

Start trusting Your BIOS

SRTM with vboot, TPM and permanent flash protection





Open Source Firmware Conference 2019

Michał Żygowski





Michał Żygowski
Firmware Engineer

-  @_miczyg_
-  michal.zygowski@3mdeb.com
-  [linkedin.com/in/miczyg](https://www.linkedin.com/in/miczyg)
-  facebook.com/miczyg1395
- PC Engines platforms maintainer
- interested in:
 - advanced hardware and firmware features
 - coreboot
 - security solutions

- SRTM
 - SRTM goals
 - SRTM in coreboot
 - Flashmap firmware layout
 - vboot
 - TPM driver
 - SPI flash lock
- Other fancy stuff
 - Persistent bootorder
 - Recovery method
 - VPD runtime configuration
- DEMO
- Future improvements
- Q&A

- **Static Root of Trust for Measurement (SRTM):**
 - root of trust within a static, immutable piece of code
 - can be achieved with SPI lock mechanisms
 - **Hardware Protected Mode** - with WP pin tied to ground
 - **One Time Programmable Mode** - permanently locked (only certain chips)
 - can be achieved with silicon vendor solutions
 - **AMD HVB** - Hardware Validated Boot based on PSP bootrom
 - **Intel Secure Boot** - based on Intel ME/TXE bootrom (older silicon generations)
 - **Intel Boot Guard** - based on Intel ME bootrom, improved Secure Boot for newer silicon generations
- **Core Root of Trust for Measurement (CRTM):** self-measurement of the immutable code responsible for all components verification

First of all why?

- unauthorized changes detection in firmware and OS
- firmware attestation with the help of measurements

Advantages:

- measurements divided for firmware use, OS use; can distinguish which component has been maliciously modified
- measurements can be used to seal/unseal secrets with the TPM
- unsigned and unverified component very unlikely to be executed in firmware
- provides attestation method of the code run on the processor in contrary to conventional boot method without verification and/or measurements

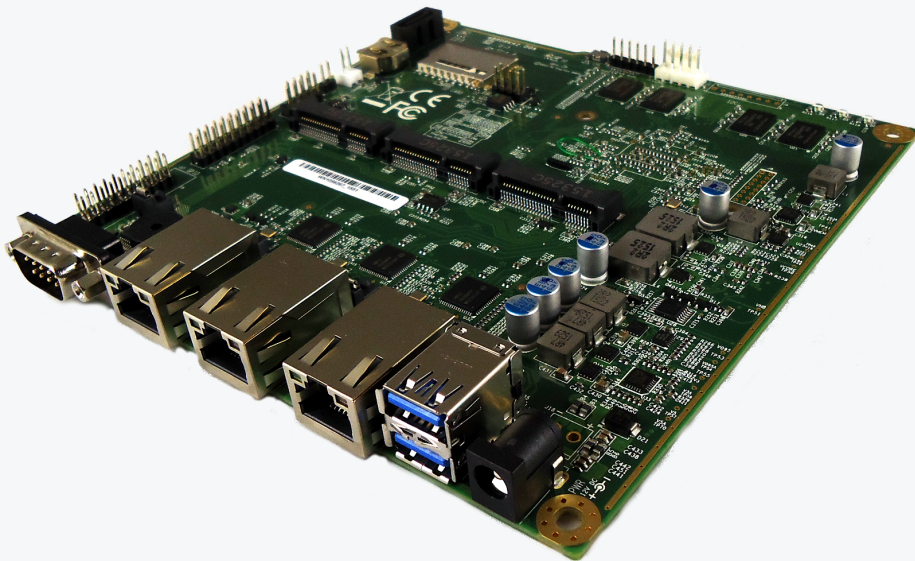
Verified and measured boot mode enabled since the end of Feb 2019 (credits to Zaolin). SRTM in coreboot comprises:

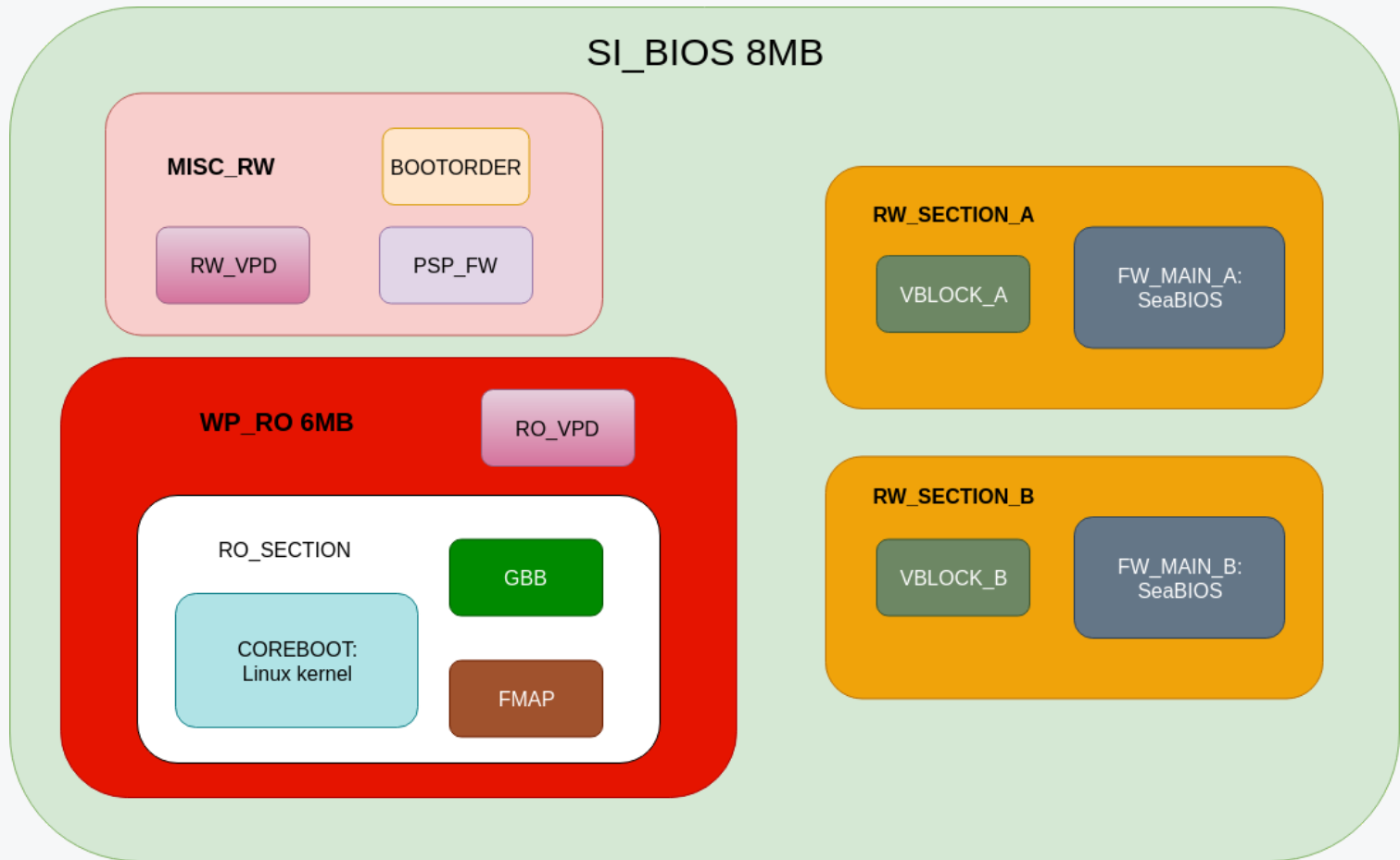
- Vboot - Google's reference verified boot implementation adapted to coreboot
- TPM driver - API to interface TPM for PCR extend operation, TPM startup, etc.
- Flashmap - firmware layout description format allowing to have multiple CBFSeS
- SPI flash protection

And everything is open-source!

Example implementation:

PC Engines APU2 with Infineon SLB9665 TPM2.0 and Adesto AT25SF641 SPI chip.



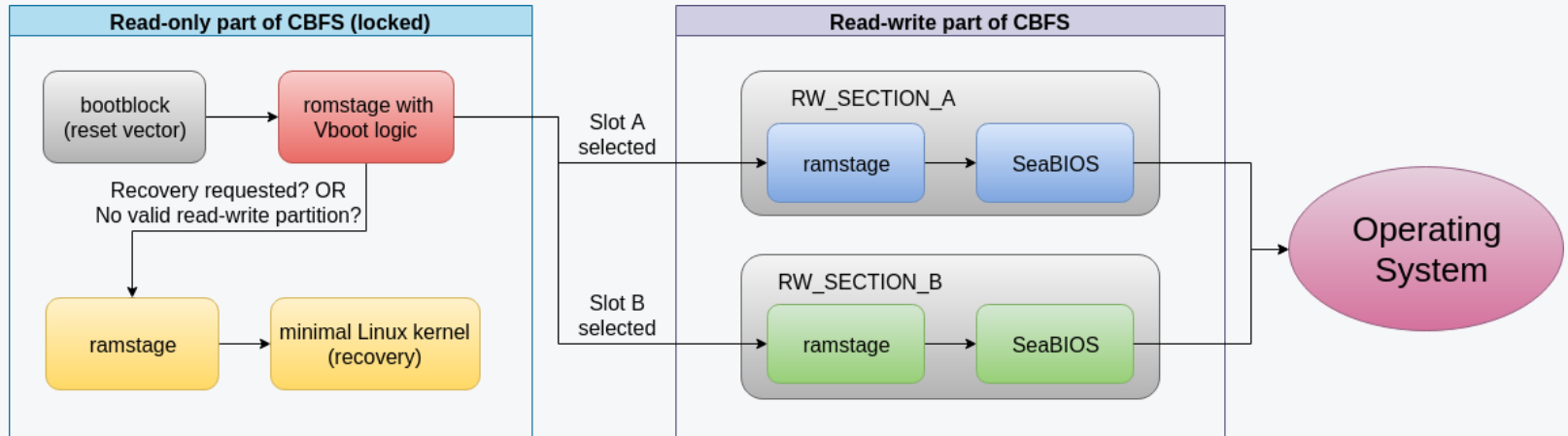


Google's reference verified boot implementation.

- the public part of root key in GBB (Google Binary Block)
- firmware signing keys in VBLOCKS in RW firmware
- two updatable firmware partitions for an easy upgrade with failure protection
- GBB and recovery CBFS in the read-only write-protected partition
- all crypto algorithms in place

Differences:

- Roof of Trust protected by a screw grounding a WP pin on Chromebooks
- Recovery requiring external USB drive with signed ChromeOS recovery image
- permanent SPI flash lock
- minimal Linux kernel in read-only CBFS for firmware recovery



1. Execution starts with the reset vector (bootblock).
2. Romstage:
 - RAM initialization
 - TPM and Vboot initialization
 - CRTM logic self-measurement of GBB, bootblock and romstage
 - firmware partition verification and selection
3. If verification passed: normal boot path to read-write partition A/B and OS
4. If verification failed: boot to recovery kernel

TPM driver is located in `src/security/tpm` and covers both TPM 1.2 and 2.0. Responsible for whole low-level access to TPM, basic commands, etc.

Integration with Vboot:

- `src/security/vboot/vboot_crtm.c:vboot_init_crtm` - does the self-measurement of the early firmware components
- `src/security/vboot/vboot_crtm.c:vboot_measure_cbfs_hook` - a hook for CBFS locator, measures the located component before use

The whole logic is invoked during program location of either ramstage or romstage (in C environment bootblock enabled solutions).

Winbond W25Q64FV - default SPI flash chip on PC Engines APU2, 5 status registers protection modes.

SRP1	SRP0	/WP	Status Register	Description
0	0	X	Software Protection	/WP pin has no control. The Status register can be written to after a Write Enable instruction, WEL=1. [Factory Default]
0	1	0	Hardware Protected	When /WP pin is low the Status Register locked and can not be written to.
0	1	1	Hardware Unprotected	When /WP pin is high the Status register is unlocked and can be written to after a Write Enable instruction, WEL=1.
1	0	X	Power Supply Lock-Down	Status Register is protected and can not be written to again until the next power-down, power-up cycle. ⁽¹⁾
1	1	X	One Time Program ⁽²⁾	Status Register is permanently protected and can not be written to.

<https://www.winbond.com/resource-files/w25q64fv%20revq%2006142016.pdf>

One Time Programmable? In God's name why? And what's this (2)???

According to the same datasheet:

Note:

1. When $SRP1, SRP0 = (1, 0)$, a power-down, power-up cycle will change $SRP1, SRP0$ to $(0, 0)$ s
2. This feature is available upon special order. Please contact Winbond for details.

available upon special order - ?!

Any attempts to enable it failed. But fortunately...

Found another SPI flash chip, **Adesto AT25SF641** with the same features and command set as Winbond. Moreover, no side notes that something is unavailable¹!

This solution may not fit everybody since it makes a part of/whole SPI flash unwritable. This option has been used for special customer needs to satisfy their requirements in terms of security.

1) AT25SF641 datasheet: https://www.adestotech.com/wp-content/uploads/AT25SF641_111.pdf

Flashmap, Vboot and TPM open many more paths to valorize the firmware.

- BOOTORDER region
 - SeaBIOS in RW firmware slots, old CBFS bootorder file may not work
 - made SeaBIOS interpret the FMAP and tell to retrieve bootorder from the region
 - bootorder is now persistent across updates!
- Recovery method
 - about 4 MiB of space left in the read-only region
 - Linux payload put inside COREBOOT region for recovery
 - minimal Linux kernel with flashrom, cbmem, gpg (for binary signature verification), network and removable storage drivers
 - can recovery firmware from a nice shell
 - only a few KiBs left :)

Runtime configuration

- coreboot currently supports CMOS as runtime options storage
- CMOS relies on coin cell battery and is also sensitive to cosmic rays etc. thus some people don't like it
- answer? VPD

VPD - Vital Product Data, platform sensitive configuration storage for Chromebooks. In our solution VPD was divided into two regions RO_VPD and RW_VPD.

- RO_VPD - unwritable runtime configuration options containing default settings
- RW_VPD - modifiable runtime configuration option for user

In order to support VPD options, done the following:

- stripped Google's VPD utility from flashrom API and placed in util/vpdtool
- when CONFIG_VPD is selected, the tool is built and formats VPD partitions
- when the binary is flashed, cbmem hooks automatically parse VPD and add them to cbmem space
- added pre-cbmem access to VPD based on FMAP for bootblock and romstage
- added VPD library to SeaBIOS to parse certain settings and influence boot flow (USB boot toggle, PXE toggle, serial console redirection toggle)
- has the potential for removing old SeaBIOS runtime configuration CBFS files in etc/ and coreboot CMOS
- VPD can be modified from user space directly on the flash with flashrom or on the coreboot image (offline method)

<https://chromium.googlesource.com/chromiumos/platform/vpd/>

DEMO time...

- implement C environment bootblock for PC Engines APU2
- prepare a generic payload for VPD and bootorder updating in BIOS
- port other SeaBIOS runtime options to VPD
- maintain chain of trust up to kernel with TrustedGRUB and Linux kernel (requires TPM2.0 driver implementation)
- fix TPM2.0 issues in coreboot:
 - TPM2.0 logs are stored in TPCA (TPM1.2 log)
 - cbmem utility can only print TPCA logs, TPM2 log not supported
 - SeaBIOS starts writing TPM2 logs at the start of TPM2 log area, make it detect the last log entry to continue logging
 - TPM2 logs are different than TPCA logs, implement correct TPM2 log format and differentiate from TPCA log API
- support for automatic disk encryption and decryption with TPM2.0 on Linux and FreeBSD (maybe OPAL disk)

Q&A



Thank you