OPEN POSSIBILITIES.

OCP Server Delta Lake OSF How-tos and Features
OCP Server Delta Lake OSF How-tos and Features

Johnny Lin, Assistant Technical Manager, Wiwynn
Christian Walter, Managing Director Firmware, 9elements
Wiwynn’s OSF Journey

2019
OSF on Mono Lake
(Intel Broadwell-DE)
IPMI, SMBIOS, ACPI and board configurations

2020
OSF on Tioga Pass (Intel Skylake-SP) and Delta Lake (Cooperlake-SP)

2021
OSF on Delta Lake
OCP OSF Approved, Pre-production readiness

2022
OSF on Sapphire Rapids-SP
CXL memory device support, Target for Production readiness

OPEN POSSIBILITIES.
OCP OSF Checklist

- https://www.opencompute.org/wiki/Open_System_Firmware/Checklist
- Delta Lake OSF Checklist for reference

1.2. Required Artifacts

a. Firmware image
b. Description of the firmware ownership model
c. Top-level build script
d. Documentation
e. Test results
f. Tool for user modification
OCP OSF Checklist (Cont.)

• 3. Ownership and Reusability
  a) Customer could choose whether to fuse or not, and how to fuse depending on the customer’s threat model.
  b) Customer could choose to use different profiles of Intel CBnT, or to use coreboot’s verified boot.
  c) If Delta Lake was fused with other owner’s OEM key, transfer of ownership is not possible. The next owner can use it without Intel CBnT.

• 6. Test Regime: cold and warm reboot to an OSI-approved OS over 100 times.
How To Download and Build

- https://github.com/opencomputeproject/OpenSystemFirmware/tree/master/Wiwynn/deltalake
- How to build: cd Wiwynn/deltalake && ./download_and_build.sh
  - Download Intel public blobs: FSP, PCH IGN, microcode
  - Download toolchain, coreboot and LinuxBoot codebase via osf-builder
  - make
OSF at Facebook - coreboot/LinuxBoot

Boot ROM

- coreboot
- FSP

Silicon init done by coreboot with FSP (UEFI PI)

Boot kernel + initramfs

LinuxBoot

Target kernel loaded via network/cloud

Target kernel loaded via local or removable storage

OPEN POSSIBILITIES.
osf-builder

- [https://github.com/linuxboot/osf-builder](https://github.com/linuxboot/osf-builder)
- *getdeps*: tool that can fetch OSF components described by JSON file, supported components are coreboot, kernel and initramfs
- Example of **config-qemu-x86_64.json**

```json
"coreboot": {
  "git": [
    {
      "label": "coreboot",
      "url": "https://review.coreboot.org/coreboot",
      "branch": "master",
      "hash": "7014f8258e6e015fe91d692..."
    }
  ],
  "files": {
    "label": "crossgcc_tarballs",
    "dest": "util/crossgcc/tarballs",
    "filelist": [
      {
        "url": "https://ftpmirror.gnu.org/gmp/gmp-6.2.0.tar.xz",
        "hash": "sha256:258e6cd51b3fbdfc185c..."
      }
    ]
  }
}
```

- It provides an example for qemu, please try it out!
Firmware Configurations

- Command line interface (u-root shell)
- **VPD**: BIOS flash region for storing key-value pairs
  - vpd: Linux userspace utility for decoding the key-value
  - RO_VPD: region holds default values
  - RW_VPD: region holds customized values
- **flashrom**: Linux userspace utility for BIOS flash read/write

- CMOS clear: Re-format RW_VPD
  - u-root commit:
    - Execute flashrom and vpd binaries to format RW_VPD region
Firmware Configurations -- Examples

- Boot order: `systembooter`
  - Boot0000='{"type":"netboot","method":"dhcpv6"}’
  - Boot0001='{"type":"localboot","method":"grub"}’
- FSP log enable: `fsp_log_enable=0` or `1`
- `vpdbootmanager`: u-root command line utility for managing VPD variables (should be renamed to `vpdmgr` in the future)
  - `vpdbootmanager get [variable name]`
  - `vpdbootmanager set [variable name] [variable value]`
  - `vpdbootmanager delete [variable name]`

OPEN POSSIBILITIES.
VPDBootManager

- Read is done via /sys/firmware/vpd kernel interface
- Write is done via executing vpd and flashrom binaries

```bash
~/$ vpdbootmanager get fsp_log_enable
fsp_log_enable(RO) => 0
```

```bash
~/$ vpdbootmanager set fsp_log_enable 1
flashrom v1.2-97-gf33f1a1 On Linux 5.2.9 (x86_64)
flashrom is free software, get the source code at https://flashrom.org
Calibrating delay loop... OK.
coreboot table found at 0x75292000.
Found chipset "Intel C620 Series Chipset (QS/PRQ)".
Enabling flash write... SPI Configuration is locked down.
FREG2: Management Engine region (0x00001000-0x02ffffff) is locked.
Not all flash regions are freely accessible by flashrom. This is most
due to an active ME. Please see https://flashrom.org/ME for details.
OK.
Found Programmer flash chip "Opaque flash chip" (65536 kB, Programmer-
Using region: "RW_VPD".
Reading flash... done.
Successfully set, it will take effect after reboot
```
Delta Lake OSF Features

- **System tables**: Most of the SMBIOS, ACPI are implemented
- **BMC Integration**: IPMI handshaking, SEL record generation, ipmidump for u-root shell utility
- **Devices**: PCIe data drives, NIC card
- **OS booting methods**: SSD local drive, IPv4/IPv6 network boot
- **Security feature**: Intel Converged Bootguard and TXT (CBnT)
- **RAS features**: ACPI EINJ and HEST, CPU, memory and PCIe error handling
- **Performance tests**: coremark, FIO, Iperf, Linpack, Intel MLC, SpecCPU, stream
Intel CBnT on the OCP Deltalake

- Full Open-Source Implementation of Intel CBnT in coreboot
- Full Open-Source Tooling to provision and validate Intel CBnT with coreboot and UEFI
- Closed-Source ACM’s from Intel needed (under CNDA)
- Profile 0T has been tested production ready
CBnT - What and Why?

- Intel Converged BootGuard and Trusted Execution Technology
- provides a Static and a Dynamic Root of Trust (SRTM/DRTM)
- The purpose of BtG is to verify that the initial BIOS startup code is “good”
- The primary objective of TXT is [...]enabling platform boot into a secure measured launch environment (MLE).
Recap Intel CBnT

- Converged BootGuard and TXT
- Moved the Trust Anchor from the TPM into the ME Fuses
- One set of ACMs for TXT and BtG
- Moved most configuration into binary structures (manifests)
  - Key Manifest
  - Boot Policy Manifest
  - FIT Table Entries
- ME Configuration
Recap Intel CBNt

Intel ME

Hash of KM
Signing Key

Key Manifest

Hash of BPM
Signing Key

Signature

Boot Policy Manifest

IBB Hash

Signature

IBB

OPEN SYSTEM FIRMWARE
coreboot Integration

- src/security/intel/{txt, cbnt}
  - Leverage most of the TXT code
  - CBnT code mostly for logging
- Integrated and Configurable via coreboot build system
- Converged Security Suite as dependency
- Build System includes
  - Generating KM/BPM
  - Signing KM/BPM
  - KM/BPM Adjustments
  - Injecting FIT Entries
Timeline - Integration

• Started in September 2020
• PoC booting into tboot in November 2020
• Tooling to generate & sign
  ○ Key Manifest
  ○ Boot Policy Manifest
• Zero → PoC in ~2 months
• Fully integrated into build system ~6 months
  ○ Build Systems generate and signs all artifacts
  ○ IBB automatically included
  ○ FSP-T / FMAP / TPM Init / … Fixes
Demo

- Can be found at OCP Demo lab
  → OCP Delta Lake
  → Open System Firmware

- Come by and check it out!
Call to Action

- Contribute to coreboot and LinuxBoot open source projects
- Please contact Wiwynn for purchasing Delta Lake
- Please contact 9elements Cyber Security for Delta Lake OSF feature development and professional support

OCP OSF Delta Lake: https://github.com/opencomputeproject/OpenSystemFirmware/tree/master/Wiwynn/deltalake

Where to buy: https://www.wiwynn.com/contact-wiwynn/

9elements Cyber Security: https://9esec.io/contact

coreboot: https://coreboot.org/

LinuxBoot: https://www.linuxboot.org/

Thank you!