LinuxBoot

Let Linux do it

Daniel Maslowski
Agenda

- Motivation
- LinuxBoot Concept
- UEFI Integration
- Implementations
- Future Work
Motivation
Firmware now vs back then

- 1999: birth of coreboot as LinuxBIOS
  - open source x86 firmware
- 2004: Tiano initial release by Intel
  - now EDK I/II, maintained by UEFI community
- 2014: Intel Haswell release
  - requires proprietary MRC (Memory Reference Code) binary
  - later on: FSP (Firmware Support Package)
- 2014: AMD Generic Encapsulated Software Architecture (AGESA) lockdown
  - binary only since then
  - was initially open sourced for coreboot in early 2011
  - an open laptop would have been nice
- 2019: UDF (UEFI Dumpster Fire™)
  - criticized by many people
  - for many years

neglected: Intel ME, AMD PSP, ARM and other SoCs
State of security

- update processes are often insecure
- vendors and firmware projects take no responsibility
- great summary by Alex Matrosov
Right to repair bill

- vendors still propose security by obscurity
  - although known to be pointless against sophisticated attackers
- repair technicians suffer from proprietary information
  - consumers and researchers alike
Platform Initialization (PI)

Platform Initialization Firmware Phases

EDK II - stages

Time

coreboot - stages

coreboot source languages

SMM (x86 only)

BL31 (ARM only)

payload

ramstage

postcar (x86 only)

romstage

verstage (optional)

bootblock

EDK II - stages

Security (SEC)

Pre-EFI Initialization Environment (PEI)

Driver Execution Environment (Dxe)

Boot Device Selection (BDS)

Power on

code/heap memory location

Cache-As-RAM

DRAM

Assembly

C

ADA SPARK (x86 only)

basic platform initialization: CPU, chipset, RAM (PEI/romstage)

▶ has to be rerun similarly for S3 resume
LinuxBoot Concept
LinuxBoot

- Linux kernel + initramfs in SPI flash
- can run on top of
  - coreboot: as payload
  - U-Boot
  - vendor UEFI firmware: remove DXEs, build Linux with EFI support

=> approach rather than implementation
Integrations

SPI Flash

UEFI PEI

coreboot romstage

u-boot SPL

Firmware HW init

Memory initialized

LinuxBoot

Linux Kernel

initramfs

Operating System
Constraints

- only few megabytes of space (8 to 16 common)
- build minimum kernel
  - disk drivers
  - filesystems
  - possibly networking
- build basic initramfs
  - core utilities like `ls`, `cat`, etc
  - bootloader(s) - need to boot an OS ;)

=> very similar to OpenWrt, except for bootloader instead of routing tools
UEFI Integration
**UEFI binary format**

PE32 / PE32+ format, without symbol tables

Three types:

- applications
  - OS loaders
  - utilities
- boot service drivers
  - disk drivers
  - network drivers
- runtime drivers
  - may remain loaded while OS is running

=> replace applications and boot service drivers with LinuxBoot
Tools

- Fiano
- utk with DXE cleaner
- UEFI Tool
Implementations
u-root

- initramfs tool written in Go
- utilities like busybox (ls, cat, …)
- offers bootloaders (SystemBoot)
Try out u-root in QEMU

go get github.com/u-root/u-root
# build an initramfs
GOOS=linux \\
~/go/bin/u-root -build=bb -o /tmp/initramfs.linux_amd64.cpio
# get a kernel
MIRROR="http://mirror.rackspace.com" REL="2019.10.01" \\
wget "$MIRROR/archlinux/iso/$REL/arch/boot/x86_64/vmlinuz"
# run it :)
qemu-system-x86_64 -kernel vmlinuz \\
  -initrd /tmp/initramfs.linux_amd64.cpio
u-root demo
Heads

- authenticated / measured boot
u-bmc

- u-root for BMCS
- alternative to OpenBMC

<table>
<thead>
<tr>
<th>Project</th>
<th>OpenBMC</th>
<th>u-bmc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td>C++, Python</td>
<td>Go</td>
</tr>
<tr>
<td>Tooling</td>
<td>Yocto, OpenEmbedded</td>
<td>u-root</td>
</tr>
<tr>
<td>Kernel</td>
<td>OpenBMC Linux fork</td>
<td>OpenBMC Linux fork</td>
</tr>
<tr>
<td>Init</td>
<td>systemd</td>
<td></td>
</tr>
<tr>
<td>IPC</td>
<td>D-Bus</td>
<td></td>
</tr>
<tr>
<td>RPC</td>
<td>IPMI, REST</td>
<td>gRPC</td>
</tr>
<tr>
<td>Metrics</td>
<td></td>
<td>OpenMetrics</td>
</tr>
</tbody>
</table>
Future Work
CHIPSEC blacklist in MFT

- UEFI Forum openly discussed security measures for firmware development and answered questions from participants

  Q: Can consumers audit the firmware? If so, how?

  A: There are a variety of tools that can allow a consumer to inspect firmware images. CHIPSEC and UEFI Tool are two tools that can analyze a firmware image and allow a consumer to inspect its contents. CHIPSEC has a blacklist of UEFI modules which include a tool that will check a ROM image for blacklisted modules.

- Mimoja released the MimojaFirmwareToolkit (MFT)
  https://firmware.doctor

  - integrate CHIPSEC blacklist in analysis?
  - contributions are welcome ;}
ACME for firmware update PKI

Since firmware updates are such an issue:

▶ we had a very similar issue on the web with secure communication
▶ leverage the ACME protocol (Let’s Encrypt) also for firmware?
▶ create issues on TianoCore GitHub org for discussion
Questions?
Thanks! :)