Security Analysis of TEEs on TrustZone

It's smarter, it's safer. It's VO.

viaccess·orca

Table of Contents

Executive Summary

The concept of Trusted Execution Environment (TEE) has been introduced more than ten years ago, and commercial TEE solutions using ARM's TrustZone technology have been used for several years now. As the number of implementations increases and they become more and more common in a large number of devices, it seems relevant to wonder about the security of these solutions.
This document's purpose is to present the state of the art regarding the security of different Trusted Execution Environment (TEE) implementations using Arm's TrustZone technology, based on available literature on TEE architecture, security, implementation details, and public information on existing vulnerabilities. By synthesizing the security functions of a TEE, presenting the attack surface and using detailed write-ups on public vulnerabilities, we aim at showing the different ways an attacker could go from an application of the Normal World to sensitive assets manipulated in the TEE. We are particularly interested in the use case of a DRM executing on a system where the TEE is the only security context available, and will guide our analysis with this in mind.
We conclude that while TEEs can indeed provide some security, the guarantees seem to significantly vary between implementations. Furthermore, the global security of the TEE relies on the Trusted OS but also on the Trusted Applications; which adds many different actors to trust. As the main vector of attack is through the Trusted Applications, it seems difficult to build a coherent secured environment in practice with only a TEE.

TEE Architecture

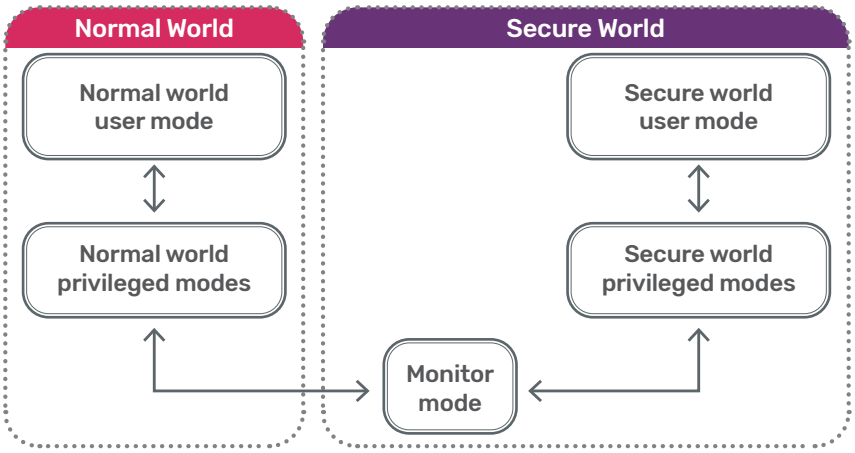
This part defines the basic concepts needed to understand the rest of the document. It also presents the main available implementations of TEE.

Arm TrustZone

In the scope of this document, we focus on TEEs using TrustZone technology, which was introduced by Arm in 2004 with their Arm1176JZ-S processor. Nowadays, it can be found in any Cortex-A processor or processor based on the Armv7-A and Armv8-A architecture, and Cortex-M processors built on the Armv8-M architecture. The TrustZone technology allows to partition the SOC's hardware and software into two worlds, called the Normal World and the Secure World. The partitioning of the two worlds is achieved by hardware logic found in the AMBA bus fabric, peripherals and processors. The hardware changes brought by TrustZone are:

- The NS (Non-Secure) bit added to the bus transactions, stored in the SCR (Secure Configuration Register),
- Two virtual cores per physical processor core (for secure and non-secure mode),
- A mechanism to switch from the Secure World to the Normal World, known as the Monitor Mode, for example with the SMC (Secure Monitor Call) instruction.

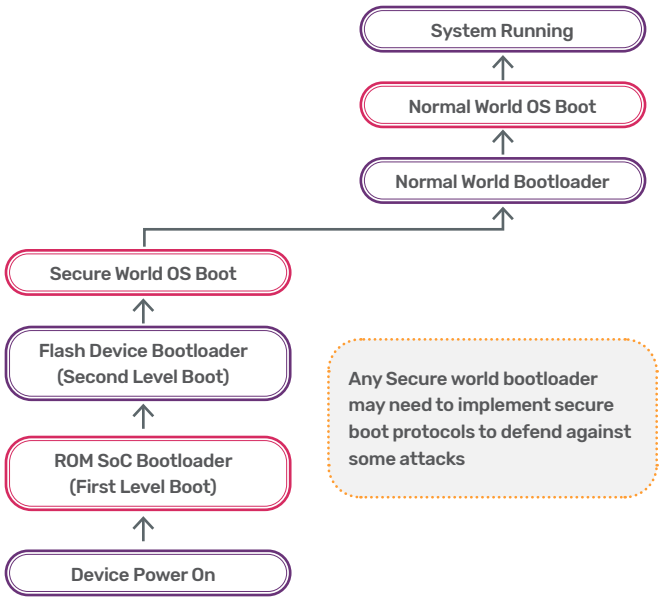
A figure representing the different modes in ARM core can be found in the reference document for the Arm TrustZone technology (Arm, 2009):



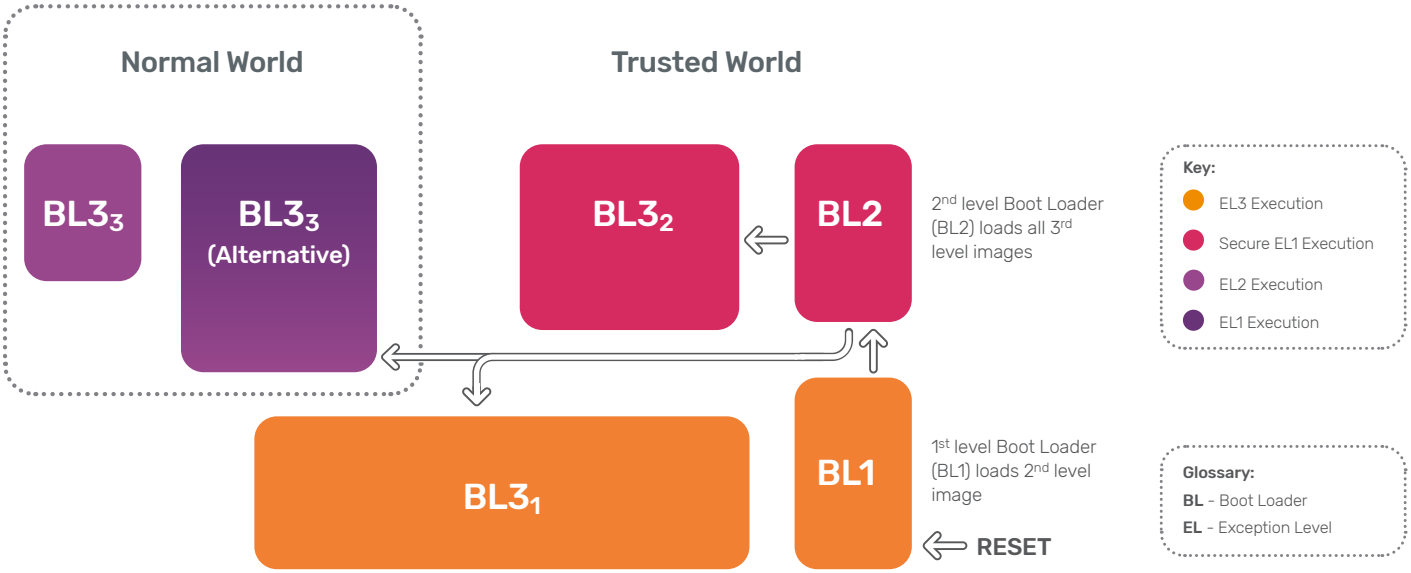
Two optional components defined in the specification can extend TrustZone security features to the memory: the TrustZone Address Space Controller (TZASC) and the TrustZone Memory Adapter (TZMA). Both help partition the memory into different secure and non-secure regions.
Arm's reference document explains that the software executing in these different modes are all implementation defined, though the document provides guidelines for these implementation. It is also specified that other security protocols, such as a secure boot, should be built on top of using the TEE to guarantee that it resists to certain types of attacks.

Secure Boot

A TrustZone-enabled processor starts in the Secure World when it is powered on. The principle of a secure boot scheme is to add cryptographic checks to each stage of the Secure World OS boot and implement a chain of trust. This guarantees that the Secure World software image has not been tampered with. Here is how Arm's documentation quickly describes the typical boot process of a TrustZone-enabled processor:



The TEE paradigm assumes a secure boot is in place; for example, here is the complete secure boot of Trusted Firmware[1] (which provides a reference implementation of Secure World for Arm processors):



1 https://www.trustedfirmware.org/

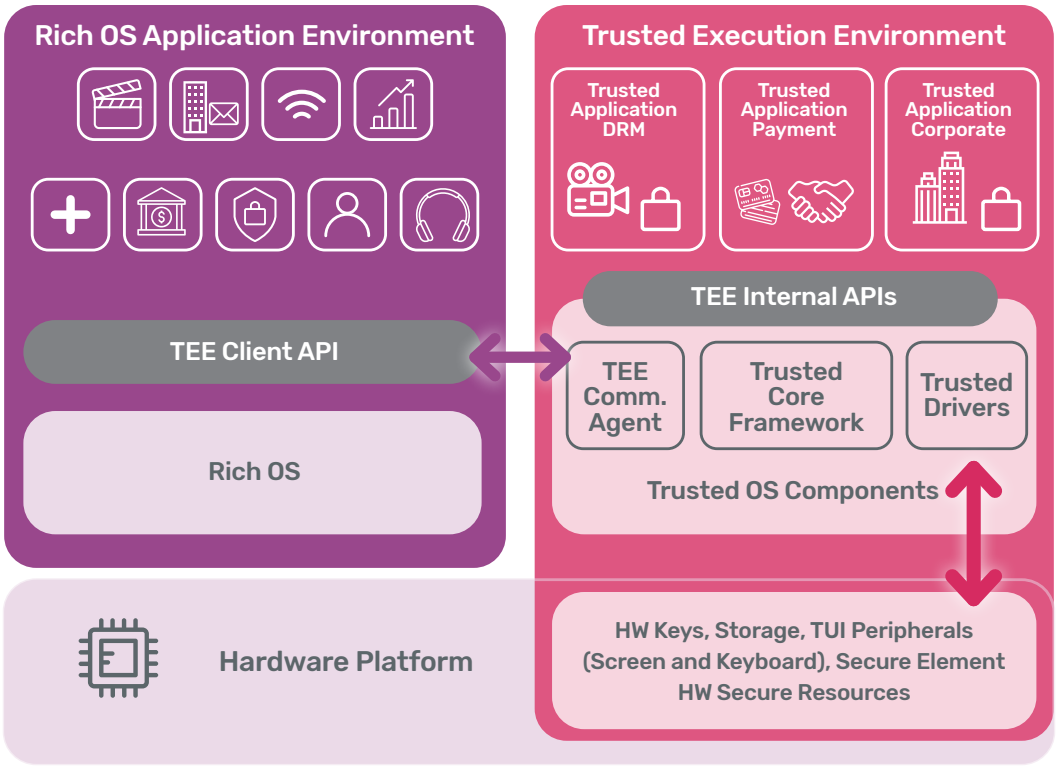
The first bootloader (BL1) is stored in secured ROM and boots the second level bootloader (BL2), which boots the secure monitor (BL31), the trusted OS (BL32) and the Normal World (BL33). We will talk a bit more about Trusted Firmware in the Implementations section.

Secure Monitor Mode

The role of the monitor mode software is to provide a robust gatekeeper which manages the switches between the Secure and Non-secure processor states. It runs at the highest privilege level (EL3 in Armv8). The secure monitor mode is not very detailed in Arm TrustZone's reference document, as it is said to be implementation defined. One may also turn to Trusted Firmware for an example of secure monitor code.

TEE

The Trusted Execution Environment, or TEE, is an isolated execution environment that runs along the Rich Execution Environment (REE) and provides trusted services to that rich environment. The standard documentation regarding TEE is provided by GlobalPlatform[2], a consortium of TEE actors aiming at designing specifications that align with the existing market requirements. Their whitepaper on TEE (Global Platform, 2015) provides a schematic of the different components of a TEE and their interactions, which shows that the trusted environment includes many subparts:



2 https://globalplatform.org/

- The trusted OS, composed of the trusted core (mainly the trusted kernel), the trusted drivers (to communicate with trusted hardware), and the TEE communication agent (to exchange with the application in the REE).
- The Trusted Applications (TA), running inside the TEE and providing security related functionalities to Client Applications (CA) outside of the TEE or to other TAs inside the TEE. The TAs communicate with the rest of the system via APIs exposed by Trusted OS components (TEE Internal APIs), thus gaining controlled access to security resources and services (cryptographic components, secure storage, secure clock…).

The specific details of the isolation between REE and TEE are dependent on each TEE implementation. It is important to note that the NS bit is the only hardware link between the TEE and the rest of the System on Chip (SoC). It is crucial that the TEE implementation uses that bit, otherwise the system would bring only software security and no hardware aspect, and offer no more security than the REE.

Implementations

In this section, we present the main current implementations of TEE using Arm TrustZone technology.

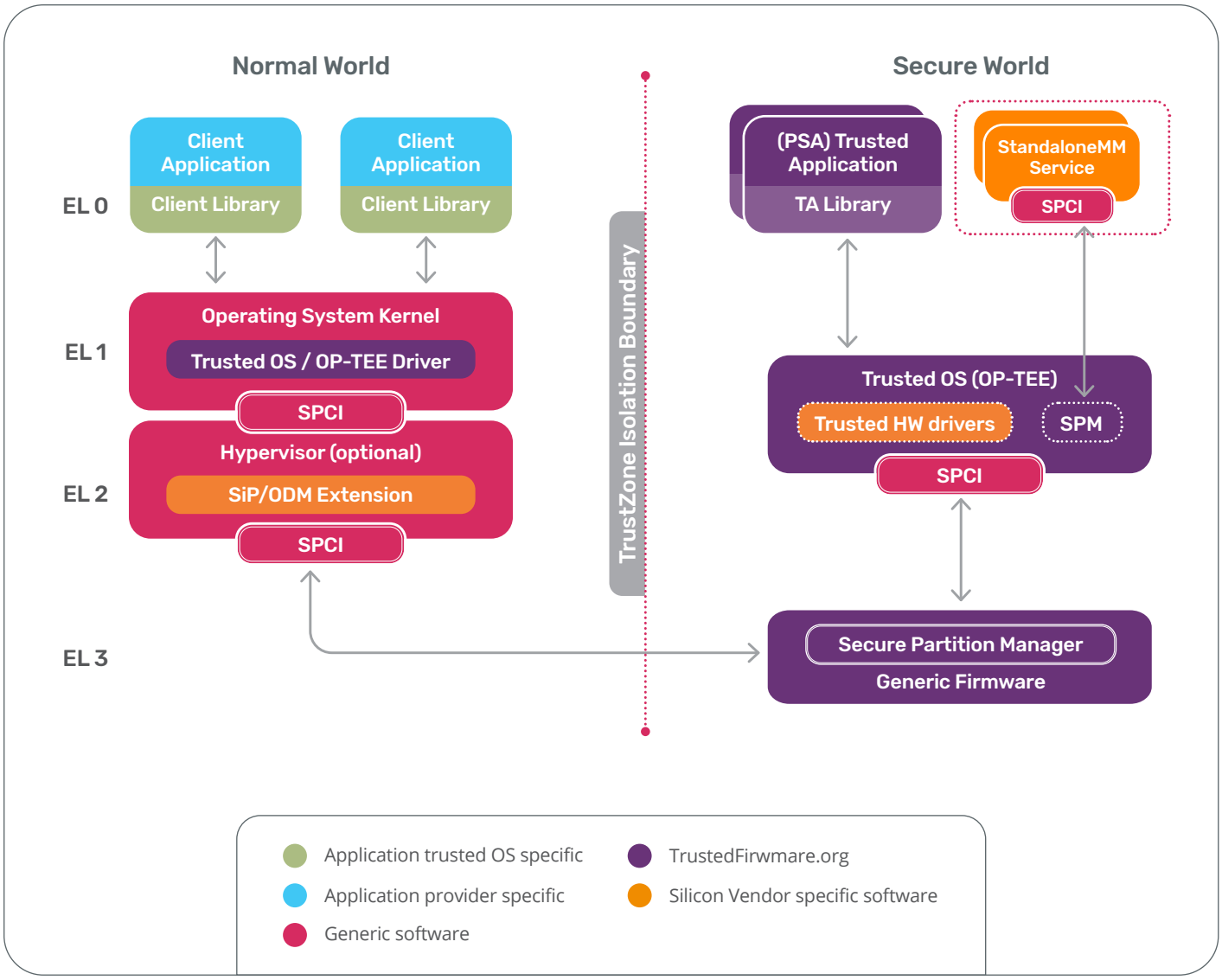
OP-TEE / Trusted Firmware

Trusted Firmware is the official reference implementation of secure world software for Armv7-A, Armv8-A and Armv8-M architectures. Created by Arm in 2013 under the name "Arm Trusted Firmware" (Thoelke, 2013), it became a community project in 2018 under the name "Trusted Firmware.org". Trusted Firmware hosts different projects, the two being relevant to this document are:

- TF-A[3], which is the reference implementation for Armv8-A and Armv7-A processors, which are the one implementing the TrustZone security extension. TF-A includes a secure monitor executing at Exception Level 3 (EL3) and implements several Arm interface standards, such as Trusted Board Boot Requirements CLIENT, or SMC Calling Conventions.
- OP-TEE[4], a TEE implementing the TEE Internal Core API and the TEE Client API provided by GlobalPlatform. It is indicated in the documentation that Armv7-A platforms should use the secure monitor provided by OP-TEE, while Armv8-A platforms should use TF-A's implementation.

[3] https://trustedfirmware-a.readthedocs.io/en/latest/index.html
[4] https://optee.readthedocs.io/en/latest/general/about.html

Here are the different components of a secure world architecture as shown by Trusted Firmware (Carlini, 2019):



Qualcomm's QTEE

Qualcomm Trusted Execution Environment or QTEE (often called Qualcomm Secure Execution Environment or QSEE) is Qualcomm's official implementation of a TEE. It is one of the oldest and most deployed implementation: articles analyzing its security date back from 2015, and it can be found on a lot of devices (Pixel, LG, Sony, HTC, Samsung…). Originally closed-source and not documented, Qualcomm has since published a few documents concerning the different technologies contributing to the secure world (Qualcomm, 2019) and the secure loading of their firmware (Qualcomm, 2019).
Due to the popularity and age of Qualcomm's implementation, it is one having the most documented attacks. We will thus use a lot of examples of vulnerabilities regarding QTEE.

Trustonic's Kinibi

Trustonic[5] was founded in 2012 by ARM, Gemalto and Giesecke & Devrient (G&D) with the purpose of developing a GlobalPlatform compliant TEE. Trustonic Secured Platform[6] (TSP) delivers a TEE that is certified by GlobalPlatform and Common Criteria. It is used by Samsung devices using the Exynos chipset, but also on a lot of other smart devices (LG, Sony …). The Secure OS is called Kibini and is based on a microkernel design. The cryptographic implementation is presented as resistant to branch prediction, dismiss timing and cache-based side-channel attacks.

Samsung apparently switched from Trustonic's implementation to their own on their latest devices, such as the Samsung Galaxy S10. This TEE implementation is called TEEGRIS[7] but there is very little information about it for now. We found only one article regarding its analysis (Tarasikov, 2019), though a search on the National Vulnerability Database yields 17 vulnerabilities in Trusted Application running on TEEGRIS[8] (search done the 27 of May 2020).

Huawei's TrustedCore

On the Hisilicon platform, Huawei devices use a TEE named TrustedCore. It is interesting to point out that Huawei gives very little documentation about their TEE implementation compared to other vendors. Most information can be found in write-ups about vulnerabilities in the Huawei TEE.

5 https://www.trustonic.com/

6 https://www.trustonic.com/solutions/trustonic-secured-platforms-tsp/

7 https://developer.samsung.com/teegris/overview.html

8 https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=TEEGRIS&search_type=all

Security Functionalities

GlobalPlatform defines the purpose of a TEE in the TEE Protection Profile (Global Platform , 2016): to host and execute Trusted Applications while enforcing their mutual isolation and isolation from other execution environments. It must also ensure integrity and confidentiality of the assets managed by the TEE. The following security functionality is defined in the document:

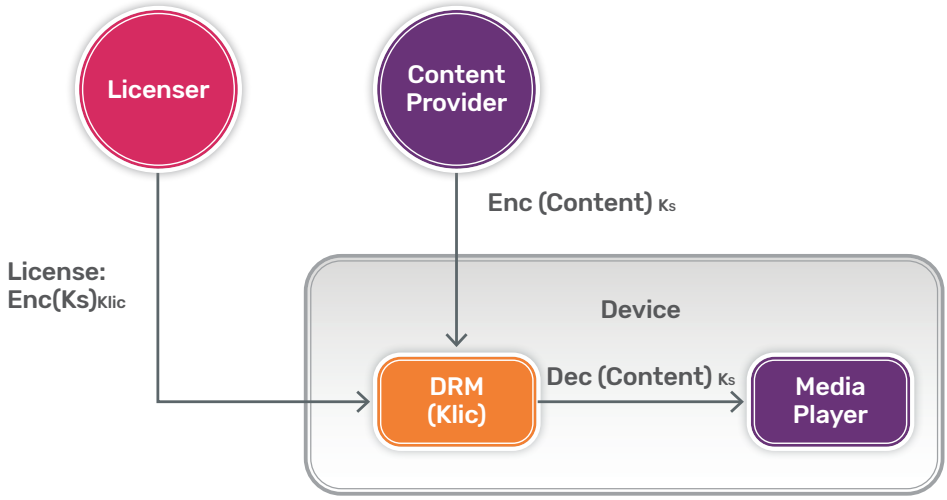
- Instantiation of the TEE through a secure initialization process that ensures the authenticity and the integrity of the TEE code (see the Secure Boot section)
- Isolation of the TEE services, resources, and all the TAs from the REE
- Isolation between TAs and isolation of the TEE from TAs
- Protected communication interface between CAs and TAs
- Trusted storage of TAs and TEE data and keys, ensuring consistency, confidentiality, atomicity and binding to the TEE
- Random Number Generator
- Cryptographic API: generation and derivation of keys and key pairs, support for cryptographic algorithms such as SHA-256, AES 128/256, T-DES, RSA 2048, etc.
- TA instantiation that ensures their authenticity and contributes to the integrity of TA code
- Monotonic TA instance time
- Correct execution of TA services
- TEE firmware integrity verification
- Prevention of downgrade of TEE firmware

A TEE that meets the protection profile thus guarantees that:

- All code executing inside the TEE has been authenticated
- The ongoing integrity and confidentiality of all TEE assets is assured through cryptography, isolation, and or other mechanisms
- Code and assets are protected from unauthorized tracing and control

Scope

This section gives more details about the scope of the security evaluation: the different assets to be protected by the TEE, the possible threats and the assumptions made in the context of this evaluation. We refine more general aspects of the TEE to our specific context, which is a DRM system in a Trusted Application. In the next figure, we very basically recall how a DRM system works:



The DRM agent manipulates a secret key (Klic) that can decrypt the license, which will in turn provide the key used to decrypt the distributed content.

Assets

The TEE protection profile defines a list of assets with the expected properties for them; for example, the TEE identification must be unique and non-modifiable, while the TEE firmware must have the properties of authenticity and integrity. In our context, the assets of interest are the DRM TA data and keys:

- License Key (Klic): integrity, confidentiality and renewability;
- User/Device specific data, used to detail permission of the user to access specific content: integrity;
- Content Key (Ks): confidentiality.

DRM implementations may use the TEE as their only security measure, or integrate a hardware keyladder and a dedicated secure processor as well. As explained previously, we are considering the implementation of a DRM using only the software security provided by the TEE with no further hardware mechanisms available. In this particular context, the protection of the assets relies on the TEE: the assets are manipulated by the TEE OS, drivers and by the Trusted Applications using them.

Of course, all the other properties of other assets defined in the protection profile must stand, as they participate in the security of these two specific components. For example, if the TA's code integrity is not guaranteed, then an attacker might use tampering as an attack to get confidential data.

Threats

We use the same threats definitions as those given in Global Platform's Protection Profile (Global Platform, 2016), slightly adapted to our context.

This security analysis targets threats to the TA's assets that arise during the end-usage phase and can be achieved by software means. Attackers are individuals or organizations with remote or physical (local) access to the device embedding the DRM solution. The user of the device becomes a potential attacker when they access a device embedding the DRM system. The motivations behind the attacks may be diverse, e.g. threaten the reputation of the service provider or decrease their revenue, but most common motivation is to illegally redistribute the content for profit.

The impact of an attack depends not only on the value of the individual assets attacked, but, in some cases, on the possibility to reproduce the attack rapidly at low cost: single attacks performed on a given agent have lower impact than massive attacks that reach many devices at the same time.

This document focuses on non-destructive software attacks that can be easily widespread, for instance through the internet, and constitute a privileged vector for getting undue access to the assets without damaging the device itself.

In many cases, a software attack involves at least two attackers: the attacker in the identification phase that discovers a vulnerability, conceives malicious software and distributes it, and the attacker at the exploitation phase that effectively exploits the vulnerability by running the malicious software (the end-user or a remote attacker on behalf of the user). The identification and the exploitation attackers may be the same person, e.g. in the case of an attack where there is no interest or no possibility of spreading the attack widely.

A lot of different threats are identified in the protection profile. A lot of them have as a consequence that the attacker has unauthorized access to the TA's sensitive data, which includes our assets.

Assumption

In addition to the assumption given by the protection profile, we provide the following:

- The device running the application can be rooted. This means the attacker has Kernel privileges and can issue SMC commands to the TEE.
- Any 3rd party application can be installed.
- All network traffic can be captured and/or filtered.

This means that the attacker has full control over the REE.

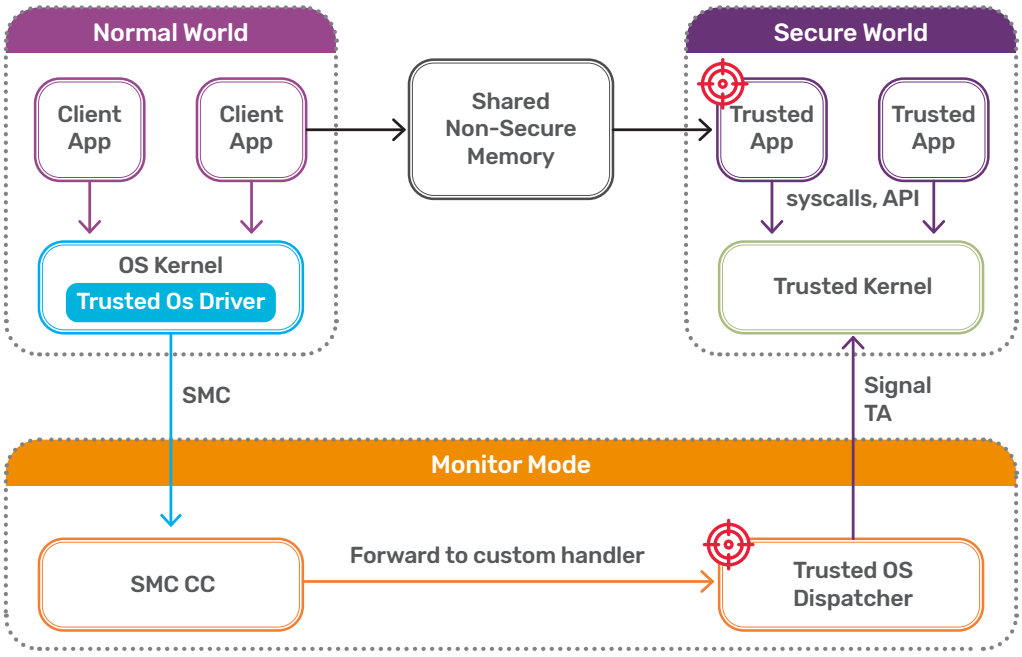
Attack Surface

As said in the Assets section, the license key (Klic) of the DRM agent is manipulated in the TEE and Trusted Application without any additional hardware protection. This means that any compromise of the TA may give access to the assets. Furthermore, attacks on other Trusted Applications may also lead to the compromise of the Trusted OS, and thus may very possibly give access to the assets of the DRM. This means that all the Trusted Applications and the Trusted OS are involved in the attack surface, and any way of accessing these components is to be considered.

Side-channel vulnerabilities and bugs based on speculative execution, such as Meltdown and Spectre, will not be covered in this document, though they can be used against the TEE as well[9]. Mitigating them requires the usage of hardware resources in the SoC but outside the Arm CPU, for instance hardware keyladders or dedicated secure processors.

From the REE, the two main communication channels to the TEE are the SMC calls used to go from Normal World to Secure World, and the shared memory used to pass data between the two worlds. Knowing that, an attacker can use two attack paths:

- Vulnerabilities in functions handling SMC calls (called SMC handlers) will give direct access to the Trusted OS due to their privilege level. Once the Trusted OS is compromised, it is trivial to access to assets of the targeted Trusted Applications.
- Vulnerabilities in any Trusted Applications handler might give access to further vulnerabilities in the Trusted OS (using syscalls for example). Again, once the Trusted OS is compromised, accessing the targeted TA's assets is trivial.



[9] https://trustedfirmware-a.readthedocs.io/en/latest/security_advisories/security-advisory-tfv-6.html#advisory-tfv-6-cve-2017-5753-cve-2017-5715-cve-2017-5754

In the following sections, we will detail several vulnerabilities to show how attackers can use bugs in either TAs handler functions of SMC handlers to gain privilege in the TEE.
Attacking the communication channel between REE and TEE might also be just a step towards the weakening of another feature, for example the secure boot or the downgrade of the TAs. This, while not directly giving access to the assets, might help the attacker later compromise the system.

Vulnerabilities

In this section, we detail several published vulnerabilities on TEE implementations to illustrate the use of bugs in SMC handlers and TAs handlers to compromise the whole Trusted OS. As a more general information, one can notice the consequent number of vulnerabilities regarding the different TEE implementations (a research on the National Vulnerability Database for "TEE" yielded 296 results on the 1st June 2020). It is worth noticing that intelligence gathering on TEE vulnerabilities is not trivial though, as they are not always indexed in the correct way (sometimes being labeled with the name of the TEE implementation, sometimes with "TrustZone").

SMC handlers

The vulnerabilities affecting SMC handlers are of great interest, as they may enable code execution in the most privileged mode of the processor: the monitor mode (EL3).
One of the first detailed articles on a QSEE vulnerability, written by Gal Beniamini, used a bug in a SMC handler to get an arbitrary null write in Trusted OS memory (Beniamini, Exploring Qualcomm's TrustZone implementation, 2015). After having analyzed the format of the TEE image, the author took a special interest in the SMC opcodes and the Secure Channel Manager (SCM), which is the channel through which the Normal World and Secure World communicate. SCM commands fall into two categories: regular (the kernel populates a structure containing the SCM command identifier and data relative to the command) and atomic (the SMC instruction is issued directly with the command ID in R0 and the arguments in other registers). When looking at a specific SCM call, the author realized that while this function was supposed to be called with a regular SCM call, it was possible to call it as an atomic call and execute it to the end. This resulted in a null write directly into an address in the trusted OS memory controlled by the attacker. In the full exploit of this vulnerability (Beniamini, Full TrustZone exploit for MSM8974, 2015), the author uses this null write to disable memory validation and then through several other steps, gets arbitrary code execution in the trusted kernel.
Another vulnerability (Rosenberg, 2014) found in the SMC handlers of QSEE uses an integer overflow in one of the handler to obtain an arbitrary write primitive in the secure memory. This allows the author to perform an SMC table extension attack, consisting on adding a fake SMC handler that allows calling any QSEE function with arbitrary arguments. The author assures that this vulnerability may be used to compromise

DRM schemes.
Trusted Firmware has also reported two vulnerabilities affecting the temporary SMC handlers contained in BL1 and used after cold reset to support the recovery mode[10] [11].

TA Command Handlers

A lot of the vulnerabilities reported regarding TEE implementations are found in Trusted Applications. While they directly only give access to the context of the TA, they are used to further compromise the system by being chained with vulnerabilities in system calls or drivers.

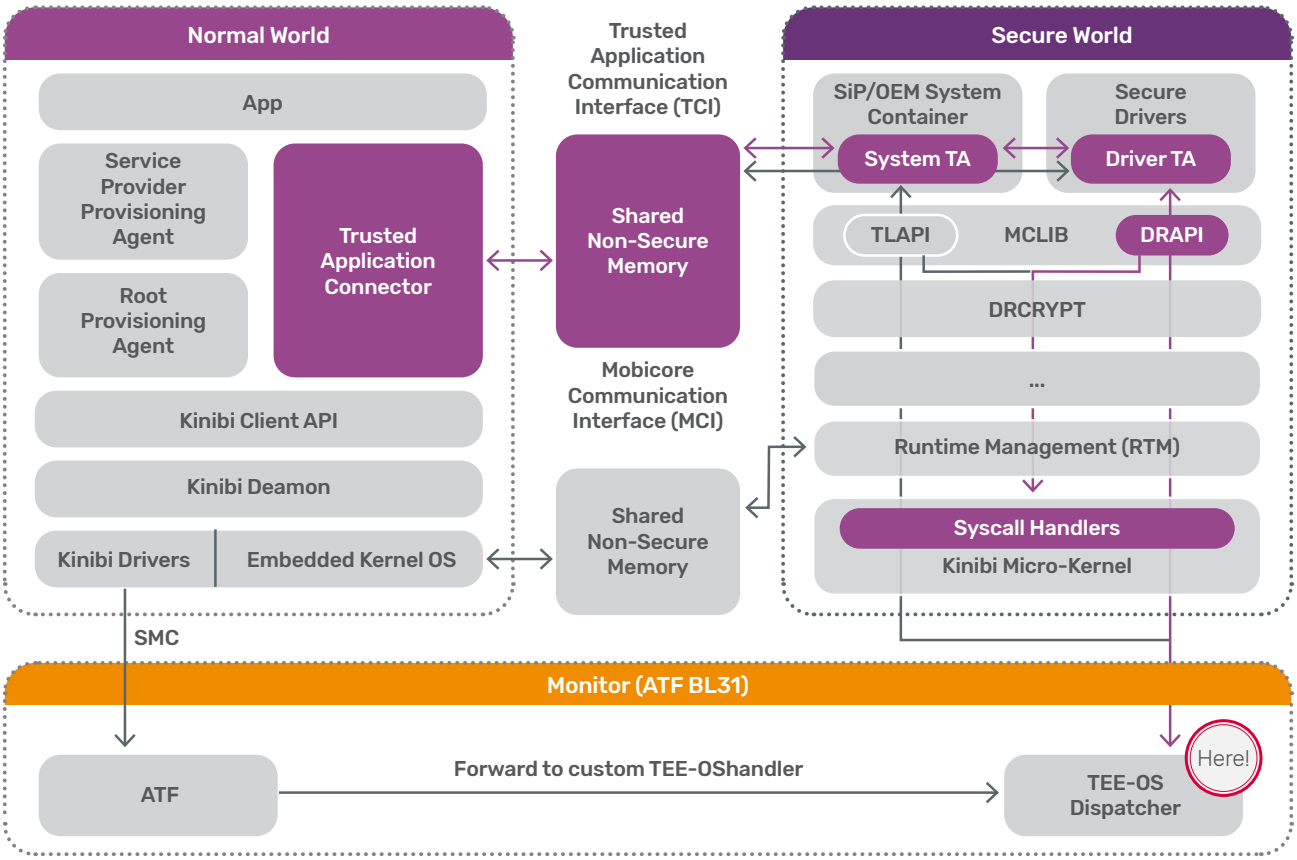
Gal Beniamini also found a buffer overflow vulnerability in the Widevine DRM (Beniamini, QSEE privilege escalation vulnerability and exploit (CVE-2015-6639), 2016) around the same time as its vulnerability in SMC handlers. This vulnerability allows code execution within QSEE, and is later used by the author to gain access to the TEE Kernel (Beniamini, TrustZone Kernel Privilege Escalation (CVE-2016-2431), 2016) and gain access to Qualcomm's KeyMaster key (Beniamini, Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption, 2016).
A complete chain of exploitation in Kinibi from kernel privilege in the REE to code execution in monitor mode was presented at Blackhat in 2019 (Maxime Peterlin, 2019): the authors used a stack-based buffer overflow in the command handler of the SEM Trusted Application (little is known about this application except that it performs cryptographic operations) to gain code execution in Secure EL0, then used it to communicate with

[10] https://trustedfirmware-a.readthedocs.io/en/latest/security_advisories/security-advisory-tfv-1.html#advisory-tfv-1-cve-2016-10319

[11] https://trustedfirmware-a.readthedocs.io/en/latest/security_advisories/security-advisory-tfv-4.html#advisory-tfv-4-cve-2017-9607

secure drivers and find another stack-based buffer overflow in the command handler of the validator driver. This gave them code execution still in Secure EL0 but with access to more syscalls.

At the time of their Blackhat submission, it was possible to map the secure monitor code in the process address space with a call to mmap. More recent versions of Kinibi contain a blacklist of physical addresses that should not be mapped (Alexandre Adamski, 2020). The authors then used mmap to map the blacklist and erase all its addresses. This gave them the possibility to modify a SMC handler and gain code execution in EL3. They provide the following figure to illustrate their attack path:



A lot of the public resources on TEE security present vulnerabilities in TA command handlers, whatever the implementation (Shen, 2015), (Berard, 2018), (Komaromy, 2018).
An example of such vulnerabilities used for other purposes than to directly compromise the Trusted OS is this article (Makkaveev, 2019) detailing the subject of Trusted Applications' fuzzing. The authors use several one-day vulnerabilities to patch an old QSEE implementation with the end goal of being able to execute recent Trusted Applications in the normal world of vulnerable devices, in order to fuzz their command handler. With this technique, they found several vulnerabilities in multiple TAs.

Revocation of TAs

One big security limitation of TEE implementation (that seems to be corrected in the latest versions) is the lack of revocation for Trusted Applications. Even when the implementation provides a revocation mechanism, it does not seem that the TA providers use it consistently (or at all).
This subject has been studied by Gal Beniamini as part of the Project Zero team (Beniamini, Trust Issues: Exploiting TrustZone TEEs, 2017), on the QSEE and Kinibi implementations. The article points out that all Trusted Applications on different images used the same image identifier, and all version counters were still at zero. Beniamini verified that the revocation system was not used by loading the vulnerable Widevine TA previously discovered without being stopped by the OS.
In the same article, the author shows that the "Service Version" field in the MobiCore Loadable Format (MCLF) does not appear to be used. The rollback prevention feature was only introduced since Samsung Galaxy S8/ S8+ devices.

Conclusion

While the TrustZone technology has been used for more than 15 years, it seems than implementations are still trying to find a common ground in standard practices. With new implementations still being released (e.g. TEEGRIS) and the standard Trusted Firmware mostly being integrated for the EL-3 components but not the trusted OS, the number of security problems and vulnerabilities related to TEE environment is still significant. This also draws the conclusion that DRM providers might find difficult to rely only on a TEE implementation to protect their assets. As shown in this document, the compromise of another Trusted Application may give access to the DRM agent's secrets, which makes it hard for the provider to assess the security level of the TEE as a whole. In general, the number and size of TAs should also be as limited as possible.
Software is, in essence, subject to vulnerabilities. While being required to implement the DRM logic, software should never manipulate the most sensitive secrets (e.g. licenser keys and content encryption keys).
Those secrets should remain isolated in hardware keyladders with chipset-unique root keys in One-Time-Programmable (OTP) fuses. Similarly, software should implement a strict "in-depth defense" approach using REE, TEE and a dedicated isolated CPU for the most critical decisions. As mentioned earlier, the TEE runs on the same physical CPU as the REE and remains subject to attacks such as Spectre, Meltdown and future similar attacks. Several articles were already published on using glitching and overclocking attacks to retrieve cryptographic secrets from the TEE or bypass the TA's signature (Adrian Tang, 2017), (Ang Cui, 2017), (Pengfei Qiu, 2019).
We therefore recommend to consider solutions using hardware resources such as keyladders and dedicated secure CPU to handle the most critical secrets and decisions in addition to the TEE.

Bibliography

Adrian Tang, S. S. (2017). CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. Retrieved from https://www.usenix.org/system/files/conference/usenixsecurity17/sec-17tang.pdf

Alexandre Adamski, J. G. (2020, Juillet). A Deep Dive Into Samsung's TrustZone (Part 3). Retrieved from https://blog.quarkslab.com/a-deep-dive-into-samsungs-trustzone-part3-.html

Ang Cui, R. H. (2017). BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection. Retrieved from https://www.usenix.org/system/files/conference/woot17/woot-17paper-cui.pdf

Arm. (2009). Building a Secure System using TrustZone Technology. Retrieved from https://documentation-service.arm.com/static/5f1ffa25bb903e39c84d7e98?token=

Beniamini, G. (2015). Exploring Qualcomm's TrustZone implementation. Retrieved from http://bits-please.blogspot.com/08/2015/exploring-qualcomms-trustzone.html

Beniamini, G. (2015). Full TrustZone exploit for MSM8974. Retrieved from http://bits-please.blogspot.com/08/2015/full-trustzone-exploit-for-msm8974.html

Beniamini, G. (2016). Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption. Retrieved from http://bits-please.blogspot.com/06/2016/extracting-qualcomms-keymaster-keys.html

Beniamini, G. (2016). QSEE privilege escalation vulnerability and exploit (CVE-2015-6639). Retrieved from http://bits-please.blogspot.com/05/2016/qsee-privilege-escalation-vulnerability.html

Beniamini, G. (2016). TrustZone Kernel Privilege Escalation (CVE-2016-2431). Retrieved from http://bits-please.blogspot.com/06/2016/trustzone-kernel-privilege-escalation.html

Beniamini, G. (2017). Trust Issues: Exploiting TrustZone TEEs. Retrieved from https://googleprojectzero.blogspot.com/07/2017/trust-issues-exploiting-trustzone-tees.html

Berard, D. (2018). Kinibi TEE: Trusted Application Exploitation. Retrieved from https://www.synacktiv.com/posts/exploit/kinibi-tee-trusted-application-exploitation.html

Carlini, M. (2019). TrustedFirmware.org - A collaborative effort into Firmware security and the path towards standardized open source Firmware on Arm. Retrieved from https://2019.osfc.io/talks/trustedfirmware-org-a-collaborative-effort-into-firmware-security-and-the-path-towards-standardized-open-source-firmware-on-arm.html

Global Platform . (2016). TEE Protection Profile. Retrieved from https://globalplatform.org/specs-library/tee-protection-profile-v3-1/

Global Platform. (2015). The Trusted Execution Environment:Delivering Enhanced Security at a Lower Cost to the Mobile Market. Retrieved from https://globalplatform.org/wp-content/uploads/04/2018/GlobalPlatform_TEE_Whitepaper_2015.pdf

Komaromy, D. (2018). Unbox Your Phone. Retrieved from https://medium.com/taszksec/unbox-your-phone-part-iii7436-ffaff7c7

Makkaveev, S. (2019). The Road to Qualcomm TrustZone Apps Fuzzing. Retrieved from https://research.checkpoint.com/2019/the-road-to-qualcomm-trustzone-apps-fuzzing/

Maxime Peterlin, A. A. (2019). Breaking Samsung's Arm TrustZone. Retrieved from https://i.blackhat.com/USA19-/Thursday/us-19-Peterlin-Breaking-Samsungs-ARM-TrustZone.pdf

Qualcomm. (2019). Guard Your Data with the Qualdcomm Snapdragon Mobile Platform. Retrieved from https://www.qualcomm.com/media/documents/files/guard-your-data-with-the-qualcomm-snapdragon-mobile-platform.pdf

Qualcomm. (2019). Secure Boot and Image Authentication. Retrieved from https://www.qualcomm.com/media/documents/files/secure-boot-and-image-authentication-technical-overview-v0-2.pdf

Rosenberg, D. (2014). Reflections on Trusting TrustZone. Retrieved from https://www.blackhat.com/docs/us14-/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf

Shen, D. (2015). Exploiting Trustzone on Android . Retrieved from https://www.blackhat.com/docs/us15-/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android-wp.pdf

Tarasikov, A. (2019). Reverse-engineering Samsung S10 TEEGRIS TrustZone OS. Retrieved from https://allsoftwaresucks.blogspot.com/05/2019/reverse-engineering-samsung-exynos9820-.html

Thoelke, A. (2013). An Introduction to ARM Trusted Firmware. Retrieved from https://www.slideshare.net/linaroorg/arm-trusted-firmareforarmv8alcu13

About Viaccess-Orca

Viaccess-Orca is a leading global solutions provider of OTT and TV platforms, content protection, and advanced data solutions for a personalized TV experience. The company offers an extensive range of innovative, end-to-end, modular solutions for content delivery, protection, discovery, and monetization. With over 20 years of industry leadership, Viaccess-Orca helps content providers and TV operators shape a smarter and safer TV and OTT experience.

Viaccess-Orca is part of the Orange Group and the company's solutions have been deployed in over 35 countries, reaching more than 27 million subscribers.

VO's security technology has been recognized in recent high-profile awards from CSI and Video Streaming Exchange and recently passed Cartesian's rigorous Farncombe Security Audit Watermark process.

Copyright