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.
Agenda

- EDKII Security Summary
- RUST Language
- Enabling RUST for EDKII
- Summary / Call to Action
EDKII Security Summary
BIOS Memory Issue in Hack Conf

Attacking Intel® BIOS
Rafal Wojtczuk and
Sam Cumpsty

Attacking Intel TXT® via SINIT code execution hijacking
Rafal Wojtczuk
rafa@invisiblethingslab.com

Extrem Privilege Escalation on Windows 8/UEFI Systems
@coreykal
@xenokovah

A New Class of Vulnerabilities in SMI Handlers
Advanced Threat Research (www.intelsecurity.com/atr)
Oleksandr Bazhaniuk, Yuriy Bulygin, Andrew Furtak, Mikhail Gorobets, John Loucaides, Alexander Matrosov, Mickey Shkatov
## BIOS Security Bug

<table>
<thead>
<tr>
<th>Top Issue</th>
<th>Open Source</th>
<th>Close Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Overflow/Integer Overflow</td>
<td>50%</td>
<td>38%</td>
</tr>
<tr>
<td>SMM</td>
<td>7%</td>
<td>18%</td>
</tr>
<tr>
<td>Variable</td>
<td>8%</td>
<td>5%</td>
</tr>
<tr>
<td>Register Lock</td>
<td>3%</td>
<td>10%</td>
</tr>
</tbody>
</table>
Vulnerabilities in C/C++

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

www.uefi.org
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

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

www.uefi.org
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/](https://www.rust-lang.org/)
RUST Project

Firecracker

libra

SGX

Rust SGX SDK

Redox OS

www.uefi.org
RUST Memory Safety

Ownership (T)

Aliasing

mutation

Shared Borrow (&T)

Aliasing

mutation

Mutable Borrow (&mut T)

Aliasing

mutation

rule out mutation in the presence of aliasing
# Memory Safety Issue in BIOS

<table>
<thead>
<tr>
<th>Type</th>
<th>Sub Type</th>
<th>Threat in BIOS</th>
<th>Risk in BIOS (Probability/Impact)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Error</td>
<td>Buffer Overflow (Write)</td>
<td>Integrity</td>
<td>High / High</td>
</tr>
<tr>
<td></td>
<td>Buffer Over-Read</td>
<td>Confidentiality</td>
<td>High / Low</td>
</tr>
<tr>
<td></td>
<td>Use After Free (Dangling Pointer)</td>
<td>Availability</td>
<td>Medium / Low</td>
</tr>
<tr>
<td></td>
<td>Double Free</td>
<td>Availability</td>
<td>Medium / Low</td>
</tr>
<tr>
<td></td>
<td>Race Condition</td>
<td>Integrity</td>
<td>Low / Low</td>
</tr>
<tr>
<td>Uninitialized Data</td>
<td>Uninitialized Variable</td>
<td>Availability</td>
<td>High / Medium</td>
</tr>
<tr>
<td></td>
<td>Wild Pointer</td>
<td>Availability</td>
<td>Medium / Low</td>
</tr>
<tr>
<td></td>
<td>NULL pointer deference</td>
<td>Availability</td>
<td>High / Medium</td>
</tr>
<tr>
<td>Memory Leak</td>
<td>Stack Exhausing</td>
<td>Availability</td>
<td>Low / Low</td>
</tr>
<tr>
<td></td>
<td>Heap Exhausing</td>
<td>Availability</td>
<td>High / Low</td>
</tr>
</tbody>
</table>
# Memory Safety – in RUST

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>RUST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Overflow</td>
<td>Addition/</td>
<td><strong>DEBUG</strong>: Runtime Check – [panic_handler]</td>
</tr>
<tr>
<td></td>
<td>Subtraction/</td>
<td><strong>RELEASE</strong>: Discard overflow data</td>
</tr>
<tr>
<td></td>
<td>Multiplication/</td>
<td><strong>Compiler Flage</strong>: -C overflow-checks=on/off</td>
</tr>
<tr>
<td></td>
<td>Division/</td>
<td><strong>Function</strong>:</td>
</tr>
<tr>
<td></td>
<td>Shift/</td>
<td>checked</td>
</tr>
<tr>
<td></td>
<td>Power Overflow</td>
<td>add</td>
</tr>
<tr>
<td>Type Cast</td>
<td>Number Cast</td>
<td><strong>Must be explicit</strong> – compile time check</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Dest Size == Source Size) =&gt; no-op</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Dest Size &lt; Source Size) =&gt; truncate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Dest Size &gt; Source Size) =&gt; {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(source is unsigned) =&gt; zero-extend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(source is signed) =&gt; sign-extend</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw Pointer</td>
<td>Unsafe</td>
<td>*const T</td>
<td>Read only Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*mut T</td>
<td>Read-write Memory</td>
</tr>
<tr>
<td>Reference</td>
<td>Memory owned by some other value</td>
<td>&amp;T</td>
<td>Shared, Immutable Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;mut T</td>
<td>Exclusive, Mutable Memory</td>
</tr>
<tr>
<td>Smart Pointer</td>
<td>Special Data Structure</td>
<td>Box&lt;T&gt;</td>
<td>Heap Memory Allocation</td>
</tr>
</tbody>
</table>
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

• **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](https://github.com/jyao1/edk2/tree/edkii-rust/RustPkg/External/FatDxeLibRust)

- **efi-lib**: memory allocation, debug log, boot services, etc

- **efi-str**: handle CHAR16 string in UEFI
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/ring/tree/uefi_support)
  - [https://github.com/jyao1/webpki/tree/uefi_support](https://github.com/jyao1/webpki/tree/uefi_support)

- **efi-random**: RDRAND, RDSEED instruction
Other EFI-Rust Project

- **r-efi**: UEFI Reference Specification Protocol Constants and Definitions
  - [https://github.com/r-util/r-efi](https://github.com/r-util/r-efi)

- **uefi-rs**: Wrapper for writing UEFI applications in RUST.
  - [https://github.com/rust-osdev/uefi-rs](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](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](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](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)
  – Uninitialized 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

• EDKII Security Bug
  – https://edk2-docs.gitbooks.io/security-advisory/content/

• Rust Type Safe Language
  – https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/

• Rust Project
  – https://github.com/libra/libra
  – 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