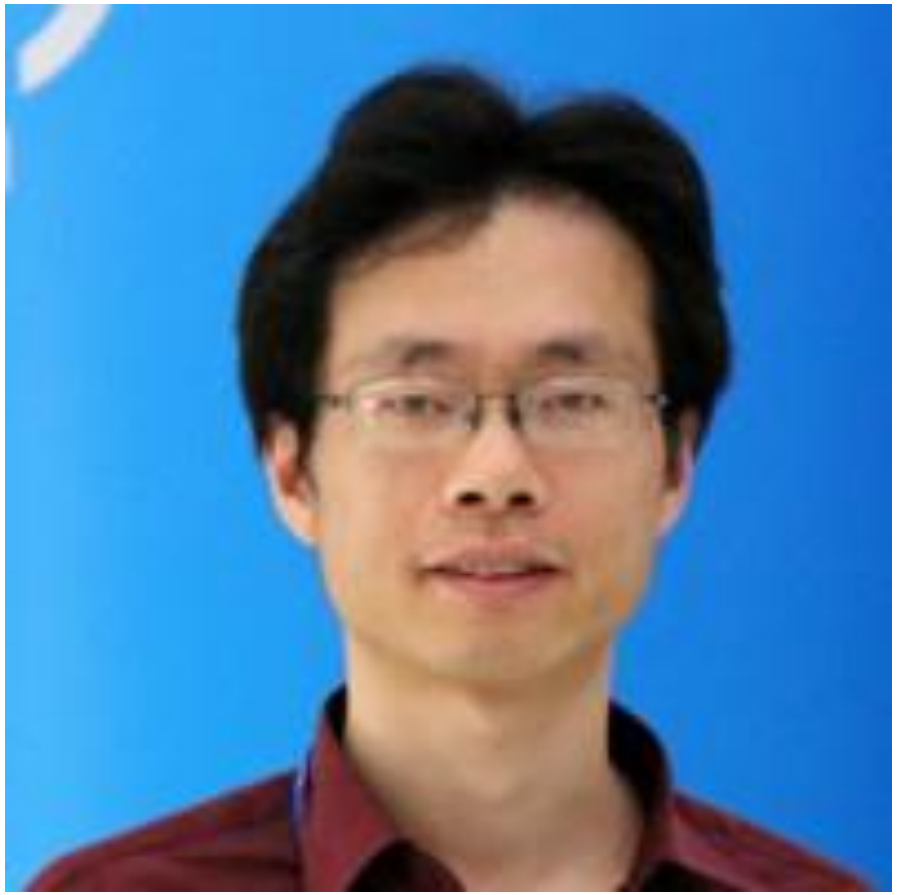
Enabling Rust for UEFI Firmware

SPEAKER

Jiewen Yao, Principal Engineer, Intel
Vincent Zimmer, Senior Principal Engineer, Intel

Jiewen Yao

- **Jiewen Yao** is a principal engineer in the Intel Architecture, Graphics, and Software Group. He has been engaged as a firmware developer for over 15 years. He is a member of the UEFI Security sub team, and the TCG PC Client sub working group.



Vincent Zimmer

- **Vincent Zimmer** is a senior principal engineer in the Intel Architecture, Graphics, and Software Group. He has been engaged as a firmware developer for over 25 years and leads the UEFI Security sub team.

Vincent Zimmer
Intel

OPEN SOURCE FIRMWARE CONFERENCE

Agenda

- EDKII Security Summary
- RUST Language
- Enabling RUST for EDKII

EDKII Security Summary

BIOS Security Bug

Top Issue	Open Source	Close Source
Buffer Overflow/ Integer Overflow	50%	38%
SMM	7%	18%
Variable	8%	5%
Register Lock	3%	10%

Firmware as Software

- **Many software issues are also firmware issues.**
  - Buffer Overflow
  - Integer Overflow
  - Uninitialized Variable
- **Software mitigation can be used for firmware mitigation.**
  - (See next page)

3 Levels of Prevention

Prevention	Method	EDKII Open Source Example
Eliminate Vulnerability	Reduce Attack Surface	SMI Handler Profile
	Static Analysis / Dynamic Analysis	Clang Static Analysis, Memory Sanitizer, KLEE
	Security Test / Fuzzing	Host-based Firmware Analyzer, Peach, AFL
	Vulnerability Scan	Chipsec
Break Exploitation	Stack Guard	MSVC:/GS, GCC:-fstack-protector
	Address Space Layout Randomization	DXE/SMM ASLR
	Non Executable Data	SMM Memory Protection
	Control Flow Guard	SMM Control-flow Enforce Technology (CET)
	Code Integrity	UEFI Secure Boot
Contain Damage	Sandbox	EBC
	Deprivilege	Ring3-based third-party Code (?)
	Isolation	(?)

What's More: Type Safe Language

Rather than providing guidance and tools for addressing flaws,

we should strive to prevent the developer from introducing the flaws in the first place.

Source: https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/

RUST Language Introduction

RUST Language

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enable you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Source: https://www.rust-lang.org /

Memory Safety – in RUST

Type	Sub Type	RUST
Access Error	Buffer Overflow (Write)	Use Offset/Index for Slice Runtime Boundary Check – [panic_handler]
	Buffer Over-Read	Use Offset/Index for Slice Runtime Boundary Check – [panic_handler]
	Use After Free (Dangling Pointer)	Ownership - Compile Time Check
	Double Free	Ownership - Compile Time Check
	Race Condition	Thread Safety - Compile Time Check
Uninitialized Data	Uninitialized Variable	Initialization - Compile Time Check
	Wild Pointer	Initialization - Compile Time Check
	NULL pointer deference	Use Option<T> enum Allocation Check – [alloc_error_handler]
Memory Leak	Stack Exhausing	N/A
	Heap Exhausing	Allocation Check – [alloc_error_handler]

Arithmetics – in RUST

Type	Method	RUST
Integer Overflow	Addition/ Subtraction/ Multiplication/ Division/ Shift/ Power Overflow	**DEBUG**: Runtime Check – [panic_handler] **RELEASE**: Discard overflow data **Compiler Flage**: -C overflow-checks=on/off **Function**: checked|overflowing|saturating|wrapping_ add|sub|mul|div|rem|shl|shr|pow()

Type Cast	Number Cast	**Must be explicit** – compile time check
		(Dest Size == Source Size) => no-op
		(Dest Size < Source Size) => truncate
		(Dest Size > Source Size) => {
		(source is unsigned) => zero-extend
		(source is signed) => sign-extend
		}

Enabling RUST for EDKII

Build RUST in EDKII

- EDKII **Staging Branch** : edkii-rust
  - https://github.com/tianocore/edk2-staging/tree/edkii-rust
  - https://github.com/jyao1/edk2/tree/edkii-rust

- **Compiler**: LLVM9.0 + RUST Cargo-xbuild

- **Target**: (supported in rust-lang master)
  - x86_64-unknown-uefi
  - i686-unknown-uefi

Build Type

- C code and RUST code mixed in EDKII INF. (driver or lib)
- Rust Module in Cargo Toml file. (driver or lib)
  - Rust Driver + Rust Lib
  - Rust Driver + C Lib
  - C Driver + Rust Lib

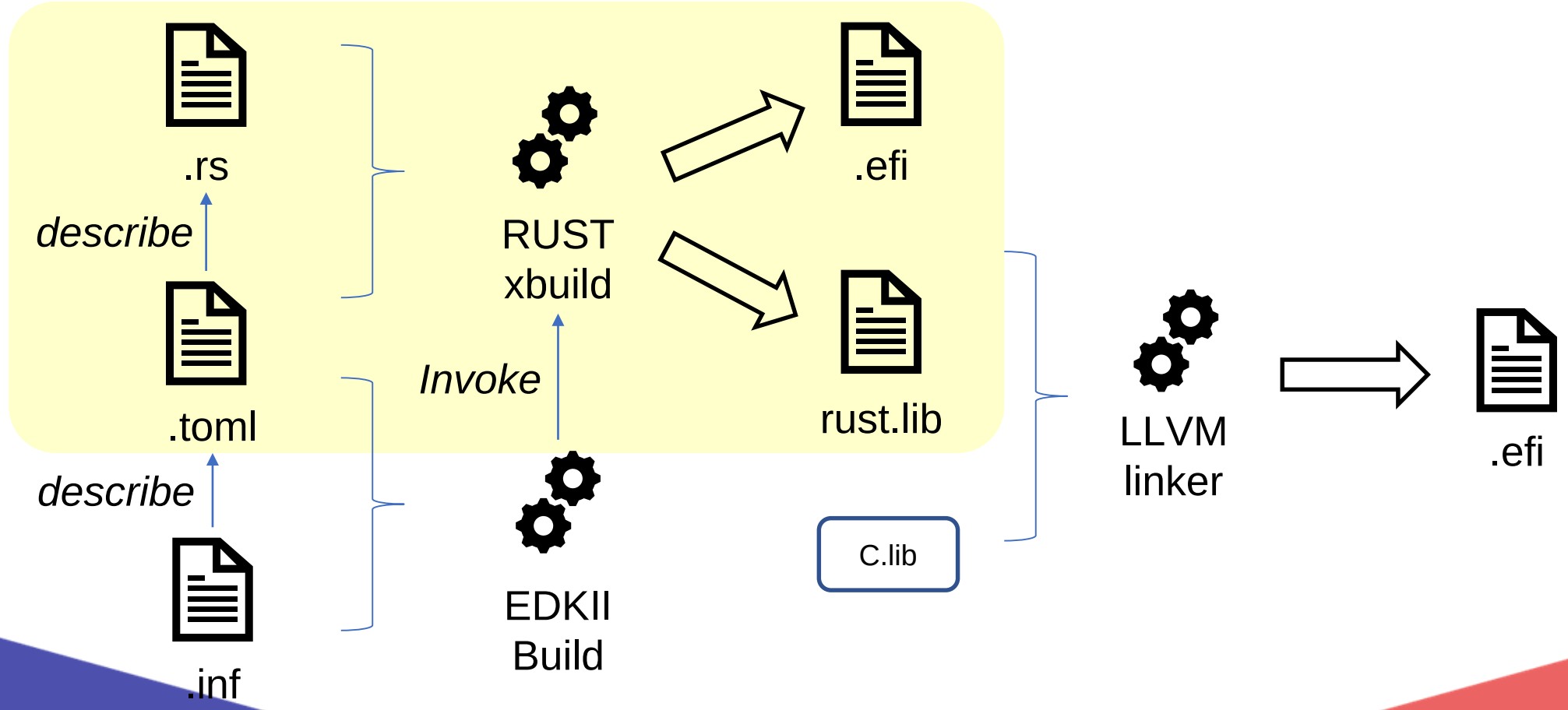
```
# BaseBmpSupportLibRust.inf
[Defines]
  INF_VERSION     = 0x00010017
  BASE_NAME       = BaseBmpSupportLibRust
  FILE_GUID       = 54708BEB-12EA-475B-86D0-A16C08A8B230
  MODULE_TYPE     = BASE
  LIBRARY_CLASS   = BmpSupportLib
# RUST_MODULE     = TRUE # to build .efi only
[Sources]
  Cargo.toml
```

```
# Cargo.toml
[package]
name = "BaseBmpSupportLibRust"
version = "0.1.0"

[lib]
name = "base_bmp_support_lib_rust"
crate-type = ["staticlib"]

[dependencies]
r-efi = { path = "../../../External/r-efi" }
base_lib = { path = "../../../MdePkg/Library/BaseLibRust"
```

Build Process



Rust Library Usage

- Use “**core**” crate.
  - Implement **[panic_handler]** for runtime check
- May use “**alloc**” crate for Box, Vec, etc.
  - Implement **[global_allocator]** – GlobalAlloc {alloc, dealloc}
  - Implement **[alloc_error_handler]** for allocation fail.
- Do not use “**std**” crate.

Rust Example for EDKII

- **fat-rust**: FAT file system library:
  - https://github.com/jyao1/edk2/tree/edkii-rust/RustPkg/External/FatDxeLibRust
- **efi-lib**: memory allocation, debug log, boot services, etc
  - https://github.com/jyao1/edk2/tree/edkii-rust/RustPkg/External/efi-lib
- **efi-str**: handle CHAR16 string in UEFI
  - https://github.com/jyao1/edk2/tree/edkii-rust/RustPkg/External/efi-str

- **RUST Initial Program Loader (IPL)**: RUST-based SEC *(in progress)*
  - https://github.com/jyao1/edk2/tree/minovmf/MinOvmf64FwPkg/RustSec

Rust Crypto Library for EDKII

- **ring**: for general purpose Cryptography (RSA, ECC, etc)
- **webpki**: for Public Key Infrastructure Certificate
  - Add extension for UEFI/EDKII.
  - https://github.com/jyao1/ring/tree/uefi_support
  - https://github.com/jyao1/webpki/tree/uefi_support

- **efi-random**: RDRAND, RDSEED instruction
  - https://github.com/jyao1/edk2/tree/edkii-rust/RustPkg/External/efi-random

Some Limitations

- UEFI specification and interfaces are defined in C.
- Cross module interaction is C-API.
- Unsafe Code is required.

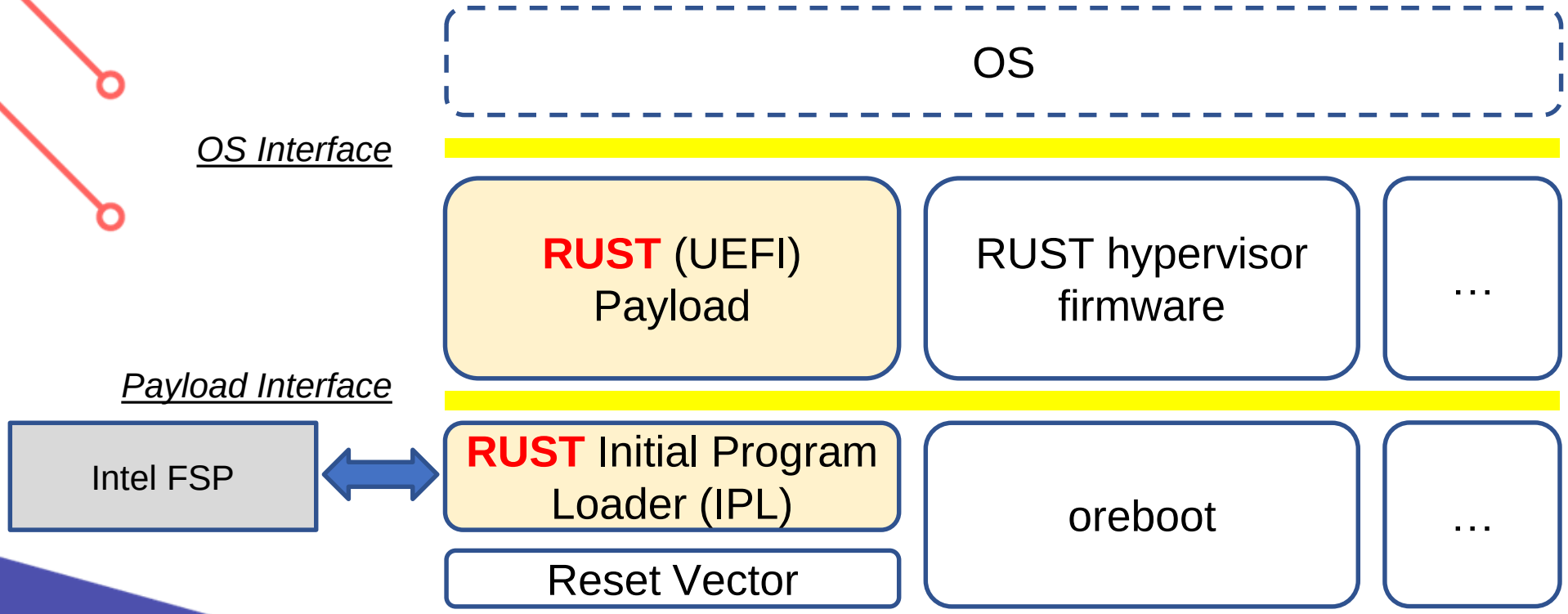
Where RUST Can Help

- 1. **Eliminate Vulnerability** (Compile Time Check)
  - Unitialized variable
  - Use After Free
  - Double Free
- 2. **Break Exploitation** (Runtime Check)
  - Memory Boundary Check
  - Integer Overflow Check
- NOTE: Boundary Check Code is required to prevent system from panic.

Where RUST Cannot Help

- **Silicon Register Lock**
  - Need Chipsec
- **Security Policy**
  - Need policy checker
- **TOC/TOU**
  - Need design review
- **SMM Callout**
  - Need hardware restriction
- **Unsafe Code Block**
  - Need careful code review
  - NOTE: Putting C code in Rust Unsafe Block helps nothing.

Thought and Current work



Reference

- UEFI Rust overview
  - https://uefi.org/sites/default/files/resources/Enabling%20RUST%20for%20UEFI%20Firmware_8.19.2020.pdf
- EDKII Security Bug
  - https://edk2-docs.gitbooks.io/security-advisory/content/
  - https://bugzilla.tianocore.org/buglist.cgi?bug_status=__all__&list_id=16941&order=bug_id&product=Tianocore%20Security%20Issues&query_format=specific
- Rust Type Safe Language
  - https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/
  - http://design.inf.unisi.ch/sites/default/files/seminar-niko-matsiakis-rustoverview.pdf
  - https://www.microsoft.com/en-us/research/project/project-verona/

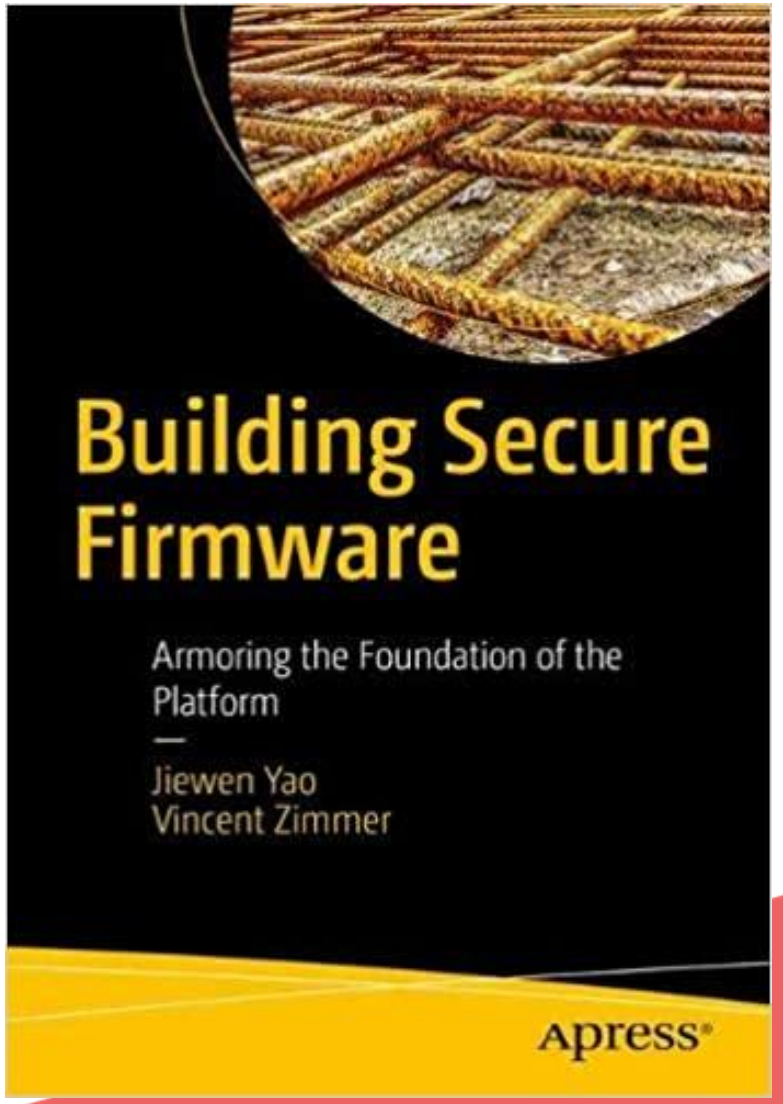
Reference

- Other EFI-Rust Project
  - **r-efi**: https://github.com/r-util/r-efi
  - **uefi-rs**: https://github.com/rust-osdev/uefi-rs
  - **Redox uefi:** https://gitlab.redox-os.org/redox-os?utf8=%E2%9C%93&filter=uefi
  - **rust-hypervisor-firmware:** https://github.com/cloud-hypervisor/rust-hypervisor-firmware
  - **Rust-based Unit Test:** https://github.com/corthon/edk2-staging/tree/rust_and_tests
- Rust Project
  - **oreboot**: https://github.com/oreboot/oreboot
  - **OSDev**: https://github.com/rust-osdev
  - **Tock OS**: https://github.com/tock/tock
  - **Redox OS**: https://gitlab.redox-os.org/redox-os
  - **OpenTitan**: https://opentitan.org/
  - **firecracker**: https://github.com/firecracker-microvm/firecracker
  - **libra**: https://github.com/libra/libra
  - **SGX SDK**: https://github.com/apache/incubator-teaclave-sgx-sdk

To Learn More About UEFI Security

Building Secure Firmware: Armoring the Foundation of the Platform

- https://www.amazon.com/gp/product/1484261054/
- https://link.springer.com/content/pdf/10.1007%2F978-1-4842-6106-4.pdf



Questions?

OPEN SOURCE FIRMWARE CONFERENCE

Thank you

OPEN SOURCE FIRMWARE CONFERENCE