TrenchBoot - Open DRTM implementation for AMD platforms

Piotr Król[1], Krystian Hebel[2], and Łukasz Wcisło[3]

[1] 3mdeb Embedded Systems Consulting, Gdansk, Poland
piotr.krol@3mdeb.com
[2] 3mdeb Embedded Systems Consulting, Gdansk, Poland
krystian.hebel@3mdeb.com
[3] 3mdeb Embedded Systems Consulting, Gdansk, Poland
lukasz.wcislo@3mdeb.com

Abstract

In this paper, we are going to explain TrenchBoot implementation for AMD and prove a boot chain leveraging it. We will outline how this solution coexists with open-source firmware like coreboot in flash, explain required bootloader extension based on GRUB2 implementation, discuss Landing Zone (LZ) secure loader implementation and required Linux kernel modifications.

Finally, we will explain what benefits this solution has over previous: OSLO, Flicker, Soft Cards and others.

1 Introduction

Trusted Computing Base (TCB) of a computer system is the part of the system (hardware, firmware, software and other) that is critical to its security and whose failure (due to bugs, vulnerabilities or any other reason) may lead to compromise of the system. It is the portion of the system that is relied on to enforce the security policy of the platform.

Trusted Computing Group (TCG) defines Root of Trust for Measurement (RTM) as a trusted implementation of a hash algorithm which is responsible for the first measurement on a platform. This measurement can be done at a boot time or later, to put a platform into a trusted state. Dynamic Root of Trust Measurement is used to measure platform initialization without a hardware platform restart. The DRTM approach is contrary to Static Root of Trust Measurement (SRTM) where measurements are taken during the boot process for the entire boot chain.

In SRTM hash algorithm measurements are recorded into PCR (Platform Configuration Register) and the code running later is unable to tamper with the earlier measurements. A PCR is a 160-bit (SHA1) or 256-bit (SHA256) wide register. It cannot be directly written, only extended. Such subsequent extension operations create a chain of trust.

In DRTM, platform hardware is already configured and memory is prepopulated. This means the process itself has to prevent any possible impact of the boot code on the TCB. The way to do this is to check that the hardware configuration is in a TCB-safe state and that the TCB can protect itself. DRTM reduces the scope of TCB – BIOS, option ROMs, bootloader and host OS are no longer part of it.

The DRTM process is presented on Figure 1, it starts with the DL Event. DCE Preamble is not measured because it runs in unknown state of environment, therefore it is not part of DRTM. As a result of TPM initialization after power on or reset, PCRs 17 to 22 begin with an initial value of -1, and the only thing that may reset them is an execution of the DL Event. It changes the value of PCRs to 0 and immediately extends it with the hash of the DCE (DRTM

Configuration Environment). Any try to reset the TPM will set PCR[17] (and following) to -1 again, which makes DRTM immune to TPM reset attack.

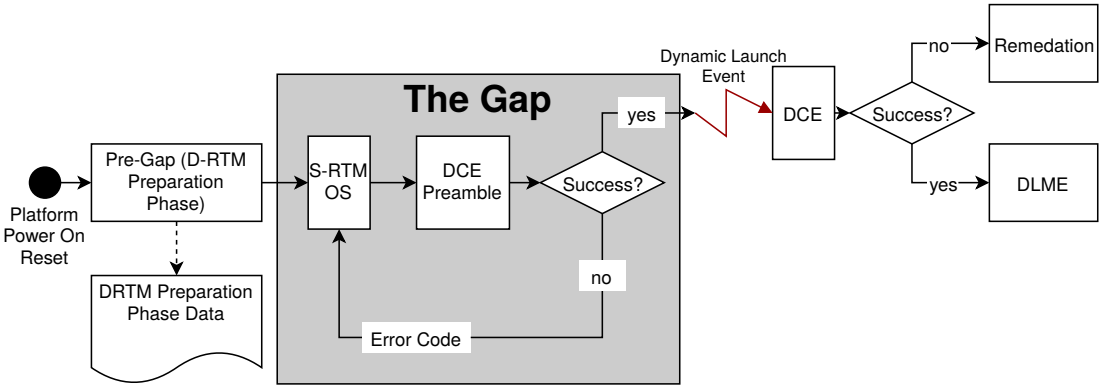

Figure 1: DRTM timeline [1]

2 Implementations

Two leading silicon vendors, AMD and Intel, enabled DRTM in their solutions. AMD provides a DRTM with its Secure Startup technology and *skinit* instruction which has been introduced with the AMD-V extension. In Intel CPUs, the Trusted Execution Technology (TXT) includes a DRTM with the *senter* instruction and many *getsec* leaf functions. We choose AMD platforms because DRTM for it may be fully open-source. In Intel solution for a reason of implementing DRTM it is necessary to use proprietary ACM module. Its usage can be questionable when a source code audit is required.

While both AMD and Intel implementations are compatible with TCG's specification, they use different terms for components of DRTM architecture. Generic flow is presented on Figure 1, and Table 1 translates TCG terms into those given by AMD and TrenchBoot. Flow of our implementation is shown on Figure 3. Intel TXT terms are out of scope of this paper.

TrenchBoot does not assume existence of Pre-Gap, i.e. no SRTM must be performed at platform reset and no assumptions about the previous state of platform (including other PCRs) can be made.

3 Memory protection

DRTM architecture must ensure that memory containing code cannot be changed by anything outside of TCB. As code of the main flow is validated before execution, this applies to other agents that can write to memory. They include DMA access and SMM code. Memory must be protected at least between hashing and execution (inclusive), otherwise a rogue agent could change code after it is validated, gaining control over platform (see Figure 2). Safer approach is to protect memory even before code is loaded. This way we can ensure that the validated code is the same as loaded image and it has not been changed after loading.

TCG	**AMD**	**TrenchBoot**
The Gap	non-trusted mode	N/A
Dynamic Launch (DL) Event	SKINIT instruction	secure launch
DMA protections	Device Exclusion Vector (DEV)	N/A[1]
DRTM Configuration Environment (DCE)	Secure Loader Block (SLB)	Landing Zone (LZ) code + data
N/A[2]	Secure Launch (SL) Image	Landing Zone (LZ) code
Dynamically Launched Measured Environment (DLME)	Secure Kernel (SK)	kernel
DCE Preamble	N/A	slaunch module

[1] uses the same name as vendor
[2] not distinguishable part of DCE

Table 1: Translation of terms

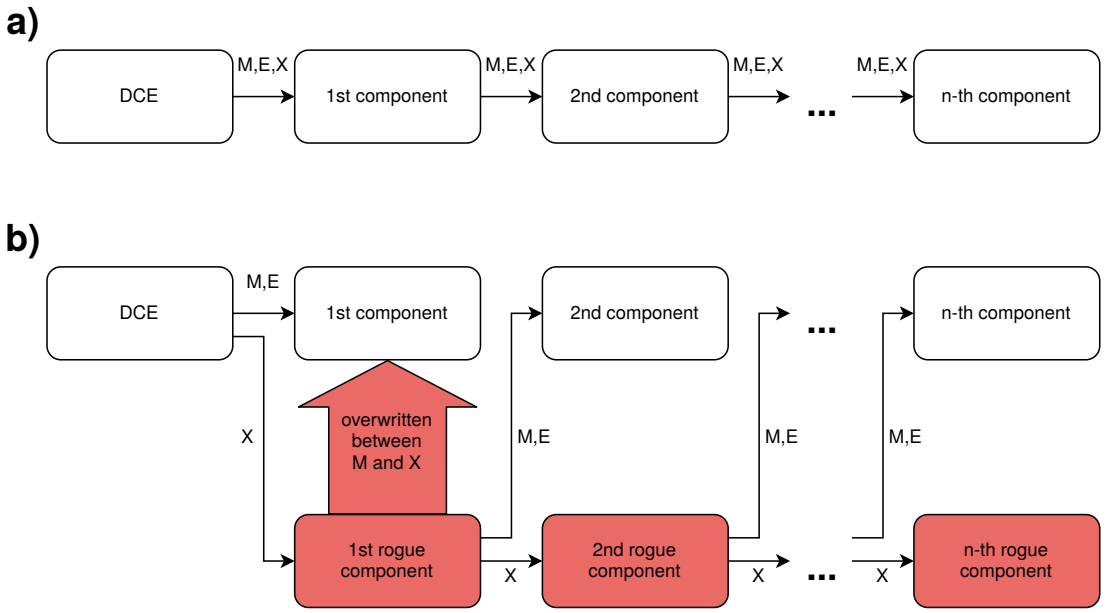

Figure 2: Assuming control over platform when no memory protection is in place; a) normal operation, b) rogue agent overwrites already validated stage. Note that PCRs are extended with original values, so all PCR-bounded secrets can still be accessed.
M – measure component, **E** – extend PCR, **X** – execute component

3.1 DMA

DMA access can be blocked by:

- exclusion – DMA cannot write to specific memory ranges,
- translation – DMA addresses are translated to different ones, outside of TCB.

The Intel VT-d implements exclusion with DMA protected range (DPR) and protected

memory regions (PMRs). First one is located right before SMRAM segment (TSEG). PMRs are two ranges of physical addresses that are protected from DMA access. One region must be in the lower 4GB of memory and the other may be anywhere in address space [2]. AMD uses Device Exclusion Vector (DEV). DEV is a continuous array of bits in physical memory; each bit in the DEV corresponds to one 4-Kbyte page in physical memory [3]. This gives better granularity than Intel's approach.

Translation is a part of IOMMU (AMD) or VT-d DMA remapping (Intel). Using translation is generally more complicated than exclusion. Care must be taken to not redirect I/O access to a range that is or will be a part of TCB. Because of that DRTM usually uses exclusion, but both approaches can be used simultaneously.

3.2 SMM

It is up to BIOS vendors to ensure safety of the SMM, including protection from external access to SMRAM and denying all callouts to unprotected memory. Due to the properties of SMM it is impossible to validate its code after it is properly locked. Sometimes SMM code is auditable thanks to being open-source. It is possible for SRTM part in the Pre-Gap state to measure SMM code into a PCR before locking SMRAM, in this case DCE can check its hash value. Note that this results in a larger Trusted Computing Base.

Even when SMM code is auditable and/or measured in SRTM phase, protection of its memory can be circumvented by ring-0 software running in the Gap [4]. Newer processors can be set that attempts to execute SMM code not within the ranges defined by the SMRR[1] will assert an unrecoverable MCE (machine check exception). This protection mechanism also has been broken [5].

Because there is no valid, universal way of measuring properly protected SMM code (not including hacks mentioned earlier), the DCE shall consider SMM to be *correct by construction* [1].

4 Goals

Our ultimate goal is to enable TrenchBoot [9], a framework that allows individuals and projects to build security engines to perform launch integrity actions for their systems on all platforms. While TrenchBoot describes theoretical framework that can cover various DRTM solutions we focus more on practical implementations and analysis of its pros and cons. This analysis covers all components required to implement DRTM, according to TrenchBoot framework, using AMD SKINIT mechanism.

We do not plan to explain TrenchBoot in details and implement all its features since it is a moving target and not yet fully defined. Our goal is to build solid foundation that can be extended in future.

Our implementation was tested on PC Engines apu2c4 that uses AMD Embedded G-series GX-412TC [10]. It has Infineon SLB 9665 TPM2.0 module connected to LPC bus.

[1] System-Management Range Register Interface. It is basically a pair of memory type range registers that define the base address and range for the SMRAM memory. It also specifies the memory type used to access it in SMM.

5 Other Solutions

Below we present other implementations of DRTM. All of them have only limited support for TPM1.2.

5.1 OSLO

OSLO [6] stands for Open Secure Loader which is a chain-loaded (started by another) bootloader and is the first implementation of DRTM using SKINIT instruction on AMD platform. OSLO bootloader implementation has 1500 lines of code approximately and shortens the trust chain by eliminating BIOS and previous-stage bootloader from TCB and establishes DRT for trusted boot. OSLO does not extend DRTM past kernel entry.

5.2 Flicker

Taking OSLO as the starting point, Flicker [7] architecture provides with *meaningful attestation* and *minimal TCB* in addition to *isolation* and *provable protection* provided by AMD SVM and Intel TXT. For this, Flicker leverages DRTM to execute security sensitive part of application in an isolated environment with extremely small TCB of 250 lines approximately. This system allows Piece of Application Logic (PAL) to execute in isolated environment and don't require Virtual Machine Monitor (VMM) or OS.

5.3 Soft Cards

An application of Flicker. Soft cards [8] are cryptographic tokens which use *secure execution technology* for providing protection against software attacks and basic protection from hardware attacks. They are virtual PKCS 11 (public-key cryptography standards) tokens running in software which makes use of 'trusted execution' for assurance of their secure and isolated execution.

6 Solution

In simple words, we will show how to extend GRUB with the ability to call the AMD SKINIT instruction with TrenchBoot. The whole solution will be open-source, auditable, and fully extendable.

In the AMD implementation, the initial code executed by the SKINIT instruction is the Secure Loader (SL). The SL is an owner-provided code base that is responsible for securely initializing the system and hand over to a Secure Kernel (SK). There are two definitions directly connected with SL: SLB (Secure Launch Block), which is a 64KBytes range of memory, and SL image, which is initial part of SLB containing header and code, but not mutable data (stack or tables filled at runtime). The SL must meet three primary conditions:

1. SL image's first two words contain the entry point offset and image size
2. SL image and stack must fit in 64KBytes
3. SLB must be loaded aligned to 64KBytes

SKINIT measures SL image and protects its memory from devices by blocking DMA access to the SLB. SL is responsible for protecting the remainder of the execution environment through measurement and memory protection of the SK it will be handing over control. Other structures accessed by a kernel also need to be validated, they include zero page[2], command line and initrd. Due to the limited size of SLB those are measured in kernel before use, but memory protection is enabled already by SL.

The AMD SKINIT instruction call requires the system to be in a specific state as enumerated below:

1. SVM check, either the feature flag CPUID Fn8000_0001_ECX[SKINIT][3] or EFER.SVME bit[4] must be set to 1,
2. The CPU must be in protected mode, system privilege level (CPL=0),
3. EAX register holds the physical address of SLB.

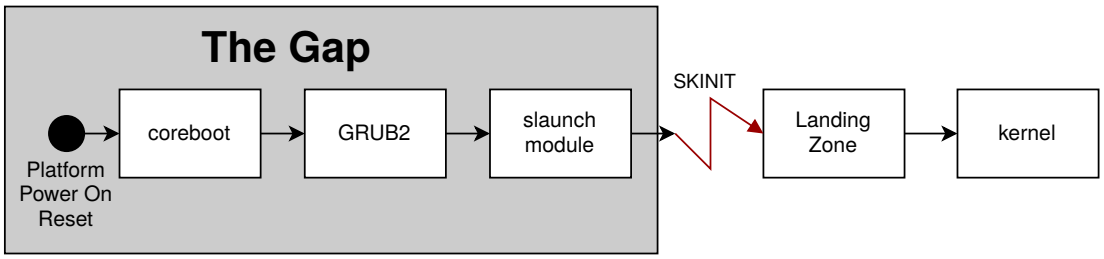

Figure 3: Timeline of TrenchBoot implementation

In our solution, depicted on Figure 3, modified GRUB initializes Secure Launch module – first component of TrenchBoot. This module adds two new instructions to GRUB: *slaunch* and *slaunch_module*. First one hooks itself into GRUB module for booting Linux kernel; it takes either *skinit* or *txt* as its only argument to support both implementations. *slaunch_module* takes path to the next element (Landing Zone) and loads it to memory. The kernel (and initrd) can be loaded as usual either before or after those commands. When Linux is about to be booted, LZ is started instead. Since then, all of the code must be validated before being used so the chain of trust is preserved.

Landing Zone is a binary file is used as an SLB for SKINIT. Its source is available in TrenchBoot repository [11]. LZ sets initial DEV table, protecting next elements from being overwritten by DMA access. After the memory is secured, kernel image is validated and executed.

A Linux kernel must have TrenchBoot-specific stub built into the early setup code in order to handle the state that the LZ leaves the BSP in[5]. Kernel validates all structures it uses (zero page, initrd, command line) before it gets to the self-decompression stage.

[2]Zero page is a special structure containing environment state that needs to be provided to the kernel before command line is accessible. Fields of zero page include, among others, video mode used by bootloader, E820 memory map table, pointers and sizes of ramdisk and command line.

[3]Reports support for SKINIT instruction.

[4]SVM enabled. This bit *enables* use of SVM instructions, it does not make SVM *active*. On processors that support the SVM-Lock feature, SKINIT and STGI can be executed even if EFER.SVME=0. Use of SKINIT when SVM is active requires extreme caution. As it resets CPU state, it can give virtual machines a way to escalate their privileges if a VM can load its own SL image and guess its host-physical address. Use of SKINIT from VMM mode is also unlikely, as it often is SKINIT that loads and validates VMM code.

[5]Regular Linux boot protocol [12] requires specific register values, but TrenchBoot requires additional data (e.g. LZ base address) to be passed to the kernel.

There are no explicit checks shown on Figure 3, as there are considered to be integral part of previous stage. This implementation also does not include *Remediation* stage, but it can be easily added if required.

Measurement and protection steps, starting from SKINIT, are:

1. Entry from the SKINIT jumps to the LZ; SL image is automatically hashed, PCR17 is set to 0 and immediately extended with that hash by SKINIT
2. Landing Zone sets up DEV protection and extends PCR17 with hash of kernel image
3. Landing Zone prepares the BSP and jumps to kernel
4. Before kernel decompresses itself, it validates the following:
   (a) zero page – page with kernel data, such as memory map tables and pointers to initrd and command line, PCR18 is extended with hash value of this page
   (b) command line, result also extends PCR18
   (c) initrd, PCR17 is extended with its hash value
5. Kernel boot finishes booting normally

6.1 System preparation

Before deploying DRTM there is a couple of requirements to fulfill:

- We should calculate space for each component that have to fit into SPI flash:
  - kernel is built with all unnecessary options disabled,
  - minimal initrd including BusyBox is used,
  - tpm2-tools and required libraries were stripped.
- We have to craft *grub.cfg* file so it automatically runs.
- Locality of TPM must be unset before SKINIT, even though AMD64 Architecture Programmer's Manual [3] does not mention this.

6.2 Issues

There is a problem that renders this solution useless for now. PCR17 is extended by SKINIT with constant values:

```
/ # tpm2_pcrread sha1:17+sha256:17
sha1:
...
  17: 0x31A2DC4C22F9C5444A41625D05F95898E055F750
...
sha256:
...
  17: 0x1C9ECEC90E28D2461650418635878A5C91E49F47586ECF75F2B0CBB94E897112
```

Those hashes do not depend on LZ code or its size specified in the header. TPM hardware issue was ruled out, we tested multiple modules and each of them produced the same values.

We have tried changing TPM firmware to TPM1.2. In this mode PCRs 17 to 22 held their initial values of -1, even when locality is relinquished. This module does not allow for firmware downgrade, so only the newest available versions of TPM1.2 and TPM2.0 images were tested (4.43.258.0 and 5.63.3144.0, respectively).

Another possibility is that different device tries to use LPC bus at the same time that SKINIT sends SL image for hash calculation. On our platform there is a SuperIO chip on this bus; unfortunately, by turning it off we are losing UART console which is our only output at this moment. Sniffing LPC bus by hardware probing might provide us with further information.

Some developers mention that microcode updates for all cores needs to be cleared (see e.g. Flicker's README). Neither clearing nor updating microcode does not help in our case. We have contacted AMD and we are working together to solve that issue.

7 Future improvements

TrenchBoot does not attempt to validate SMM, instead it assumes that it is to be trusted unconditionally. As mentioned earlier, it is impossible to validate this code after it is locked. There are other options, described in *Containerizing Platform SMM* section of *AMD64 Architecture Programmer's Manual Volume 2* [3], however they require use of virtualization. Basically, there are 3 options for handling SMM:

1. SMI signals are not intercepted at all. This is default state, in which pre-SKINIT, installed by firmware SMM handlers are used.

2. A hypervisor emulates all instructions that cause SMI signal assertion. This way, whenever SMI happens, it is known to be asynchronous, external interrupt, such as an overheating warning. In this case hypervisor should temporary enable interrupt handling (by using STGI instruction), so the pre-SKINIT handler will be called. While this approach reduces SMM impact on the system, it still includes SMM code in TCB.

3. A VMM (virtual machine monitor) can containerize SMM by creating its own trusted SMM hypervisor and use that handler to run the platform SMM code in a container. The SMM hypervisor emulates SMM entry, including setup of the SMM save area, and emulates RSM at the end of SMM operation. The guest executes the platform SMM code in paged real mode with appropriate virtualization intercepts in place, thus ensuring security. Unfortunately, this requires access to SMM control registers, so SMM must be unlocked, in which case its code could be accessed and measured anyway.

To summarize, there is no secure way to use SMM handlers, especially without making changes to underlying firmware. One such change would be to leave SMM unlocked in firmware (either for validation or ability to replace untrusted SMM handler with a new one), but this leaves it unprotected from software running in the Gap; it also requires a high level of knowledge about hardware platform. Another option is to measure it during boot and expanding a PCR, creating a pre-Gap phase.

ACPI tables are also not validated by TrenchBoot. They can include AML (ACPI Machine Language) bytecode, which must run with the highest privilege mode. ACPI is selected in example kernel configs for TrenchBoot, which means it is used without validation. DRTM specification [1] mentions that ACPI Name Space must be protected, but details for such protection are outside of that specification.

Host firmware should also create DRT (DRTM Resources Table) – an ACPI table which provides the address of the DCE preamble for starting the DRTM process. It also holds information such as the location of the DRTM event log and is the primary way information is passed from the DCE to the DLME.

Right now, Secure Launch module supports booting only to Linux kernel. It could be extended (along with LZ code) to enable use of different payloads, such as a binary file containing a hypervisor, loaded in a secure manner before any OS starts.

8 Conclusion

Open-source solutions are the future of firmware and security. There is no escape from it, even if someone for some reason states something different. For last decade consequent digitalization of our lives and rapidly growing market of IoT has made security crucial like never before. One of its features is enabling secure encryption of vulnerable data and provide a safe upgrade online. We consider our paper a small step for a brighter future, where our devices will be trusted and reliable.

References

[1] https://trustedcomputinggroup.org/resource/d-rtm-architecture-specification/

[2] https://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf

[3] https://www.amd.com/system/files/TechDocs/24593.pdf

[4] http://www.lepointdeau.fr/CONF%20SLIDES%20AND%20PAPER/us-15-Domas-The-Memory-Sinkhole-Unleashing-An-x86-Design-Flaw-Allowing-Universal-Privilege-Escalation-wp.pdf

[5] https://www.synacktiv.com/posts/exploit/code-checkmate-in-smm.html

[6] Bernhard Kauer, *OSLO: improving the security of trusted computing.*, In Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium (SS'07), Niels Provos (Ed.). USENIX Association, Berkeley, CA, USA, Article 16, 9 pages

[7] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, Hiroshi Isozaki, *Flicker: an execution infrastructure for tcb minimization.*, In Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (Eurosys '08). ACM, New York, NY, USA, 315-328. DOI=10.1145/1352592.1352625 http://doi.acm.org/10.1145/1352592.1352625

[8] Brasser, Franz Ferdinand, et al. *"Softer Smartcards." Financial Cryptography and Data Security.*, Springer Berlin Heidelberg, 2012. 329-343.

[9] https://github.com/TrenchBoot/documentation

[10] https://pcengines.ch/apu2c4.htm

[11] https://github.com/3mdeb/travail/tree/master/trenchboot/lz

[12] https://www.kernel.org/doc/html/latest/x86/boot.html