiano_plus
- mu_plus

🏛 Core
- mu硅icon_arm_tiano
- mu硅icon_intel_tiano
- mu_common_min_platform

🚀 Features
- mu_feature_config
- mu_feature_dfci
- mu_feature_ipmi
- mu_feature_mm_supv

🖥 Platform
- mu_tiano_platforms
- mu_oem_sample

Project Mu (microsoft.github.io)

www.uefi.org
Firmware Security

WE’VE INSTALLED A TWO-KEY SYSTEM TO PREVENT ACCIDENTAL MISSILE LAUNCHES.

WE’VE DEVELOPED A DUAL-TURNER DEVICE TO ALLOW A USER TO EFFICIENTLY TURN MULTIPLE KEYS.

WE’VE INSTALLED A TWO-KEY LOCK ON THE DUAL TURNER DEVICE TO PREVENT ACCIDENTAL USE.

xkcd: Two Key System
Overview

"Firmware Vulnerabilities as a Percentage of New Vulnerabilities Added to the NVD"

Takeaway: This is getting worse

Source: DHS CISA Strategy to Fix Vulnerabilities Below the OS Among Worst Offenders

www.uefi.org
Recent Security Focus Areas

Attackers are increasingly focusing on firmware

- **Memory Protections**
  - Reduce UEFI gap to other system software
  - Address UEFI ecosystem compatibility challenges

- **Code Correctness**
  - Leverage CodeQL to identify implementation bugs

- **OS Runtime**
  - Expand usage of policy for UEFI variables
  - Explore SMM alternatives

- **Secure Boot**
  - Address limited revocation space
  - Coordinate certificates nearing expiration
  - Evolve 3rd Party UEFI CA signing requirements
  - Plan for the future

- **Reduce MM Attack Surface**
  - Reduce MM memory access
  - Eliminate excessive DXE coupling
  - Launch MM earlier in boot
  - Build an open-source MM supervisor

www.uefi.org
System Management Mode (SMM)

SMM - A special-purpose operating mode in x86 architecture used to monitor and manage various system resources and perform manufacturing tasks.

- Main benefits:
  - Provides a distinct & easily isolated processor environment
  - Operates transparently to the OS & software applications

- SMM is entered via a System Management Interrupt (SMI)
1. Enable memory protections
   – Protect against buffer overflow, stack overflow, overwriting code sections, etc.
   – See “UEFI Memory Protections” presentation.
   – Effort to enable: Medium

2. SMM drivers: Follow best practices
   – Use communicate buffers, validate input, reduce memory map exposure to SMM, etc.
   – Effort to enable: Low

3. SMM core: Use Standalone MM
   – Prevent “SMM callouts”.
   – Load MM core earlier.
   – Effort to enable: Medium

4. Enable an MM Supervisor
   – Reduce capabilities of MM code to what is required.
   – Effort to enable: Medium - High
Traditional MM

System Boot

- DxeCore loads PiSmmIpl

PiSmmIpl Initialize

- PiSmmCore runs inside SM RAM

PismmCore loads other SMM drivers

- Notify

DxeCore loads other DXE drivers

- Notify

Ready to lock event

- Notify

Hand off to OS

- More SMI events during runtime

Standalone MM

System Boot

- PiMmpiIpl Initialize

PiMmpiIpl Initialize MM/RAM

PeiCore loads other PEI drivers

- Standalone MM Core runs inside MM/RAM

DxeCore loads other DXE drivers

- DXE phase drivers

Ready to lock event

- Ready to lock event

Hand off to OS

- More MMI events during runtime

Early Boot Phase

Late Boot Phase

MMI Environment

www.uefi.org
# Standalone MM – Protocol Access

<table>
<thead>
<tr>
<th>Traditional MM Initialization</th>
<th>Standalone MM Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UEFI</strong></td>
<td><strong>MM</strong></td>
</tr>
<tr>
<td>- Boot Services</td>
<td>- MM Services</td>
</tr>
<tr>
<td>- DXE Services</td>
<td>- MM protocols</td>
</tr>
<tr>
<td>- Runtime Services</td>
<td>Note: Drivers do not have access to interfaces outside the MM environment.</td>
</tr>
<tr>
<td>- DXE protocols</td>
<td></td>
</tr>
<tr>
<td><strong>MM</strong></td>
<td></td>
</tr>
<tr>
<td>- MM Services</td>
<td></td>
</tr>
<tr>
<td>- MM protocols</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traditional MM Runtime</th>
<th>Standalone MM Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MM</strong></td>
<td><strong>MM</strong></td>
</tr>
<tr>
<td>- MM Services</td>
<td>- MM Services</td>
</tr>
<tr>
<td>- MM protocols</td>
<td>- MM protocols</td>
</tr>
<tr>
<td>Note: Drivers do not have access to interfaces outside the MM environment.</td>
<td></td>
</tr>
</tbody>
</table>
MM Supervisor

MM Supervisor - A kernel like module that operates in Standalone MM mode with policy-based resource access restrictions.

Performs the responsibilities of PiSmmCore and PiSmmCpuDxeSmm:
- Initial MM environment setup
- Memory management
- Standalone MM driver dispatching
- MMI handlers dispatching
- ...

Provides the following unique features:
- Customizable Security Policy (Open-Source Python tools)
- PEI Launch Capability
- Privilege Separation for Resource Access Operations
- Memory Isolation

microsoft/mu_feature_mm_supv - MM Supervisor
UEFI Variable Policy

- Code should implement strict UEFI variable protections using variable policy.
- All UEFI variables that are no longer consumed should be locked as soon as possible during boot.
- UEFI variables should enforce that variable attributes are set to expected values.

More Information: UEFI Variable Policy Whitepaper - Project Mu (microsoft.github.io)

Tianocore Variable Lock to Variable Policy Transition Documentation:
VariablePolicy Protocol Enhanced Method for Managing Variables · tianocore/tianocore.github.io Wiki

Source: Data-Only Attacks Against UEFI BIOS
Firmware Features

IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPU'S, UP FROM THE OLD LIMIT OF 1,024.

DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?

NO, BUT WHO USES THAT?

xkcd: Supported Features
Due to the diverse application of SMM, an incremental approach is taken to reduce its usage.

Begin by classifying SMI handlers:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software SMI handlers that do not require SMM privileges</td>
<td>Address translation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>From System Physical Address (SPA) to DIMM Address (DA)</td>
</tr>
<tr>
<td>2</td>
<td>Software SMI handlers that require SMM privileges</td>
<td>UEFI Authenticated Variables, UEFI Capsule Update</td>
</tr>
<tr>
<td>3</td>
<td>Hardware SMI handlers that do not require SMM privileges</td>
<td>Memory error correction handling</td>
</tr>
<tr>
<td>4</td>
<td>Hardware SMI handlers that do require SMM privileges</td>
<td>CPU hot add and remove</td>
</tr>
</tbody>
</table>

*Note: SMM privileges means certain hardware resources such as registers can only be accessed from SMM execution context.*

We can most easily eliminate SMI handlers that do not depend upon SMM privileges.
Platform Runtime Mechanism (PRM)

Introduces the capability to move SW SMI handlers (Category 1) and a sub-set of HW SMI handlers (Category 3) that do not require SMM privileges out of SMM and into OS/VMM execution context.

It is common for Category 1 SW SMI to occur from a _DSM in ACPI.

This means:
1. PRM is backward compatible with this interface
2. PRM can leverage this abstraction to substitute _DSM -> SMI with _DSM -> PRM
3. At a high-level, an SMI handler becomes a PRM handler
PRM Invocation

Two types of invocation:

1. **Direct** – A PRM aware OS driver calls into the ACPI Bridge driver directly to invoke a PRM handler.

2. **ACPI** – An OS driver continues to call a `_DSM` which is implemented to invoke a PRM handler by writing to a PRM OpRegion instead of triggering a software SMI.

- PRM modules are PE/COFF binaries that list their PRM handlers in an export table.
- PRM handlers are discoverable within a module by both firmware and operating systems.
- PRM modules can be updated at OS runtime (independent of UEFI firmware).

PRM Specification: Platform Runtime Mechanism (uefi.org)
PRM Firmware Code: edk2/PrmPkg · tianocore/edk2 (github.com)
Firmware Policy

A flexible way for platforms to define and manage configuration data.

- **Policy** – An opaque data block.
  - Identified by **GUID**
  - Controlled by **attributes**
  - Useful for data passing across ownership boundaries (e.g. platform to silicon).
  - Can control dispatch order on policy presence.

- **Policy Service** – PI-phase independent interfaces to:
  - Get/Set/Remove policies
  - Register & Unregister for policy events
    - E.g. addition, modification, or removal
  - Work with “verified policies”
    - Data access via generated accessor functions
    - Checks data access details such as size and version between consumers and producers.
Test & Debug Tools

I was trying to figure out why my browser was acting weird.

 Turns out it wasn't the browser—the issue was with my keyboard driver.

Debugging that led me to a mysterious error message from a system utility...

Anyway, long story short, I found the sword of Martin the Warrior.

I think at some point there you switched puzzles.

xkcd: Debugging
UEFI 3rd Party CA PE/COFF Tests

Microsoft 3rd Party UEFI CA memory mitigation requirements added November 30th, 2022:

1. Page aligned sections. For example, 4KB or a larger power of 2 (64KB).
2. Section flags must not combine IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE for any given section.
3. If targeting NX compatible firmware, DLL Characteristics must include IMAGE_DLLCHARACTERISTICS_NX_COMPAT

---

PE/COFF Image Validation Tool

The PE/COFF image validation tool is a command line tool used to verify that memory protection requirements such as section alignment and write / execute settings are applied correctly. This tool also provides the ability to check, set, and clear the NX_COMPAT flag found in OPTIONAL_HEADER.DllCharacteristics.

Synopsis


image_validation · tianocore/edk2-pytool-extensions (github.com)
TPM Replay

Replays a custom event log for testing operating system features dependent on measurements.

- Exclusive control over PCR digests
- Easy conversion between JSON/YAML and binary event logs
- JSON schema that makes it easy to understand input format and validate logs
- Logs can be passed through FFS file, UEFI variable, or QEMU FW CFG

[TpmTestingPkg/TpmReplay · microsoft/mu_plus (github.com)]
CodeQL is open-source and free for open-source projects.

- Uses a semantic code analysis engine to discover vulnerabilities.
- CodeQL queries are open-source and easy to write.
- Can be run in CI and locally with the CodeQL CLI.
- Integrated with GitHub Code Scanning and IDEs like VS Code.

CodeQL (github.com)
Future Investments
We see value in Rust as a viable alternative to C for UEFI firmware.

**Brings:**
- Memory safety with no garbage collector
- Productivity improvements
  - High-level multi-paradigm concepts such as generics and traits
  - An official package management system with first-class support for formatters and linters
  - Ability to produce and consume crates with a broader community of developers increasing overall development velocity

**Status:**
- Integrated Rust build support in the edk2 build environment
  - Including containers that are regularly tested in CI
  - Including Rust unit testing and code coverage support
- Started publishing crates in several Project Mu repos
- Demonstrated UEFI Rust (DXE) driver execution in QEMU

**Currently:** Porting more code to Rust
Firmware Database

**Edk2DB** – A tool that builds a sqlite3 database from an edk2 workspace.

- Generates databases that can be shared.
- “Parsers” are registered into a generic framework to build the database.

- Includes built-in parsers for:
  - Workspace Environment – Env vars, Git repo context, etc.
  - Source code – source code files (C, asm, etc.)
  - Modules – Metadata from INF files
  - Instance info – Active library and component instances (DSC)
  - FV info – Component and size info for a given (FDF)
Firmware Database Tooling

Off-the-shelf and custom tools help provide data insight.

Generic Sqlite3 Tooling
- VSCode extensions (SQLite, SQLite Viewer)
- Command line (SQLite CLI, sqlite-analyzer)
- Applications (SQLiteStudio, SQLite Expert)

Edk2DB Tooling (Reports and Queries)
- By-INF code coverage
- Recursive INF dependency graph
- Platform usage report
- Others
## Code Coverage

<table>
<thead>
<tr>
<th>INF File</th>
<th>Lines</th>
<th>Branches</th>
<th>Functions</th>
<th>Lines Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseMemoryLib.inf</td>
<td>151</td>
<td>113</td>
<td>264</td>
<td>1131 57.1%</td>
</tr>
<tr>
<td>CompareMemWrapper.c</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>61  88.8%</td>
</tr>
<tr>
<td>CopyMem.c</td>
<td>77</td>
<td>0</td>
<td>77</td>
<td>148 100%</td>
</tr>
<tr>
<td>CopyMemWrapper.c</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>59  100%</td>
</tr>
<tr>
<td>MemLibGeneric.c</td>
<td>10</td>
<td>58</td>
<td>68</td>
<td>307 14.7%</td>
</tr>
<tr>
<td>MemLibGuid.c</td>
<td>12</td>
<td>18</td>
<td>30</td>
<td>167 40%</td>
</tr>
<tr>
<td>SetMem.c</td>
<td>24</td>
<td>1</td>
<td>25</td>
<td>80  96%</td>
</tr>
<tr>
<td>SetMem16Wrapper.c</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>58  0%</td>
</tr>
<tr>
<td>SetMem32Wrapper.c</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>58  0%</td>
</tr>
<tr>
<td>SetMem64Wrapper.c</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>58  0%</td>
</tr>
<tr>
<td>SetMemWrapper.c</td>
<td>5</td>
<td>7</td>
<td>12</td>
<td>85  41.6%</td>
</tr>
<tr>
<td>ZeroMemWrapper.c</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>50  85.7%</td>
</tr>
</tbody>
</table>

**Code Coverage Per INF**
Platform Usage Report

Build Information
- Platform: OvmfPkg/OvmfPkgia32X64.dsc
- Target: DEBUG
- Architectures: IA32 X64
- Toolchain: GCC5
- Commit Sha: f36e1ec1f0a5fd3be84913e09181d7813444b5620

Components and Libraries From Other Packages
Virtual Platforms

All the features covered are integrated in these easy-to-use virtual platforms (derived from OvmfPkg):

- **QemuQ35Pkg** – IA32/X64 development
- **QemuSbsaPkg** – AARCH64 development

Includes support for:
- Advanced debug logging capabilities
- CodeQL
- Device Firmware Configuration Interface (DFCI)
- Firmware Config & Policy
- Graphical front-page w/ on-screen keyboard
- MM Supervisor (with PEI launch)
- Platform Runtime Mechanism (PRM)
- Rust integration
- Telemetry / WHEA reporting
- TPM Replay
- Utility apps – Paging audits, UEFI variable policy info, …
- UEFI debugger extension (WinDbg extension)
- UEFI memory protections
- UEFI variable policy
Thanks for attending the UEFI Fall 2023 Developers Conference & Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

Microsoft