Spill the TeA: An Empirical Study of Trusted Application Rollback Prevention on Android Smartphones

Marcel Busch Philipp Mao Mathias Payer

EPFL, Lausanne, Switzerland

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.

1 Introduction

On Android, Trusted Applications (TAs) are vendor-signed dynamically loadable modules that extend the features provided by a Trusted Execution Environment (TEE). The TA loader is responsible of verifying the TA's signature. Thus, only authentic TAs can be loaded. TAs provide many security-critical tasks including secret-based authentication (e.g., login via PIN, pattern, or password), biometric authentication (e.g., fingerprint and face identification), playback of protected digital content (e.g., digital rights management), disk encryption, mobile banking, and generic cryptographic key management. In this study, we are concerned with vulnerabilities in TAs putting these security-critical tasks at risk.

TEEs on Android devices leverage the ARM TrustZone architectural extension. TrustZone creates an isolated execution context separated from the commonly known Android software stack. This separated context provides integrity and confidentiality guarantees for its components (TEE OS and TAs). Code in the normal world can only interact with TAs through well-defined and protected interfaces. TAs can interact with the TEE OS through system calls.

Rollback Exposure. TAs are an attractive target for attackers. Their number and complexity is ever-increasing. TAs are a prime attack surface to breach the TEE because TAs are directly accessible from the Android context. Either an attacker breaches a specific TA and has direct access to confidential data, or the breach serves as a initial foothold to compromise the entire TEE [1, 35, 49]. Unsurprisingly, TAs of *all* major TEE implementations have been exploited in the past [1, 8, 31–33, 35]. Such breaches are usually followed by the rollout of updates that fix the underlying vulnerabilities. However, attackers retain access to the original, signed, vulnerable TA. Without *rollback prevention*, meaning to increase a *rollback counter*, this vulnerable TA may still be loaded into the TEE and exploited.

In this paper, we extend the existing vulnerability lifecycle model [2, 27, 43] to cover this underexplored rollback attack vector. The vulnerability lifecycle captures the phases from the point of introduction of a vulnerability into a codebase until the point where all systems affected by the vulnerability are patched. The study of the vulnerability lifecycle is important as it helps to mature software security practices and workflows, and provides insights into the impact of security efforts. While prior measurement studies on different targets than TAs have produced a number of valuable insights into the vulnerability lifecycle [2, 27], our study provides novel insights on the *rollback exposure* for TAs.

Challenges and Solutions. Studying the TA rollback attack vector poses several unique challenges.

- Capturing the complexity of the TA ecosystem. Several parties including vendors, chipset providers, third-party

TEE implementations, and third-party TAs as well as their interdependencies have to be taken into account.

- Understanding and measuring the implication of security-related events, like vulnerability disclosures and rollback counter increases. The impact of these events on individual products and the ecosystem is underexplored.
- Identifying reasonable past and present industry trends to extrapolate the future development of the ecosystem.
- Obtaining a representative ground truth of TAs and past security-related events. The (lack of) availability of historical firmware, proprietary file formats, and vague information on public TA vulnerabilities pose difficulties.

Our study is the first large-scale systematic measurement of its kind and addresses all of these challenges. To address these challenges, we systematize and model the TA ecosystem in Section 5 by surveying and reverse engineering the proprietary TEE software stacks of a representative subset of device series. This ecosystem model allows us to reason about the implications of security-related events and guides our discovery of real-world instances of rollback-counter-related side effects in Section 7. For these real-world instances, we obtain a representative ground-truth dataset of TAs and security-related events as described in Section 6.1 and Section 7.1. This dataset required us to define the criteria for representativeness, obtain and process 1,330 proprietary firmware images, and aggregate the heterogeneous information across these sources. Finally, from the past and current industry trends, we extrapolate and discuss future developments of the ecosystem in Section 8.

Study Outline. We present a large-scale cross-vendor study on the effectiveness of TA rollback prevention surveying the prevalence of rollback mitigations. First, we collect a representative dataset of 1,330 firmware images containing 35,541 TAs deployed on recent mobile devices spanning the five most common vendors. Next, we reverse engineer and fully automate the extraction of TAs and their metadata from these firmware images. Then, we establish a ground truth of 37 TA vulnerabilities that we leverage to investigate the effectiveness of TA rollback prevention. Conversely, we form a subset of TAs with rollback counter usages and investigate the corresponding leakage of security-critical information and its implications for the ecosystem. Lastly, we conduct a study on the effectiveness of TA rollback prevention by investigating if we can load and breach vulnerable TAs in the most widely-used TEEs, namely Samsung's TEEGRIS, Trustonic's Kinibi, Qualcomm's QSEE, and Beanpodtech's BeanPod.

Results. Our results indicate the lack of effective TA rollback prevention on an industry-wide scale and discovers a net-negative security impact on the entire ecosystem in the case of uncoordinated rollback prevention. Based on our ground truth of 37 vulnerabilities, we discovered 2,582 cases of rollback exposure out of which 265 remain vulnerable on the latest firmware images (note that one vulnerable TA can affect multiple device models). These rollback exposure cases affect all major TEE platforms. Consequently, all platforms can potentially be breached using n-day exploits.

Counterintuitively, we discover that the uncoordinated usage of rollback counters for individual products has implications for other products by *the same vendor* and potentially *the ecosystem as a whole*. We observe that the ad-hoc and uncoordinated usage of rollback counters correlates with severe vulnerabilities in TAs and, thus, leaks security-critical information affecting all products making use of the affected TA. We observe this phenomenon across products by the same vendor and anticipate this leakage to occur across products by different vendors.

Finally, to demonstrate the severity of the flawed state of TA rollback prevention, we carry out two case studies where we breach two major TEE platforms on fully updated devices.

Contributions. Our contributions are the following:

- We establish the largest dataset of TA images and vulnerable TAs in existence to investigate TA rollback prevention.
- We investigate the TA rollback exposure using 37 manually verified vulnerabilities by conducting a large-scale cross-vendor study on the effectiveness of TA rollback prevention. We discover that rollback prevention is not used securely across the entire industry.
- We discover that the uncoordinated usage of TA rollback prevention of individual products implies the leakage of security-critical information potentially leading to a net-negative security for the entire ecosystem.

Responsible Disclosure. In this study, we discovered several TA rollback attacks affecting recent fully-updated mobile devices. We responsibly disclosed our findings to all affected vendors. Samsung confirmed that TA rollback prevention is applied to models released after the Galaxy S22, since the patch required a huge update that included internal infrastructure. According to Samsung, this patch cannot be applied to the devices we reported to be vulnerable to TA rollback attacks. Although several recent devices stay vulnerable to TA rollback attacks, we were not assigned an official identifier for these issues (i.e., CVE or SVE). Further, we disclosed our findings to Xiaomi, Oppo, Vivo, and MediaTek, and inquired about how they will address TA rollback attacks on their products. We did not receive a response to our inquiry yet and are waiting for a response from these vendors.

2 Motivation for Up-to-Date TAs

In this section, we motivate the need for up-to-date TAs by reviewing ARM TrustZone and TA rollback prevention.

ARM TrustZone Preliminaries. ARM is the dominant CPU architecture on mobile devices. Thus, the vast majority

of available devices leverage ARM TrustZone as the hardware primitive for their TEE. Modern ARMv8-A CPUs, as typically used for mobile devices offer four privilege levels (i.e., exception levels in ARM terminology) where EL3 denotes the highest and EL0 the lowest level.

ARM TrustZone splits these exception levels into two execution modes, the untrusted normal world and the trusted secure world. For the OS level (EL1), this mode also results in a dedicated way to use the memory management unit, leading to the world split and logical components as illustrated in Figure 1. We commonly refer to the combination of the TEE OS (S-EL1) and TAs (S-EL0) as the *TEE*. Additionally, dedicated hardware protects the TEE memory and peripherals from normal world accesses.

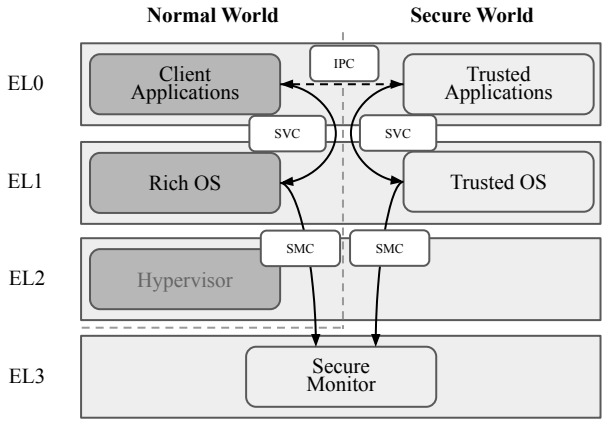

Figure 1: ARMv8 privilege levels.

Cryptographically signed TA images reside in the rich OS's file system and are dynamically loaded into the TEE at runtime. By design, TAs expose their services to the untrusted normal world. Thus, in comparison to the TEE OS, TAs comprise an attack surface that is directly accessible to attackers. Additionally, TAs constitute the largest attack surface of the TEE. As we show in Section 7, we identified 187 unique TAs available for Samsung devices, and 106 unique TAs available for Xiaomi devices, two popular brands in the mobile market.

Vulnerabilities in TAs have severe consequences. TAs have been used to directly compromise the integrity and confidentiality guarantees expected by security-sensitive services, including mobile payment [32], full-disk encryption [25], and biometric authentication [49]. Further, TA vulnerabilities served as initial footholds to compromise the entire TEE [1, 35, 49]. These breaches emphasize that vulnerable TAs can subvert the purpose of the TEE, and threaten the security guarantees expected by TEE-protected assets like biometric identifiers, payment data, and cryptographic keys. Given this threat potential, users and vendors have a great interest in fixing vulnerable TAs quickly, and deploying these fixes to devices as soon as possible.

TA Rollback Prevention. Figure 2 illustrates a simplified vulnerability lifecycle for a TA vulnerability. The solid section of the arrow indicates the time in which the vulnerable TA can potentially be exploited. The red solid section is especially interesting to attackers since publicly known vulnerabilities, or even readily available n-day exploits, can be used to compromise a vulnerable non-revoked version of a TA. Consequently, vendors *must* enforce the loading of up-to-date TAs.

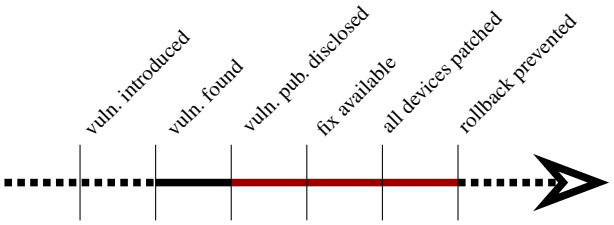

Figure 2: The TA vulnerability lifecycle. Attackers can leverage publicly disclosed vulnerabilities up until the rollback to a vulnerable TA version is prevented.

TAs, in contrast to the TEE OS, are dynamically loaded and unloaded from the untrusted normal world. Hence, the vendor has to ensure that *only up-to-date TAs can be loaded*. The mechanism of choice for OP-TEE and the majority of commercial TEEs (see Section 5) is *TA rollback prevention* based on a secure storage.

Rollback prevention is typically desired on a per-TA basis, as opposed to a group of TAs or all TAs on a given device. One major property achieved with per-TA rollback prevention is modularity. If certain features are not used, then the corresponding TA is not loaded, thus the trusted computing base is not unnecessarily increased. Additionally, TAs are developed individually and may originate from different sources as we discuss in Section 5. Per-TA rollback prevention facilitates the coordination of patching and increasing rollback counters. Thus, the TA loader needs to (1) uniquely *identify* a TA and (2) have access to a protected TA-specific *version counter*. This information needs to be part of a signed segment of each TA and securely persisted on the device. After verifying the TA's authenticity, the TA loader can leverage this information to distinguish TAs and keep track of each TA's version counter. If an attacker tries to load an outdated version, the loader will reject the corresponding TA.

In practice, taking OP-TEE as an example, platform designers can leverage a replay-protected memory block (RPMB) as a secure storage to enable TA rollback prevention [52]. RPMB is a feature of embedded multi-media cards (eMMCs), and introduced in the eMMC4.4 standard, a widely available technology. Figure 3 illustrates how a Client Application requests the helper daemon (`supplicant`) to load a TA from the regular filesystem (`FS`) in steps (1a)-(1b)-(1c). After verifying the TA's authenticity, the TA Loader requests the `ta_ver.db` from the `supplicant` ((2a)-(2b)). This file contains all TA identifiers and TA versions. The RPMB and the TA loader have

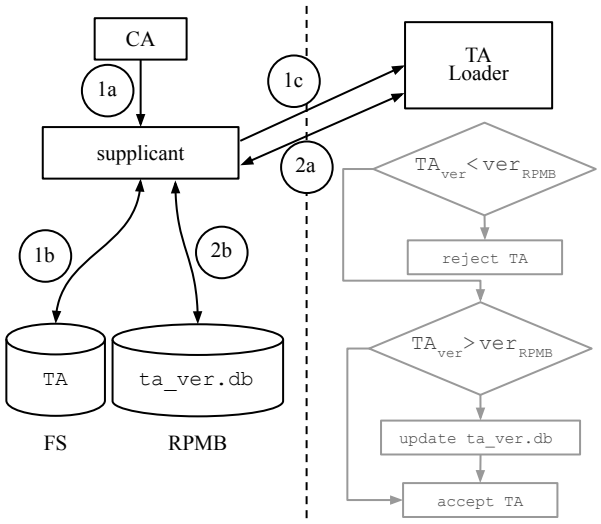

Figure 3: When a Client Application (CA) loads a TA (1a-1b-1c), the TA Loader keeps track of the TA versions using an RPMB-based database (2a-2b).

access to a shared key deployed by the manufacturer. This key, which is kept secret from the Normal World, is used to generate hash-based message authentication codes (HMACs) for messages exchanged between the RPMB and the TA Loader to guarantee authenticity. Thus, the TA loader can leverage the authentic `ta_ver.db` for TA version bookkeeping. The TA Loader must handle three scenarios. If the version found in the TA header (TA_{ver}) is lower than the version maintained in the `ta_ver.db` file (ver_{RPMB}), the TA is rejected. If the TA_{ver} is greater than the ver_{RPMB}, the TA Loader writes to the RPMB to update the `ta_ver.db`. This write operation makes use of the shared key and a write counter present in the RPMB to guarantee authenticity and freshness. Lastly, if the TA_{ver} is equal to the ver_{RPMB}, the Loader accepts the TA.

3 Research Overview

In this section, we outline the challenges of this measurment study and briefly discuss how we addressed them.

TA Ecosystem Complexity. The complexity of the TEE ecosystem is hidden by its lack of transparency, closed-source nature, absence of documentation, and inherent security-senstitive nature, making it challenging to comprehend and analyze without privileged insider knowledge. In fact, even when this insider knowledge is given, researchers are restrained by the secrecy requirements imposed by industry collaborators. For instance, the PartEmu prototype [22] cannot be publicly released because of non-disclosure agreements.

This lack of transparency makes it challenging to draw a clear picture of the ecosystem that includes vendors, chipset providers, third-party TEE implementations, and third-party TAs as well as their interdependencies.

Implications of Security-Related Events. Given the fact that TA code is shared amongst different parties in the ecosystem, security-related events originating from one party may affect another party. These events include the release of security advisories, rollback counter increases, third-party vulnerability reports, and updates in general. The implications of these events are unexplored, and require a thorough modeling and empirical validation. Creating a robust model that yields interesting insights for real-world TAs is a challenging task.

Identify Trends. Already in 2017 Google P0 discussed the issue of TA rollback attacks [4]. Since then, new players entered the TEE ecosystem and established ones evolved. The specific question regarding the effectiveness of TA rollback prevention in the ecosystem requires to gather, aggregate, and interpret historical and current data to distill the evolution and future trends, which is a non-trivial task.

Representative Dataset. The core of this study is a large and representative dataset of TAs and past security-related events. The availability of historical firmware, proprietary file formats, and vague information on public TA vulnerabilities pose difficulties for this task.

This study is the first to 1) systematize and model the TA ecosystem (Section 5), 2) relate this model to the implications of security-related events and discover real-world instances of rollback-counter-related side effects (Section 7), 3) obtain a representative ground-truth dataset of TAs and security-related events (Section 6.1 and Section 7.1), and 4) extrapolate and discuss potential future developments of the ecosystem (Section 8).

4 Threat Model

We base our threat model on the typical guarantees given to the TEE by ARM TrustZone, and refine the model in the specific context of a production mobile device running Android.

Attack Surface Characterization. ARM TrustZone-based TEEs benefit from strong hardware-assisted isolation that provides integrity and confidentiality guarantees for all the components running in the secure world. In particular, this isolation prevents the execution of any unauthenticated code within the TEE (e.g., loading unsigned code or modifying existing code), and ensures that secrets that are not shared with the normal world stay confidential (e.g., cryptographic keys or biometric identifiers). Consequently, this attacker model assumes an attacker capable of executing code on N-EL0 and N-EL1, having access to the TEE-exposed interfaces.

On a typical production device running Android, this attacker model is realistic, given the history of privilege escalation exploits available for the Linux kernel (e.g., CVE-2016-5195, or CVE-2018-9568, CVE-2022-2602). Moreover, this attacker model can be refined and weakened, since typically a large list of user-space daemons have access to the TEE driver interface exposed by the kernel. In the best case, access

to this interface (N-EL0 only) is all an attacker needs to carry out the attacks described in our work.

Attack Objectives. In our specific scenario, the attacker is interested in breaching the integrity and confidentiality of a TA, and potentially use the TA's capabilities to further compromise the TEE (e.g., breach the TEE OS). Note that for the latter, an exploitable bug in *a single* TA (single point of failure) is enough to achieve this goal.

Bug Classes and Rollback. All TAs for major TEE platforms are implemented in memory-unsafe languages, namely C/C++. The C/C++ code is compiled to aarch32 or aarch64 ARM machine code. We found that most TEE platforms for ARMv8-based SoCs support 32-bit and 64-bit TAs, and follow standard calling conventions. These design choices expose the TEE to traditional memory corruption bugs like buffer overflows and dangling pointers that can lead to arbitrary code execution within the virtual address space of a TA, in the worst case.

The "*classic*" approach for an attacker to compromise the TEE is to search for unknown exploitable vulnerabilities (0-days) in TAs, which is non-trivial and requires deep knowledge of the target. The second attack vector, the focus of our work, is to load a properly signed but vulnerable TA. This attack vector option is easier, as in the best case, the attacker can simply reuse existing publicly-available proof-of-concept exploits. However, this option *must* be prevented by effective TA rollback prevention mechanisms, which leads us to the centerpiece of our research: the industry-wide effectiveness of TA rollback prevention mechanisms.

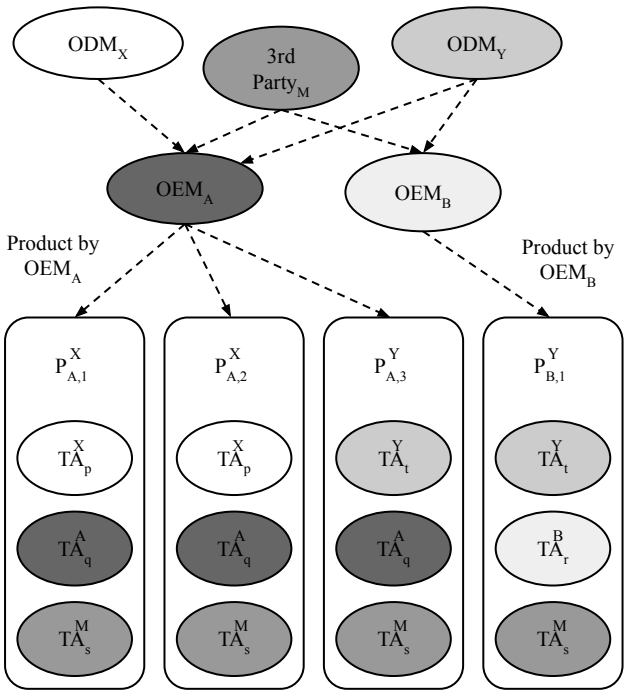

Figure 4: Ecosystem of Android mobile device TEEs with a focus on the origins of TAs. The shading indicates the origin of a TA used within a product. For instance, Product $P^X_{A,1}$ is a product by OEM_A based on ODM_X's chipset and ships with three TAs, TA^X_p (by the ODM), TA^A_q (by the OEM), and TA^M_s (by a third party).

5 Security Analysis of the TEE Ecosystem

The Android TEE ecosystem consists of several complex components. We focus our description on TAs and analyze security implications related to the effectiveness of TA rollback prevention that directly derive from the characteristics of the ecosystem.

5.1 TA Origins and Anti-Rollback Challenges

The Android TEE landscape is complex and growing. To understand this ecosystem, we concentrate on two important entities. The *Original Design Manufacturer (ODM)* provides the hardware platform (e.g., the SoC for a mobile device) and the initial firmware/software provisioning. This platform is rebranded by the *Original Equipment Manufacturer (OEM)* and further customized or provisioned with software components to create the customer-facing branded product. The following real-world examples illustrate the ecosystem further.

Xiaomi Mi series. The Xiaomi Mi series is a line of high-end mobile devices sold worldwide, where the two above-mentioned entities are relatively clear. The hardware of these phones is provided by Qualcomm, the ODM in this case. The consumer-facing branding and final customizations are done by Xiaomi, the OEM. The ODM usually determines low-level components. Thus, the TEE deployed on these phones is the Qualcomm Secure Execution Environment (QSEE). Further, a subset of TAs could originate from the ODM. OEMs can extend their TEEs by either developing their own TAs or integrating third-party TAs. For instance, a popular TA for digital content protection is Google's Widevine [55].

Samsung S series. We observe instances where the ODM leverages third-party TEEs. For instance, on the Samsung Galaxy S9 sold outside of the US, we find an Exynos chipset by Samsung and the Kinibi TEE by Trustonic. Samsung introduced its own TEEGris TEE in later models (e.g., the Samsung Galaxy S10).

Xiaomi Redmi series. We observe OEMs selling several series of products based on different ODM platforms. For instance, besides Xiaomi's high-end Mi series based on Qualcomm SoCs, their Redmi series is based on MediaTek SoCs. MediaTek utilizes a TEE stack called BeanPod by Beanpodtech for their chips.

All of these non-exhaustively mentioned TEE implementations (e.g., QSEE, Kinibi, TEEGris, or Beanpod) differ drastically in terms of features and, thus, affect the focus of our research—the capability to prevent TA rollbacks. We provide an overview of the different TA header formats indicating their rollback capabilities in Tables 8a, 8b, 8c, 5, and 7 in the Appendix. Furthermore, considering that TAs can have several origins, it is a challenge for the OEM to compose the final product's firmware image containing the latest TAs *and* ensure that rollback counters are updated accordingly. We illustrate the dependencies of TA deployment in Figure 4.

For the purpose of our research, surveying this landscape yields the following high-level insights:

- TAs are developed by several entities (e.g., ODM, OEM, or third-party).
- The ODM determines the TEE OS, which can be provided by a third party.
- The customer-facing brand (the OEM) alone does not reveal the underlying TEE implementation.
- The TEE OS implementations differ in terms of supported features.

5.2 Hypotheses for TA Anti-Rollback

Given the Android TEE ecosystem structure as described above, we derive a set of hypotheses that directly affect the effectiveness of TA rollback prevention.

HYP1: Leakage. When TA rollback counters are used in an uncoordinated way to enforce anti-rollback prevention for severely vulnerable TAs, the counter increase can inadvertently disclose information about the existence of a vulnerability in the TAs.

HYP2: Cross-Product Leakage. The leakage due to a rollback-counter increase affects devices of the same product series. Vendors often update their phone models yearly. Most of the software stack deployed on the previous generation within a product series remains the same. *Consequently, an increased TA rollback counter on one product of a series will directly leak security-critical information about the other products in this series.* In a product-centric organization, where rollback counters are increased in an ad-hoc and uncoordinated way, the vulnerability leakage will directly affect other products. Figure 5 illustrates this hypothesis.

HYP3: Cross-OEM Leakage. If two different OEMs are using the same ODM, the set of common TAs is prone to cross-OEM leakage in case of a TA rollback counter increase. *Consequently, the uncoordinated usage of TA rollback counters by one OEM affects other OEMs and may leak TA vulnerabilities affecting other OEMs' products.* Figure 6 illustrates this hypothesis.

HYP4: Cross-ODM Leakage. If a TA is supported on multiple ODMs, a rollback counter increase on one ODM

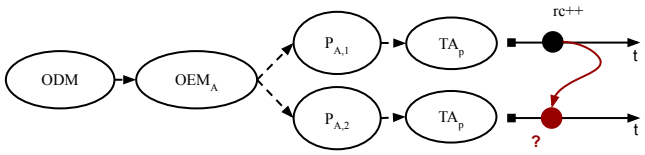

Figure 5: Cross-product leakage of security-critical information affecting TA_1^2 deployed on product P_2 due to increasing a rollback counter ($rc++$) for TA_1^1 on product P_1.

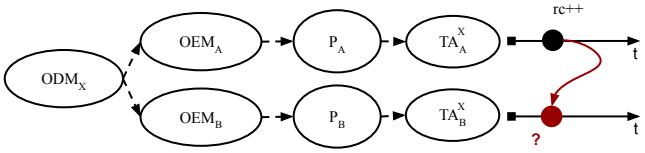

Figure 6: Cross-OEM leakage of security-critical information affecting TA_X^B deployed on product P_B by OEM_B due to increasing a rollback counter ($rc++$) for TA_X^A on product P_A by OEM_A. Both OEMs use the platform by ODM_X.

platform will leak security-critical information affecting the other ODMs' platforms. This leakage is especially relevant if ODM platforms lack support for proper rollback prevention or employ a different (weaker) set of exploit mitigations, putting this weaker platform at risk. Figure 7 illustrates this hypothesis.

6 Study Design

Our goal is to assess the industry-wide effectiveness of TA rollback prevention for Android smartphones. This section describes our analysis pipeline. Figure 8 serves as an overview. To carry out this study, we collect a representative dataset of firmware images containing TAs distributed to the most popular devices. We present our dataset selection criteria, elaborate on the extraction of rollback counters, and describe the collection of vulnerable TAs.

6.1 TA Dataset Criteria

In this study, we want to understand the current industry-wide state of TA rollback prevention for Android smartphones. The foundation for this assessment is a dataset addressing the following timeliness, representativeness, and diversity criteria.

Timeliness. Given that the average smartphone lifecycle is around 2.5 to 3 years [46, 48], we can establish a conservative upper bound by considering smartphone models released within the last 4 years to capture a set of phones that are relevant today.

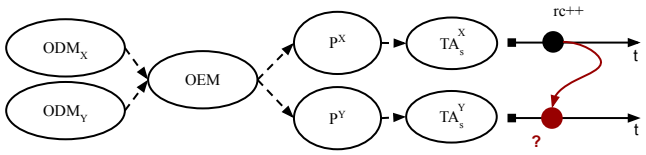

Figure 7: Cross-ODM leakage of security-critical information affecting TA_1^Y deployed on product P_Y based on the platform by ODM_Y due to increasing a rollback counter ($rc++$) for TA_1^X on product P_X based on the platform by ODM_X. The same TA is used on two different platforms.

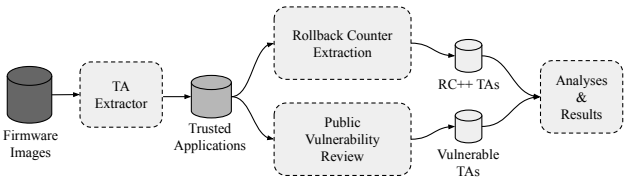

Figure 8: Overview of our firmware processing pipeline.

Representativeness. Further, our guiding principle to claim representativeness for our dataset is that the investigated devices are widely used. Thus, we base our selection on the market share of Android smartphone vendors [47].

Diversity. Each OEM typically employs a variety of platforms based on different ODMs. As explained in Section 5, the ODM determines the TEE. Thus, our dataset should take the diversity of TEEs into account.

6.2 Rollback Counter Usage

Our dataset contains TAs deployed on a diverse set of TEE implementations. Each implementation handles rollback prevention differently, but generally, all the relevant information can be found in the header of a given TA. Hence, we collected the publicly available information on TA header formats and reverse-engineered those formats that are not publicly known. The details on these header formats can be found in Tables 8a, 8b, 8c, 5, and 7 in the Appendix.

As a result of our rollback counter extraction, we obtain a chronologically ordered sequence of TAs for every phone in our dataset that shows the rollback counters for each TA.

6.3 TA Vulnerabilities

To establish the ground truth of vulnerable TAs, we survey known vulnerabilities broadly across all vendors in our dataset. The set of vulnerable TAs is based on publicly available information. Our collection of publicly known vulnerabilities relies on information from three different input sources. First, we systematically survey the programs of (non-academic) security conferences (e.g., DEF CON [13], Black-Hat [12], OffensiveCon [36], or C3 [9]). Second, we diligently review the blogs of companies and individuals known to conduct research on TEEs (e.g., Riscure [39], Quarkslabs [38], Google P0 [37], or Blue Frost Security [44]). Third, we scraped the security bulletins of vendors for TEE-related keywords (e.g., TEE, TA, TrustZone, and various vendor-specific keywords). Lastly, we aggregate this information, manually match the vulnerabilities to the TAs in our dataset, and verify the described issue.

This subset of vulnerable TAs is our ground truth and starting point to assess the rollout of patches and rollback prevention measures.

7 Analyses and Results

After introducing the study design in Section 6, this section discusses the study results. We first describe our dataset in detail and then focus on our research questions:

RQ1 What are TA rollback counters used for?

RQ2 Are TA rollback counters used correctly and consistently?

RQ3 What are the consequences of uncoordinated TA rollback increases?

RQ4 Can rollbackable TAs be loaded and exploited under real-world conditions?

7.1 Dataset Collection and Characterization

To create a timely, representative, and diverse dataset as explained in Section 6.1, we first take into account that smartphones have a lifetime of about two-and-a-half to three years [46,48]. Thus, we choose a conservative upper bound of smartphones released within the last four years to fulfill our timeliness criterion. Further, we focus on the top five Android smartphone vendors as reported bi-quarterly by Statista [47] over the last four years. These vendors are Samsung, Oppo, Vivo, Xiaomi, and Transsion covering a share of >65% of the Android smartphone global market for each reported quarter. Next, we leverage `gsmarena.com` to identify all smartphone models released within the last four years by these vendors. To address our diversity criterion, we group all phones by ODM for each vendor. `gsmarena.com` provides information on the chipset for each phone which allows us to perform this grouping. We select at least two phones from each of these groups to investigate their firmware images. We collect at least five firmware images for each of these phones spanning the firmware evolution of at least two years. We list the sources of these firmware images in Table 6 in the Appendix.

Firmware Dataset								Rollback Analysis		
ODM	TEE	Vendor	#Devices	Σ #Firmware	Σ #TAs	Σ #vulnerable TAs	Σ #TAs with rollback-counter change	average #days of rollback exposure†	Σ #rollbackable TAs§	Σ #rollbackable devices*
Qualcomm	QSEE	Samsung	6	88	2,834	132	13	0‡	13	4
Qualcomm	QSEE	Xiaomi	5	299	6,982	0	0	N/A	0	0
Qualcomm	QSEE	Oppo	4	28	215	0	0	N/A	0	0
Qualcomm	QSEE	Vivo	3	13	92	0	0	N/A	0	0
MediaTek	BeanPod	Xiaomi	4	173	3,514	584	0	N/A	63	4
MediaTek	BeanPod	Transsion	4	22	104	0	0	N/A	0	0
MediaTek	Kinibi	Oppo	5	28	832	97	0	N/A	20	5
MediaTek	Kinibi	Vivo	4	29	1,512	166	0	N/A	22	4
MediaTek	TEEGRIS	Samsung	2	16	658	3	83	58	0	0
Exynos	Kinibi	Samsung	6	325	10,523	1,090	22	0	90	6
Exynos	TEEGRIS	Samsung	8	309	8,490	510	72	245	99	6
All	All	All	51	1,330	35,541	2,582	190	99	265	29

Table 1: The dataset used for our study. (†) This column measures the average number of days between advisory publication and rollback counter increase. N/A denotes that there are no instances of a vulnerable TA whose rollback counter is increased. (§) A rollbackable TA is a TA whose outdated and vulnerable version can be loaded due to a lack of rollback counter increase on the newest firmware version. (*) A device whose newest firmware version contains at least one rollbackable TA is considered to be rollbackable. (‡) All of the TA rollback counter increases for Samsung QSEE devices are on newer devices without any associated public vulnerabilities for TAs.

For each downloaded firmware image, we extract all enclosed TAs, parse the available metadata (e.g., rollback counters), and manually mark TAs with publicly known vulnerabilities. Table 9 in the Appendix shows all publicly known vulnerabilities systematically obtained from several resources as described in Section 6.3 [1, 5, 23, 24, 31, 32, 35, 40–42, 45, 56].

Table 1 summarizes our dataset. In total, we collect 1,330 firmware images recently deployed on 51 different phone models spanning five vendors (OEMs) and three ODMs. We obtained 35,541 TAs (293 unique TAs) for four different TEE implementations. We were able to mark 2,582 vulnerable TAs and found 190 rollback counter changes. We focus on the 65% most common vendors to represent the market. The remaining 35% consists of the long tail of the market. Including an additional vendor would only marginally contribute to the representativeness of our dataset, while obtaining and processing their historical firmware imposes a disproportional effort. For instance, vendors like Google, LG, Motorola, HTC, and Huawei each have a market share of lower than 5% [47].

Although the firmware for Google devices is readily available [19], we exclude Google devices from our study for the following reasons. Older QSEE based Google devices do not use rollback counters to support arbitrary firmware downgrades [4]. With the Trusty TEE, which replaced QSEE on newer Pixel devices (Pixel 6 and later), Google disabled runtime-loadable TAs. TAs are shipped as part of signed firmware images and cannot be modified [18]. This decision is a trade-off where Google sacrifices third-party TA support and their business opportunities for security.

Security-focused Android ROMs such as GrapheneOS [21] or LineageOS [29] rely on the device's existing TEE software stack. Hence, our findings apply equally to these ROMs.

7.2 RQ1: Rollback Counter Use Cases

Table 1 summarizes our results regarding the usage of TA rollback counters. Our first observation is that all vendors in our dataset except Samsung are *not* making use of rollback prevention. The reason is either that the TEE does not support rollback prevention (e.g., BeanPod) or the vendor does not effectively employ the TEE's rollback counter capabilities to prevent rollback—even if there are public security advisories for several TAs available.

Exploring the history of Samsung devices reveals an episode of ad-hoc and uncoordinated usage of rollback prevention, followed by a period of consistent rollback counter increases with every other firmware update shipped to customer devices. First, this observation shows that Samsung (the biggest player in the Android smartphone market by market share) attempts to address TA anti-rollback enforcement. Second, the history of ad-hoc and uncoordinated anti-rollback enforcement indeed shows a correlation between security-critical vulnerabilities and the increase of TA rollback counters. In Table 2 we list TAs with an increased rollback counter that correlates with a public security advisory. As we will discuss in Section 7.4, the threat of unwanted disclosure affecting other devices of the same vendor, or potentially other vendors in the ecosystem is real.

7.3 RQ2: Correct and Consistent Usage of Rollback Counters

To answer RQ2, we analyze TAs across all of a device's firmware versions to assess if the device was or still is exposed to rollback attacks. To track updates to a TA we compute the hash of the TA's `.text` section as a heuristic. If this hash changes in a firmware version, we register this as an update

TEE	TA	Vendor ID	CVE	Published	RC++
Kinibi	ESECOMM	SVE-2017-8889	2017-18655	2017-08	2017-10
Kinibi	TEE_keymaster	SVE-2019-14126	2019-20607	2019-05	2019-02
TEEGRIS	WVDRM	SVE-2020-17117	2020-13832	2020-06	2020-05
TEEGRIS	SKPM	SVE-2019-14892	2019-14892	2019-08	2021-01

Table 2: Four critical security advisories released by Samsung. TA rollback counter increases often coincide with advisories indicating critical vulnerabilities. The RC++ column shows the date of the rollback counter increase.

to the TA. This heuristic serves as upper bound and may have false positives, i.e., it may signal an update if the TA is recompiled in a different environment. For each firmware version, we also check if the rollback counter was increased and correlate this to the vulnerabilities in our dataset from Table 9. We consider a TA vulnerable in the time frame after a vulnerability disclosure and before a rollback counter increase. If no rollback counter increase takes place then the vulnerable version can still be loaded in the newest firmware version. Note that even if a vendor removes a TA from newer firmware releases, and the rollback counter of that TA was never increased, the attacker can still load this deprecated TA.

Figure 9(a)-(d) shows the updates and vulnerabilities for the TAs of four BeanPod devices in our dataset. Each of these devices has at least 3 vulnerable TAs that may be loaded on a fully updated device. Note that all TA vulnerability lifecycles ending with a ○ in Figure 9 represent TAs vulnerable to rollback attacks. The reason for this high number is that the BeanPod TA format does not contain a rollback counter. Even though the vulnerability may be patched in a newer firmware version, it is still available to the attacker in our threat model, who can simply load the old vulnerable TA.

Figure 9(e)-(h) shows the updates and vulnerabilities for the TAs for four Kinibi devices in our dataset. Note that there are a few version counter increases but no public vulnerability corresponding to the TA. In contrast, the version counter for vulnerable TAs is only increased in one instance.

Figure 9(i)-(m) shows the updates and vulnerabilities for the TAs for five TEEGRIS devices in our dataset. For all phones older than the S21, there are at least two vulnerable TAs that can be loaded and there are five occurrences of a rollback counter increase preventing the loading of a vulnerable TA. On the newer S21 device, Samsung periodically increased the rollback counter.

Figure 9(n)-(p) shows the updates and vulnerabilities for the TAs for three QSEE devices in our dataset. Note the difference of rollback counter increases between older devices and the more recent S21. The S21 also shows an example of deprecated TAs that still pose a threat to the device. The two QSEE example TAs, smplap64/smplap32 were shipped in the first firmware version and then removed from later firmware. However, the attack surface exposed by these TAs remains accessible to attackers who can load the TAs shipped in the first firmware version on a device with the newest firmware.

An overview of the results of this analysis can be seen in Table 1. In summary, we observe that rollback counters are rarely used. There are only 190 instances of a TA with a rollback counter increase out of the 35,541 TAs in our dataset. In total, we found 2,582 vulnerable TAs and only 22 TAs whose rollback counter increase prevents loading one of these vulnerable TAs. Leaving us with 265 rollbackable TAs on the latest fully-updated devices. As a consequence, we can load vulnerable TAs into *all* fully-updated TEE implementations and into 29 out of the 51 devices in our dataset.

We also observe that, in cases where the rollback counter was used to prevent rollback attacks, on average the update with the rollback counter increase is released 99 days after the publication of the vulnerability. Interestingly, for older Samsung devices using Kinibi, the average number of days is 0. This means that for these devices, on average the update with the counter increase was before the advisory publication. In these cases, there is no rollback exposure as the rollback counter increase precedes the advisory publication. It appears that Samsung moved away from these best practices on newer devices using TEEGRIS, resulting in an average rollback exposure time of 245 days. However, the newest generation of devices (Samsung S21 and Samsung S22) follows a systematic approach and increases rollback counters frequently.

7.4 RQ3: Consequences of Uncoordinated TA Rollback Increases

The uncoordinated usage of TA rollback counters can lead to the leakage of security-critical information which can affect several entities in the ecosystem. As we can see in Figure 9(j) the WVDRM TA on the Samsung A10 received a rollback counter increase after a critical vulnerability was fixed. Analyzing the other smartphone models by the same vendor during the time of the counter increase reveals that no other model received this update. The rollback counter for the WVDRM TA on the Samsung S10, A40, and S20 was never increased. The WVDRM TAs deployed on these models share the same codebase as the TA on the A10 with the counter increase. Thus, these models remain vulnerable to a rollback attack. We carry out a rollback attack against the S10 in Section 7.5.

Consequently, we can clearly see that cross-product leakage affects products of *the same* OEM, as hypothesized in HYP2 Section 5.2, is a real threat within the ecosystem.

Our dataset does not contain clear cases for cross-OEM and cross-ODM leakage as described in HYP3 and HYP4 in Section 5.2. To estimate the potential impact of HYP3 and HYP4 we estimate a lower bound on the number of TAs shared between OEMs. These shared TAs are either shipped by the ODM or third-party TEE provider. For this measurement we match the TA filename, and the strings in the TA. We then manually confirm matches by decompiling the TA and comparing the decompilation. Table 3 shows the number of TAs shared by different vendors. MediaTek maintains 14 TAs

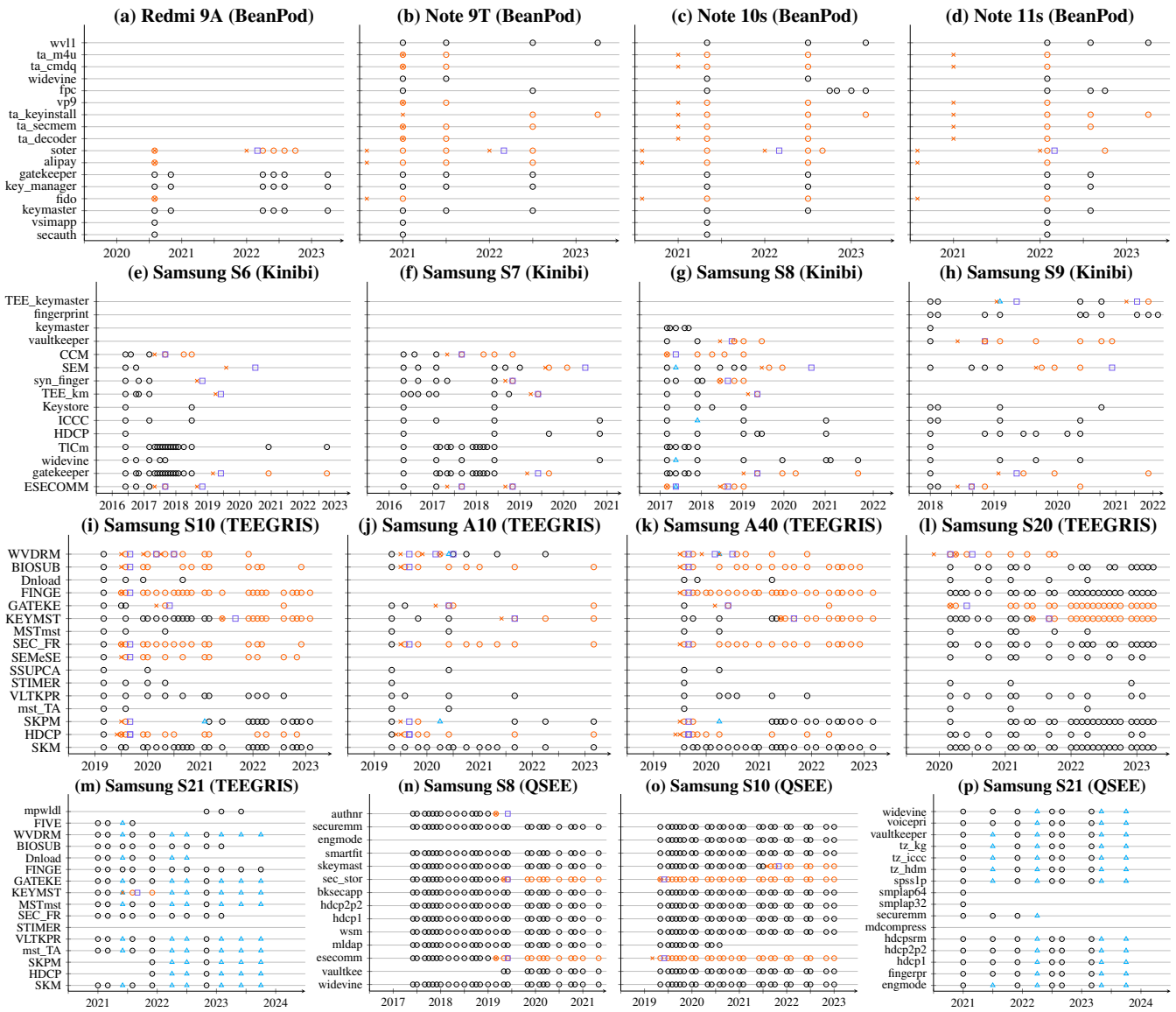

Figure 9: The TA vulnerability lifecycle for TAs found on 16 devices covering the four most widely-used TEE implementations. A circle ○ represents an update to the TA. A × represents a vulnerability. A square □ represents a published advisory. A triangle △ represents an update with a rollback counter increase. Orange circles ○ are updates to a vulnerable TA that do not increase the rollback counter. If the last update to a TA is marked with an orange circle, then a vulnerable version of that TA may be loaded on the latest firmware of that phone.

shared between Oppo, Vivo and Xiaomi. These TAs are deployed on both Kinibi and BeanPod TEEs. BeanPod does not support anti-rollback while Kinibi does. Thus, cross-ODM leakage from Kinibi to BeanPod would expose BeanPod devices to rollback attacks.

We anticipate that the demand for security as a feature will drive vendors to enforce rollback prevention. This upcoming episode will likely be uncoordinated amongst different entities in the ecosystem and result in the leakage of security-critical information that puts the TEEs of the ecosystem at risk.

7.5 RQ4: Rollback Exploitation under Real-World Conditions

After assessing issues with rollback counters quantitatively, we now assess them qualitatively by testing if outdated TAs can be loaded under real-world conditions, answering RQ4. We attempt to load, crash, and exploit TAs identified as vulnerable and rollbackable on our fully patched test devices. To simulate a compromised normal-world EL1, we manually root our devices using Magisk [51] with the newest available firmware version. To load old TAs we remount the vendor

Origin	Vendors	# shared TAs
MediaTek	Oppo, Vivo, Xiaomi	14
MediaTek	Transsion,Xiaomi	4
Kinibi	Oppo, Samsung, Vivo	1
Kinibi	Oppo, Samsung	1
QSEE	Oppo, Samsung, Vivo, Xiaomi	2
QSEE	Samsung, Vivo, Xiaomi	5
QSEE	Oppo, Vivo, Xiaomi	4
QSEE	Oppo,Vivo	5
QSEE	Vivo, Xiaomi	1
QSEE	Samsung, Xiaomi	3

Table 3: The shared TAs across different OEMs and third-party TEE implementations. Origin denotes the entity that provides the TA.

partition as writable and copy the old TAs to the folders where the TAs are stored. We then use a small program that interacts with the kernel driver to load the TA and invoke commands.

We use our Samsung S10 test device (FW version: G973FXXSGHWA1 from January 2023) and attempt to load the seven vulnerable TAs identified in Figure 9(i). We verify that the rollback counter increase does indeed prevent loading vulnerable TAs by attempting to load a vulnerable version of the SPKM TA. Table 4 shows the results. We successfully load all the vulnerable TAs and trigger a crash for six of the seven loaded TAs. To successfully rollback the FINGE TA, we need to first enable biometric authentication. Otherwise, the TEE refuses to load the TA, both the vulnerable and the current version. The SPKM TA cannot be loaded anymore due to the rollback counter increase.

We use our Xiaomi Redmi 9A test device (FW version: MIUI 12.5.7_RCDEUXM from November 2022) and attempt to load all vulnerable TAs identified in Figure 9(a)-(d). Note that this experiment is possible due to cross-device loading in devices using the Beanpod TEE. Table 4 shows the results. Overall, we successfully load all vulnerable TAs and trigger crashes in seven out of nine loadable TAs.

We use our Xiaomi Redmi Note 11 test device (FW version: 13.0.14 from February 2023). Since there are no public vulnerabilities for this vendor/TEE platform we attempt to load TAs from the earliest firmware version. We are able to load all the old TAs, see Table 4.

We use our Samsung S9 test device (FW version: G960FXXUHFVB4 from March 2022) and attempt to load the five vulnerable TAs identified in Figure 9(h). Table 4 shows the results. We successfully load all vulnerable TAs and trigger crashes for two out of five TAs.

Exploitation Case Studies. To further demonstrate the feasibility and impact of the rollback attack, we demonstrate two instances where we rollback to a vulnerable TA version on a fully patched device and then use the vulnerability to gain control of the TA's program counter.

We use our TEEGRIS Samsung S10 test device on the newest firmware version and exploit the vulnerable HDCP TA [35]. Since we can successfully install the vulnerable TA, we can rollback to the same TA version as used in the blog post and follow the detailed exploitation steps, gaining control over the program counter. Since exploitation is trivial and well-documented we omit the details of the exploit here.

For a second case study, we use our Xiaomi Redmi 9A test device on the newest firmware version, which uses the Beanpod TEE, to rollback, load and exploit the `ta_keyinstall` TA. Note that this TA is not shipped with the firmware of our test device but we found that we could nevertheless load the TA from another device's firmware. The `ta_keyinstall` is vulnerable to a type-confusion bug (CVE-2023-32835). We exploit this bug to obtain an ASLR oracle and an arbitrary write primitive. With these primitives, we get control over the program counter.

Figure 1 shows the relevant code inside the `ta_keyinstall` TA. When the TA is invoked with commandID 1, the TA calls the `DRMKEY_QUERY` function. This function reads data from the input buffer and writes

```
TA_InvokeCommandEntryPoint(void* sessionContext,
    uint32_t commandID, uint32_t paramTypes,
    TEE_Param params[4]) {
  ...
  if(commandID == 1) {
    pInput = params[0];
    nInputSize = params[1];
    pOutput = params[2];
    r = TEE_CheckMemoryAccessRights(5,pInput,nInputSize);
    if(r==0) {
      DRMKEY_QUERY(pINput, &count, pOutput);
    }
  }
  ...
}
DRMKEY_QUERY(void* keyblock, int* count,
    void* pOutput) {
  int keycount = *(int*)(keyblock + 0x44);
  void* src = (void*)(keyblock + 0x48);
  int buffer[0x16];
  if(keycount < 0x201){
    pOutput[0] = keycount;
    int ct = 0;
    while(ct != keycount){
      memcpy(buffer, src, 0x58);
      int encDrmKeySize = buffer[3];
      int keyblockLeng = encDrmKeySize + 0x60;
      keyid = buffer[0];
      // vuln: arbitrary write
      *(void*)(pOutput + 4*ct) = keyid;
      src = src + keyblockLeng;
      ct++;
    }
  }
  ...
}
```

Listing 1: Code in the `ta_keyinstall` TA relevant for the `DRMKEY_QUERY` function. The `paramTypes` parameter is never checked, thus the TA has no way of knowing whether the values in params are integers or pointers to shared memory. By supplying integers that are treated as pointers by the TA we obtain an ASLR oracle and arbitrary write.

	TEE_keymaster	gatekeeper	ESECOMM	SEM	vaultkeeper	WVDRM	GATEKE	BIOSUB	FINGE	SEC_FR	SEMeSE	HDCP	SPKM (fixed)	soter	fido	alipay	secmem	keyinstall	decoder	cmdq	m4u	vp9	soter64	mlipay	cardapp	mfido	vsimapp	smplap64	hdcspsrm
	Samsung S9					Samsung S10								Redmi 9A										Redmi Note 11						
loaded	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
crash triggered	✗	✗	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 4: The results of attempting to load vulnerable TAs on our fully-patched devices.

part of it to the output buffer. Due to the missing parameter type check, the TA cannot ensure that the input or output buffers point to valid shared memory. We can use the `TEE_CheckMemoryAccesRights` call as an ASLR oracle. The function checks if `pInput` (the first of our parameters) points to valid memory. By setting the first parameter to an arbitrary pointer we can figure out if that pointer points to valid memory or not. For the arbitrary write, we simply supply the pointer where we want to write to as the third parameter (`pOutput`). The `DRMKEY_QUERY` function will then write data from the input buffer under our control to the output buffer. To get control over the program counter we first use the ASLR oracle to find the location of the stack. Then we use the arbitrary write to overwrite the return address of the `TA_InvokeCommand` by pointing `pOutput` to the corresponding stack address.

8 Takeaways

We now discuss the main takeaways, highlighting the primary results and their implications for the security community.

The State of TA Rollback Prevention. We examine the evolution of TAs along their lifetimes on several Android smartphones. Our study covers a representative set of TEE implementations deployed on the most popular smartphones, namely QSEE, Kinibi, TEEGris, and BeanPod.

We observe a division of the market. Either vendors have rollback attacks in scope (e.g., Samsung), or they remain oblivious to the problem (e.g., Xiaomi, Oppo, Vivo, and Transsion).

Samsung has a long history of advertising security features to the consumer market and especially the enterprise market. Thus, most of their recent models address the problem of rollback attacks. Although even on the newest Samsung devices, we still identified deficiencies (i.e., TAs that are not yet equipped with a version counter), Samsung effectively protects most TAs against rollback attacks.

However, Samsung's development until reaching effective TA rollback prevention presents a controversial picture. It is noteworthy that several of their still-supported smartphone models remain vulnerable to TA rollback attacks. Contrary, their newest generation of phones receives continuous rollback counter updates. The episode of uncoordinated, ad-hoc usage of TA rollback counters as depicted by many past Samsung TAs, provides the opportunity to gain several major insights for effective anti-rollback. First, and most obvious, the fix of a vulnerable TA, the increase of the corresponding rollback counter, and the rollout of these changes should precede the public disclosure of the vulnerability. Depending on the interdependencies of the organizational units involved, even this obvious observation needs to be explicitly taken into account. Second, when rollback counters are used to prevent rollback to severely vulnerable TAs, this effort should consider *all* affected products. Otherwise, the information about severe vulnerabilities is leaked for all products that did not receive a rollback counter increase. In our study, we demonstrated the cross-product dimension of this problem, and we anticipate seeing cross-OEM and cross-ODM instances of this problem occur frequently in the future because many code bases for TAs are shared amongst OEMs.

For the remaining vendors, we observe that rollback counters never change, and especially that they *do not* change when publicly known vulnerabilities are fixed in TAs. Note that while our ground-truth dataset of vulnerable TAs is likely incomplete, as no public repositories exist, we nevertheless identify several cases of ineffective rollback prevention across *all* TEE implementations. We conclude that, with the exemption of the latest Samsung models, *TA rollback prevention is ineffective at an industry-wide scale*.

TA Vulnerability Practices. Aside from our focus on TA rollback prevention, we encountered several supply chain flaws (Section 5). Vendors manage to successfully patch TAs internally but fail to roll out the patches to their full fleet of products. For instance, across all Xiaomi devices, we found patched versions of the soter, fido, and alipay TAs, but several individual devices remain vulnerable. The latest version of the firmware is vulnerable for, e.g., the Xiaomi Note 11S, Redmi

Note 9T, and Redmi 10A. A similar inconsistency applies to the Samsung S10 and A10 in our dataset. The widevine TA was found vulnerable on both devices in 2020. While the vulnerability was fixed on both devices in the latest firmware for these devices, the A10 widevine TA received a rollback counter update and the S10 version did not. This leaves the S10 vulnerable to a rollback attack.

A further unfortunate vendor practice is to patch security-critical bugs silently. We observed this practice for the soter, fido, and alipay TAs on Xiaomi devices employing Beanpod.

A further hiccup that might be overlooked by vendors is discontinued TAs that turn out to be vulnerable but remain loadable. For the Xiaomi Note 11S, we observed that a vulnerable version of the ta_decoder TA was deployed in 2021. This TA was discontinued mid-2022 and is not present in the latest firmware images for this device. However, the vulnerable version deployed in 2021 remains loadable and potentially exploitable on the latest firmware.

Transparency Regarding TA Rollback Prevention. A major challenge of our study was to establish a ground-truth dataset of vulnerable TAs, to extract their corresponding rollback counters, and to track these version counters over time. Monitoring this information reveals the lifetime of exposure to a potential exploit leveraging a rollback attack, and it sheds light on rollback prevention being effective at all, in case the counters are never updated. A widely adopted transparency covering this information would help consumers and vendors.

As a means of public communication and to keep track of security patches, Google established the Android security bulletins [20]. These bulletins summarize security-related patches on a monthly basis and provide details on each patched vulnerability, including CVE ID, type of vulnerability, severity, and affected components. Android users can check their current security patch level on their devices in the "Settings" app searching for "Security Patch Level". This patch level shows the system property `ro.build.version.security_patch` which corresponds to the security patch level dates published in the security bulletins (typically in YYYY-MM-DD format).

A common practice for vendors is to reference these security patch levels by Google and publish their own security patch levels containing vendor-specific patches. These vendor patch levels usually subsume the patch levels by Google.

To establish more transparency regarding the TA rollback prevention problem, we suggest two approaches. The first approach requires the cooperation of the vendor who just has to report the increase of the TA rollback counter (or other rollback countermeasures) as part of the publicly disclosed advisory. Typically, these advisories contain a "patch description" that can be used to communicate the handling of TA rollback. This practice would transparently communicate the consideration of TA rollback attacks.

The second approach does not require the vendor's cooperation and relies on community effort. For our empirical study, we created the infrastructure to continuously monitor new firmware releases for five vendors and automatically track TA version counters. Using this growing database, we can determine if a rollback counter was increased whenever a TA-related vulnerability is disclosed. As an artifact of this paper, we release the dataset used for this study at `http://hexhive.epfl.ch/spill_the_tea`.

Both of these solution sketches increase the transparency regarding TA rollback attacks and communicate the problem clearly to consumers and vendors.

9 Threats to Validity

Dataset Representativeness. The primary threat to the validity of our study is the representativeness of our dataset. Our dataset is representative as it covers a broad selection of the total market and focuses on recent Android-based mobile devices. The vendors and phones in our dataset cover over 65% of the Android market share for the last four years. We extended this dataset with phone models by vendors that received security audits with publicly available results. The resulting dataset ensures sufficient vulnerability data to make our study conclusive—which is the case given the large number of 37 vulnerable TAs in our dataset. Besides the representative market share and timeliness of our device selection, we also considered the diversity of hardware platforms used by all vendors in our dataset. Since the hardware platform often determines the TEE, we made sure to include a representative set of hardware platforms for each vendor.

Missing Firmware Images. In Section 7.3, we analyze firmware versions over time to demonstrate how vendors use rollback counters. However, we cannot guarantee that we have all firmware versions, i.e., we may miss some intermediary versions. These missing images are not a threat to the conclusiveness of our study, since in a realistic scenario, an attacker can always retrieve these intermediary images to obtain a specific version of a TA. For our study, it is sufficient to observe the usage of TA rollback counters over a longer period of time and associate images with the release of public vulnerability disclosures to derive our conclusions concerning the effectiveness of TA rollback prevention.

Other Rollback Prevention Mechanisms. In our study, we focus on the rollback counter as the primary defense against an attacker trying to load an outdated TA version. However, there may be other mechanisms to prevent TAs from being loaded. One straightforward mechanism is to re-sign a TA and not use the public key of the old signature in the TA loader anymore. We observed TA resigning for eight TAs out of 35,541 TAs in total, which suggests that resigning is not used as an anti-rollback mechanism. Moreover, we try to mitigate this threat to validity by running the experiments described in Section 7.5. However, due to a lack of physical

devices, we could not run this experiment for all devices in our dataset and acknowledge this practical limitation.

TA Vulnerability. We use public advisories to identify TA vulnerabilities. These advisories often specify the vulnerable TA and the affected firmware versions. Additional data points to pinpoint a vulnerable TA are the reporting and the advisory release dates. We assume that firmware images released past the release date contain patched TAs and firmware images released before the reporting date contain unpatched TAs. In Section 7.5 we investigate a range of vulnerable TAs selected using this method to demonstrate real-world rollback attacks.

10 Related Work

The issue of TA rollback prevention on TZ-based TEEs was discussed before. Beniamini studied rollback prevention in QSEE and Kinibi and demonstrated that in 2017 rollback counters were not used [4]. Also in 2017, Chen et al. presented a rollback attack and its consequences on a Nexus 6 device [11]. Unlike these works, we assess the current status of TA rollback prevention six years later and focus on a large-scale cross-vendor study encompassing several recent and popular devices to assess the effectiveness of current industry-wide practices. Beyond the study of TA rollback prevention effectiveness, we discover the leakage effects resulting from ad-hoc and uncoordinated rollback counter increases aimed at preventing rollback to severely vulnerable TAs on individual devices. To the best of our knowledge, we are the first to explore the negative leakage effects concerning different entities in the ecosystem (i.e., cross-product, cross-OEM, and cross-ODM vulnerability leakage).

Beyond TA rollback prevention, several researchers have studied and exploited vulnerabilities in TZ-based TEEs, including Beanpod [32], QSEE [24, 31, 40], TEEGRIS [35, 42, 45], Kinibi [1, 5, 23], and TrustedCore [8]. Moreover, researchers investigated software design flaws [30] and side-channel attacks [6, 26, 50, 54]. Most of these works were summarized and systematized by Cerdeira et al. [10].

The growing number of reported memory corruptions led to the work on automated vulnerability discovery in TEEs. While TEEzz [7] follows an on-device black-box fuzzing approach, PartEmu uses an emulation-based approach to enable coverage-guided fuzzing of TAs [22]. Instead of automatically finding memory corruptions, Wan et al. attempt to establish Rust as a memory-safe alternative for TA development [53].

Android's popularity has attracted many security researchers who have conducted studies on various aspects of Android's security architecture. Farhang et al. [17] study Android security bulletins across various vendors. Several studies investigate how updates are applied to either apps or libraries [3, 14, 34]. Further, multiple studies have focused on various Android security vulnerabilities [16, 28]. Finally, Egele et al. [15] studied how developers misuse cryptographic APIs in Android apps.

11 Conclusions

In our large-scale cross-vendor study, we evaluated the pervasiveness and effectiveness of rollback prevention on Android devices for TAs running in the TEE. We augmented the study with publicly known TA vulnerabilities to form a ground-truth dataset containing vulnerable TAs. These datasets allowed us to track the handling of patches and rollback prevention over time. We uncovered a severe ineffectiveness of TA rollback prevention across all TEE implementations covered in our study. Furthermore, our study shows that TA rollback counters are never updated in the majority of cases across the industry, with the exception of the latest Samsung devices. Our case studies show that ineffective TA rollback prevention can be leveraged to gain code execution inside the TEEs of two popular up-to-date mobile devices. Our findings call for an immediate improvement of update strategies to protect the mobile ecosystem against n-day attacks against patched but rollbackable versions of TAs. Our dataset is publicly available at http://hexhive.epfl.ch/spill_the_tea.

Acknowledgments

We thank the anonymous reviewers and our shepherd for their feedback on the paper. This work was supported, in part, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850868), SNSF PCEGP2_186974, and DARPA HR001119S0089-AMP-FP-034.

References

[1] Maxime Peterlin Alexandre Adamski, Joffrey Guilbon. A deep dive into samsung's trustzone (part 1 - part 3). https://blog.quarkslab.com/a-deep-dive-into-samsungs-trustzone-part-1.html, 2019. Accessed: April 2023.

[2] Nikolaos Alexopoulos, Manuel Brack, Jan Philipp Wagner, Tim Grube, and Max Mühlhäuser. How long do vulnerabilities live in the code? A large-scale empirical measurement study on FOSS vulnerability lifetimes. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 359–376. USENIX Association, 2022.

[3] Sumaya Almanee, Arda Ünal, Mathias Payer, and Joshua Garcia. Too quiet in the library: An empirical study of security updates in android apps' native code. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1347–1359. IEEE, 2021.

[4] Gal Beniamini. Trust issues: Exploiting trustzone tees. https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html, 2017. Accessed: April 2023.

[5] David Berard. Kinibi tee: Trusted application exploitation. https://www.synacktiv.com/en/publications/kinibi-tee-trusted-application-exploitation.html, 2018. Accessed: April 2023.

[6] Sébanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. How trustzone could be bypassed: Side-channel attacks on a modern system-on-chip. In *Workshop in Information Security Theory and Practice*, 2017.

[7] Marcel Busch, Aravind Machiry, Chad Spensky, Giovanni Vigna, Christopher Kruegel, and Mathias Payer. Teezz: Fuzzing trusted applications on cots android devices. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 220–235. IEEE Computer Society, 2022.

[8] 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, August 2020.

[9] CCC. Ccc event blog. https://events.ccc.de/congress/, 2024. Accessed: March 2024.

[10] 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, San Francisco, CA, USA, May 18-21, 2020*, pages 1416–1432. IEEE, 2020.

[11] Yue Chen, Yulong Zhang, Zhi Wang, and Tao Wei. Downgrade attack on trustzone, 2017.

[12] DEF CON. Black hat website. https://www.blackhat.com/, 2024. Accessed: March 2024.

[13] DEF CON. Def con website. https://defcon.org/, 2024. Accessed: March 2024.

[14] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2187–2200, 2017.

[15] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84, 2013.

[16] Sadegh Farhang, Mehmet Bahadir Kirdan, Aron Laszka, and Jens Grossklags. Hey google, what exactly do your security patches tell us? a large-scale empirical study on android patched vulnerabilities, 2019.

[17] Sadegh Farhang, Mehmet Bahadir Kirdan, Aron Laszka, and Jens Grossklags. An empirical study of android security bulletins in different vendors. In *Proceedings of The Web Conference 2020*, WWW '20, page 3063–3069, New York, NY, USA, 2020. Association for Computing Machinery.

[18] Google. Trusty tee. https://source.android.com/docs/security/features/trusty, 2022. Accessed: April 2023.

[19] google. Factory images for nexus and pixel devices. https://developers.google.com/android/images, 2023. Accessed: July 2023.

[20] Google. Android security bulletin. https://source.android.com/docs/security/bulletin/asb-overview, 2024. Accessed: March 2024.

[21] graphenos.org. Grapheneos. https://grapheneos.org/, 2024. Accessed: January 2024.

[22] Lee Harrison, Hayawardh Vijayakumar, Rohan Padhye, Koushik Sen, and Michael Grace. Partemu: Enabling dynamic analysis of real-world trustzone software using emulation. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association.

[23] kutyacica. Unbox your phone (part 1 - part 3). https://labs.taszk.io/articles/post/unbox_your_phone_1/, 2018. Accessed: April 2023.

[24] laginimaineb. Exploring qualcomm's secure execution environment. http://bits-please.blogspot.com/2016/04/exploring-qualcomms-secure-execution.html, 2016. Accessed: April 2023.

[25] laginimaineb. Extracting qualcomm's keymaster keys - breaking android full disk encryption. http://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html, 2016. Accessed: April 2023.

[26] Paul Leignac, Olivier Potin, Jean-Baptiste Rigaud, Jean-Max Dutertre, and Simon Pontié. Comparison of side-channel leakage on rich and trusted execution environments. In *Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems*, CS2 '19,

page 19–22, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2201–2215. ACM, 2017.

[28] Mario Linares-Vásquez, Gabriele Bavota, and Camilo Escobar-Velásquez. An empirical study on android-related vulnerabilities. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 2–13, 2017.

[29] lineageos.org. Lineageos. https://lineageos.org/, 2024. Accessed: January 2024.

[30] Aravind Machiry, Eric Gustafson, Chad Spensky, Christopher Salls, Nick Stephens, Ruoyu Wang, Antonio Bianchi, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. Boomerang: Exploiting the semantic gap in trusted execution environments. In *NDSS*, 2017.

[31] Slava Makkaveev. The road to qualcomm trustzone apps fuzzing. https://research.checkpoint.com/2019/the-road-to-qualcomm-trustzone-apps-fuzzing/, 2019. Accessed: April 2023.

[32] Slava Makkaveev. Researching xiaomi's tee to get to chinese money. https://research.checkpoint.com/2022/researching-xiaomis-tee/, 2022. Accessed: April 2023.

[33] Alexandre Adamski Maxime Peterlin. Hara kirin, dissecting the privileged components of huawei mobile devices. https://www.hexacon.fr/slides/22-Hexacon-Hara-Kirin_Dissecting_the_Privileged_Components_of_Huawei_Mobile_Devices.pdf, 2022. Accessed: October 2023.

[34] Stuart McIlroy, Nasir Ali, and Ahmed E Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, 21:1346–1370, 2016.

[35] Frederico Menarini. Breaking tee security part 1-part 3. https://www.riscure.com/tee-security-samsung-teegris-part-1/, 2019. Accessed: April 2023.

[36] Offensivecon. Offensivecon website. https://www.offensivecon.org/, 2024. Accessed: March 2024.

[37] Google P0. Google project zero blog. https://googleprojectzero.blogspot.com/, 2024. Accessed: March 2024.

[38] Quarkslab. Quarkslab blog. https://blog.quarkslab.com/, 2024. Accessed: March 2024.

[39] Riscure. Riscure blog. https://www.riscure.com/blog/, 2024. Accessed: March 2024.

[40] Dan Rosenberg. Reflections on trusting trustzone. Black Hat 2014, 2014. Accessed: April 2023.

[41] Samsung. Samsung mobile security - security update. https://security.samsungmobile.com/securityUpdate.smsb, 2023. Accessed: April 2023.

[42] Eloi Sanfelix. Tee exploitation - exploiting trusted apps on samsung's tee. https://downloads.immunityinc.com/infiltrate2019-slidepacks/eloi-sanfelix-exploiting-trusted-apps-in-samsung-tee/TEE.pdf, 2019. Accessed: April 2023.

[43] Bruce Schneier. Full disclosure and the window of exposure. https://www.schneier.com/crypto-gram/archives/2000/0915.html, 2020. Accessed: April 2023.

[44] Blue Frost Security. Blue frost security blog. https://labs.bluefrostsecurity.de/blog/, 2024. Accessed: March 2024.

[45] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust dies in darkness: Shedding light on samsung's TrustZone keymaster design. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 251–268, Boston, MA, August 2022. USENIX Association.

[46] Statista. Smartphone life cycles are changing. https://www.statista.com/chart/8348/smartphone-life-cycles-are-changing/, 2017. Accessed: October 2023.

[47] Statista. Global smartphone market share from 4th quarter 2009 to 2nd quarter 2023. https://www.statista.com/statistics/271496/global-market-share-held-by-smartphone-vendors-since-4th-quarter-2009/, 2023. Accessed: October 2023.

[48] Statista. Replacement cycle length of smartphones in the united states 2013-2027. https://www.statista.com/statistics/619788/average-smartphone-life/, 2023. Accessed: October 2023.

[49] Nick Stephens. Behind the pwn of a trustzone. https://www.slideshare.net/GeekPwnKeen/nick-stephenshow-does-someone-unlock-your-phone-with-nose, 2016. Accessed: April 2023.

[50] Adrian Tang, Simha Sethumadhavan, and Salvatore J Stolfo. Clkscrew: Exposing the perils of security-oblivious energy management. In *USENIX Security Symposium*, volume 2, pages 1057–1074, 2017.

[51] topjohnwu. Magisk. https://github.com/topjohnwu/Magisk, 2023. Accessed: April 2023.

[52] TrustedFirmware.org. Op-tee documentation - trusted applications. https://optee.readthedocs.io/en/latest/architecture/trusted_applications.html, 2023. Accessed: April 2023.

[53] Shengye Wan, Mingshen Sun, Kun Sun, Ning Zhang, and Xu He. Rustee: Developing memory-safe ARM trustzone applications. In *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*, pages 442–453. ACM, 2020.

[54] Jie Wang, Kun Sun, Lingguang Lei, Shengye Wan, Yuewu Wang, and Jiwu Jing. Cache-in-the-middle (citm) attacks: Manipulating sensitive data in isolated execution environments. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1001–1015, 2020.

[55] Widevine. Widevine. https://www.widevine.com/, 2023. Accessed: April 2023.

[56] Xiaomi. Xiaomi security center - security bulletins. https://trust.mi.com/misrc/bulletins?tab=advisory, 2023. Accessed: April 2023.

A Appendix

Size	Description
-	ELF
0x19	magic bytes
-	signatures

Table 5: Beanpod TA format (no version counter).

Vendor	Firmware Download Sources
Xiaomi	https://xiaomifirmwareupdater.com/
Samsung	https://samfw.com/ https://www.sammobile.com/firmwares/
Oppo	https://oppostockrom.com/ https://www.gsmmafia.com/oppo-flash-file/ https://mobifirmware.com/category/oppo-firmware/
Vivo	https://vivofirmware.com/ https://www.gsmmafia.com/vivo-flash-file/ https://mobifirmware.com/category/vivo-firmware/

Table 6: The websites from which we download the firmware for our study.

Size	Description
-	ELF header
0x30	metametadata
0x74	secboot metadata
0x4	*anti_rollback*
-	certificates and signature

Table 7: QSEE TA format. The TAs are split into multiple files. The metadata is stored in a .mdt file, whose structure is shown above. The rollback counter is stored in the `anti_rollback` field.

Size	Description
0x4	magic
0x4	version
0x40	various ELF and MCLF fields
0x4	*ServiceVersion*
-	ELF
-	certificates and signature

(a) Kinibi TA format.

Size	Description
0x4	SEC version
0x4	ELF size (s)
s	ELF
-	certificates and signature

(b) TEEGris SEC2 TA format.

Size	Description
0x4	SEC version
0x4	ELF size (s)
s	ELF
0x4	rollback length
0x4	*version*
-	certificates and signature

(c) TEEGris SEC3 TA format.

Identifier	Origin	TA	Type	Disclosure Date	Patch Date
SVE-2017-{8889, 8891, 8892}	Samsung	ESECOMM	stack overflow	4.2017	8.2017
SVE-2017-8890	Samsung	ESECOMM	OOB read	4.2017	8.2017
SVE-2017-8893	Samsung	ESECOMM	arbitrary write	4.2017	8.2017
SVE-2018-12852	Samsung	ESECOMM	stack overflow	8.2018	10.2018
SVE-2018-12855	Samsung	vaultkeeper	double fetch	8.2018	11.2018
SVE-2018-12853	Samsung	fingerprint	invalid free	8.2018	10.2018
SVE-2017-{9008, 9009}	Samsung	CCM	integer overflow	4.2017	8.2017
SVE-2017-10638	Samsung	CCM	session hijack	9.2017	1.2018
SVE-2019-16665	Samsung	SEM	stack overflow	7.2019	6.2020
SVE-2019-13958	Samsung	gatekeeper	information disclosure	2.2019	5.2019
SVE-2019-14126	Samsung	keymaster	heap overflow	3.2019	5.2019
SVE-2021-21948	Samsung	keymaster	IV reuse	5.2021	8.2021
SVE-2019-14847	Samsung	EXT_FR	parameter confusion	6.2019	8.2019
SVE-2019-14850	Samsung	HDCP	parameter confusion	6.2019	8.2019
SVE-2019-14665	Samsung	HDCP	stack overflow	5.2019	8.2019
SVE-2019-14851	Samsung	SEC_FR	parameter confusion	6.2019	8.2019
SVE-2019-14864	Samsung	FINGERPRINT	parameter confusion	6.2019	8.2019
SVE-2019-14867	Samsung	MLDAP	parameter confusion	6.2019	8.2019
SVE-2019-14885	Samsung	WVDRM	parameter confusion	6.2019	8.2019
SVE-2019-14892	Samsung	SPKM	parameter confusion	6.2019	8.2019
SVE-2019-14891	Samsung	SEM	parameter confusion	6.2019	8.2019
SVE-2019-15873	Samsung	WVDRM	arbitrary write/read	11.2019	2.2020
SVE-2020-16908	Samsung	GATEKE	credential bruteforce	2.2020	5.2020
SVE-2021-22658	Samsung	KEYMST/skeymast	crypto downgrade	7.2021	10.2021
CVE-2020-13832	TEEGRIS	WVDRM	multiple	3.2020	6.2020
SVE-2019-13952	Samsung	sec_store	integer underflow	2.2019	5.2019
SVE-2019-13949	Samsung	Authnr	null dereference	4.2019	5.2019
SVE-2019-13950	Samsung	ESECOMM	null dereference	2.2019	5.2019
CVE-2020-14125	Xiaomi	soter	heap overflow	12.2021	2.2022
None	Xiaomi	soter	parameter confusion	-	12.2020
None	Xiaomi	fido	parameter confusion	-	4.2021
None	Xiaomi	alipay	parameter confusion	-	4.2021
CVE-2023-32834	Xiaomi, Vivo, Oppo	ta_secmem	parameter confusion	5.2023	10.2023
CVE-2023-32835	Xiaomi, Vivo, Oppo	ta_keyinstall	parameter confusion	5.2023	10.2023,
MSV-828	Xiaomi, Vivo, Oppo	ta_decoder	parameter confusion	-	5.2023
CVE-2023-32848	Xiaomi, Vivo, Oppo	ta_decoder2	parameter confusion	5.2023	10.2023
CVE-2023-32849	Xiaomi, Vivo, Oppo	ta_cmdq	parameter confusion	5.2023	10.2023
CVE-2023-20722	Xiaomi, Vivo, Oppo	ta_m4u	parameter confusion	-	5.2023

Table 9: An overview of the public vulnerabilities we use in our study.