Downgrade Attack on TrustZone

Yue Chen[1], Yulong Zhang[2], Zhi Wang[1], Tao Wei[2]   Florida State University[1], Baidu X-Lab[2]

ABSTRACT

Security-critical tasks require proper isolation from untrusted software. Chip manufacturers design trusted execution environments (TEEs) in their processors to secure these tasks. The integrity and security of the software in the trusted environment depend on the verification process of the system.

We find a form of attack that can be performed on the current *implementations* of the widely deployed ARM TrustZone technology. The attack exploits the fact that the trustlet (TA) or TrustZone OS loading verification procedure may **lack proper rollback prevention (or configuration) and may use the same verification key across versions**. If an exploit works on an out-of-date version, but the vulnerability is patched on the latest version, an attacker can still use the same exploit to compromise the update-to-date system, by downgrading the software to an older and exploitable version**.**

We did experiments on popular devices on the market including those from Google, Samsung and Huawei, and found that all of them have the risk of being attacked. Also, we show a real-world example to exploit Qualcomm's QSEE.

In addition, in order to find out which device images share the same verification key, pattern matching schemes for different device vendors are analyzed and summarized.

Background

Hardware manufacturers have introduced a new security mechanism called trusted execution environment (TEE), such as ARM TrustZone (TZ) technology, which has a widespread deployment in the digital world.

Basically, it has two separate worlds: one is called normal world and another one is secure world. Each world has its own operating system (OS) and user applications as shown in Figure 1. In practice, the normal world contains untrusted software, and the secure world contains trusted software. By design, the normal world is isolated from the secure world by hardware-enforced mechanisms. Simply put, the user applications and normal OS in the normal world are the traditional ones, while the ones in secure world have dedicated usages (e.g., digital rights management, authentication, etc.). We call the OS (privileged) in the secure world trusted OS, and the user-space applications (unprivileged) are called trustlets (also called trusted applications (TAs)). The two worlds communicate through the secure monitor.

When the trusted OS loads a trustlet from its unprivileged mode, it firstly **checks its signature** to see if it is signed by the right party and if the software is modified. This integrity check aims at removing the risk of loading tampered trustlets.

By design, starting from the hardware (e.g., eFuse/qFuse or ROM), a chain of trust is formed. This chain includes the bootloaders, the trusted OS, and derived certificates or keys. This step-by-step verification procedure ensures the integrity of the secure world. Potentially, the security and integrity of the whole system relies on this kind of verification.

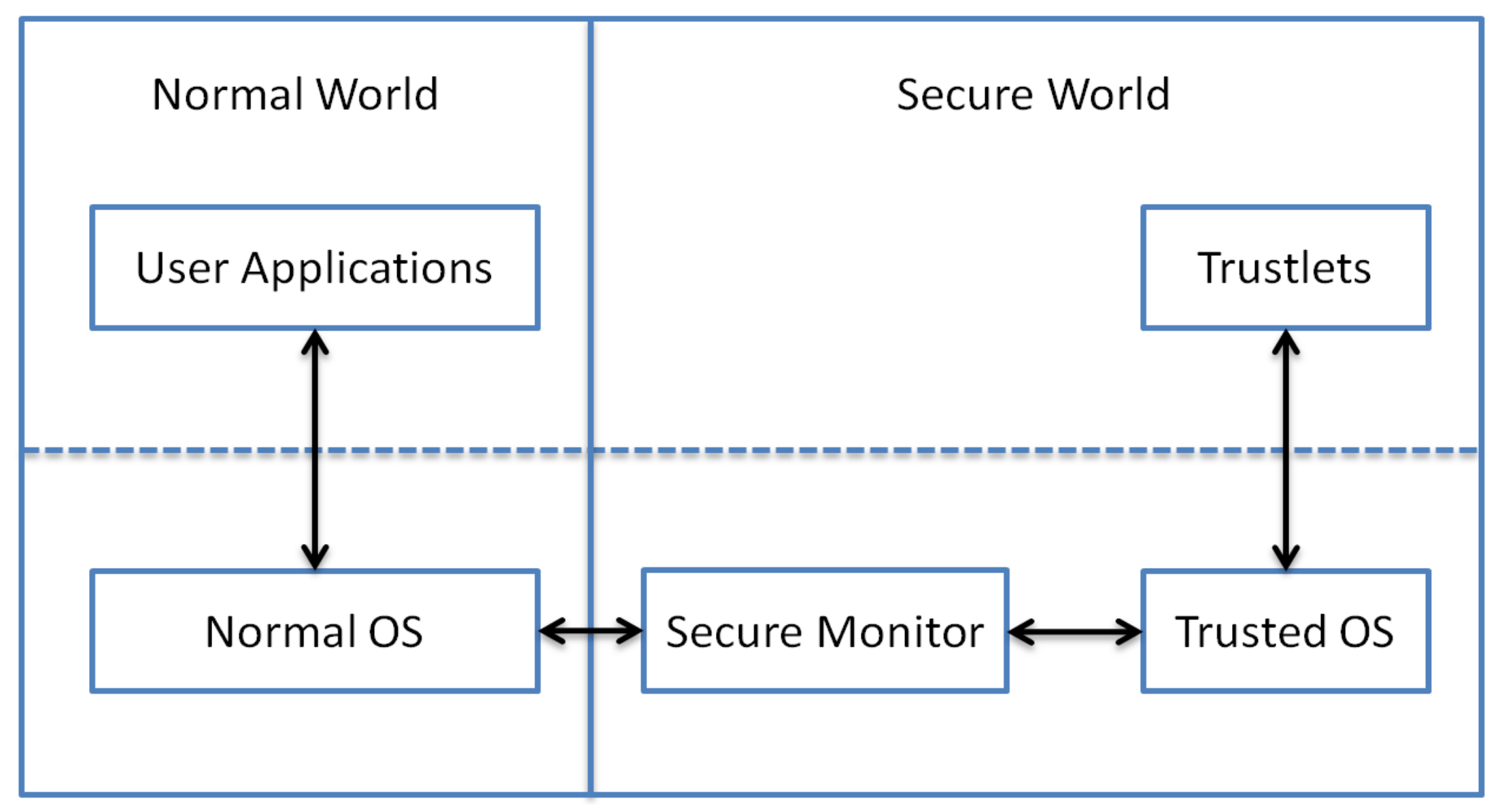

Figure 1: Overview of the typical TEE architecture

Attack Description

When a trustlet is being loaded, its cryptographic digital signature will be validated. Usually this signature is computed using a private key to encrypt the hash of the trustlet (or certain parts of the trustlet). Only the correct public key can decrypt it, and then the trustlet can be verified. For security purposes, the key pairs are based on the chain of trust, originating from the hardware.

Ideally, if the system is updated, the older software like trustlets cannot be loaded into the newer system. However, the trustlets can be replaced with their corresponding older versions. If the older version has a vulnerability and the newer version is patched, the attacker can still exploit the patched version by replacing it with an older one. We call it downgrade attack or rollback attack.

A successful exploit first needs to have the root privilege of the device (e.g., exploit another vulnerability), and then use this issue combined with other vulnerabilities to exploit the device, potentially compromising the TrustZone/TEE (even its kernel).
This vulnerability potentially impacts all the devices that are on the current market, though we do not have a thorough test and experiment.

We run a **real-world** exploit on Nexus 6 and successfully use the downgrade attack to exploit a QSEE privilege escalation vulnerability (CVE-2015-6639). This vulnerability could be exploited in version LMY48M, but not N6F26Y. We replace N6F26Y's widevine trustlets with the ones from LMY48M. Then the exploit works.

The secure boot procedure has a chain of trust. In its boot sequence, each software image to be executed is authenticated by software that was previously verified. This design prevents unauthorized or tampered code from being run. Different devices could have different boot stages, including different bootloaders. For example, for Qualcomm's MSM8960 chipset, SBL1 loads SBL2, and SBL2 loads tz and SBL3. Here the SBL stands for secondary bootloader.

Similar to the loading verification of trustlets, the TrustZone OS also needs loading verification. Hence, it is under the haze of downgrade attack, too. In our experiments with Google Nexus 5, the **TrustZone OS, even the bootloader, can be downgraded** without failure. We believe that this problem exists on more components in the chain of trust, potentially affecting the fundamental security of the whole system.

Compatibility Matching

If one needs to write a scanner to see if two image versions can load their trustlets mutually, or to cluster images into such groups, here we provide a pattern matching analysis on the TEE implementations of different vendors. To ensure the compatibility of the trustlets across versions, they may need the API compatibility with the TEE OS, and may need to pass the version check if there is any. Qualcomm does have the version control parameter `SW_ID`, but it is not properly configured, as most of the images have the value of zero (Trustonic's situation is similar). So the most important is signature verification. This procedure requires the correct public key or certificate to verify the trustlet. If it fails, the trustlet cannot be loaded. Until now in our experiments, we find that the keys are the determining factor for load verification. So next we will focus on this factor and describe our findings on different platforms.

Through reverse engineering and analysis, we try to extract the patterns about the trustlet/image pair replaceability.

Google Nexus:
If the hexadecimal pattern is
`30 82 [XX] [XX] 30 82 ...`
We can say that it is a beginning of a certificate with DER (Distinguished Encoding Rules) encoding. Note that **`30 82`** are **`0x30`** and **`0x82`** in hex, and the two bytes in between can be random.

Samsung (two TZ vendors):
Qualcomm:
The same as that of Google Nexus, as they both use Qualcomm's QSEE.

Trustonic / Mobicore:
`00 01 00 00 [256 bytes] 01 00 00 00 03 ...`
What needs to be compared is the 256-byte blob between "**`00 01 00 00`**" and "**`01 00 00 00 03`**". A standalone 256-byte blob cannot form an RSA-2048 public key or a standard certificate. However, since the RSA public exponent is usually set as **`65537`** or **`3`**, which can be easily embedded elsewhere, we guess this 256-byte (2048-bit) blob is the modulus of an RSA public key, which is enough for the comparison purpose.

Huawei:
The desired keys are not statically embedded.