

**M+E**

**JOURNAL**

Today's localization challenges are enormous. The opportunities are unprecedented. Is the industry ready for the mayhem?

# GIVING VOICE TO CHAOS

## **SECURITY SOLUTIONS**

The threats to our most valuable assets are many. M+E vendors are on top of it.

## **WORKFLOWS AND THE CLOUD**


Much has changed in the way we track, access, move and store everything we deal with.

## **SMART CONTENT**

The many ways the industry adopts new technologies to make content smarter.

22.02

# ATTACKING REAL-WORLD APPLICATION OF TEE SECURITY



Most modern devices are equipped with TEE

**ABSTRACT:** In the last few years, Trusted Execution Environments (TEEs) have gained popularity in the Android ecosystem. Riscure analyzed the TEE security of a real-world application available in the market. For Samsung's TEEGRIS TEE OS as implemented in their Galaxy S10, we identify vulnerabilities and discuss techniques used by attackers to exploit them.

By Jasmina Omic, Product Manager Services, Riscure

The Trusted Execution Environment (TEE), a technology enabling developers to delegate security functions to a separate secure environment apart from the normal execution environment, has gained significant interest and is widely adopted by the payment industry, media, and entertainment as well as the Internet of Things (IoT).

Most modern devices including general-purpose computers, smartphones, and TVs are equipped with TEE. The main advantage of delegating such security functions to an isolated environment such as TEE is its logical and physical separation from the Rich Execution Environment (REE) which can be prone to insecure software. Developing secure TEEs is paramount for the secure application of TEE technology within the automotive industry.

Riscure experts and the rest of the security industry have investigated TEE security in-depth over the last few years. One of such investigations looked into how strong Samsung's TEE OS is and whether it can be compromised to obtain runtime control and extract all protected assets, allowing, e.g. decryption of user data. This research was conducted by Federico Menarini in 2019. You can find the full blog series named "Breaking TEE Security" on our website. All identified vulnerabilities were reported to Samsung and fixed at the end of 2019.

In this article, we share how we found vulnerabilities in TAs running in TEE-



GRIS, and how we exploited one TA to gain runtime control and further escalate privileges and gain access to the full TEE memory. To start, let's understand what TEE is and how it works in OS first.

In short, there are three types of separations that a robust TEE is expected to implement both in hardware and software:

- *Separation between TEE and REE*
- *Separation between TAs and TEE kernel*
- *Separation between TAs*

The transition between secure/non-secure modes is managed by a component called "secure monitor." This monitor is the primary interface between the TEE and REE and is the only component that can change the security state of a core.

While a fully isolated environment would be very secure, for it to be practically useful, it needs to communicate with other untrusted components running in Android. Communication between the REE and the TEE is done with the "Secure Monitor Call" (SMC). This instruction can be invoked by both worlds at EL > 0, which means that Android applications cannot directly initiate communication with the secure TEE. What normally happens is that the Linux kernel acts as a proxy and exposes a driver that can be used by apps to interact with the TEE.

The TEEGRIS kernel is a small component running in secure EL1. Even though it is small, it is not exactly a microkernel, and for instance, it integrates several drivers that can be used by TAs. Since the kernel is stored in plain text in the boot partition, it can be easily extracted and disassembled.

Although TAs in TEEGRIS can be easily disassembled to search for vulnerabilities, there are a set of protection mechanisms that prevent exploitation of those vulnerabilities some of which include ASLR, anti-roll-back and stack canaries.

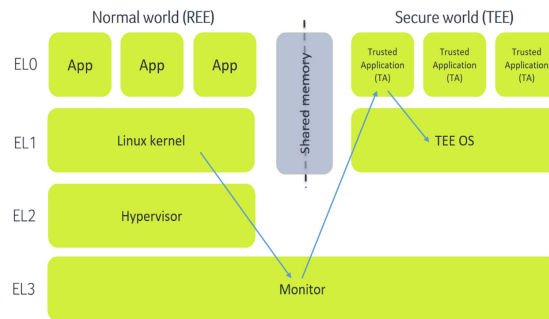
During the investigation, we overcame some used countermeasures in kernel and TAs. We found that by exploiting a type confusion in variable delivered to

*RISCURE EXPERTS AND THE REST OF THE SECURITY INDUSTRY have investigated TEE security in-depth over the last few years.*

TA enabled us to chaining three commands, we could obtain both read and write primitives in a particular TA which is a significant achievement since TAs are supposed to be secure and completely isolated from the untrusted Android OS.

**EXPLOITED MITIGATIONS XN (EXECUTE NEVER)**

**THE ATTACK WAS EXECUTED THROUGH MULTIPLE STEPS:**



*The attack route.*

This countermeasure is used in both kernel and TAs, so data memory is never executable, and code is never writable. We, however, obtained arbitrary read using type confusion vulnerability, write, and code execution within the TA, but XN only allows reusing existing code.

**ASLR AND KASLR**

We overcame this address space randomization of TAs and kernel by attempting multiple times as nothing prevents us from trying again to talk to the same TA, access the same random address, and see if we hit mapped memory. This can be repeated several times until the address we tried to access is mapped. Since the possible



*Jasmina Omic handles product manager services for Riscure and has worked in the security field for more than 14 years. In her current position she is driving security services, and focuses on providing guidance for Riscure customers, based on eight years of hands-on experience as a security analyst for IC and embedded devices, working under different certification schemes and testing approaches. [inforequest@riscure.com](mailto:inforequest@riscure.com) @Riscure*

random numbers are only 32k, finding lucky random can be usually achieved within less than one minute. PAN and PXN are in place to prevent the kernel from accessing or executing user-mode memory.

### STACK CANARIES (IN THE KERNEL AND TAs)

Stack canaries protect against buffer overflow vulnerabilities. We found a textbook stack-based buffer overflows, in which we control the size of the copy and the buffer contents. In total, up to 1275 bytes can be copied, enough for storing shellcode. However, the TA uses stack canaries, therefore exploitation of this vulnerability is not trivial. Since we have arbitrary read and ASLR bypass, we can simply read the value of `__stack_chk_guard` and fill it in our shellcode so that the canary verification succeeds.

### PRIVILEGE ESCALATION AND ACCESS TO FULL TEE MEMORY

We take our investigation one step further to gain runtime control of the TEE kernel. Historically, exploit mitigations in TEE OSes have been lackluster compared to other modern OSes. However, for our attack success multiple vulnerabilities in TEE need to be exploited.

The kernel exposes a driver that can be used by privileged TAs to map physical memory into the TA memory space. We will leverage this driver from the hacked TA to map secure registers and unprotect the TEE memory. Finally, we use the same hacked

TA to modify the hypervisor page tables and allow Android apps to map the (now unprotected) TEE memory with complete read/write access.

We focused our attack on registers since they contain all the configurations of peripherals, including the ones used to secure the TrustZone. The two registers that are commonly used for configuring TrustZone are TZASC and TZPC. If a TA could access any of them, it would be possible to read the contents of such registers but also write to them, removing the protection of TEE memory. In principle, modifying any of the two could allow the REE to access the TEE, effectively compromising the security provided by the TEE. We decided to target the TZASC using our exploit and remove the protection of the TEE memory space. We managed to map all the TEE memory into an Android application, meaning that we can:

- *Modify the code of TAs and TEE kernel since the permissions restrictions do not apply to the Android application;*
- *Bypass countermeasures implemented in the kernel such as KASLR, PAN, and PXN;*
- *Gaining full TEE control and performing attacks like modifying the phone unlock functionality implemented in the TEE (fingerprint or face recognition) to bypass the screen lock. ☒*



Media and game providers should consider security not only to protect their solutions but also to protect their users' privacy and company reputation.

One of the easiest ways to achieve high-level security is to collaborate with third-party labs like Riscure.

**riscure**

driving your security forward

Discover more about security and Riscure on our website, or scan the code:  
[riscure.com/market/media-and-multiple-service-operators](https://riscure.com/market/media-and-multiple-service-operators)

