Poster: TEEzz – Fuzzing Trusted Applications on COTS Android Devices

Marcel Busch, EPFL
Aravind Machiry, Purdue University
Chad Spensky, Allthenticate
Giovanni Vigna, UC Santa Barbara
Christopher Kruegel, UC Santa Barbara
Mathias Payer, EPFL

***Abstract*—Security and privacy-sensitive smartphone applications use trusted execution environments (TEEs) to protect sensitive operations from malicious code. By design, TEEs have privileged access to the entire system but expose little to no insight into their inner workings. Moreover, real-world TEEs enforce strict format and protocol interactions when communicating with trusted applications (TAs), which prohibits effective automated testing.**

TEEzz is the first TEE-aware fuzzing framework capable of effectively fuzzing TAs *in situ* on production smartphones, i.e., the TA runs in the encrypted and protected TEE and the fuzzer may only observe interactions with the TA but has no control over the TA's code or data. Unlike traditional fuzzing techniques, which monitor the execution of a program being fuzzed and view its memory after a crash, TEEzz only requires a limited view of the target. TEEzz overcomes key limitations of TEE fuzzing (e.g., lack of visibility into the executed TAs, proprietary exchange formats, and value dependencies of interactions) by automatically attempting to infer the field types and message dependencies of the TA API through its interactions, designing state- and type-aware fuzzing mutators, and creating an in situ, on-device fuzzer.

Due to the limited availability of systematic fuzzing research for TAs on commercial-off-the-shelf (COTS) Android devices, we extensively examine existing solutions, explore their limitations, and demonstrate how TEEzz improves the state-of-the-art. First, we show that general-purpose kernel driver fuzzers are ineffective for fuzzing TAs. Then, we establish a baseline for fuzzing TAs using a ground-truth experiment. We show that TEEzz outperforms other blackbox fuzzers, can improve greybox approaches (if TAs source code is available), and even outperforms greybox approaches for stateful targets. We found 13 previously unknown bugs in the latest versions of OPTEE TAs in total, out of which TEEzz is the only fuzzer to trigger three. We also ran TEEzz on popular phones and found 40 unique bugs for which one CVE was assigned so far.

***Index Terms*—Fuzzing, Android, TEE, ARM TrustZone**

I. Introduction

Smartphones operate on private user data and perform sensitive functionality. To defend against various application- and kernel-level exploits, applications leverage TEEs [3] (e.g., ARM TrustZone (TZ)) as an additional hardware-based defense. TEEs enforce the integrity and confidentiality of their applications. Partially due to recent research that demonstrated the usefulness of TEE applications [4], [6], called TAs, their number, as well as their complexity, is steadily increasing, leading to more TA-based vulnerabilities [1], [2]. Unlike regular applications, where the vulnerability affects only the application, a vulnerability in a TA compromises the security of the entire system, potentially even the secure boot process [7].

While the security of these TAs is foundational to the security of the device, performing effective testing (e.g., fuzzing) remains an open challenge. Smartphones ship with the trusted OS (tOS) and numerous pre-installed TAs, prohibiting the normal world (e.g., Android) from inspecting their code at runtime. TA interactions are stateful and use complex proprietary message formats [5]. The entities in the secure world (TEE and TAs) are often encrypted and get decrypted in secure memory at runtime, prohibiting the use of static analysis-based vulnerability detection techniques. Dynamic analysis, i.e., fuzzing, is an effective alternative.

There are two principled approaches for fuzzing TAs: *rehosting through emulation* or *on-device instrumentation*.

Rehosting the TEE in an emulated environment overcomes the inaccessibility of the TEE's internal state. PartEmu [5] rehosts Samsung's proprietary TEE software stacks. They rehost the tOS and its TAs, to an emulated system-on-a-chip (SoC), gaining unrestricted access to the TEE's internal state. Limitations to this approach are (1) the reverse engineering and implementation effort for emulated software and hardware components, (2) the inaccuracy of these implementations, (3) the lack of public data sheets, and (4) industry involvement leading to non-disclosure agreements for existing solutions.

The second approach, on-device fuzzing, mitigates these limitations and inaccuracies of emulation approaches. However, it lacks access to the TEE's internal state and must fall back to blackbox fuzzing techniques. Unlike typical fuzzing techniques, which can analyze the binary, system memory, and executed instructions, an on-device TEE fuzzer must infer bugs from a far more limited view of the execution. Interactions with TAs happen through a vendor-provided interface (e.g., a driver in the rich OS (rOS), which ultimately generates an secure monitor call (SMC) to communicate with the secure world. The only observable execution effects are returned data (e.g., return values) and the status of the TA.

We present TEEzz, a fuzzing framework for TAs running on COTS smartphones. TEEzz targets three popular TEE implementations: the Qualcomm Secure Execution Environment (QSEE), used on Nexus and Pixel devices; TrustedCore (TC), found on Huawei devices; and the Open Portable Trusted Execution Environment (OPTEE), the de-facto reference implementation for TZ-based TEEs. The analysis first identifies the TAs within the TEE and then manually triggers interactions with them. During these interactions, TEEzz records the data passed both into and out of the TEE to automatically reconstruct the message format and complex message and value

dependencies. Lastly, this message format, along with the dependencies of the interaction (i.e., generating a cryptographic key before it is used for encryption), are fed into our fuzzer. The fuzzer explores the TA while continuously checking for liveness and monitoring for crashes.

TEEzz necessitates diverse contributions. First, the complex and proprietary data structures of TAs require fuzzed inputs to be well-formed, or else the parsing logic in the tOS will reject them. Thus, we designed TEEzz as a *mutation-based* fuzzer that operates on type- and state-aware seeds generated from legitimate interactions with TAs. To infer the necessary knowledge of the API, we design an inference mechanism that maps high-level abstractions to low-level messages used to communicate with the TA. TEEzz *automatically* generates memory introspection logic for each parameter type of the exposed interface and then abstracts the interaction protocol from the recorded traces. At runtime, we dynamically instrument this interface, parse the values corresponding to each type on-the-fly from memory, and save the type-aware token sequence to disk. The observed type- and state-aware interactions become the specification for efficient mutation.

Second, we *automatically* generate type-aware mutators for the enriched seeds. We convert the type definitions used by TA-facing interfaces into type-aware mutator plugins for TEEzz's mutator engine. While fuzzing, TEEzz leverages these type-specific mutators to manipulate input tokens.

Third, many TAs are stateful, and value dependencies between invocations need to be resolved, i.e., a value returned from one invocation *must* be used as an input for a future invocation. Leveraging the previously recorded type-enriched interaction sequences, which include ingoing and outgoing data, TEEzz employs a novel value dependency inference technique to add state-awareness to its seeds.

Finally, TEEzz is the first end-to-end solution capable of continuously fuzzing TAs on COTS Android devices. Although we use known techniques such as dynamic binary instrumentation-based introspection (DBII) and semantic reconstruction, the novelty of TEEzz stems from solving technical challenges and applying these techniques to a restricted environment of a COTS device with no direct access to secure world entities. In addition, TEEzz features an extensible type-aware mutation engine, a state-aware fuzzing paradigm that considers entire interaction sequences as seeds and resolves interaction dependencies during runtime. Further, it supports state reset mechanisms to deterministically build up TA state to facilitate the reproduction of crashes.

We evaluated TEEzz's performance in terms of coverage and bug-finding capabilities in a ground-truth experiment. For this purpose, we extended the OPTEE platform with (1) permanently shared memory between client applications (CAs) and TAs, (2) TA instrumentation to populate the coverage bitmap, (3) TA instrumentation to collect program counters during post-processing, and (4) support for TA constructors to initialize the instrumentation. Due to the non-availability of related fuzzers, we truthfully replicate the state-of-the-art based on AFL++ and compare TEEzz against three TA-aware AFL++ variants. Our results show that TEEzz finds bugs that are unreachable for existing fuzzers. In fact, TEEzz was the only fuzzer capable of finding three previously unknown bugs in OPTEE TAs. Further, we tested TEEzz on 18 TAs covering four popular Google and Huawei phones. TEEzz successfully generated enriched seeds, inferred interaction dependencies, and fuzzed each TA. Across these proprietary targets, TEEzz successfully found 40 unique bugs that we responsibly reported to the corresponding vendors. One CVE (CVE-2019-10561) was assigned so far, and we await further replies. Some of these crashes force the phone into a factory reset—wiping all user data— to resume normal functionality. In contrast, others allowed us to extract protected cryptographic keys from the TEE, a stepping stone to launch brute-force attacks against a device's disk encryption.

In summary, our contributions are as follows:

- We developed TEEzz, available at https://github.com/HexHive/teezz-fuzzer, the first end-to-end automated fuzzing framework capable of fuzzing TAs on commercially available smartphones;
- an automated, dynamic-analysis-based technique for inferring field types of messages, as well as their dependencies, to facilitate type-aware fuzzing of stateful TAs;
- type and state-aware fuzzing mutators that leverage the message and dependency information inferred from analyzing the interactions with the TAs; and
- a thorough evaluation of TEEzz against other state-of-the-art fuzzing techniques on production TAs.

This poster builds on work accepted for publication at the 44th IEEE Symposium on Security and Privacy.

References

[1] Marcel Busch, Johannes Westphal, and Tilo Müller. Unearthing the trustedcore: A critical review on huawei's trusted execution environment. In Yuval Yarom and Sarah Zennou, editors, *Proceedings of the Workshop on Offensive Technologies, WOOT*. USENIX Association, 2020.

[2] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 18–20, 2020.

[3] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. Trusted execution environments on mobile devices. In *Proceedings of the ACM SIGSAC conference on Computer & communications security (CCS)*, pages 1497–1498. ACM, 2013.

[4] Le Guan, Peng Liu, Xinyu Xing, Xinyang Ge, Shengzhi Zhang, Meng Yu, and Trent Jaeger. Trustshadow: Secure execution of unmodified applications with arm trustzone. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 488–501. ACM, 2017.

[5] Lee Harrison, Hayawardh Vijayakumar, Rohan Padhye, Koushik Sen, Michael Grace, Rohan Padhye, Caroline Lemieux, Koushik Sen, Laurent Simon, Hayawardh Vijayakumar, et al. Partemu: Enabling dynamic analysis of real-world trustzone software using emulation. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, 2020.

[6] Jinsoo Jang, Changho Choi, Jaehyuk Lee, Nohyun Kwak, Seongman Lee, Yeseul Choi, and Brent Byunghoon Kang. Privatezone: Providing a private execution environment using arm trustzone. *IEEE Transactions on Dependable and Secure Computing*, 15(5):797–810, 2018.

[7] Nilo Redini, Aravind Machiry, Dipanjan Das, Yanick Fratantonio, Antonio Bianchi, Eric Gustafson, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Bootstomp: on the security of bootloaders in mobile devices. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2017.