USENIX Security '24 Artifact Appendix: Spill the TeA: An Empirical Study of Trusted Application Rollback Prevention on Android Smartphones

Marcel Busch Philipp Mao Mathias Payer
EPFL, Lausanne, Switzerland

A Artifact Appendix

A.1 Abstract

The number and complexity of Trusted Applications (TAs, applications running in Trusted Execution Environments — TEEs) deployed on mobile devices has exploded. A vulnerability in a single TA impacts the security of the entire device. Thus, vendors must rapidly fix such vulnerabilities and revoke vulnerable versions to prevent rollback attacks, i.e., loading an old version of the TA to exploit a known vulnerability. In this paper, we assess the state of TA rollback prevention by conducting a large-scale cross-vendor study. First, we establish the largest TA dataset in existence, encompassing 35,541 TAs obtained from 1,330 firmware images deployed on mobile devices across the top five most common vendors. Second, we identify 37 TA vulnerabilities that we leverage to assess the state of industry-wide TA rollback effectiveness. Third, we make the counterintuitive discovery that the uncoordinated usage of rollback prevention correlates with the leakage of security-critical information and has far-reaching consequences potentially negatively impacting the whole mobile ecosystem. Fourth, we demonstrate the severity of ineffective TA rollback prevention by exploiting two different TEEs on fully-updated mobile devices. In summary, our results indicate severe deficiencies in TA rollback prevention across the mobile ecosystem.

Our artifact aims to show that our study on TA rollback prevention can be reproduced.

A.2 Description & Requirements

We obtained our dataset from multiple terabytes of Android firmware images. This artifact appendix demonstrates where to obtain these images from and provides the extraction tools to obtain the information relevant for our dataset. Further, we provide evidence for the major claims in our paper.

A.2.1 Security, privacy, and ethical concerns

We do not expect any security, privacy, or ethical concerns from executing our code.

A.2.2 How to access

Visit https://github.com/HexHive/spill_the_tea/tree/sec-ae.

A.2.3 Hardware dependencies

No special hardware requirements needed.

A.2.4 Software dependencies

We performed all of our processing on a Ubuntu 22.04 machine and require Python 3.10 plus the dependencies references in our GitHub repository. The proprietary firmware images that our dataset is based on can be obtained from the various sources described in the paper.

A.2.5 Benchmarks

Our dataset is extracted from mutliple terabytes of Android firmware images that we obtained from various online sources. This artifact appendix demonstrates how to use our extraction tools to recreate or extend our dataset.

Our artifact contains the names of the firmware images (`data/firmware.txt`) we used. Based on these names, it should be possible to obtain the corresponding firmware image from the internet and reproduce our extraction pipeline.

A.3 Set-up

A.3.1 Installation

Follow the installation steps in the `README.md` of our GitHub repository.

A.3.2 Basic Test

Run the Trusted Application extraction on the firmware image as described in the `README.md`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Created functional tooling to create the TA dataset.
(C2): Identified vulnerable and rollbackable TAs (Table 1 shows this data in aggregated form and Figure 9 on a per device model level over time).
(C3): Captured state of ecosystem TA rollback.
(C4): Uncoordinated TA rollback increases have negative side effects for the TA ecosystem.

A.4.2 Experiments

(E1): *[TA Extraction and Parsing] [30 human-minutes + 15 compute-minutes (depending on bandwidth) + 15GB disk]*:
This experiment shows how to extract TAs from firmware images. Our artifacts document the vendors, models, and firmware versions used for our study. Since our dataset spans multipe terabytes of firmware images, we demonstrate the reproduction of our firmware extraction based on two representative examples and provide the result of the extraction for the whole dataset.
The ODM, TEE, Vendor combination in Table 1 are documented in `data/config.py`. The individual firmware images of our study are documented in `data/firmware.txt`.
For instance, `data/config.py` reveals that Xiaomi has a phone model with code name `dandelion` (sold under the name "Xiaomi Redmi 9A" according to gsmarena.com[1]) and that this model is using the MediaTek/BeanPod TEE. Further, all `/fw/xiaomi/dandelion/*` entries in `data/firmware.txt` list the firmware images for this model that we used in our study.
Additionally, `data/config.py` contains the Samsung Galaxy S10 (SM-G973F[2]) which is running the TEEGris TEE. The `/fw/samsung/SM-G973F/*` entries in `data/firmware.txt` list the firmware images for this model.
Preparation: Follow the SETUP steps in our `README.md`.
Execution: Follow the steps in `Example 1` and `Example 2` in our `README.md`.
Results: The experiment is successful if the TAs are extracted successfully for the dandelion image and the SM-G973F image. Additionally, the `report.json` for the SM-G973F should contain the metadata for the TAs contained in this image. This result demonstrates our tooling to automatically process firmware images (**C1**).

(E2): *[Reproducing TA Rollback Analysis] [5 human-minutes + 5 compute-minutes]*:
In E1, we demonstrated how to use our tooling to automatically extract and parse TAs from proprietary Android firmware images. In this experiment, we aim to reproduce the TA Rollback Analysis captured in Table 1 and Figure 9 in our paper.
Table 9 (Appendix) contains the vulnerable TAs we used in our study. We use this data in machine-readable form, see `data/samsung_vuln_db.py` and `data/xiaomi_vuln_db.py`.
Preparation: The `fw/` directory in our repository contains the metadata for all TAs in our dataset organized by OEM, device, region, and firmware version.
To focus on a single device and a single region, we prepare the Samsung SM-G973F for the VD2 region as follows. `mkdir -p fw2/samsung/SM-G973F/`, `cp -r fw/samsung/SM-G973F/VD2 fw2/samsung/SM-G973F/`, and change the base directory for the dataset in `data/config.py` to `fw_path = "./fw2"`.
Execution: Run the script `paper_scripts/gen_number_better.py` and compare the `teegris` entries for `samsung` in the generated `numbers.json` with Figure 9i, for instance `cat numbers.json | python -m json.tool`.
Results: The output should indicate one neutralized rollback attack (`"nr_TAs_neutralized": 1`) due to an increased rollback counter (blue triangle for SKPM in Figure 9i), 12 vulnerabilities (`"nr_public_vulns": 12`) indicated by orange X marks, and 9 TAs that are rollbackable (`"nr_TAs_rollbackable": 9`) in the latest firmware image (all TAs in Figure 9i that end with an orange circle). Note that Figure 9i does not show all TAs for this device due to space constraints, and the 9th rollbackable TA is the `00000000-0000-0000-0000-000000000046` found on this device.
With this experiment, we show that our tooling can identify vulnerable and rollabckable TAs (**C2**).
Further, `paper_scripts/numbers.json` contains the aggregated raw data of this experiment based on all the firmware images listed in `data/firmware.txt` (Table 1). Changing `data/config.py` back to `fw_path = "./fw"` and running `paper_scripts/gen_number_better.py` allows to recreate the results in `paper_scripts/numbers.json`. Given the full dataset, our analysis reveals that 29 out of 51 fully-updated devices are vulnerable to at least one vulnerable TA that can be rolled back. We capture the state of TA rollback attacks with these numbers (**C3**).

(E3): *[Negative Side Effects] [5 human-minutes + 5 compute-minutes]*: This experiment aims to show instances of cross-OEM leakage and, thus, provides evidence for negative side effects of uncoordinated TA rollback counter usage.

[1] https://www.gsmarena.com/xiaomi_redmi_9a-10279.php
[2] https://www.gsmarena.com/samsung_galaxy_s10-9536.php

As can be seen in Figure 9i and j, the WVDRM TA on the Samsung S10 (SM-G973F) and Samsung A10 (SM-A105F) have received uncoordinated rollback counter increases. In this experiment, we show that the Samsung S10 is negatively impacted by this leakage.

Preparation: In this experiment, we focus on the first (G973FXXS3ASJG) and the latest (G973FXXUGHVK1) firmware images for the Samsung S10 (SM-G973F, region: VD2) in `fw/samsung/SM-G973F/VD2/`, and the first (A105FDDS3ATC1) and last (A105FDDU8CVH4) images for the Samsung A10 (SM-A105F, region: XFA) in `fw/samsung/SM-A105F/XFA/`.

Execution: Find the information for the `WVDRM` TA in all four firmware images. For instance, for the G973FXXS3ASJG image `cat report.json | python -m json.tool` in `fw/samsung/SM-G973F/VD2/G973FXXS3ASJG/` which should format and print the report. We are looking for the entry where `"human_name": "WVDRM"`.

Results: The `WVDRM` TA of the SM-A105F changes its header format from SEC2 to SEC3, enabling and setting the TA rollback counter for this TA after a sequence of vulnerabilities (see Figure 9j). In contrast, the `WVDRM` for the SM-G973F does not receive a rollback counter increase during the entire lifetime of this TA. Consequently, the rollback counter usage of the SM-A105F `WVDRM` leaks information about the availability of vulnerable `WVDRM` versions on the SM-G973F (**C4**).

A.5 Notes on Reusability

Our Android firmware extraction tooling can be used and extended to automate tasks related to the processing of proprietary firmware images.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.