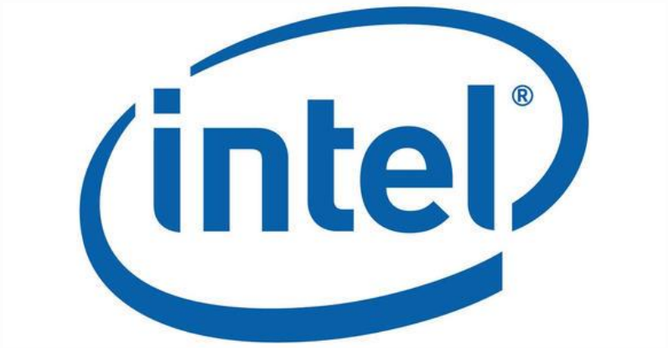
presented by



Enabling RUST for UEFI Firmware

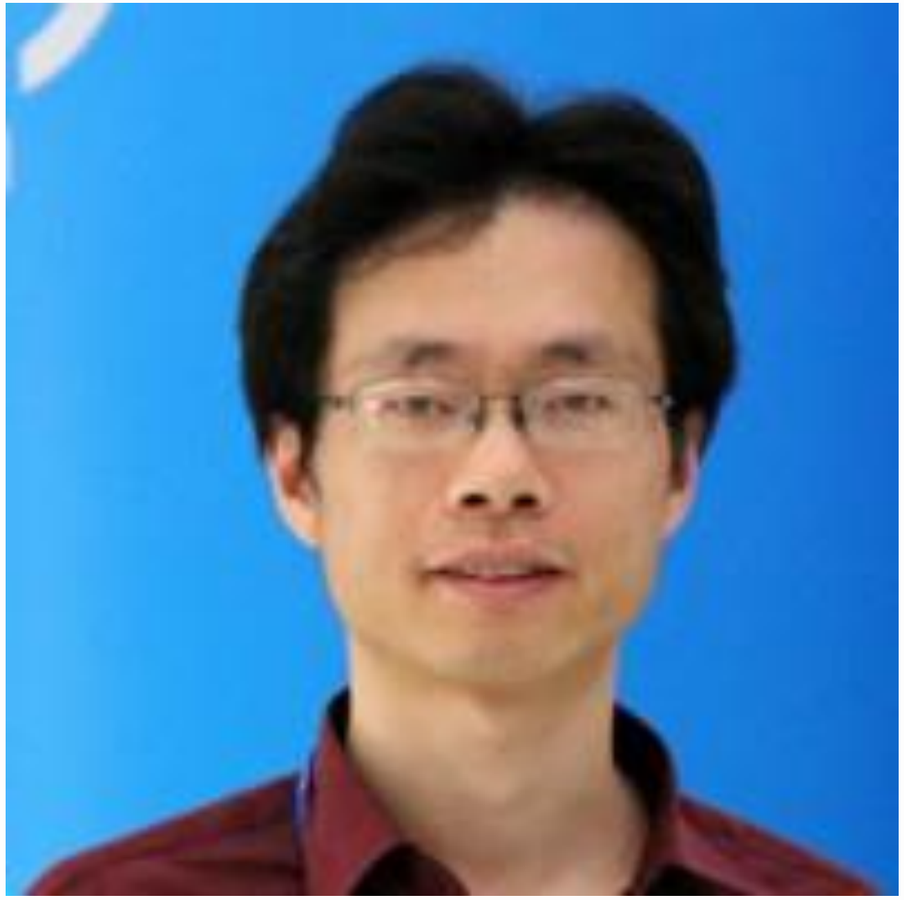
UEFI 2020 Virtual Plugfest

August 20, 2020

Jiewen Yao & Vincent Zimmer, Intel Corporation

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.

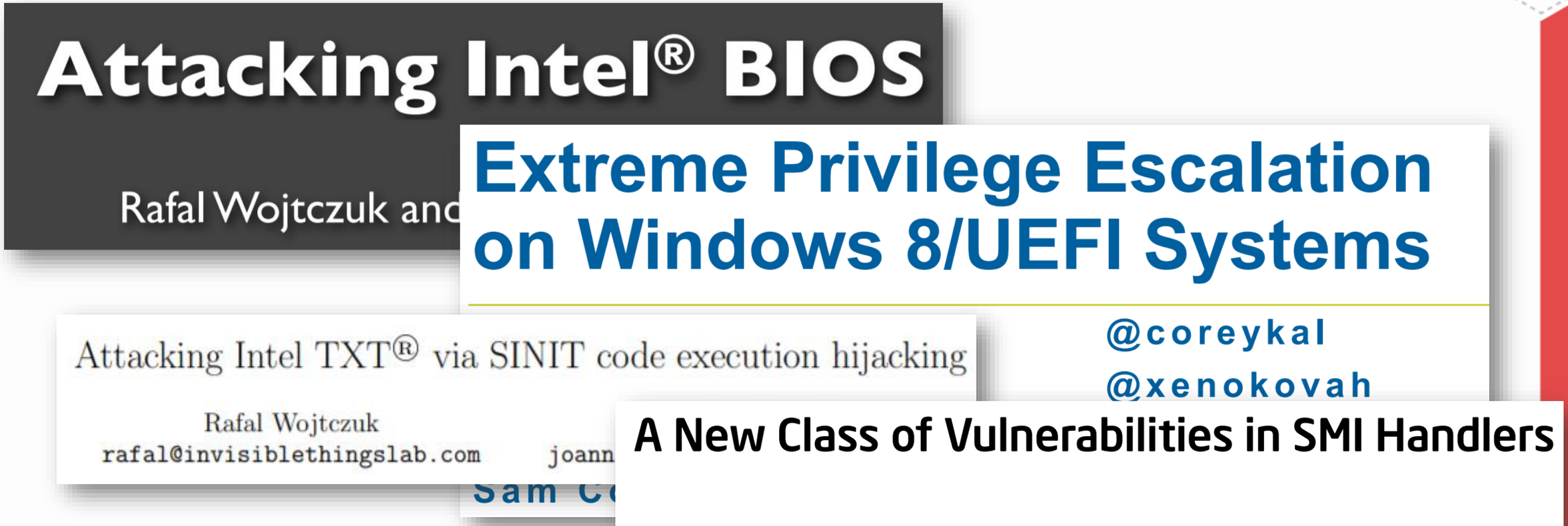
Vincent Zimmer
Intel

Agenda

- EDKII Security Summary
- RUST Language
- Enabling RUST for EDKII
- Summary / Call to Action

EDKII Security Summary

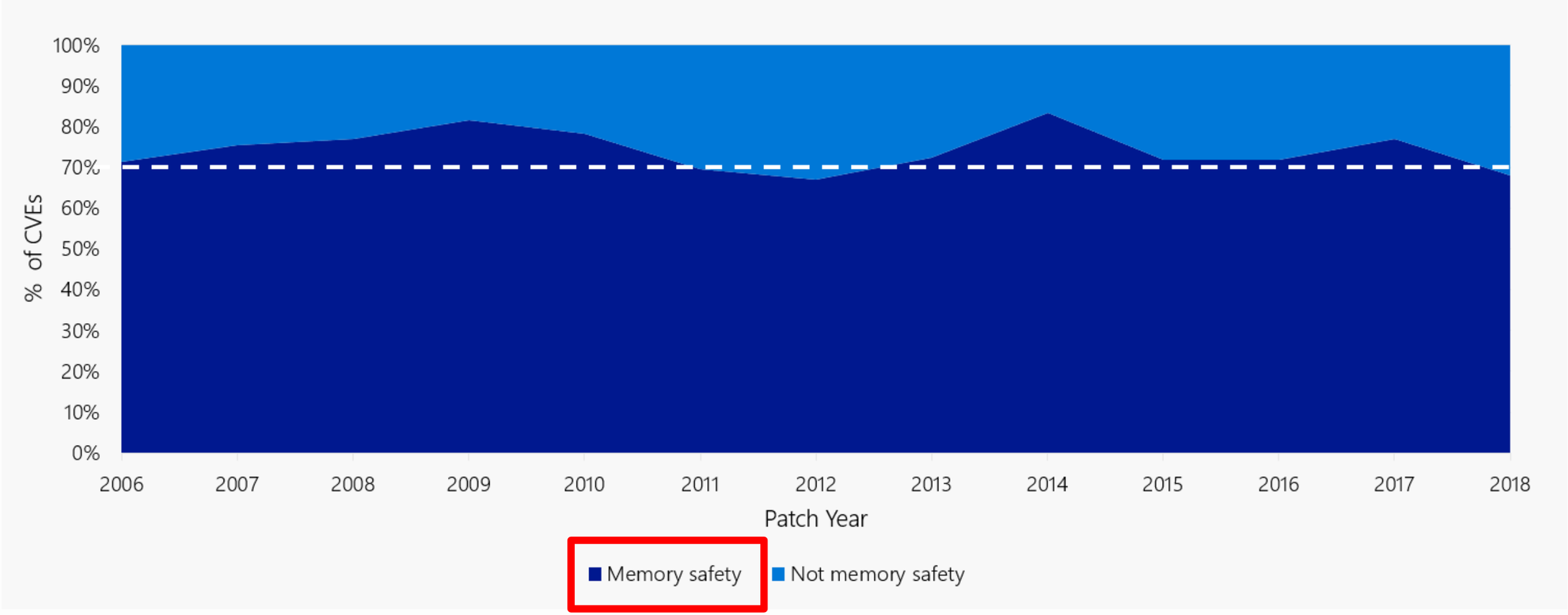
BIOS Memory Issue in Hack Conf



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%

Vulnerabilities in C/C++



Source: Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape – Microsoft, Bluehat IL 2019

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 /

RUST Project

Firecracker

libra

CircleCI license Apache codecov 57%

chat on gitter

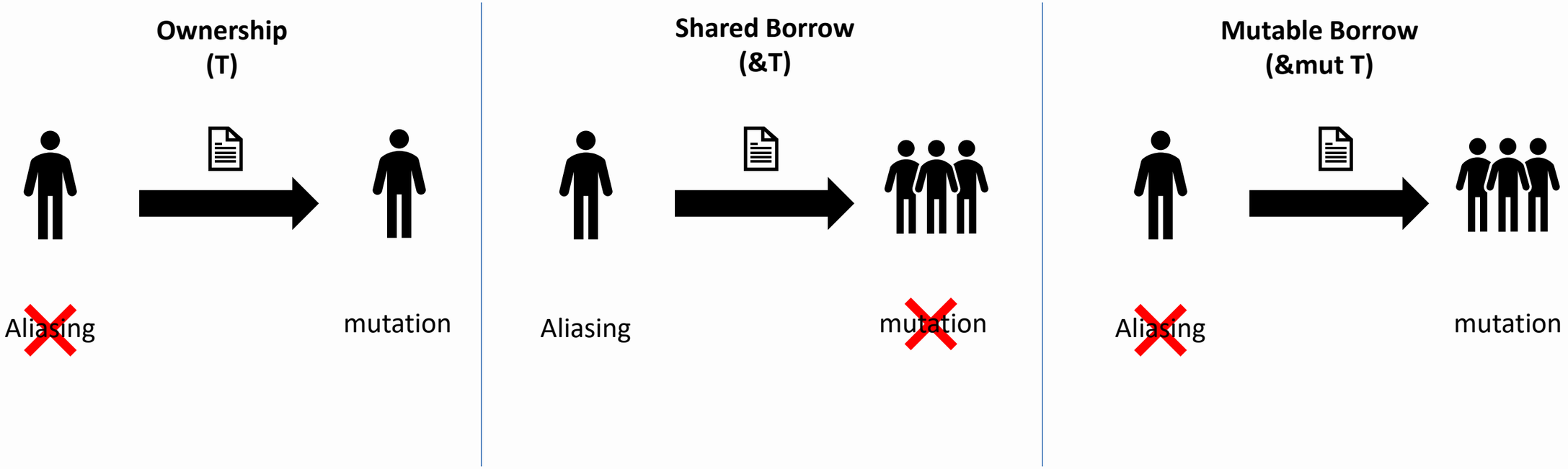
Rust SGX SDK

redox-os

Group ID: 3

Redox OS

RUST Memory Safety



*rule out **mutation** in the presence of **aliasing***

Memory Safety Issue in BIOS

Type	Sub Type	Threat in BIOS	Risk in BIOS (Probability/Impact)
Access Error	Buffer Overflow (Write)	Integrity	High / High
	Buffer Over-Read	Confidentiality	High / Low
	Use After Free (Dangling Pointer)	Availability	Medium / Low
	Double Free	Availability	Medium / Low
	Race Condition	Integrity	Low / Low
Uninitialized Data	Unitialized Variable	Availability	High / Medium
	Wild Pointer	Availability	Medium / Low
	NULL pointer deference	Availability	High / Medium
Memory Leak	Stack Exhausing	Availability	Low / Low
	Heap Exhausing	Availability	High / Low

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
		}

Pointer in RUST

Type	Description	Example	Usage
Raw Pointer	Unsafe	*const T	Read only Memory
		*mut T	Read-write Memory
Reference	Memory owned by some other value	&T	Shared, Immutable Memory
		&mut T	Exclusive, Mutable Memory
Smart Pointer	Special Data Structure	Box<T>	Heap Memory Allocation

Unsafe Code

- Dereference raw pointer (from C function)
- Call unsafe functions (to C function)
- Access RUST mutable global static variable
- Implement RUST unsafe trait
- Access RUST Union

In this case, you trust me.
- Dev To Rust

Enabling RUST for EDKII

Build RUST in EDKII

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

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

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

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 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

Other EFI-Rust Project

- **r-efi**: UEFI Reference Specification Protocol Constants and Definitions
  - https://github.com/r-util/r-efi
- **uefi-rs**: Wrapper for writing UEFI applications in RUST.
  - https://github.com/rust-osdev/uefi-rs
- **Redox uefi support**: Wrapper for UEFI services.
  - https://gitlab.redox-os.org/redox-os?utf8=%E2%9C%93&filter=uefi
- **rust-hypervisor-firmware**: A simple KVM firmware for cloud hypervisor with minimal UEFI support.
  - https://github.com/cloud-hypervisor/rust-hypervisor-firmware
- **Rust-based Unit Test in EDKII**: Rust-based UefiVariablePolicyLib with Unit Test.
  - https://github.com/corthon/edk2-staging/tree/rust_and_tests

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 still 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.

Summary & Call for Action

Summary & Call for Action

- 50% of EDKII security issues are memory issues.
- RUST can help to mitigate memory issues.
- Write critical firmware modules in RUST.

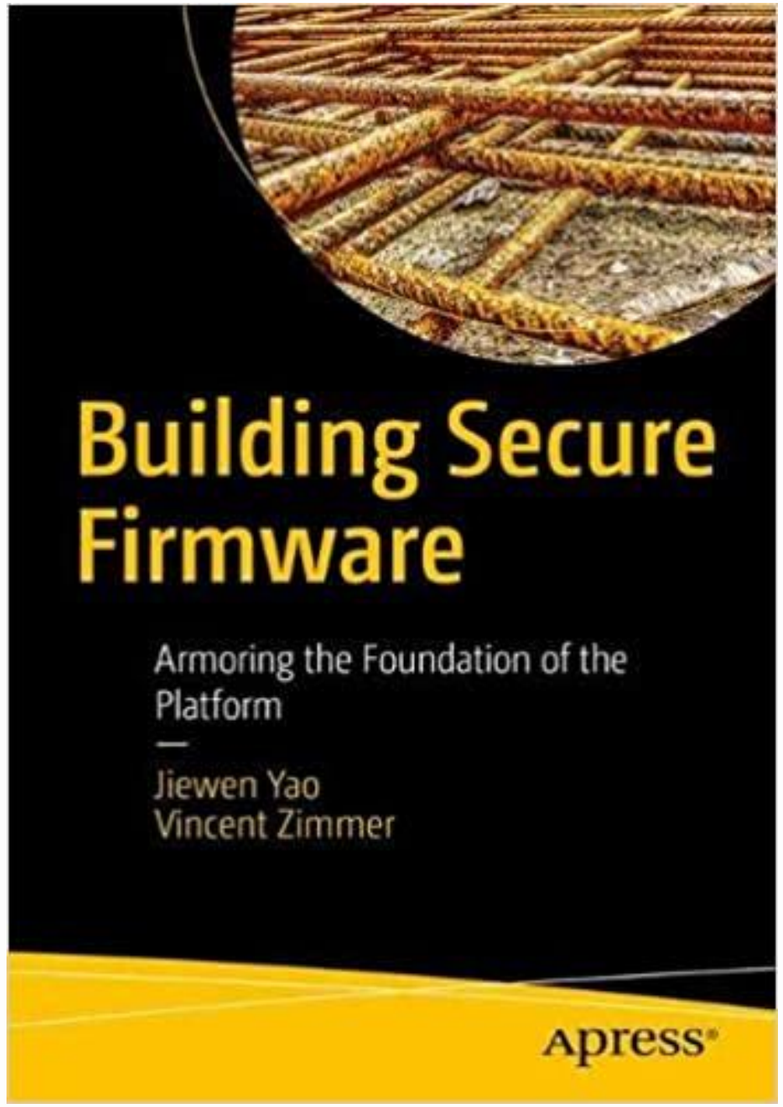
Reference

- Attack
  - https://www.blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf
  - https://www.mitre.org/sites/default/files/publications/14-2221-extreme-escalation-presentation.pdf
  - https://invisiblethingslab.com/resources/2011/Attacking_Intel_TXT_via_SINIT_hijacking.pdf
  - http://www.c7zero.info/stuff/ANewClassOfVulnInSMIHandlers_csw2015.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
- Rust Project
  - https://github.com/libra/libra
  - https://github.com/apache/incubator-teaclave-sgx-sdk
  - https://github.com/firecracker-microvm/firecracker
  - https://github.com/oreboot/oreboot
  - https://gitlab.redox-os.org/redox-os

To Learn More About UEFI Security

Building Secure Firmware: Armoring the Foundation of the Platform

https://www.amazon.com/gp/product/1484261054/



Questions?

Thanks for attending the UEFI 2020 Virtual Plugfest

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

presented by