

Enabling TPM 2.0 on coreboot based devices




European Coreboot Conference 2017

Piotr Król and Kamil Wcisło








Piotr Król

-  @pietrushnic
-  piotr.krol@3mdeb.com
-  [linkedin.com/in/krolpiotr](https://www.linkedin.com/in/krolpiotr)
-  [facebook.com/piotr.krol.756859](https://www.facebook.com/piotr.krol.756859)



Kamil Wcisło

-  @mek_xgt
-  kamil.wcislo@3mdeb.com
-  [linkedin.com/in/kamil-wcislo-86a83189](https://www.linkedin.com/in/kamil-wcislo-86a83189)
-  [facebook.com/mek.xgt](https://www.facebook.com/mek.xgt)

Why?

- security is important
- in times of IoT devices have to be hardened against attacks
- devices have to have some kind of proof they run good software

Why?

- security is important
- in times of IoT devices have to be hardened against attacks
- devices have to have some kind of proof they run good software

What we want to show?

- some of our struggles and conclusions when trying to enable secure boot in open firmware device

Why?

- security is important
- in times of IoT devices have to be hardened against attacks
- devices have to have some kind of proof they run good software

What we want to show?

- some of our struggles and conclusions when trying to enable secure boot in open firmware device

Some comments...

- this is still WIP, there are many things to be done
- we're just the beginners in this area

- Security concepts overview
- Introduction to TPM 2.0
- TPM hardware
- State of support in UEFI and SeaBIOS
- State of Linux kernel and user space tools
- Verified boot in coreboot
- Conclusions

The CIA triad

- Confidentiality
- Integrity
- Authorization

The CIA triad

- Confidentiality
- Integrity
- Authorization

and...

- Availability
- Non-repudation

- cryptochip for boot verification, platform verification, disk encryption, etc.
- slow by design
- contains cryptographic functions, hashing algorithms, key generation and protection mechanisms, RNG, integrity measurement mechanisms
- talks with the host platform using various interfaces (LPC, I2C, SPI)
- PCRs
 - shielded locations
 - modifiable only by using Extend function
 - resettable only by rebooting system
- has secure storage
- it's pasive!
- has possibility to restrict access to certain keys, if measurements don't match

secure boot vs measured boot

secure boot

- uses cryptographic functions
- secure boot prevents booting other fw
- needs some kind of root of trust
- asserts the boot

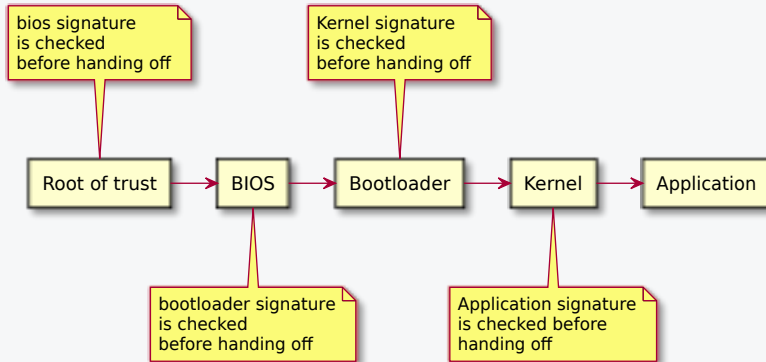
secure boot vs measured boot

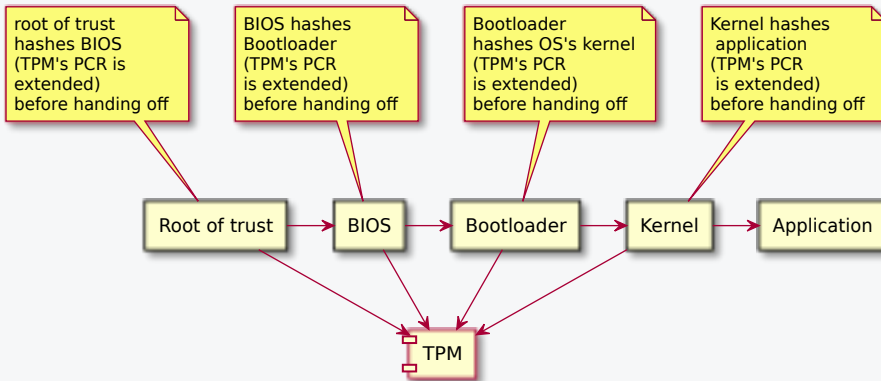
secure boot

- uses cryptographic functions
- secure boot prevents booting other fw
- needs some kind of root of trust
- asserts the boot

measured (trusted) boot

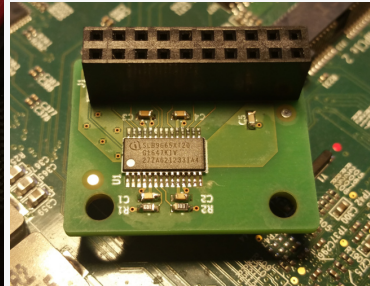
- uses hashes (cryptographic hashes)
- measured boot notifies about compromised fw
- TPM is needed (extend PCRs) to be secure
- proves the boot
- could be used to certify that system is in expected state





- TPM is international standard for a secure microprocessor
- There are many hw implementations
- TPM 2.0 is not backwards compatible
- Differences
 - supported algos
 - private keys vs. primary "seeds" and KDF
 - API

- We're using PC Engines APU2 board as a reference platform
- TPM is Infineon SLB9665 chip with LPC interface



- UEFI was first to have TPM2.0 support
- coreboot supports TPM 2.0 (CONFIG_TPM && CONFIG_TPM2)
 - drivers for devices are in `src/drivers/{pc80,i2c,spi}/tpm`
 - TPM API is in `src/lib/tpm2_t1cl.c`
- SeaBIOS also supports TPM2.0 (CONFIG_TCGBIOS)
 - drivers are in `src/hw/tpm_drivers.c`
 - API (TCGBIOS) is in `/src/tcgbios.c`

- support for TPM 2.0 in kernel added in 4.0 and improved in 4.4
 - tpm_tis kernel module in our case
 - resource manager added in 4.12
- 3 userspace stacks available
 - (Intel) <https://github.com/01org/tpm2-tss.git>
 - (IBM) <https://sourceforge.net/projects/ibmtpm20tss/>
 - (Google) https://chromium.googlesource.com/chromiumos/third_party/tpm2/

After changing the board's Kconfig to select CONFIG_TPM2 and CONFIG_MAINBOARD_HAS_TPM2 board booted, but Linux had problems initializing the module.

In Liunx, we had to insert the `tpm_tis` module manually:

```
$ modprobe tpm_tis interrupts=0 force=1
[ 29.203243] tpm_tis tpm_tis: 2.0 TPM (device-id 0x1A, rev-id 16)
[ 29.237198] tpm tpm0: A TPM error (256) occurred continue selftest
[ 29.243596] tpm tpm0: starting up the TPM manually
```

After this, the device was somewhat usable, but not in the full potential.

`tpm2-tss` and `tpm2-tools` (Intel's TPM2.0 stack) was able to communicate with the device.

It's seems that TPM is not getting initialized correctly in the firmware.

After checking the debug output, it was obvious our TPM chip was detected as the wrong one:

```
...  
TPM initialization.  
TPM: Init  
Found TPM SLB9660 TT 1.2 by Infineon  
TPM: Open  
TPM: Startup  
TPM: command 0x99 returned 0x1e  
TPM: Error code 0x1e.  
...
```

It detected the chip as the TPM1.2 compliant one and the startup sequence returned errors.

TPM2.0 has different startup sequence than 1.2.

TPM2.0 spec is not easily readable, despite having code examples.

In 4.13, Linux has quite good support for TPM 2.0.

Also contains TPM 2.0 starting sequence.

`drivers/char/tpm` is the pretty good source of knowledge now.

Because Linux is trying to fix firmware shortcomings, it's got startup procedure (i.e. "manual startup" used when inserting the tpm module with force). E.g.:

```
int tpm_startup(struct tpm_chip *chip) {
    /*...*/
    if (chip->flags & TPM_CHIP_FLAG_TPM2) {
        rc = tpm_buf_init(&buf, TPM2_ST_NO_SESSIONS, TPM2_CC_STARTUP);
        if (rc < 0)
            return rc;

        tpm_buf_append_u16(&buf, TPM2_SU_CLEAR);
    } else {
        rc = tpm_buf_init(&buf, TPM_TAG_RQU_COMMAND, TPM_ORD_STARTUP);
        if (rc < 0)
            return rc;

        tpm_buf_append_u16(&buf, TPM_ST_CLEAR);
    }
    /*...*/
}
```

- It seems that SLB9665 has the same VID/DID as SLB9660
- Added the conditional to identify the chip as SLB9665, when using TPM2.0
- Added additional conditional to use startup sequence for TPM2.0

Done in `src/drivers/pc80/tpm`. It seems this driver has support only for TPM1.2.

...

TPM initialization.

TPM: Init

Found TPM SLB9665 TT 2.0 by Infineon

TPM: Open

TPM2: Startup

TPM: command 0x144 returned 0x0

TPM: OK.

...

Now Linux inserts the `tpm_tis` module automatically.

```
$ tpm2_dump_capability -T device -c properties-fixed
TPM_PT_FAMILY_INDICATOR:
  as UINT32:      0x08322e3000
  as string:      "2.0"
TPM_PT_LEVEL:      0
TPM_PT_REVISION:    1.16
TPM_PT_DAY_OF_YEAR: 0x000000d1
TPM_PT_YEAR:        0x000007df
TPM_PT_MANUFACTURER: 0x49465800
TPM_PT_VENDOR_STRING_1:
  as UINT32:      0x534c4239
  as string:      "SLB9"
TPM_PT_VENDOR_STRING_2:
  as UINT32:      0x36363500
  as string:      "665"
...

$ tpm2_getrandom -T device 10
0x84 0xCF 0xA4 0xF8 0xEC 0x43 0x11 0xA4 0x7D 0xE8
```


There is TPM2.0 driver in `src/lib/tpm2_t1c1.c`

As far as I understand it, it's used exclusively in vboot, at least for now.

Maybe this could be leveraged to remove redundancy with drivers? (Maybe even that's the plan... ;-))

SeaBIOS also doesn't detect TPM2.0 chip correctly.

After hacking the `tpmhw_probe()` to return TPM2.0 version, we got TPM menu in SeaBIOS's bootmenu. We can clear the TPM there.

I can see that measurements are being done, e.g. functions like:

```
tpm_option_rom  
tpm_add_bcv  
tpm_add_cdrom
```

are used. They contain code to hash the target and extend the TPM.

- vboot
 - concepts
 - firmware image
 - usage in Chromebooks
- coreboot
- SeaBIOS/depthcharge

1. starts in RO part, where bootblock exists
2. verstage is where vboot is
3. control from bootblock is passed to verstage
4. vboot verifiw rw section of firmware
5. if this fails, falls back to ro firmware and boots recovery

We wanted to have the verified booting capabilities.

It wasn't so trivial to implement.

vboot uses different flash layout to work:

- several CBFS regions: (RO) COREBOOT, FW_MAIN_A, FW_MAIN_B, LEGACY

Our platform uses hardware blobs.

- AGESA
- amdfw (PSP)

1. First we need to create FMD file (FMAP source)
 - need to place blobs in correct places
 - need to correctly place RO section (bootblock has to be at the end of flash)
 - takes a bit of hacking
2. Second need to add some boilerplate function to mainboard dir (e.g. write protect detection routine - `get_write_protect_state(void)`). vboot requires them in order to detect that RO area is protected.
3. Third we have to enable vboot, `CONFIG_COLLECT_TIMESTAMPS`. We have to point `CONFIG_FMDFILE` to our flash layout file.
4. Some hacking with blobs adding was required.

```
FLASH@0xff800000 0x800000 {
  SI_BIOS@0x0 0x800000 {
    RO_APUFW@0x20000 0x80000

    RW_SECTION_A@0x100000 0xf0000 {
      VBLOCK_A@0x0 0x10000
      FW_MAIN_A(CBFS)@0x10000 0xdffc0
      RW_FWID_A@0xeffc0 0x40
    }

    RW_SECTION_B@0x1f0000 0xf0000 {
      VBLOCK_B@0x0 0x10000
      FW_MAIN_B(CBFS)@0x10000 0xdffc0
      RW_FWID_B@0xeffc0 0x40
    }

    RW_LEGACY(CBFS)@0x300000 0x200000

    WP_RO@0x570000 0x290000 {
      RO_VPD@0x0 0x4000
      RO_UNUSED3@0x4000 0xc000
      RO_SECTION@0x10000 0x280000 {
        FMAP@0x0 0x800
        RO_FRID@0x800 0x40
        RO_FRID_PAD@0x840 0x7c0
        GBB@0x1000 0x7f000
        RO_AGESA@0x80000 0x90000
        COREBOOT(CBFS)@0x110000 0x170000
      }
    }
  }
}
```

```
...
Found TPM SLB9665 TT 2.0 by Infineon
setup_tpm():512: TPM: SetupTPM() succeeded
...
FMAP: area GBB found @ 581000 (520192 bytes)
VB2:vb2_report_dev_firmware() This is developer signed firmware
FMAP: area VBLOCK_A found @ 100000 (65536 bytes)
FMAP: area VBLOCK_A found @ 100000 (65536 bytes)
VB2:vb2_verify_keyblock() Checking key block signature...
FMAP: area VBLOCK_A found @ 100000 (65536 bytes)
FMAP: area VBLOCK_A found @ 100000 (65536 bytes)
VB2:vb2_verify_fw_preamble() Verifying preamble.
Phase 4
FMAP: area FW_MAIN_A found @ 110000 (917440 bytes)
...
Slot A is selected
creating vboot_handoff structure
Copying FW preamble
CBFS: 'VB00T' located CBFS at [110000:12f6c0)
CBFS: Locating 'fallback/ramstage'
...
```


- enable verified boot capabilities in coreboot, SeaBIOS
- TrustedGRUB is there
- investigate the usage of vboot
- prove it works
- test, test, test...
- Is it possible to harden it even more? (what about the PSP?)

- Adding new chip in coreboot is quite easy. TPM support is there.
- Using the source brings better results than using the huge spec.
- Understanding the platform boot process and blob placement is crucial, when trying to enable vboot.
- One can have verified boot capabilities on custom coreboot platform.

- recent news - vulnerable RSA key generated by Infineon's hardware
- can decode RSA private key from public key
- if the RSA primes are generated as truly random numbers attack is not viable
- ECC keys are not affected
- key test tools available: https://crocs.fi.muni.cz/public/papers/rsa_ccs17

- Linux and coreboot (obvious ;) source code
- <https://danielmiessler.com/study/infosecconcepts/>
- <https://forums.juniper.net/t5/Security-Now/What-s-the-Difference-between-Secure-Boot-and-Measured-Boot/ba-p/281251>
- <https://blog.hansenpartnership.com/tpm2-and-linux/>
- <https://firmwaresecurity.com/2016/01/15/seabios-gets-tpm2-security/>
- <https://www.coreboot.org/git-docs/Intel/vboot.html>
- [https://www.coreboot.org/images/f/f1/Tpm - Philipp.pdf](https://www.coreboot.org/images/f/f1/Tpm_-_Philipp.pdf)
- [https://elinux.org/images/6/6e/ELC2017 TPM2-and-TSS Tricca.pdf](https://elinux.org/images/6/6e/ELC2017_TPM2-and-TSS_Tricca.pdf)

Thanks!

Thanks!

Q/A