Improving interoperability between Linux and UEFI using LUV

Megha Dey
Intel Open Source Technology Center
AGENDA

• Introduction to UEFI
• UEFI implementation bug crashing Linux
• Linux EFI subsystem bug affecting system stability
• Introduction to Linux UEFI Validation (LUV) project
• Improve Linux readiness of UEFI systems with LUV
Introduction to UEFI

• UEFI (Unified Extensible Firmware Interface) defines a software interface between an OS and platform firmware.

• It offers various enhancements in terms of boot speed, performance security and modularity.

• UEFI forum members – Biggies like AMD, Apple, Dell, IBM, Microsoft Intel Cavium and Linaro itself! (over 300!!!). ARM Limited is part of the board of directors.
UEFI Internals
UEFI Services

- Implemented as API’s – has code and data regions in memory.
- UEFI offers two types of services:
  1. **Boot time services:** API’s available to OS loader before ExitBootServices(). These services assist the UEFI OS loader in preparing to boot the operating system.
  2. **Runtime services:** API’s available to OS loader before and after ExitBootServices(). These services provide OS with a mechanism to communicate with the firmware.
Boot Process – Before ExitBootServices

- Firmware controls system memory
- Memory accesses in physical mode

---

Firmware owns system memory

<table>
<thead>
<tr>
<th>Memory as seen by firmware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Code</td>
</tr>
<tr>
<td>Conventional Memory</td>
</tr>
<tr>
<td>ACPI Memory NVS</td>
</tr>
<tr>
<td>Boot Data</td>
</tr>
<tr>
<td>Loader Data</td>
</tr>
<tr>
<td>Conventional Memory</td>
</tr>
<tr>
<td>Loader Code</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>Runtime Data</td>
</tr>
<tr>
<td>Runtime Code</td>
</tr>
<tr>
<td>Boot Code</td>
</tr>
</tbody>
</table>
Boot Process – After ExitBootServices

- Kernel controls system memory
- Physical addressing mode
- Before calling SetVirtualAddressMap()

Memory as seen by kernel:

- Boot Code: 0x7ff70000
- Conventional Memory: 0x7ed74000
- ACPI Memory NVS: 0x7e48d000
- Boot Data: 0x7e3ba000
- Loader Data: 0x5c920000
- Conventional Memory: 0x0544e000
- Loader Code: 0x01300000
- Reserved: 0x00800000
- Runtime Data: 0x00100000
- Runtime Code: 0x00001000
- Boot Code: 0x00000000
# Boot Process – Virtual Address space

## Kernel Virtual Address space

<table>
<thead>
<tr>
<th>Module</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Modules</td>
<td>0xffffffff000000</td>
</tr>
<tr>
<td>Modules</td>
<td>0xfffffffffa000000</td>
</tr>
<tr>
<td>High Kernel Mapping</td>
<td>0xffffffff80000000</td>
</tr>
<tr>
<td>EFI Runtime Services</td>
<td>0xffffffffef000000</td>
</tr>
<tr>
<td>ESPfix Area</td>
<td>0xffffffff8100000000</td>
</tr>
<tr>
<td>vmemmap</td>
<td>0xffffffff81800000000</td>
</tr>
<tr>
<td>vmalloc() area</td>
<td>0xffffffff80800000000</td>
</tr>
<tr>
<td>Low Kernel mapping</td>
<td>0xffffffff80000000000</td>
</tr>
<tr>
<td>Kernel space</td>
<td>0x0000000000000000</td>
</tr>
</tbody>
</table>

### Memory as seen by firmware

- **Boot Code**
  - 0x7ff70000
  - 0x7ed74000
- **Conventional Memory**
  - 0x7e48d000
  - 0x7e3ba000
- **ACPI Memory NVS**
  - 0x5c920000
- **Boot Data**
  - 0x0544e000
  - 0x01300000
- **Loader Data**
  - 0x00800000
  - 0x00100000
- **Runtime Data**
  - 0x00001000
- **Runtime Code**
  - 0x00000000
- **EFI Runtime Code**
  - 0x00000000
- **EFI Runtime Data**
  - 0x00000000

---

Linaro Connect 2018
Illegal accesses by firmware

Kernel Virtual Address space

<table>
<thead>
<tr>
<th>Module</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Modules</td>
<td>0xfffffffff000000</td>
</tr>
<tr>
<td>Modules</td>
<td>0xfffffffffa000000</td>
</tr>
<tr>
<td>High Kernel Mapping</td>
<td>0xffffffff80000000</td>
</tr>
<tr>
<td>EFI Runtime Services</td>
<td>0xfffffffffe000000</td>
</tr>
<tr>
<td>ESPfix Area</td>
<td>0xffffffff00000000</td>
</tr>
<tr>
<td>vmemmap</td>
<td>0xffffffff818000000000</td>
</tr>
<tr>
<td>vmalloc() area</td>
<td>0xffffffff810000000000</td>
</tr>
<tr>
<td>Low Kernel mapping</td>
<td>0xffffffff808000000000</td>
</tr>
<tr>
<td>Kernel space</td>
<td>0xffffffff800000000000</td>
</tr>
<tr>
<td>User space</td>
<td>0x0000000000000000</td>
</tr>
</tbody>
</table>

Memory as seen by firmware

- **Boot Code**: 0x7ff0000
- **Conventional Memory**: 0x7ed74000
- **ACPI Memory NVS**: 0x7e48d000
- **Boot Data**: 0x7e3ba000
- **Loader Data**: 0x5c920000
- **Conventional Memory**: 0x0544e000
- **Loader Code**: 0x01300000
- **Reserved**: 0x00800000
- **Runtime Data**: 0x00100000
- **Runtime Code**: 0x00010000
- **Boot Code**: 0x00000000

Linaro Connect 2018
Illegal access by EFI Runtime Service

EFI Runtime Service that caused illegal access

Illegally accessed address

Unmapped region

EFI Runtime Service referencing EFI Conventional Memory
Faulty UEFI implementation bricks systems

PC enthusiast reports “Booting Linux bricks specific Samsung laptops”

Kernel uses EFI backed pstore

Driver triggers H/W error

BIOS code (SPI or Flash)

EFI variables overwrites BIOS

Mathew Garett found astonishing facts

Kernel hacker starts investigation

Buggy firmware

Reference: https://mjg59.dreamwidth.org/22855.html
Kernel bugs in EFI subsystem causing panic

Kernel panics on BGRT enabled platform

It’s impossible to test for all cases

Enable all possible tables/features in OVMF and use Qemu + OVMF to find bugs

No worries! We have a solution

Reference: https://lkml.org/lkml/2015/12/10/599
BGRT - Boot Graphics Resource Table
OVMF - Opensource Virtual Machine Firmware
Buggy firmware causes issues

ESRT parsed successfully

Failed to parse ESRT

Same kernel booted on two different machines

Enable all possible tables/features in OVMF and use Qemu + OVMF to find bugs

QA Engineer

LUV team

No worries! We have a solution

It’s tough to catch these errors

ESRT - EFI System Resource Table
UEFI implementation is critical

Things can go wrong during interactions between firmware and Linux kernel

Faulty implementation of generic UEFI hurts Linux

Linux UEFI Validation (LUV) project was started
Why LUV?

- Fragmented validation strategy
- Linux workarounds, firmware not fixed
- No test-suite tests FW-OS interactions
Current Validation Strategy

Pre-boot

Boot

Run-time

Test gap

UEFI SCT/PI

UEFI Boot Services

UEFI Runtime Services

Userspace

Linux Kernel

Execution time

FWTS
LUV: A unified framework

- Continuous Integration framework of upstream open source projects:
  - Linux kernel
  - Yocto Project
  - Open source test-suites

LUV is a Linux distro
LUV: Covering the entire spectrum (*x86 systems)
Some more details:

**CHIPSEC**
- X86 specific platform security assessment framework
- Find firmware Vulnerabilities
- Detect firmware implants
- Explore low level system assets
- ARM should also start a security oriented test suite

**FWTS**
- Endorsed by the UEFI forum as an ACPI test
- Critical for OS power management
LUV: Detect bugs early

At boot time:

LUV uses hardened kernel to work around these issues (during boot time and run time)

Kernel fixing illegal access by firmware when EFI Runtime Service is invoked

LUV uses hardened kernel to work around these issues (during boot time and run time)
LUV: Easy to use

Plug and play

Diskless boot

Boot on Virtual Machine
LUV: Easy to add test-suites

LUV is open source!
LUV: Increase collaboration

- LUV is an open source project!
- Generic architecture-agnostic framework to which architectural specific test-suites can be added (e.g., chipsec, FWTS)
- Already have contributions from Linaro! Expect to see more!
- LUV is used as part of the **ARM specific ARM Enterprise ACS**
- ACS uses the flexibility of LUV by using the EFI stub instead of grub!
LUV: Available features

- Results are saved for viewing
  1. If (USB Stick) - Saved in LUV_RESULTS folder
  2. If (Net Boot) - Saved to HTTP server
  3. If (VM) - Saved in the results partition of luv.img
- Debug via network using Netconsole.
- Telemetrics mechanism to send error report to the LUV server incase of crash.
- Lists all available DSMs on a platform.
- *Future feature: Add support for more bootloaders.
LUV: Path to better firmware

Collaborate
Consolidates various open source test-suites

Comprehensive
Test the entire software stack

Quality
Detect firmware/kernel bugs early

Community
Easy to use
Easy to contribute
What do you need?
Tell us your needs to better validate your firmware!

Join US
https://01.org/linux-uefi-validation

Credits
Ricardo Neri
Matt Fleming
Gayatri Kammela
Naresh Bhat
And many more contributors…
Thank you.

Any Questions?