École Polytechnique Fédérale de Lausanne

Investigating Downgrade Attacks of Trusted Applications on COTS Android Devices

by Michele Lizzit

Master Project Report

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer
Master Project Advisor

Dr. Marcel Busch
Master Project Supervisor

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

June 14, 2022

Abstract

Trusted Execution Environments (TEE) are a widely deployed technology that is present in most consumer-level smartphones to provide higher trust guarantees which additionally allows execution of Trusted Applications (TA). Most modern TEEs provide for anti-rollback features that prevent version downgrade attacks on TAs. This project is a comprehensive analysis of the current state of the art and TA anti-rollback implementations of major commercially available off-the-shelf (COTS) smartphone devices, with a particular focus on Samsung products and on Samsung's TEEGRIS TEE system.

As part of this analysis we have fully reverse engineered the TEEGRIS TA format and we have developed a software suite for TEEGRIS TA analysis. In the final part of this project we have collected a large and comprehensive dataset of firmware images from various sources in order to obtain a wider perspective on the current status of TA anti-rollback deployment and determine if TA anti-rollback features are correctly used by major vendors. We finally concluded that TA anti-rollback counters on TEEGRIS are never set to values other than 1, defeating the whole purpose of anti-rollback version counters. We also found that TA signing certificates are often shared across devices, thereby exposing a much larger attack surface than needed.

Introduction

TEEs are a widely deployed technology that is present in most consumer-level smartphones to provide higher trust guarantees and that allows execution of Trusted Applications (TAs). Most modern TEEs provide for anti-rollback features that prevent version downgrade attacks on TAs. Version downgrade attacks are of particular concern within TEEs since an attacker might be able to load an old, vulnerable, TA and exploit that TA to compromise the whole TEE. Therefore TEEs should provide for strong guarantees against version downgrade attacks and prevent loading of older TA versions once a new TA version is released.

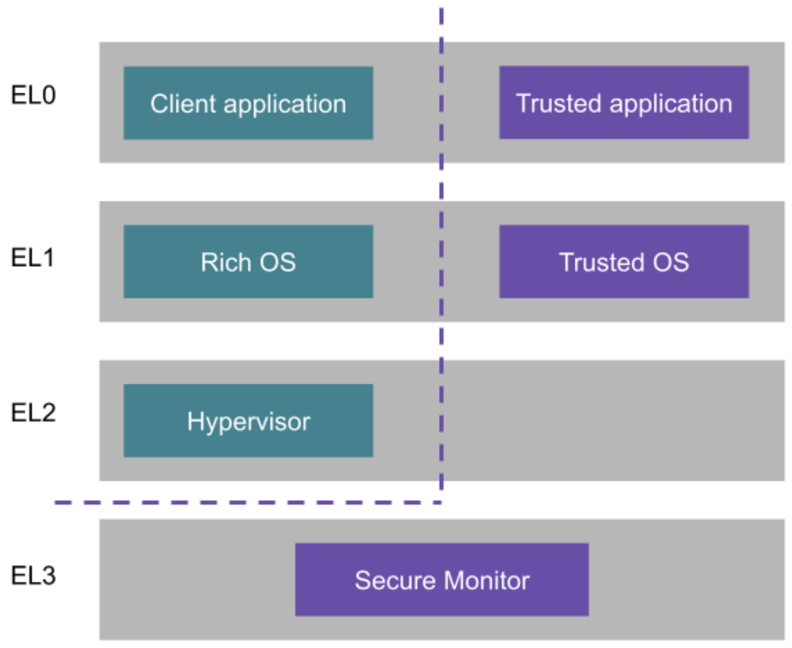
In this project we want to independently analyze the implementation of rollback prevention systems in common TEEs. This work focuses particularly on Samsung's own proprietary TEE *TEEGRIS*. TEEGRIS has already been subject to independent research [10, 14], however little has been studied in regard to TA rollback attacks and TA rollback prevention. Our goal in this project is to understand the implementation of TA anti-rollback systems in TEEGRIS and other TEEs. In particular, we are interested in (1) how TA images are signed and how many different keys are used to sign them, (2) how keys and certificates are shared across different models and regions, (3) if it possible to load cross-model TAs and (4) how TA evolve in time, how their rollback counters are incremented and how TAs differ across models distributed in different geographic regions. Finally we want to answer the question of whether Rollback Prevention is effective and to what degree. We formulate all these questions in a set of Research Questions (RQ) in Section results. As part of our analysis we have fully reverse engineered the TEEGRIS TA format, knowledge of which was incomplete according to previous research [10]. As a result of this research we are able to validate TA signatures, extract TA certificates and their corresponding public key from a TA image, read rollback counters, and have identified a flaw in the design of TA images that could potentially allow a Denial of Service attack. We have also developed a software suite for TEEGRIS TA analysis that allowed us to conveniently analyze a large amount of TA images. In the final part of this project we have collected a large and comprehensive dataset of firmware images from various sources in order to obtain a wider perspective on the current status of TA anti-rollback deployment and determine if TA rollback prevention features are correctly used by major vendors. A significant amount of engineering effort has been spent to efficiently store a large dataset of images that present a large percentage of duplicated content. Our software solution automatically downloads and recursively decompresses firmware images, in order to allow for content deduplication. The resulting tarball files are stored on a VDO device [12] that provides for block-level deduplication and data compression. Finally, our TA analyzer has been run on the dataset we collected, giving us an overview on how rollback counters are used and how widely TA anti-rollback protection is enabled within TAs.

We found out that in most smartphone models TEEGRIS TA anti-rollback is not enabled for the vast majority of TAs. Additionally, TAs that are built with support for rollback prevention never set their version counter to anything other than 1, effectively defeating its original purpose. However, we found out that TEEGRIS implements proper support for TA anti-rollback and therefore just updating the TAs by setting a proper version counter could result in strong TA anti-rollback guarantees.

Background

Trusted Execution Environments are a secure environment providing strong integrity guarantees against compromises of the Rich OS. As Rich OSes grow in complexity they become more prone to software bugs, therefore certain signed TAs might benefit from running in an isolated environment to protect against such attacks. TAs handle sensitive operations such as key management or secret handling, so that sensitive data is never directly accessible from the Rich OS. Despite the widespread use of TEEs in various products, security analyses related to these technologies are rarely discussed in public. Because of the secrecy surrounding the implementations of these systems by major vendors, such as Huawei, Samsung, and Qualcomm, the exact implementation of these technologies is not always discussed. Due to the complexity of implementing a TEE, the correct implementation can be challenging for vendors, as has already been highlighted by past research [4] [5]. TEEGRIS is a recent effort from Samsung to develop its own TEE, leveraging Arm TrustZone, for its own Android products. TEEGRIS has a structure similar to other common TEEs. The TEEGRIS kernel runs in EL1, while TAs run in EL0. Each TA generally has one or more companion apps running in the Normal World, that run in EL0 and interact with TAs via the TEE driver in the Rich OS (Linux) that in turn issues a SMC (Secure Monitor Call). TEEGRIS TAs are signed, and optionally encrypted, in a custom TA image format. Two TAs (`root_task` and `ACSD`) do not follow this scheme and are included in the `startup.tzar` archive in `boot.img`, which is instead verified by Secure Boot. These two TAs are responsible for loading and verifying additional TAs. They include a whole X509 certificate parser and perform other checks to ensure that the TA image has not been tampered with.

Figure 1: Diagram of privilege levels in Arm TrustZone



TEEs are susceptible to rollback attack, in which an older, vulnerable TA is loaded and then exploited to compromise the entire TEE. For this reason most TEE implement rollback protection mechanisms that prevent the loading of deprecated TA versions.

In TEEGRIS, each TEEGRIS TA image is identified by its UUID and can optionally contain an anti-rollback version counter. When such a TA is loaded a field is updated in the eMMC RPMB (Replay Protected Memory Block). Subsequent attempts to load the same TA UUID with a lower or absent anti-rollback version counter will fail.

Analysis and Design Choices

In this section, we will discuss the design choices made for collecting and analyzing a large dataset of TAs in order to answer our research questions. First, we cover the image format of TEEGRIS TAs. Then, we present the challenges that we faced in order to efficiently store a large amount of images, that include a significant percentage of duplicate content, and our solution to the problem. Finally we describe our approach for locating and extracting TAs from firmware images.

Samsung TEEGRIS TA format Samsung TEEGRIS has introduced a new image format, that differs significantly from the image format employed by previous TEE OSes such as Kinibi (that employed the MCLF TA image format) [9]. The new TEEGRIS TA image format is undocumented, but some public work on reverse engineering has been done in 2019[10]. Our work builds on previous publicly known information, reverse engineering effort and static analysis of the TEEGRIS image loader to provide a complete understanding of the TEEGRIS TA image format. TEEGRIS loads images from three different formats, called *SEC2*, *SEC3* and *SEC4*. Each format has its own unique features that are illustrated below, but images can be identified by looking at the first 4 magic bytes (that can be either `SEC2`, `SEC3` or `SEC4`). The `SEC1` format seems to be deprecated, and we were not able to find any TA image using this format. We can realistically assume that the `SEC1` format has never reached production stage and therefore is not interesting for what concerns our research.

NOTE: *Only items marked in italic are part of the signed blob within each TA image*

File structure (SEC2)

Field location	**Description**	**Samsung name**
0x00 - 0x04	SEC version	PKG_TYPE / magic
0x04 - 0x08	ELF size (s)	SIZE / ELF size
0x08 - +s	*ELF file*	*OUTPUT / content*
+0x00 - +0x01	Metadata size (m)	CN len
+0x00 - +m	Metadata	CN value
+0x00 - +0x04	Signature len (l)	SIGNATURE len
+0x00 - +l	Signature	SIGNATURE
+0x00 - +0x04	Cert. len (l)	CERT len
+0x00 - +l	X509 Certificate	CERT / CERTIFICATE

Table 1: Structure of the `SEC2` TA image format

File structure (SEC3)

Field location	**Description**	**Samsung name**
0x00 - 0x04	SEC version	PKG_TYPE / magic
0x04 - 0x08	ELF size (s)	SIZE / ELF size
0x08 - +s	*ELF file*	*OUTPUT / content*
+0x00 - +0x04	*rollback_1*	*Version len*
+0x00 - +0x04	*rollback_2*	*VERSION*
+0x00 - +0x01	Metadata size (m)	CN len
+0x00 - +m	Metadata	CN value
+0x00 - +0x04	Signature len (l)	SIGNATURE len
+0x00 - +l	Signature	SIGNATURE
+0x00 - +0x04	Cert. len (l)	CERT len
+0x00 - +l	X509 Certificate	CERT / CERTIFICATE

Table 2: Structure of the `SEC3` TA image format

File structure (SEC4)

Field location	**Description**	**Samsung name**
0x00 - 0x04	SEC version	PKG_TYPE / magic
0x04 - 0x08	ELF size (s)	SIZE / ELF size
0x08 - +s	*Encrypted ELF file*	*OUTPUT / content*
+0x00 - +0x04	*rollback_1*	*Version len*
+0x00 - +0x04	*rollback_2*	*VERSION*
+0x00 - +0x04	*TA IV length*	*TA IV len*
+0x00 - +0x10	*TA IV value*	*TA IV*
+0x00 - +0x04	*AUTH IV length*	*AUTH IV len*
+0x00 - +0x10	*AUTH IV value*	*AUTH IV*
+0x00 - +0x04	*SHA1 length*	*SHA1 length*
+0x00 - +0x14	*SHA1 checksum*	*SHA1 checksum*
+0x00 - +0x04	*AES key len*	*AES key len*
+0x00 - +0x20	*Encrypted AES key*	*Encrypted AES key*
+0x00 - +0x01	Metadata size (m)	CN len
+0x00 - +m	Metadata	CN value
+0x00 - +0x04	Signature len (l)	SIGNATURE len
+0x00 - +l	Signature	SIGNATURE
+0x00 - +0x04	Cert. len (l)	CERT len
+0x00 - +l	X509 Certificate	CERT / CERTIFICATE

Table 3: Structure of the `SEC4` TA image format

SEC version

This field indicates which TA image format is used. It can be `SEC2`, `SEC3` or `SEC4`:

- `SEC2` has no rollback protection and no TA encryption.
- `SEC3` has rollback protection and no TA encryption.
- `SEC4` has rollback protection and AES TA encryption.

ELF file

This is the main TA content, it is unencrypted for `SEC2` and `SEC3`.

Rollback counters (SEC3/4 only)

`rollback_1` is the length of the `rollback_2` field and is fixed to value `0x04`.
Checks are present in the `root_task` loader to ensure that `rollback_1` $== 4$
`rollback_2` is the actual version counter (often set to value 0x01).

Metadata

This field is a copy of the common name used in the certificate.
It is reasonable to assume that the loader uses the field to select which CA root certificate to validate the image with.
Samsung calls this field `CN`.

NOTE: *This is not part of the signed part of the image in neither SEC2 nor SEC3*

Signature

Signature scheme for `SEC2` is `PKCS#1 v1.5`.
Signature scheme for `SEC3` is PSS (`-sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:-1`).
Hash is computed using `SHA256`.
Additional resources: GP TEE API[7].

Certificate

This field is a standard X509 certificate in DER format.

TZAR

Certain TAs are located in an archive referred to as `startup.tzar`. `startup.tzar` is embedded in the kernel image within `boot.img`. TAs located in `startup.tzar` do not follow the same structure as other TAs and often are just composed of an unsigned ELF file. This is possible since `boot.img` is already verified trough other means. Our code repository includes a working TA extractor for `startup.tzar` that takes as input a raw `boot.img`, that was developed by building on previous work [15] and our research.

Sample TZAR archive within a `boot.img` (archive begins at `0x01b339c0`):

```
01b33950: d220 000a 0000 0000 0000 0000 0000 0000  . ..............
01b33960: 1c01 0000 630a 0009 d320 000a 0000 0000  ....c.... ......
01b33970: 0000 0000 0000 0000 1d01 0000 640a 0009  ............d...
01b33980: d420 000a 0000 0000 0000 0000 0000 0000  . ..............
01b33990: 1c01 0000 640a 0009 0000 0000 0000 0000  ....d...........
01b339a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
01b339b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
01b339c0: 7fa5 5441 0000 0001 9a0a 4d00 1b00 0000  ..TA......M.....
01b339d0: 0f00 0000 687d 0000 6269 6e2f 6c69 6274  ....h\}..bin/libt
01b339e0: 7a6c 642e 736f 007f 454c 4601 0101 0000  zld.so..ELF.....
01b339f0: 0000 0000 0000 0003 0028 0001 0000 0058  .........(.....X
01b33a00: 2600 0034 0000 0010 7b00 0000 0200 0534  &..4....\{......4
01b33a10: 0020 0006 0028 000f 000e 0001 0000 7084  . ...(........p.
01b33a20: 7900 0084 7900 0084 7900 0008 0000 0008  y...y...y.......
01b33a30: 0000 0004 0000 0004 0000 0001 0000 0000  ................
01b33a40: 0000 0000 0000 0000 0000 008c 7900 008c  ............y...
```

Our extractor works by looking for the TZAR magic bytes, then parsing the TZAR header to extract the TZAR length and finally, one begin position and archive length are known, cutting the `startup.tzar` from the `boot.img`.

Data storage Storing thousands of firmware images presents several challenges from a data storage standpoint. For each device model multiple firmware versions exist. Additionally for each firmware version multiple regions exists. Therefore each firmware image is identified by the (model, version, region) tuple. As a result the dataset will contain overall a lot of duplicated content that needs to be deduplicated for efficient storage. Samsung FW images are packed in a zip file, each file contains several archives. An example is presented below for the S10e (SM-G970F):

```
8496824320 AP_G973FXXUCFUH3_CL22340597_QB42324606_REV01_user_low_ship_meta_OS11.tar.md5
  14366720 BL_G973FXXUCFUH3_CL22340597_QB42324606_REV01_user_low_ship.tar.md5
  51271680 CP_G973FXXUCFUH3_CP19998134_CL22340597_QB42324606_REV01_user_low_ship.tar.md5
 127242240 CSC_OMC_OVF_G973FOVFCFUH3_CL22344568_QB42334723_REV01_user_low_ship.tar.md5
 127180800 HOME_CSC_OMC_OVF_G973FOVFCFUH3_CL22344568_QB42334723_REV01_user_low_ship.tar.md5
```

A naive approach might be to attempt deduplication directly of the TAR archive, this however will result in very poor performance as the TAR archive themselves contain compressed content, as shown below.

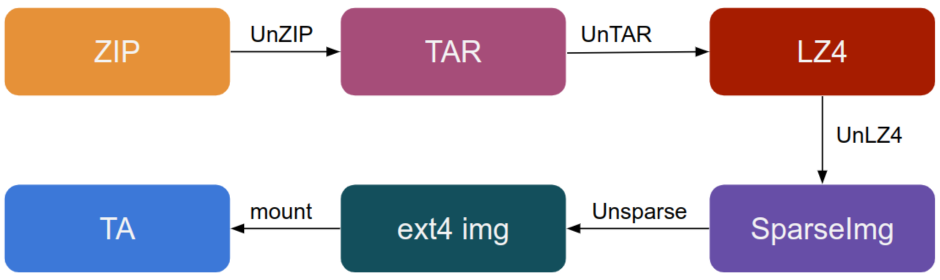
```
  22805059 ott 23  2020 boot.img.lz4
      1156 ott 23  2020 dqmdbg.img.lz4
    537929 ott 23  2020 dtbo.img.lz4
    108551 ott 23  2020 dt.img.lz4
         0 ott 23  2020 meta-data
  35251876 ott 23  2020 recovery.img.lz4
3699394998 ott 23  2020 system.img.lz4
 830391363 ott 23  2020 userdata.img.lz4
      3638 ott 23  2020 vbmeta.img.lz4
 735385955 ott 23  2020 vendor.img.lz4
```

Therefore, our approach aims at improving deduplication by decompressing all LZ4 compressed images that appear within each TAR archive. Since LZ4 compression is an optional feature in the Samsung firmware image format, decompressing images does not break the compatibility with other existing tools. The main limitation with this approach is that files within TAR archives are not necessarily block-aligned, this could result in performance issues if deduplication is performed using a fixed block size. To overcome this issue a future idea could be to block-align files within tarballs by adding extra "padding" files with the sole purpose of achieving block-alignment. This approach however, in order to obtain optimal performance, would also require adding padding within each system image (which is usually in ext4 format), since the files that we aim to dedeplicate are contained there. We also discovered that it is common for similar zip firmware images to contain some fully identical tarballs. To better handle this case and to avoid relying too much on the deduplication layer we leverage hardlinks. Each tarball is hashed upon downloading, before performing our optimizations, and its hash stored in an SQLite archive, if another identical tarball is found on the filesystem the tarball is discarded and a hardlink is created instead.

To perform deduplication and block-level compression we rely on VDO [12]. We briefly considered other deduplication alternatives, such as BorgBackup [3], and found to provide marginally better performance than VDO for this type of content, but their use was not investigated in-depth. Our full stack is composed of an XFS filesystem of logical size 80TB, provisioned on a virtual block-device managed by VDO. VDO is configured to use a single 8TB physical disk as a backing store.

TA extraction process TAs are stored in both the `system` and `vendor` images, in multiple paths. To access and extract such TAs we first need to extract the relevant tarball files (`"AP_"` and `"BL_"` tarballs), then decompress the LZ4-compressed images. We only decompress `system.img` and `vendor.img` since those are the two only two images that we determined to contain TAs. If the archive contains a `super` image, we unpack the super image first [1]. Each image is an `ext4` or `ext2` filesystem image in sparse image format, as specified in [2]. Therefore we first unsparse the image to obtain a valid ext2/ext4 image, then we parse the ext2/ext4 image looking for files that match one of the following patterns: `[ ".*-0000-0000-0000-.*", ".*\.tlbin" ]`. Matching files are copied to the final results folder. If a `startup.tzar` archive is found in `boot.img` then also the content of the `startup.tzar` archive is copied in the final results folder.

Figure 2: TA extraction process from a zip firmware image



This whole extraction process is performed using a Python-only implementation of common parser and decompression libraries, some of which we improved or rewrote from scratch. The advantage of our approach is that we are able to keep all intermediate files in memory, mostly by using Python BytesIO buffers, and therefore the extractor only performs one sequential read of the relevant tarball archives in the whole extraction process. This is particularly relevant since deduplicating filesystems such as `VDO` are very compute-heavy and offer low read and write performance, therefore minimizing reads and seeks is crucial to obtain a reasonable running time of the extractor software.

Results

To understand the implementation of TA anti-rollback systems in TEEGRIS and other TEEs we want to answer the following research questions:

RQ1: How are TA images signed and how many different keys are used to sign them?

RQ2: Are keys and certificates shared across different models and regions and is it possible to load cross-model Trusted Applications?

RQ3: How do TA evolve in time? How are their rollback counters incremented and how TAs differ across models distributed in different geographic regions?

RQ4: Is Rollback Prevention is effective and to what degree?

To answer the aforementioned questions we collected a large and comprehensive dataset of firmware images. Since each firmware image is identified by *device model*, *version* and *region*, the following set of *regions* was selected to provide for a diverse enough sample, while at the same time following to the need of minimize space utilization:

Samsung country code	**Country name**
ACR	Ghana
ATO	Austria
CTU	Uruguay
KSA	Saudi Arabia
MXO	Mexico
SWC	Switzerland
VAU	Australia
XEO	Poland
XEU	United Kingdom
XXV	Vietnam

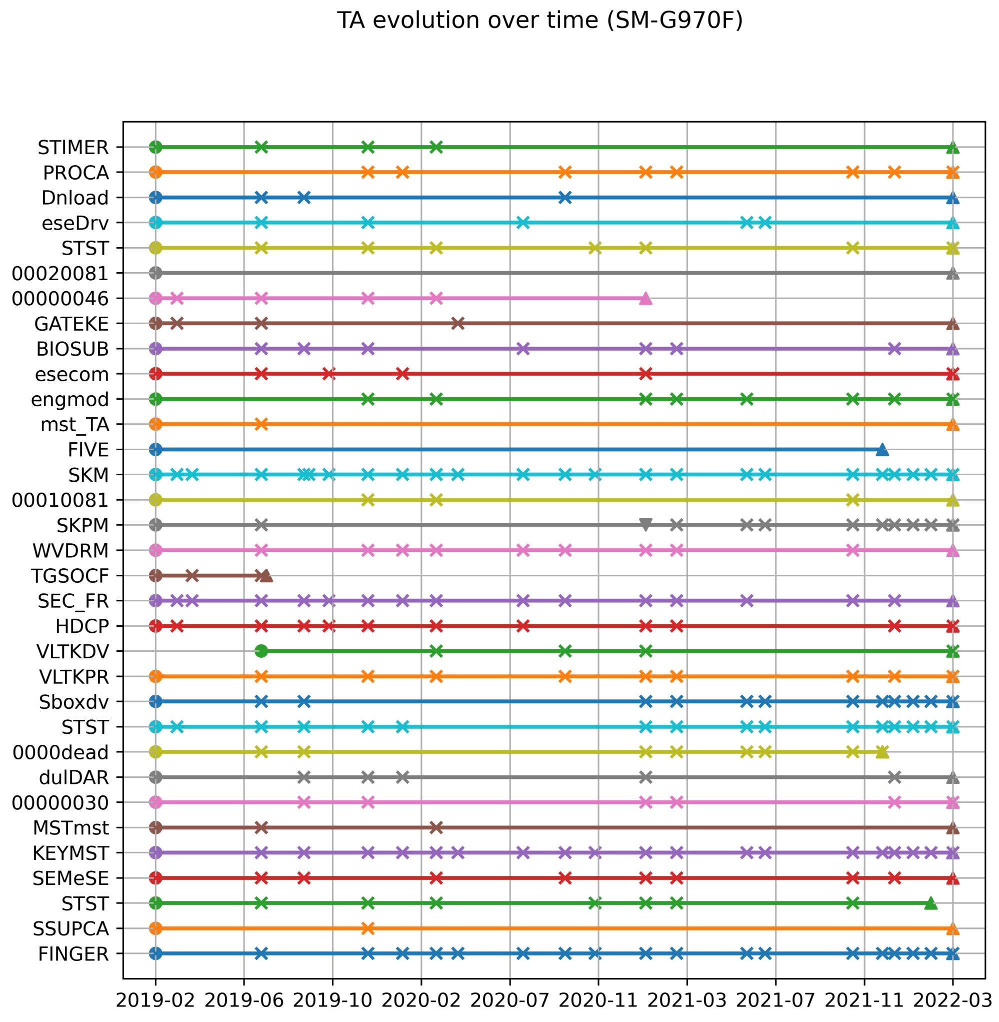
Table 4: List of the *region* sample that we selected for analysis

We also restricted firmware image collection to the list of Samsung devices that officially support Samsung KNOX [13]. Firmware versions were selected at random, as this was determined to be the best way of creating a diverse sample for the purpose of our analysis. In total we collected 1675 firmware images, from 122 unique device models. On the collected sample we run our automated analysis tool and extracted a total of 33198 TAs with 47 unique UUIDs.

SEC version	Number of TAs
SEC2	10039
SEC3	5060
SEC4	0
Other (Kinibi)	18099

Table 5: SEC version summary

Figure 3: TA evolution over time on the S10e (G970F). Marked locations indicate a change in the TA binary.



TA UUID	**Friendly name**	**Unique signature certificates**
00000000-0000-0000-0000-53626f786476	Sboxdv	2
00000000-0000-0000-0000-736e61707370	snapsp	1
00000000-0000-0000-0000-4d53546d7374	MSTmst	5
00000000-0000-0000-0000-46494e474552	FINGER	6
00000000-0000-0000-0000-53454d655345	SEMeSE	6
00000000-0000-0000-0000-4b45594d5354	KEYMST	10
00000000-0000-0000-0000-564c544b5052	VLTKPR	10
00000000-0000-0000-0000-535355504341	SSUPCA	4
00000000-0000-0000-0000-64756c444152	dulDAR	3
00000000-0000-0000-0000-5447534f4346	TGSOCF	4
00000000-0000-0000-0000-474154454b45	GATEKE	10
00000000-0000-0000-0000-0050524f4341	PROCA	7
00000000-0000-0000-0000-505256544545	PRVTEE	3
00000000-0000-0000-0000-656e676d6f64	engmod	7
00000000-0000-0000-0000-564c544b4456	VLTKDV	9
00000000-0000-0000-0000-545241545453	TRATTS	2
00000000-0000-0000-0000-000000020081		7
00000000-0000-0000-0000-000000010081		7
00000000-0000-0000-0000-0000534b504d	SKPM	9
00000000-0000-0000-0000-00575644524d	WVDRM	8
00000000-0000-0000-0000-00535453540f	STST	1
00000000-0000-0000-0000-000000534b4d	SKM	10
00000000-0000-0000-0000-534258505859	SBXPXY	1
00000000-0000-0000-0000-5354494d4552	STIMER	4
ffffffff-0000-0000-0000-000000000030	0	6
00000000-0000-0000-0000-000000000046	F	7
00000000-0000-0000-0000-657365636f6d	esecom	7
00000000-0000-0000-0000-446e6c6f6164	Dnload	6
00000000-0000-0000-0000-54412d48444d	TA-HDM	1
00000000-0000-0000-0000-00535453540a	STST	1
00000000-0000-0000-0000-544545535355	TEESSU	1
00000000-0000-0000-0000-000046495645	FIVE	7
00000000-0000-0000-0000-6d73745f5441	mst$_{T}A$	5
00000000-0000-0000-0000-42494f535542	BIOSUB	9
00000000-0000-0000-0000-000048444350	HDCP	8
00000000-0000-0000-0000-00535453540b	STST	10
00000000-0000-0000-0000-534543535452	SECSTR	1
00000000-0000-0000-0000-00535453540d	STST	1
00000000-0000-0000-0000-00535453540c	STST	10
00000000-0000-0000-0000-46494e474502	FINGE	6
00000000-0000-0000-0000-5345435f4652	SEC$_{F}R$	9
00000000-0000-0000-0000-0053545354ab	STST	9
00000000-0000-0000-0000-494363447256	ICcDrV	1
00000000-0000-0000-0000-657365447276	eseDrv	3
00000000-0000-0000-0000-53465453494d	SFTSIM	2
00000000-0000-0000-0000-534543445256	SECDRV	2
00000000-0000-0000-0000-00000000dead		10

Table 6: X509 fingerprint report. A total of 28 unique certificate fingerprints were found.

The following raw results were additionally obtained:

```
KINIBI/TEEGRIS summary:
5 models are KINIBI-only
31 models are TEEGRIS-only
86 models are HYBRID
Of which:
8 models have upgraded from KINIBI to HYBRID
3 models have upgraded from HYBRID to TEEGRIS-only
0 models have upgraded from KINIBI to TEEGRIS-only

Rollback counters values (aggregated):
None: 10039
1: 5060
Rollback counters values (model_averages):
None: 66.60 %
1: 33.40 %

11/47 TAs have upgraded from SEC2 to SEC3:
33/47 TAs are only SEC2
3/47 TAs are only SEC3

Upgraded from SEC2 to SEC3:
00000000-0000-0000-0000-505256544545 (PRVTEE)       (present in 7/57 models with 64.80 % frequency)
00000000-0000-0000-0000-000000020081 ( )            (present in 45/57 models with 98.81 % frequency)
00000000-0000-0000-0000-46494e474552 (FINGER)       (present in 21/57 models with 100.00 % frequency)
00000000-0000-0000-0000-4b45594d5354 (KEYMST)       (present in 57/57 models with 100.00 % frequency)
00000000-0000-0000-0000-0000534b504d (SKPM)         (present in 52/57 models with 98.08 % frequency)
00000000-0000-0000-0000-0053545354ab (STST)         (present in 52/57 models with 100.00 % frequency)
00000000-0000-0000-0000-00575644524d (WVDRM)        (present in 49/57 models with 97.96 % frequency)
00000000-0000-0000-0000-000000534b4d (SKM)          (present in 56/57 models with 100.00 % frequency)
00000000-0000-0000-0000-000000010081 ( )            (present in 45/57 models with 98.81 % frequency)
00000000-0000-0000-0000-474154454b45 (GATEKE)       (present in 55/57 models with 100.00 % frequency)
00000000-0000-0000-0000-00000000dead ()             (present in 55/57 models with 90.90 % frequency)
SEC2-only:
00000000-0000-0000-0000-736e61707370 (snapsp)       (present in 2/57 models with 60.77 % frequency)
00000000-0000-0000-0000-4d53546d7374 (MSTmst)       (present in 42/57 models with 100.00 % frequency)
00000000-0000-0000-0000-53454d655345 (SEMeSE)       (present in 23/57 models with 98.70 % frequency)
00000000-0000-0000-0000-564c544b5052 (VLTKPR)       (present in 55/57 models with 100.00 % frequency)
00000000-0000-0000-0000-535355504341 (SSUPCA)       (present in 25/57 models with 100.00 % frequency)
00000000-0000-0000-0000-64756c444152 (dulDAR)       (present in 18/57 models with 93.84 % frequency)
00000000-0000-0000-0000-5447534f4346 (TGSOCF)       (present in 8/57 models with 20.81 % frequency)
00000000-0000-0000-0000-0050524f4341 (PROCA)        (present in 42/57 models with 100.00 % frequency)
00000000-0000-0000-0000-656e676d6f64 (engmod)       (present in 36/57 models with 95.41 % frequency)
00000000-0000-0000-0000-564c544b4456 (VLTKDV)       (present in 49/57 models with 99.08 % frequency)
00000000-0000-0000-0000-545241545453 (TRATTS)       (present in 5/57 models with 74.52 % frequency)
00000000-0000-0000-0000-53465453494d (SFTSIM)       (present in 2/57 models with 66.67 % frequency)
00000000-0000-0000-0000-00535453540f (STST)         (present in 7/57 models with 77.38 % frequency)
00000000-0000-0000-0000-534258505859 (SBXPXY)       (present in 10/57 models with 100.00 % frequency)
00000000-0000-0000-0000-5354494d4552 (STIMER)       (present in 32/57 models with 71.43 % frequency)
ffffffff-0000-0000-0000-000000000030 (0)            (present in 42/57 models with 94.84 % frequency)
00000000-0000-0000-0000-000000000046 (F)            (present in 41/57 models with 56.78 % frequency)
00000000-0000-0000-0000-657365636f6d (esecom)       (present in 43/57 models with 100.00 % frequency)
00000000-0000-0000-0000-446e6c6f6164 (Dnload)       (present in 37/57 models with 97.68 % frequency)
00000000-0000-0000-0000-00535453540a (STST)         (present in 7/57 models with 100.00 % frequency)
00000000-0000-0000-0000-544545535355 (TEESSU)       (present in 1/57 models with 10.00 % frequency)
00000000-0000-0000-0000-000046495645 (FIVE)         (present in 42/57 models with 91.89 % frequency)
00000000-0000-0000-0000-6d73745f5441 (mst_TA)       (present in 42/57 models with 100.00 % frequency)
00000000-0000-0000-0000-42494f535542 (BIOSUB)       (present in 48/57 models with 100.00 % frequency)
00000000-0000-0000-0000-000048444350 (HDCP)         (present in 46/57 models with 100.00 % frequency)
00000000-0000-0000-0000-00535453540b (STST)         (present in 55/57 models with 98.57 % frequency)
```

```
00000000-0000-0000-0000-534543535452 (SECSTR)        (present in 5/57 models with 100.00 % frequency)
00000000-0000-0000-0000-00535453540d (STST)          (present in 7/57 models with 80.95 % frequency)
00000000-0000-0000-0000-00535453540c (STST)          (present in 55/57 models with 97.27 % frequency)
00000000-0000-0000-0000-5345435f4652 (SEC_FR)        (present in 48/57 models with 100.00 % frequency)
00000000-0000-0000-0000-494363447256 (ICcDrV)        (present in 1/57 models with 28.57 % frequency)
00000000-0000-0000-0000-657365447276 (eseDrv)        (present in 11/57 models with 100.00 % frequency)
00000000-0000-0000-0000-534543445256 (SECDRV)        (present in 4/57 models with 75.00 % frequency)
SEC3-only:
00000000-0000-0000-0000-46494e474502 (FINGE)         (present in 25/57 models with 99.81 % frequency)
00000000-0000-0000-0000-53626f786476 (Sboxdv)        (present in 17/57 models with 100.00 % frequency)
00000000-0000-0000-0000-54412d48444d (TA-HDM)        (present in 10/57 models with 100.00 % frequency)

Region report:
00000000-0000-0000-0000-53626f786476 (Sboxdv)        (present in 8/8 regions with 66.02 % frequency)
00000000-0000-0000-0000-736e61707370 (snapsp)        (present in 4/8 regions with 3.06 % frequency)
00000000-0000-0000-0000-4d53546d7374 (MSTmst)        (present in 8/8 regions with 92.38 % frequency)
00000000-0000-0000-0000-46494e474552 (FINGER)        (present in 8/8 regions with 26.86 % frequency)
00000000-0000-0000-0000-53454d655345 (SEMeSE)        (present in 8/8 regions with 72.18 % frequency)
00000000-0000-0000-0000-4b45594d5354 (KEYMST)        (present in 8/8 regions with 100.00 % frequency)
00000000-0000-0000-0000-564c544b5052 (VLTKPR)        (present in 8/8 regions with 99.70 % frequency)
00000000-0000-0000-0000-535355504341 (SSUPCA)        (present in 8/8 regions with 55.02 % frequency)
00000000-0000-0000-0000-64756c444152 (dulDAR)        (present in 8/8 regions with 63.23 % frequency)
00000000-0000-0000-0000-5447534f4346 (TGSOCF)        (present in 8/8 regions with 5.60 % frequency)
00000000-0000-0000-0000-474154454b45 (GATEKE)        (present in 8/8 regions with 99.70 % frequency)
00000000-0000-0000-0000-0050524f4341 (PROCA)         (present in 8/8 regions with 93.49 % frequency)
00000000-0000-0000-0000-505256544545 (PRVTEE)        (present in 4/8 regions with 2.94 % frequency)
00000000-0000-0000-0000-656e676d6f64 (engmod)        (present in 8/8 regions with 86.18 % frequency)
00000000-0000-0000-0000-564c544b4456 (VLTKDV)        (present in 8/8 regions with 93.29 % frequency)
00000000-0000-0000-0000-545241545453 (TRATTS)        (present in 7/8 regions with 7.59 % frequency)
00000000-0000-0000-0000-000000020081 ( )             (present in 8/8 regions with 93.64 % frequency)
00000000-0000-0000-0000-000000010081 ( )             (present in 8/8 regions with 93.64 % frequency)
00000000-0000-0000-0000-0000534b504d (SKPM)          (present in 8/8 regions with 98.16 % frequency)
00000000-0000-0000-0000-00575644524d (WVDRM)         (present in 8/8 regions with 96.51 % frequency)
00000000-0000-0000-0000-00535453540f (STST)          (present in 3/8 regions with 4.74 % frequency)
00000000-0000-0000-0000-000000534b4d (SKM)           (present in 8/8 regions with 99.81 % frequency)
00000000-0000-0000-0000-534258505859 (SBXPXY)        (present in 7/8 regions with 33.43 % frequency)
00000000-0000-0000-0000-5354494d4552 (STIMER)        (present in 8/8 regions with 70.86 % frequency)
ffffffff-0000-0000-0000-000000000030 (0)             (present in 8/8 regions with 89.49 % frequency)
00000000-0000-0000-0000-000000000046 (F)             (present in 8/8 regions with 47.96 % frequency)
00000000-0000-0000-0000-657365636f6d (esecom)        (present in 8/8 regions with 93.63 % frequency)
00000000-0000-0000-0000-446e6c6f6164 (Dnload)        (present in 8/8 regions with 89.57 % frequency)
00000000-0000-0000-0000-54412d48444d (TA-HDM)        (present in 7/8 regions with 33.43 % frequency)
00000000-0000-0000-0000-00535453540a (STST)          (present in 4/8 regions with 5.76 % frequency)
00000000-0000-0000-0000-544545535355 (TEESSU)        (present in 1/8 regions with 1.00 % frequency)
00000000-0000-0000-0000-000046495645 (FIVE)          (present in 8/8 regions with 83.32 % frequency)
00000000-0000-0000-0000-6d73745f5441 (mst_TA)        (present in 8/8 regions with 92.38 % frequency)
00000000-0000-0000-0000-42494f535542 (BIOSUB)        (present in 8/8 regions with 96.30 % frequency)
00000000-0000-0000-0000-000048444350 (HDCP)          (present in 8/8 regions with 94.94 % frequency)
00000000-0000-0000-0000-00535453540b (STST)          (present in 8/8 regions with 98.95 % frequency)
00000000-0000-0000-0000-534543535452 (SECSTR)        (present in 3/8 regions with 3.90 % frequency)
00000000-0000-0000-0000-00535453540d (STST)          (present in 3/8 regions with 5.12 % frequency)
00000000-0000-0000-0000-00535453540c (STST)          (present in 8/8 regions with 96.63 % frequency)
00000000-0000-0000-0000-46494e474502 (FINGE)         (present in 8/8 regions with 72.20 % frequency)
00000000-0000-0000-0000-5345435f4652 (SEC_FR)        (present in 8/8 regions with 96.30 % frequency)
00000000-0000-0000-0000-0053545354ab (STST)          (present in 8/8 regions with 97.87 % frequency)
00000000-0000-0000-0000-494363447256 (ICcDrV)        (present in 2/8 regions with 1.12 % frequency)
00000000-0000-0000-0000-657365447276 (eseDrv)        (present in 8/8 regions with 42.18 % frequency)
00000000-0000-0000-0000-53465453494d (SFTSIM)        (present in 2/8 regions with 2.62 % frequency)
00000000-0000-0000-0000-534543445256 (SECDRV)        (present in 4/8 regions with 3.71 % frequency)
00000000-0000-0000-0000-00000000dead ( )             (present in 8/8 regions with 88.49 % frequency)
```

Answer 1 (RQ1): *How are TA images signed and how many different keys are used to sign them?*

Each TA image embeds an X509 certificate in DER format. The certificate is used to sign the signed part of the TA image and not all fields in the TA image are signed. The signature scheme varies depending on the TA image version. `SEC2` images follow the PKCS1 v1.5 signature scheme while `SEC3` use RSA Probabilistic Signature Scheme (PSS) (OpenSSL: `-sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:-1`). Hash is computed using the SHA256 algorithm, regardless of the TA image version. This is compatible with the information present in the GP TEEGRIS certification [8].

The following unique CA key is used to sign all TAs:
`AuthorityKeyIdentifier: 79e590a7a3bf9a4be3620823295aec7a0e5bbff6`
Certificate CN field can be called `samsung_drv` or `samsung_ta`, these two CN also match the two permission groups present in TEEGRIS. The integrity of each TA image is validated against the CA certificate embedded in the TEEGRIS kernel. This process is performed by the `root_task` and `ACSD` TAs.

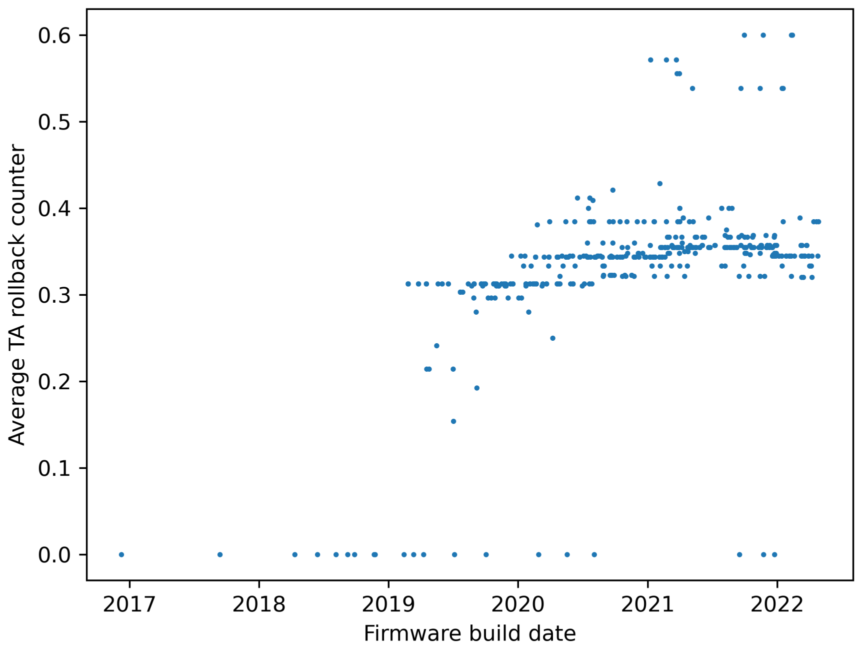
Answer 2 (RQ2): *Are keys and certificates shared across different models and regions and is it possible to load cross-model Trusted Applications?*

The following CA certificate is shared across all devices:
`AuthorityKeyIdentifier: 79e590a7a3bf9a4be3620823295aec7a0e5bbff6`
Additionally, TA certificates are often reused across multiple TAs and multiple models. Only 28 different TA certificates exist in total, all signed with the same CA key. We were able to successfully load cross-model TAs, that is a TA that was not built for that specific device. Apart from incompatibilities that may arise from different versions of the TEEGRIS kernel we were not able to find any mechanism to prevent cross-device TA loading. This needlessly increases attack surface, as a vulnerable TA can be loaded also on a device that did not originally ship with that TA.

Answer 3 (RQ3): *How do TA evolve in time? How are their rollback counters incremented and how TAs differ across models distributed in different geographic regions?*

We compared TAs from images related to the same model and same firmware version, across different regions. From our analysis we were not able to identify any difference in TAs between regions for the same model and same firmware version. We can therefore conclude that TAs do not differ across geographical regions. It is important to note that we explicitly excluded all Qualcomm-based devices from the analysis, as we focused only on Exynos models. While tracking the evolution of TAs over time we have observed that many devices ship with both Kinibi and TEEGRIS TAs, in the report we have used the word `HYBRID` to describe these models, however based on current public knowledge we can speculate that Kinibi and TEEGRIS are not compatible and cannot run on the same device. We can therefore also speculate that in HYBRID models only TEEGRIS TAs are actually loaded by the system. The purpose of leaving Kinibi TA images within the firmware image, without a Kinibi TEE is not known. During our analysis we have seen that over time models have upgraded, through an OTA firmware upgrade, from Kinibi to HYBRID (8/122 models) and others (3/122 models) have upgraded from HYBRID to TEEGRIS-only. We have also observed that newer models ship with only TEEGRIS and older models have not been upgraded at all.

Figure 4: TA anti-rollback counter evolution over time. An anti-rollback counter value of 0 indicates no rollback protection (SEC2). Average values are computed by considering all Trusted Applications that are packed within the same firmware image.



When considering only TEEGRIS-enabled systems we observed that the vast majority of TAs (10039/15099 in total, and 66.6% of the TAs in a model on average) is of type SEC2. As detailed in the Analysis section, SEC2 TAs lack any form of rollback protection, but their loadability can be prevented by upgrading to SEC3 and loading a SEC3 TA with the same UUID once. We have seen this happening on a small number of TAs (11/47 total unique TAs), that have upgraded from SEC2 to SEC3, this list however includes some of the most critical TAs such as GATEKEEPER and KEYMASTER.

Answer 4 (RQ4): *Is Rollback Prevention is effective and to what degree?*

Overall TEEGRIS provides for strong rollback protection mechanisms. However, TAs do not correctly make use of the features provided by TEEGRIS, by not enabling rollback protection

(SEC2 TAs) or by improperly setting rollback counters (all counters are currently set to a value of 1). This exposes TEEGRIS to downgrade attacks in which a vulnerable version of a TA is loaded onto the system and exploited in order to compromise the entire TEEGRIS environment. Additionally TAs and devices do not use device-unique CA certificates, therefore increasing attack surface: if a vulnerable TA is discovered on a device it can be loaded and exploited on any device. This is particularly relevant since a device that has never seen a TA will not have rollback counters set for that specific TA and will therefore allow loading regardless of whether a new patched TA has been released or not.

android-extract *android-extract* is a set of Python 3 tools that we have developed as part of this research for extracting TA binary files from .zip Android system images, collecting images from various sources and efficiently storing firmware images.

Repository structure:

Component	Description
analyze	Analyze a single TA file
extractor	Extract TA files from a firmware image
fetch	Fetch firmware images from various public sources

Table 7: Structure of the *android-extract* repository

The final report for each TA is available both in JSON format, and outputted to stdout in a human-readable format. Here is a sample run of the analyzer:

```
TA b'HDCP' : SEC2
No rollback protection (SEC2)
Metadata: b'samsung_drv:samsung_drv'
67748
Success
True
Signature verified
CRT fingerprint: 4e6686a70330b673401be41302ce872bdc18ab5e
CRT subject:  samsung_drv:samsung_drv
CA ID:  79e590a7a3bf9a4be3620823295aec7a0e5bbff6

[
    {
        "filename": "/tmp/k/00000000-0000-0000-0000-000048444350",
        "success": true,
        "header_bytes": "53454332000108a4",
        "is_mclf_format": false,
        "header_valid": true,
19          "human_name": "HDCP",
20          "sec_version": 2,
21          "elf_last8": "0100000000000000",
22          "rollback_version": null,
23          "signature_ok": true,
24          "crt_fingerprint": "4e6686a70330b673401be41302ce872bdc18ab5e
                "
25      }
26  ]
```

On line 1 the output shows the SEC version and the friendly TA name. The friendly TA name is obtained by taking the UUID as a hex byte representation and converting it to ASCII. Line 2 shows rollback counters (if any). Line 3 shows the metadata field. Following, on line 4, is the length of the signed component, and on line 5 to 7 the outcome of the signature validation process. On line 8 and following is the certificate information, with the computed certificate fingerprint, subject field and ID of the signing CA (`AuthorityKeyIdentifier` field). The final part (from line 12 on) shows the the machine-friendly JSON output.

TEEGRIS TA image flaws We found that the nor `SEC2`, nor `SEC4`, nor `SEC4` formats sign the whole image. In particular signed part does not include the header, or the ELF length fields. In our opinion this is a major design flaw. A trivial attack would be to change the `SEC2` header to `SEC3`, and adjust the unsigned ELF length field accordingly as to have the last 8 bytes of the ELF become the new rollback counters field. This is however mitigated since `SEC2` and `SEC3` use different signature schemes. Nevertheless, signature schemes of `SEC3` and `SEC4` are the same, therefore it might be possible to downgrade an encrypted `SEC4` to `SEC3`, have the system crash while trying to execute an encrypted ELF without decrypting, and setting anti-rollback counters in such a way to prevent the correct TA from loading anymore, leading to a permanent DoS on the whole device. This however has not been tested as no `SEC4` TA is known. The opposite however is not true, as an upgrade from `SEC3` to `SEC4` would simply result in a decryption failure and would not set the rollback counters in the RPMB region of the device eMMC.

Future work As part of our research we have identified several interesting areas, not within the scope of this research, and that therefore we were unable to explore in depth. An interesting target for future research could be the `ACSD` and `root_task` loader TAs, their critical role in the functioning of TEEGRIS, and the ability to access them directly from the Rich OS make them a promising research target. Additionally, both can be efficiently tested by fuzzing. Another interesting component is the TEEGRIS OS itself, in particular in its handling of the eMMC RPMB and RPMB key management. It is currently undocumented and to our knowledge has never been studied independently.

Related Work

The choice of the topic for this project has been influenced by Riscure's blog post "Breaking TEE Security Part 1: TEEs, TrustZone and TEEGRIS"[10], dated 2021, where the authors describe how TA antirollback protection mechanisms have only recently been implemented on TEEGRIS. The article provides for a global overview of the TEEGRIS system, its security mitigations and an initial analysis of its TA image format. Other sources were of particular inspiration while developing this project. In particular Alexander Tarasikov, in 2019 [14] provides for a good introduction to Samsung's TEEGRIS OS. Another source of inspiration was the talk from Federico Menarini and Martijn Bogaard at OffensiveCon22 "Bug Hunting S21's 10ADAB1E FW" [11], which provides from a very recent analysis of certain features of the TEEGRIS OS. We would also like to highlight the importance of Google Project Zero blog post "Trust Issues: Exploiting TrustZone TEEs" [6], that was among the first to publicly raise the attention on rollback attacks in TEEs and the importance of robust rollback prevention mechanisms.

Conclusion

TEEs have become ubiquitous among COTS Android smartphones, and their importance is constantly increasing given the stronger security needs of modern consumers. We also see the increasing impact of TA rollback attacks and highlight the importance of properly implementing mitigations against TA rollback attacks. We focus our work on Samsung's own TEE "TEEGRIS" and analyze its antirollback protection features. By building on public past knowledge and static analysis of the TEEGRIS TA loader we reverse-engineer the TEEGRIS TA image format. We then build a large and comprehensive repository of TEEGRIS-enabled firmware images and analyze TAs within each image to obtain a global understanding of the current deployment status of anti-rollback protection in TEEGRIS. Unfortunately, while anti-rollback protection overall is correctly implemented in TEEGRIS, we find that TAs do not properly use the anti-rollback capabilities offered by TEEGRIS by not setting version counters to anything else than 1. This practice has been observed among all devices and all firmware versions. This completely defeats the original purpose of anti-rollback protection and exposes TEEGRIS to TA rollback attacks. We also find out that CA certificates are shared among multiple devices, allowing for loading TAs even on devices for which the TA was not originally intended, thereby needlessly increasing the attack surface.

Bibliography

[1] android.com. In: Accessed: 02/06/2022. URL: https://source.android.com/devices/tech/ota/dynamic%5C_partitions/implement.

[2] android.googlesource.com. In: Accessed: 02/06/2022. URL: https://android.googlesource.com/platform/system/core/+/9640b30dd78d71ca56ac6d701f8c7874cbd51038/fastboot/fastboot.c.

[3] borgbackup.org. In: Accessed: 02/06/2022. URL: https://www.borgbackup.org/.

[4] Marcel Busch, Johannes Westphal, and Tilo Mueller. "Unearthing the TrustedCore: A Critical Review on Huawei's Trusted Execution Environment". In: *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020. URL: https://www.usenix.org/conference/woot20/presentation/busch.

[5] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. "SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems". In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, pp. 1416–1432. DOI: 10.1109/SP40000.2020.00061.

[6] Project Zero Gal Beniamini. In: 24/07/2017, accessed: 02/06/2022. URL: https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html.

[7] globalplatform.org. In: Accessed: 02/06/2022. URL: https://globalplatform.org/wp-content/uploads/2020/10/GP-TEE-2020%5C_01-CR-1.0%5C_GP200007-Certificate-and-Certification-Report%5C_20200922.pdf.

[8] globalplatform.org. In: Accessed: 02/06/2022. URL: https://globalplatform.org/wp-content/uploads/2020/10/GP-TEE-2020_01-CR-1.0_GP200007-Certificate-and-Certification-Report_20200922.pdf.

[9] Lee Harrison, Hayawardh Vijayakumar, Rohan Padhye, Koushik Sen, and Michael Grace. "PARTEMU: Enabling Dynamic Analysis of Real-World TrustZone Software Using Emulation". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 789–806. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/harrison.

[10] Federico Menarini. In: 23/02/2021. URL: https://www.riscure.com/blog/tee-security-samsung-teegris-part-1.

[11] Federico Menarini and Martijn Bogaard. In: *OffensiveCon22 - Bug Hunting S21's 10ADAB1E FW*. 2022. URL: https://www.youtube.com/watch?v=XvmtEwkG_Cc.

[12] redhat.com. In: Accessed: 02/06/2022. URL: https://access.redhat.com/documentation/en-us/red%5C_hat%5C_enterprise%5C_linux/7/html/storage%5C_administration%5C_guide/vdo.

[13] samsungknox.com. In: Accessed: 02/06/2022. URL: https://www.samsungknox.com/en/knox-platform/supported-devices.

[14] Alexander Tarasikov. In: Accessed: 02/06/2022. URL: https://allsoftwaresucks.blogspot.com/2019/05/reverse-engineering-samsung-exynos-9820.html.

[15] Alexander Tarasikov. In: 26/03/2021. URL: https://gist.github.com/astarasikov/f47cb7f46b5193872f376fa0ea842e4b.