presented by

Microsoft

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

edk2
edk2-platforms
edk2-pytool-extensions
edk2-pytool-library

Open-Source

Project Mu

mu_basecore
mu_tiano_plus
mu_plus
mu_tiano_platforms
. . .

"Core UEFI"

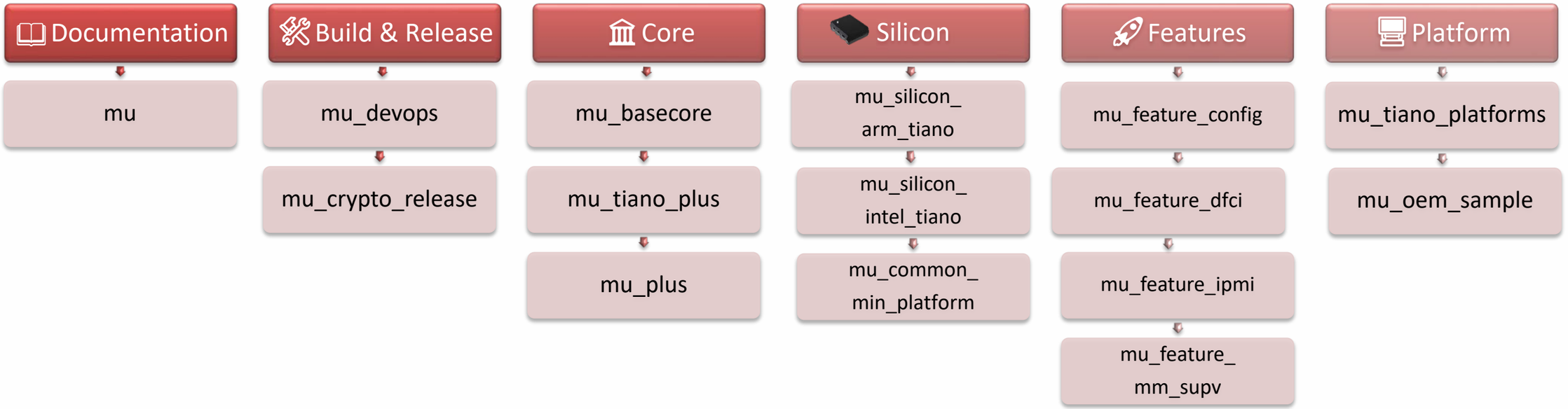
Microsoft Azure

Windows

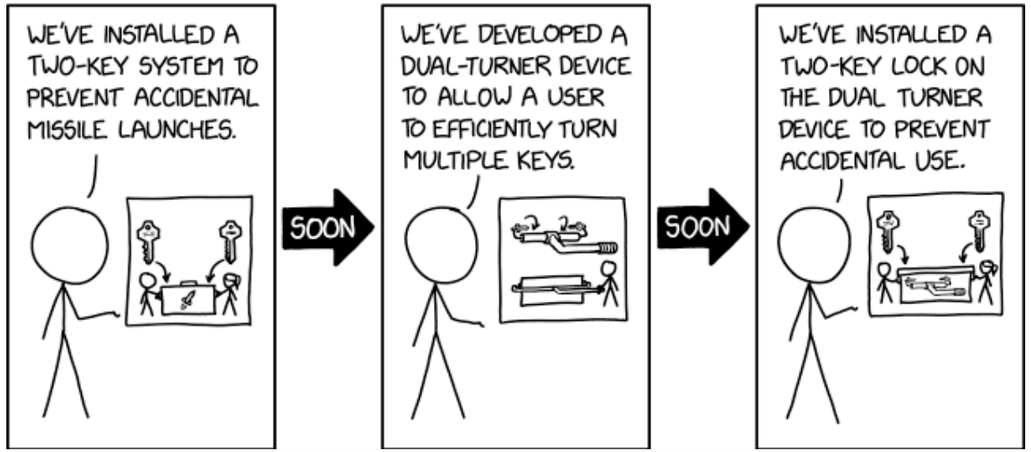
Microsoft Security

Microsoft Surface

Open-Source Firmware Repos



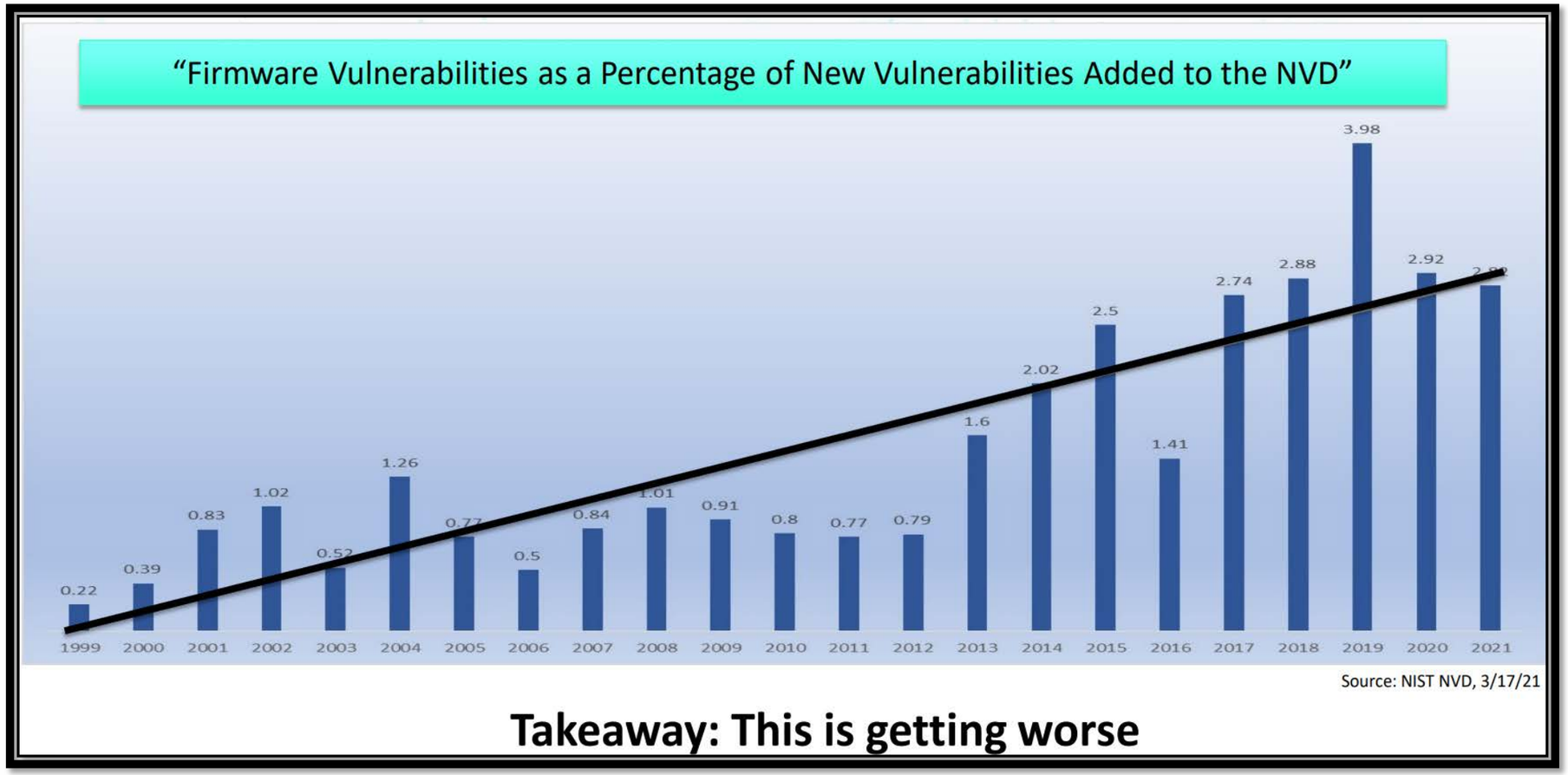
Project Mu (microsoft.github.io)



xkcd: Two Key System

Firmware Security

Overview



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

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

Reduce MM Attack Surface

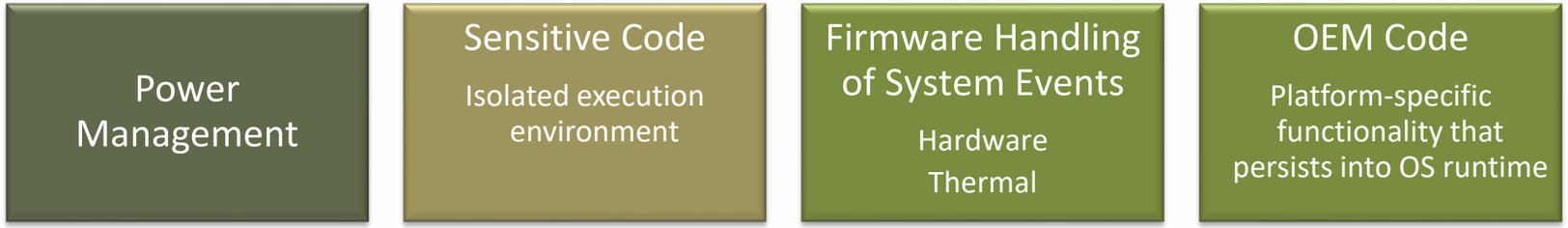
Reduce MM memory access
Eliminate excessive DXE coupling
Launch MM earlier in boot
Build an open-source MM supervisor

Secure Boot

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

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)

Containing MM

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

[BRLY-2021-021] The stack buffer overflow vulnerability leads to arbitrary code execution in UEFI DXE driver on BullSequana Edge server. (binarly.io)

The vulnerability exists due to incorrect use of the `gRT->GetVariable()` service:
- after the first call to `gRT->GetVariable()`, the value of the `DataSize` local variable will be updated (`DataSize` will contain the size of the value of the `PrimaryDisplay` NVRAM variable)
- after the second call to `gRT->GetVariable()`, the value of the `PrimaryDisplayValue` local variable will be updated (`PrimaryDisplayValue` will contain the value of the `PrimaryDisplay` NVRAM variable)
- if value of the `PrimaryDisplay` NVRAM variable more than 1 byte, a stack overflow may occur, followed by execution of arbitrary code

[BRLY-2021-011] SMM memory corruption vulnerability in combined DXE//SMM driver on Fujitsu device (SMRAM write). (binarly.io)

However, the following checks are missing:
- checking the size located at `*(CommBuffer + 3)`

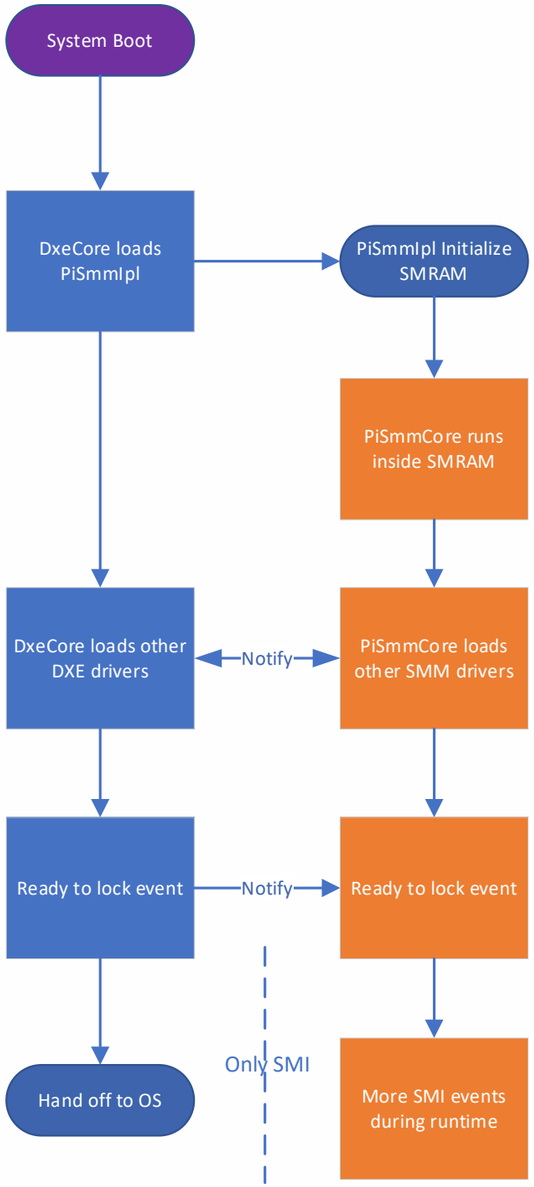
Thus, a potential attacker can write fixed data to SMRAM to corrupt some data inside this memory (for example, change SMI handler's code or modify Smram Map structures to break input pointer validation for other SMI handlers, hence to completely make this mitigation inefficient). This could lead to gaining arbitrary code execution in SMM.

To fix this vulnerability, it is essential to wrap all the input pointers (including the nested pointers) for SMI handlers with sanity checks to make sure they are not pointing into SMRAM and add a check for the size located in `*(CommBuffer + 3)`.

[BRLY-2021-008] SMM callout vulnerability in SMM driver on Fujitsu device (SMM arbitrary code execution). (binarly.io)

The `g_ReadyToBootRegistered` flag will be set (by the `SetReadyToBootRegisteredFlag()` callback) after the `SmmReadyToBoot` event is signaled. Due to this fact the vulnerability cannot be exploited from the operating system. However, using `EFI_BOOT_SERVICES` and `EFI_RUNTIME_SERVICES` services such as `SetVariable()` is unsafe inside code intended to run in SMM (from SMRAM) because an attacker capable of executing code in DXE phase could exploit this vulnerability to escalate privileges to SMM (ring -2).

Traditional MM

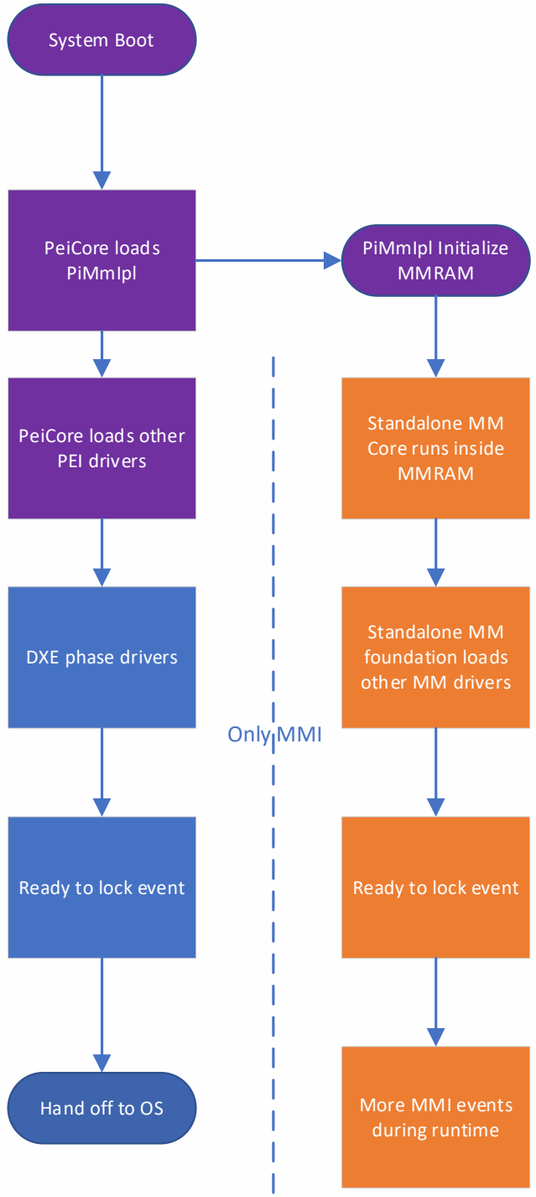


Early Boot Phase

Late Boot Phase

MMI Environment

Standalone MM



Standalone MM – Protocol Access

Traditional MM Initialization	Standalone MM Initialization
UEFI ▪ Boot Services ▪ DXE Services ▪ Runtime Services ▪ DXE protocols **MM** ▪ MM Services ▪ MM protocols	**MM** ▪ MM Services ▪ MM protocols Note: Drivers do not have access to interfaces outside the MM environment.

Traditional MM Runtime	Standalone MM Runtime
MM ▪ MM Services ▪ MM protocols	**MM** ▪ MM Services ▪ MM protocols Note: Drivers do not have access to interfaces outside the MM environment.

MM Supervisor

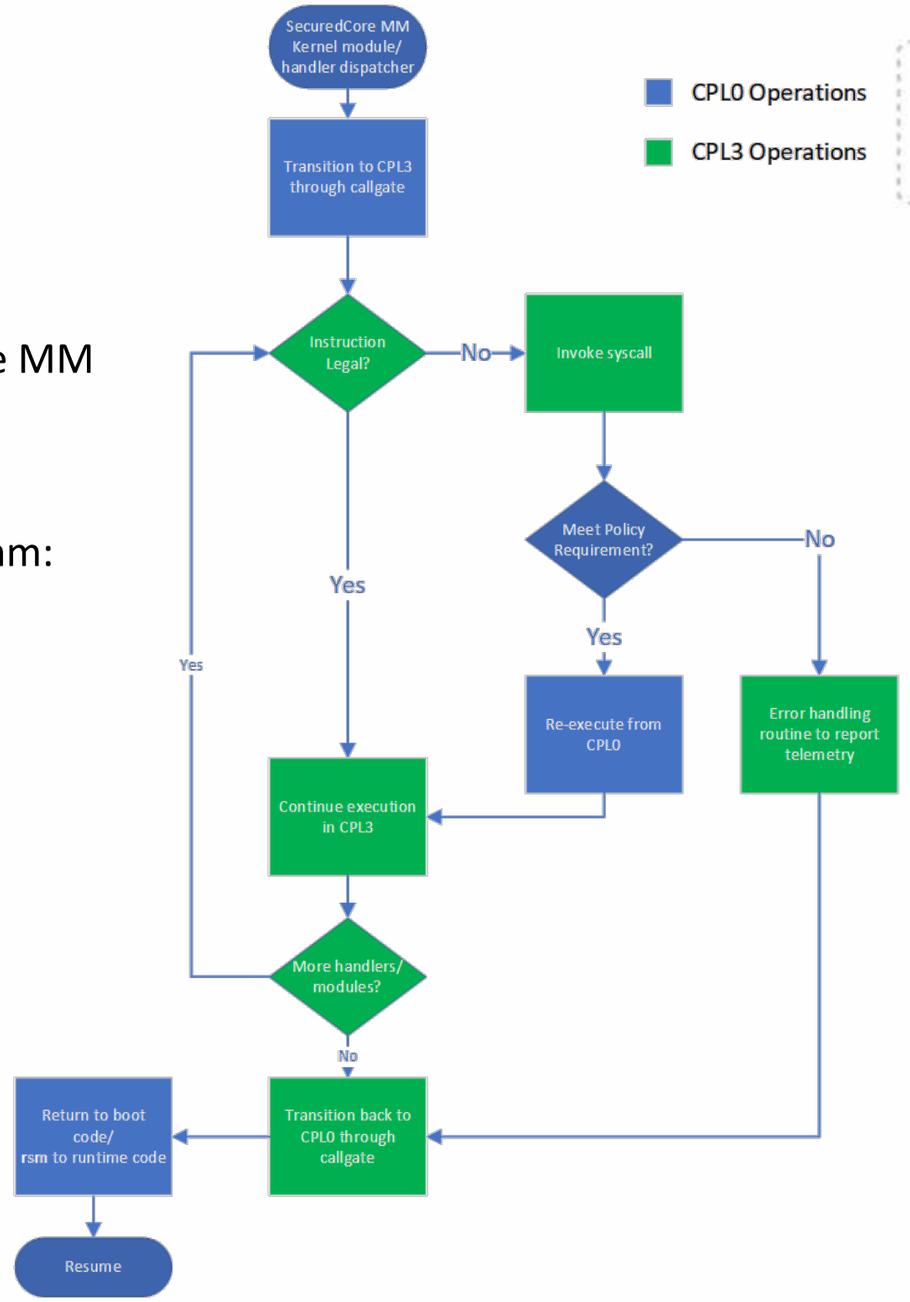
MM Supervisor - A kernel like module that operates in Standalone MM mode with policy-based resource access restrictions.

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:

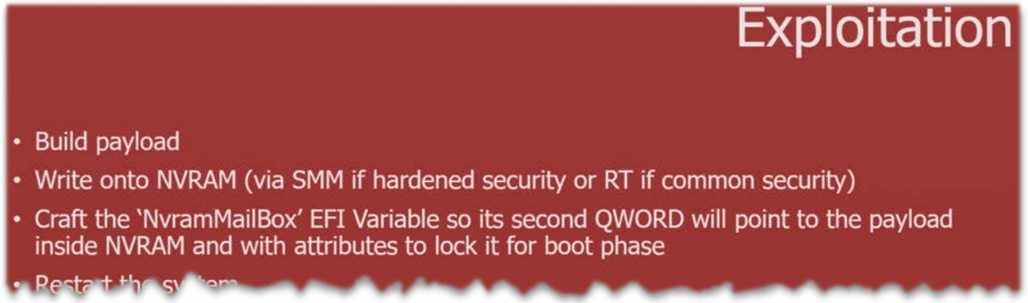
- Customizable Security Policy (Open-Source Python tools)
- PEI Launch Capability
- Privilege Separation for Resource Access Operations
- Memory Isolation



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.



Source: Data-Only Attacks Against UEFI BIOS

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

xkcd: Supported Features

Firmware Features

Platform Runtime Mechanism (PRM)

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

Begin by classifying SMIs:

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

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)

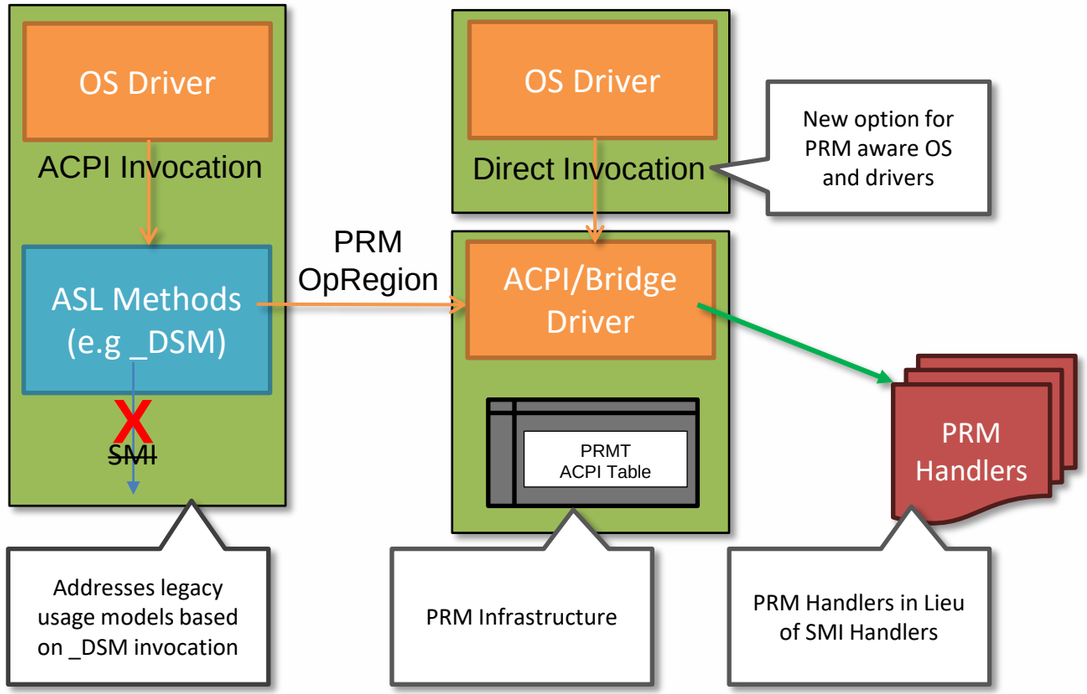
Introduces the capability to move SW SMI handlers (Category 1) and a sub-set of HW SMI handlers (Category 3) that do not require SMM privileges out of SMM and into OS/VMM execution context.

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)

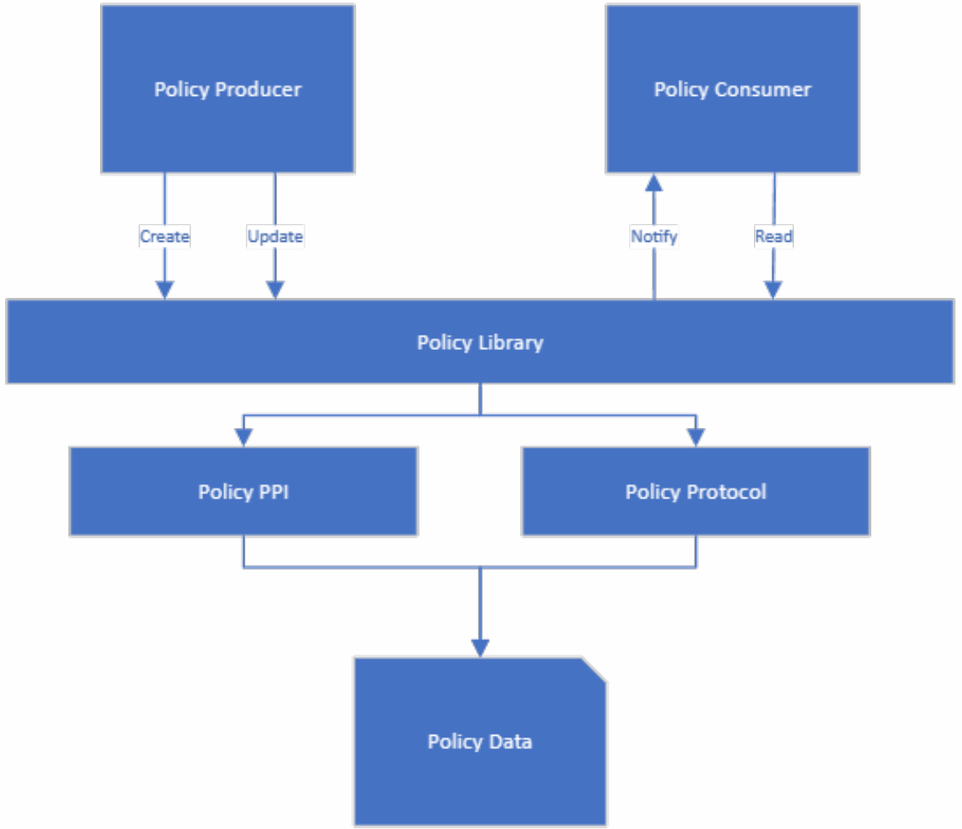
PRM Firmware Code: edk2/PrmPkg · tianocore/edk2 (github.com)

Firmware Policy

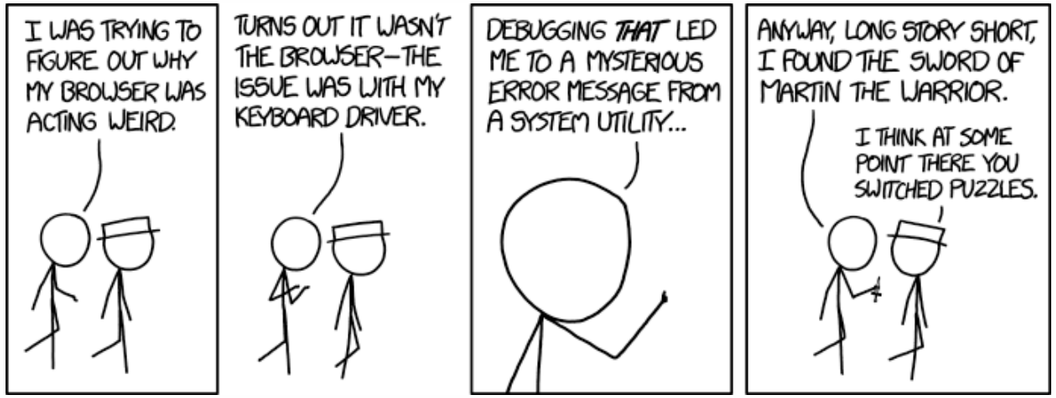
A flexible way for platforms to define and manage configuration data.

- 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.



PolicyServicePkg · microsoft/mu_basecore (github.com)



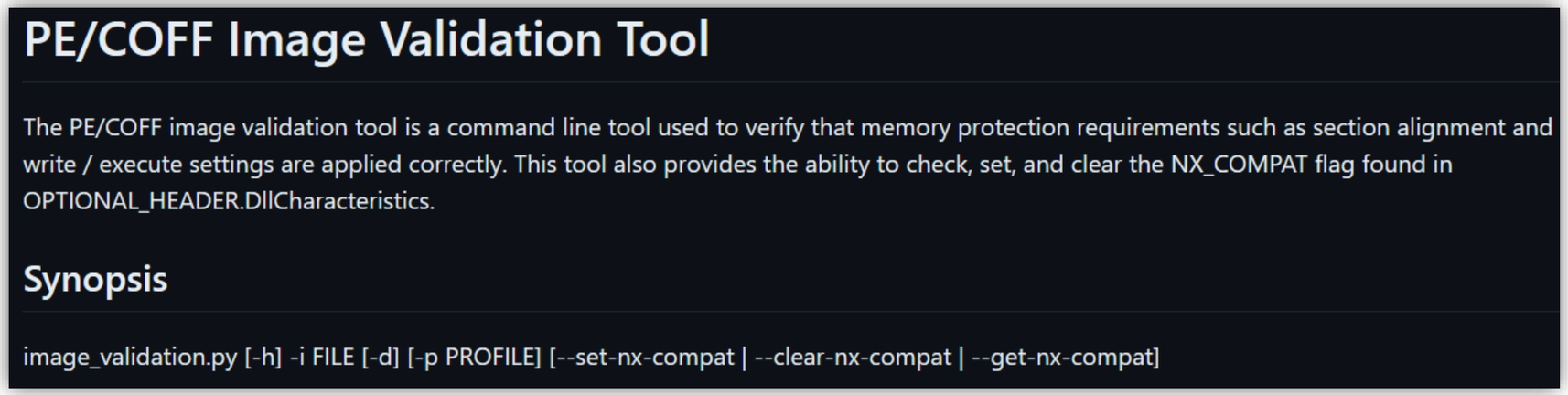
xkcd: Debugging

Test & Debug Tools

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



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

TPM Replay

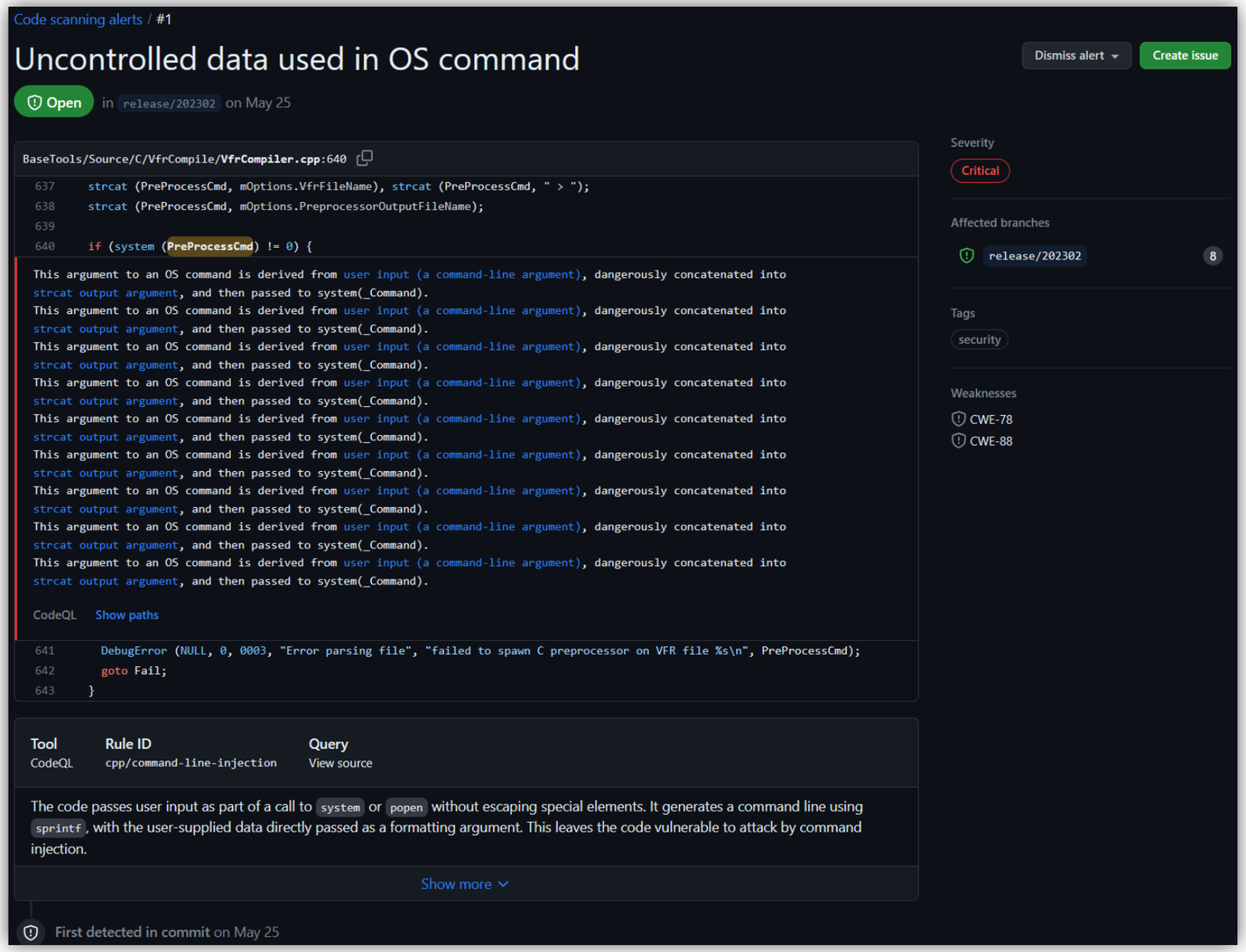
Replays a custom event log for testing operating system features dependent on measurements.

- 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)

```yaml
events:

  - type: EV_S_CRTM_VERSION
    description: "Descriptions are optional. Hash the UTF-8 string with SHA256 in PCR0."
    pcr: 0
    hash:
      - sha256
      - sha384
    data:
      type: string
      value: |-
        Example event data

  - type: EV_S_CRTM_VERSION
    description: "Descriptions are optional. Hash the UTF-16 string with null character data with SHA256 & SHA384 in PCR0."
    pcr: 0
    hash:
      - sha256
      - sha384
    data:
      type: string
      encoding: utf-16
      include_null_char: true
      value: |-
        More example data

  - type: EV_S_CRTM_VERSION
    description: "Descriptions are optional. Hash the UTF-8 string with SHA256 in PCR7."
    pcr: 7
    hash:
      - sha256
    data:
      type: string
      encoding: utf-8
      value: |-
        Data in PCR7 to prevent UEFI var measurements

  - type: EV_S_CRTM_VERSION
    description: "Descriptions are optional. Use a pre-hash SHA256 value. Event data is a UTF-16 string without a null characte
    pcr: 0
    prehash:
      sha256: "0xF97326281EABD9A5B64DD757540355165D4BED9A35B13126ED36D3A9F28A10AB"
    data:
      type: string
      encoding: utf-16
      value: |-
        Some more example data

  - type: EV_NO_ACTION
    description: "Descriptions are optional. Hash the base64 data with SHA256 and SHA394 in PCR6."
    pcr: 6
    hash:
      - sha256
      - sha384
    data:
      type: base64
      value: |-
        U2FtcGxlIGV2ZW50AA==
```

CodeQL



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

Rust

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

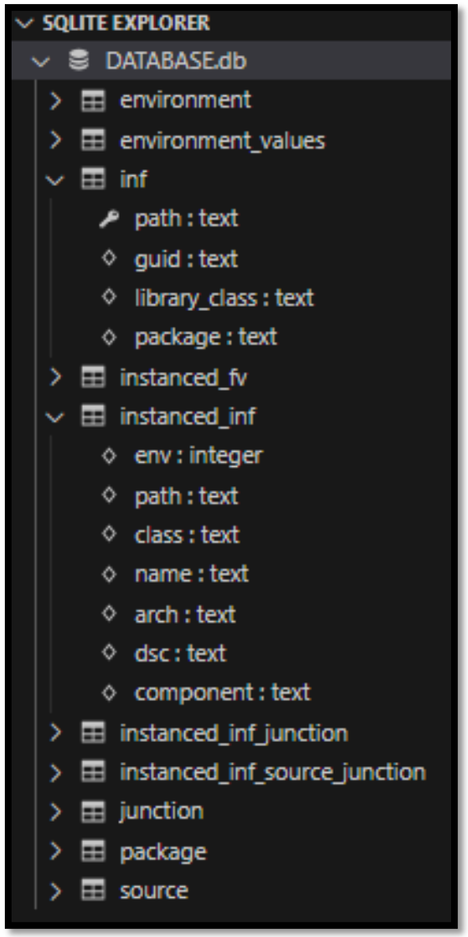
Rust - Project Mu (microsoft.github.io)

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)



features/edk2_db · tianocore/edk2-pytool-library (github.com)

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

Dependencies Between INF Files

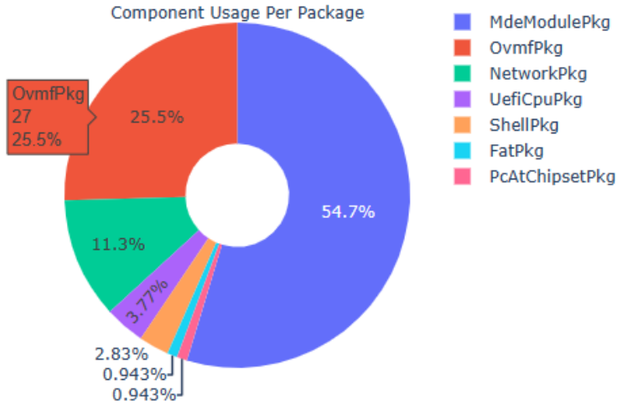
Code Coverage

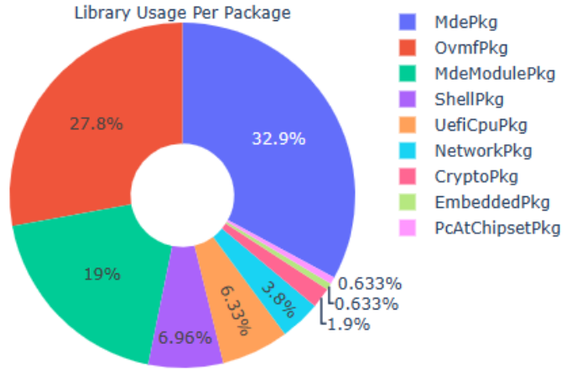
– BaseMemoryLib.inf	151	113	264	1131	57.1%
CompareMemWrapper.c	8	1	9	61	88.8%
CopyMem.c	77	0	77	148	100%
CopyMemWrapper.c	9	0	9	59	100%
MemLibGeneric.c	10	58	68	307	14.7%
MemLibGuid.c	12	18	30	167	40%
SetMem.c	24	1	25	80	96%
SetMem16Wrapper.c	0	9	9	58	0%
SetMem32Wrapper.c	0	9	9	58	0%
SetMem64Wrapper.c	0	9	9	58	0%
SetMemWrapper.c	5	7	12	85	41.6%
ZeroMemWrapper.c	6	1	7	50	85.7%

Code Coverage Per INF

features/edk2_db · tianocore/edk2-pytool-library (github.com)

Platform Usage Report





Build Information
Platform: OvmfPkg/OvmfPkgIa32X64.dsc
Target: DEBUG
Architectures: IA32 X64
Toolchain: GCC5
Commit Sha: f36e1ec1f0a5fd3be84913e09181d7813444b620

Components and Libraries From Other Packages

features/edk2_db · tianocore/edk2-pytool-library (github.com)

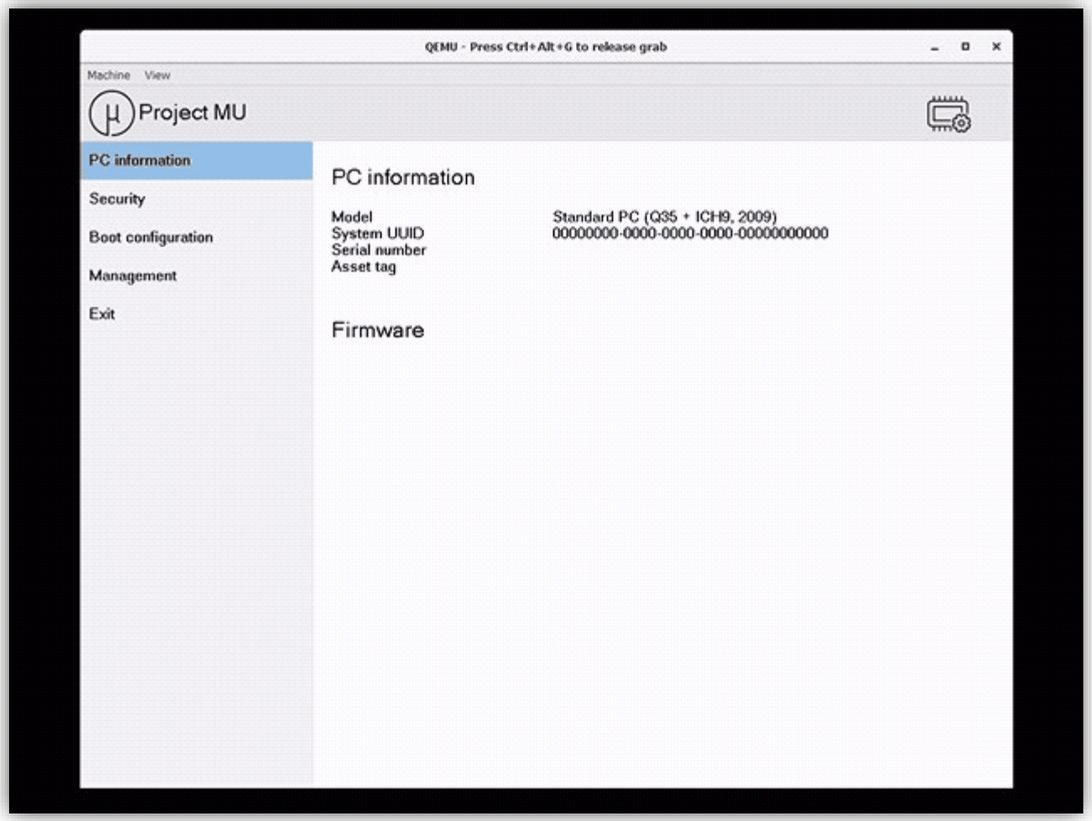
Virtual Platforms

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



Project Mu Virtual Platform Firmware (github.com)

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