Poster: SyzTrust: State-aware Fuzzing on Trusted OS Designed for IoT Devices

***Abstract*—Trusted Execution Environments (TEEs) embedded in IoT devices provide a deployable solution to secure IoT applications at the hardware level. By design, in TEEs, the Trusted Operating System (Trusted OS) is the primary component. It enables the TEE to use security-based design techniques, such as data encryption and identity authentication. Once a Trusted OS has been exploited, the TEE can no longer ensure security. However, Trusted OSes for IoT devices have received little security analysis, which is challenging from several perspectives: (1) Trusted OSes are closed-source and have an unfavorable environment for sending test cases and collecting feedback. (2) Trusted OSes have complex data structures and require a stateful workflow, which limits existing vulnerability detection tools.**

To address the challenges, we present SYZTRUST, the first state-aware fuzzing framework for vetting the security of resource-limited Trusted OSes. SYZTRUST adopts a hardware-assisted framework to enable fuzzing Trusted OSes directly on IoT devices as well as tracking state and code coverage non-invasively. SYZTRUST utilizes composite feedback to guide the fuzzer to effectively explore more states as well as to increase the code coverage. We evaluate SYZTRUST on Trusted OSes from three major vendors: Samsung, Tsinglink Cloud, and Ali Cloud. These systems run on Cortex M23/33 MCUs, which provide the necessary abstraction for embedded TEEs. We discovered 70 previously unknown vulnerabilities in their Trusted OSes, receiving 10 new CVEs so far. Furthermore, compared to the baseline, SYZTRUST has demonstrated significant improvements, including 66% higher code coverage, 651% higher state coverage, and 31% improved vulnerability-finding capability. We report all discovered vulnerabilities to vendors and open source SYZTRUST.

I. INTRODUCTION

Trusted Execution Environments (TEEs) are essential to securing important data and operations in IoT devices. GlobalPlatform, the leading technical standards organization, has reported a 25-percent increase in the number of TEE-enabled IoT processors being shipped quarterly, year-over-year [1]. Recently, major IoT vendors such as Samsung have designed TEEs for low-end Microcontroller Units (MCUs) [2], [3] and device manufacturers have embedded the TEE in IoT devices such as unmanned aerial vehicles and smart locks, to protect sensitive data and to provide key management services. A TEE is composed of Client Applications (CAs), Trusted Applications (TAs), and a Trusted Operating System (Trusted OS). Among them, the Trusted OS is the primary component to enable the TEE using security-based design techniques, and its security is the underlying premise of a reliable TEE where the code and data are protected in terms of confidentiality and integrity. Unfortunately, implementation flaws in Trusted OSes violate the protection guarantees, bypassing confidentiality and integrity guarantees. These flaws lead to critical consequences, including sensitive information leakage (CVE-2019-25052) and code execution [4]. Once attackers gain control of Trusted OSes, they can launch critical attacks, such as creating a backdoor to the Linux kernel [5] and extracting full disk encryption keys of Android's KeyMaster service [6].

While TEEs are increasingly embedded in IoT devices, the security of Trusted OS for IoT devices remains under studied. Considering the emerging amount of diversified MCUs and IoT devices, manual analysis, such as reverse engineering, requires significant expert efforts and is therefore infeasible at scale. Recent academic works use fuzzing to automate TEE testing. However, unlike Trusted OSes for Android devices, Trusted OSes for IoT devices are built on TrustZone-M with low-power and cost-sensitive MCUs, including NuMicro M23. Thus, Trusted OSes for IoT devices are hardware-dependent and resource-constrained, complicating the development of scalable and usable testing approaches. Below, we identify two challenges in fuzzing IoT Trusted OSes.

Challenge I: The inability of instrumentation and restricted environment. Most Trusted OSes are closed-source. Additionally, TEE implementations, especially the Trusted OSes are often encrypted by IoT vendors, which implies the inability to instrument and monitor the code execution in the secure world. Accordingly, classic feedback-driven fuzzing cannot be directly applied to the scenario of testing TEEs including TAs and Trusted OSes. Existing works either rely on on-device binary instrumentations [7] or require professional domain knowledge and rehosting through proprietary firmware emulation [8] to enable testing and coverage tracking. However, as for the Trusted OSes designed for IoT devices, the situation is more challenging due to the following two reasons. First, IoT devices are severely resource-limited, while existing binary instrumentations are heavy-weight for them and considerably limit their execution speed. Second, as for rehosting, IoT devices are mostly hardware-dependent, rendering the reverse engineering and implementation effort for emulated software and hardware components unacceptable. In addition, rehosting faces the limitation of the inaccuracy of modeling the behavior of hardware components. To our best knowledge, the only existing TEE rehosting solution PartEmu [8] is not publicly available and does not support the mainstream TEE based on Cortex-M MCUs designed for IoT devices.

Challenge II: Complex structure and stateful workflow. Trusted OSes for IoT devices are surprisingly complex. Specifically, Trusted OSes implement multiple cryptographic algorithms, such as AES and MAC, without underlying hardware support for these algorithms as would be present on Cortex-

A processors. To implement these algorithms in a secure way, Trusted OSes maintain several state diagrams to store the execution contexts and guide the execution workflow. To explore more states of a Trusted OS, a fuzzer needs to feed syscall sequences in several specific orders with different specific state-related argument values. Without considering such statefulness of Trusted OSes, coverage-based fuzzers are unlikely to explore further states, causing the executions to miss the vulnerabilities hidden in a deep state. Unfortunately, existing fuzzing techniques lack state-awareness for Trusted OSes. Specifically, they have trouble understanding which state a Trusted OS reaches since there are no rich-semantics response codes to indicate it. In addition, due to the lack of source code and the inability of instrumentation, it is hard to infer and extract the state variables by program analysis.

Our solution. To address the above key challenges, we propose and implement SyzTrust, the first fuzzing framework targeting Trusted OSes for IoT devices, supporting state and coverage-guided fuzzing. Specifically, we propose an on-device fuzzing framework and leverage a hardware-in-the-loop approach. To support in-depth vulnerability detection, we propose a composite feedback mechanism that guides the fuzzer to explore more states and increase code coverage.

SyzTrust necessitates diverse contributions. First, to tackle Challenge I, we propose a hardware-assisted fuzzing framework to execute test cases and collect code coverage feedback. Specifically, we decouple the execution engine from the rest of the fuzzer to enable the direct execution of test cases in the protective TEE secure world on the resource-limited MCU. To support coverage tracking, we present a selective trace collection approach that enables tracing instructions on a target MCU instead of costly code instrumentation. In particular, we leverage the ARM Embedded Trace Macrocell (ETM) feature to collect raw trace packets by monitoring instruction and data buses on MCU with a low-performance impact. We offload heavy-weight tasks to a PC and carefully schedule the fuzzing subprocesses in a parallel way. Furthermore, we present an event- and address-based trace filter to handle the noisy trace packets that are not generated by the Trusted OS.

Second, as for the Challenge II, the vulnerability detection capability of coverage-based fuzzers is limited, and a more effective fuzzing strategy is required. Therefore, we propose a composite feedback mechanism that enhances code coverage with state feedback. Specifically, we utilize state variables that control the execution contexts to present the states of a Trusted OS. However, such state variables are usually stored in closed-source and customized data structures within Trusted OSes. Existing state variable inference methods either use explicit protocol packet sequences [9] or require source codes of target software [10], [11], which are unavailable for Trusted OSes. Therefore, to identify the state-related members from those complex data structures, SyzTrust collects some heuristics for Trusted OS and utilizes them to perform an active state variable inference algorithm. After that, SyzTrust monitors the state variable values during the fuzzing procedure as the state feedback. Finally, SyzTrust is the first end-to-end solution capable of fuzzing Trusted OSes for IoT devices. Moreover, the design of the on-device fuzzing framework and modular implementation make SyzTrust more extensible. With several MCU-specific configurations, SyzTrust scales to Trusted OSes on different MCUs from different vendors.

Evaluation. We evaluate SyzTrust on real-world Trusted OSes from three leading IoT vendors Samsung, Tsinglink Cloud, and Ali Cloud. The evaluation result shows that SyzTrust is effective at discovering new vulnerabilities and exploring new states and codes. As a result, SyzTrust has discovered 70 new vulnerabilities. Among them, vendors confirmed 28, and assigned 10 CVE IDs. The vendors are still investigating others. Compared to state-of-the-art approaches, SyzTrust finds more vulnerabilities, hits 66% higher code branches, and 651% higher state coverage.

Summary and contributions.

• We propose SyzTrust, the first fuzzing framework targeting Trusted OSes for IoT devices, supporting effective state and code coverage guided fuzzing. With a carefully designed hardware-assisted fuzzing framework and a composite feedback mechanism, SyzTrust is extensible and configurable to different IoT devices.

• With SyzTrust, we evaluate three popular Trusted OSes on three leading IoT vendors and detect several previously unknown bugs. We have responsibly reported these vulnerabilities to the vendors and got acknowledged from vendors such as Samsung. We release SyzTrust as an open-source tool for facilitating further studies at https://github.com/SyzTrust.

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

References

[1] GlobalPlatform, "GlobalPlatform TEE spec adoption to reach 10 billion," https://globalplatform.org/latest-news/globalplatform-tee-spec-adoption-to-reach-10-billion.

[2] Samsung, "mTower," https://github.com/Samsung/mTower.

[3] "About Alibaba Cloud Link TEE," https://iot.aliyun.com/products/tee.

[4] G. Beniamini, "TrustZone kernel privilege escalation," http://bits-please.blogspot.com/2016/06/trustzone-kernel-privilege-escalation.html.

[5] G. Beniamini, "Hijacking the linux kernel from QSEE," https://bits-please.blogspot.com/2016/05/war-of-worlds-hijacking-linux-kernel.html.

[6] G. Beniamini, "Extracting Qualcomm's keymaster keys - breaking android full disk encryption," https://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html.

[7] M. Busch, A. Machiry, C. Spensky, G. Vigna, C. Kruegel, and M. Payer, "TEEzz: Fuzzing Trusted Applications on COTS Android devices," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2023.

[8] L. Harrison, H. Vijayakumar, R. Padhye, K. Sen, and M. Grace, "PARTEMU: Enabling dynamic analysis of real-world TrustZone software using emulation," in *Proceedings of USENIX Security Symposium (SEC)*, 2020.

[9] C. McMahon Stone, S. L. Thomas, M. Vanhoef, J. Henderson, N. Bailluet, and T. Chothia, "The closer you look, the more you learn: A grey-box approach to protocol state machine learning," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2265–2278.

[10] K. Kim, T. Kim, E. Warraich, B. Lee, K. R. Butler, A. Bianchi, and D. J. Tian, "FUZZUSB: Hybrid stateful fuzzing of USB gadget stacks," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2022.

[11] B. Zhao, Z. Li, S. Qin, Z. Ma, M. Yuan, W. Zhu, Z. Tian, and C. Zhang, "StateFuzz: System call-based state-aware linux driver fuzzing," in *Proceedings of USENIX Security Symposium (SEC)*, 2022.

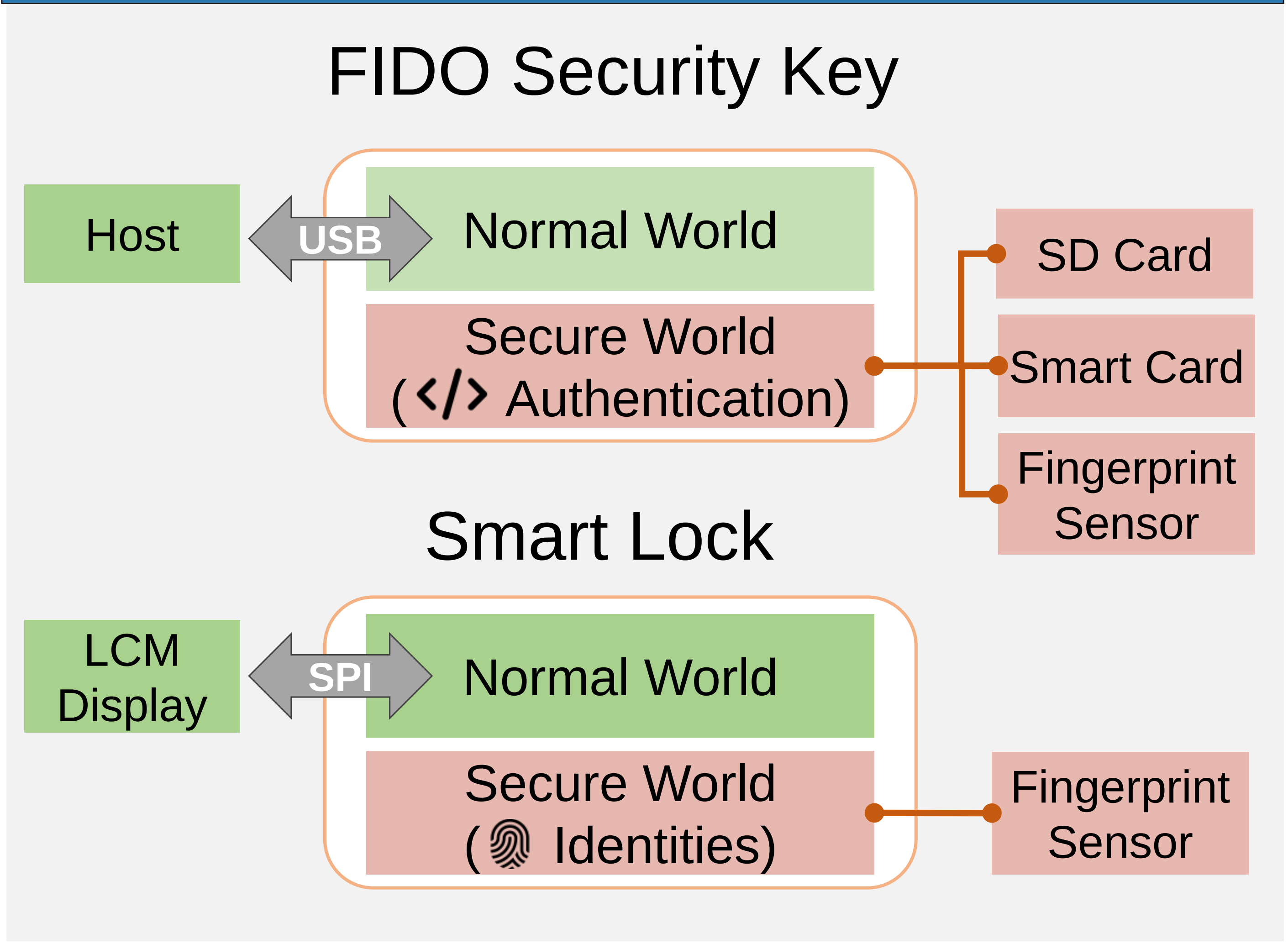
Accepted at *IEEE S&P 2024*

浙江大学 ZHEJIANG UNIVERSITY   EPFL   UCLA   Georgia Tech

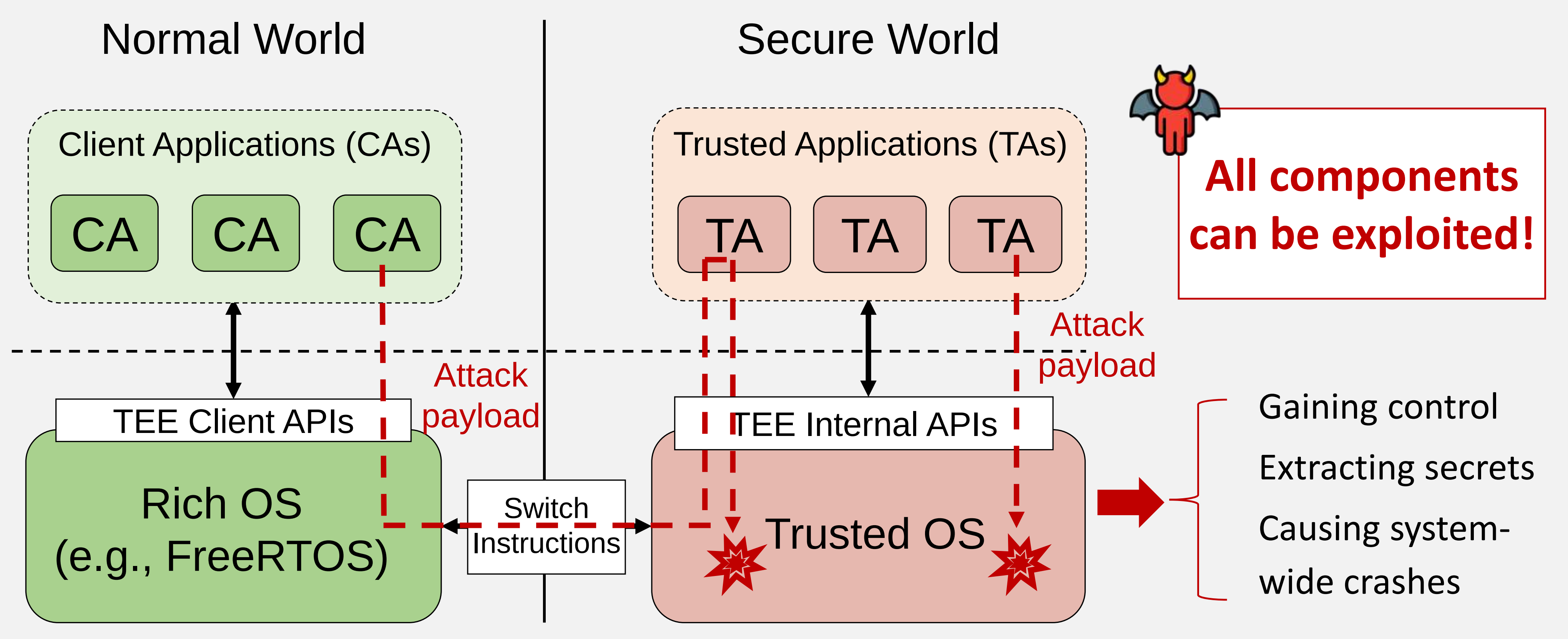
SyzTrust: State-aware Fuzzing on Trusted OS Designed for IoT Devices

Qinying Wang, Boyu Chang, Shouling Ji, Yuan Tian, Xuhong Zhang, Binbin Zhao, Gaoning Pan, Chenyang Lyu, Mathias Payer, Wenhai Wang, Raheem Beyah

TEE Use-cases in IoT Devices

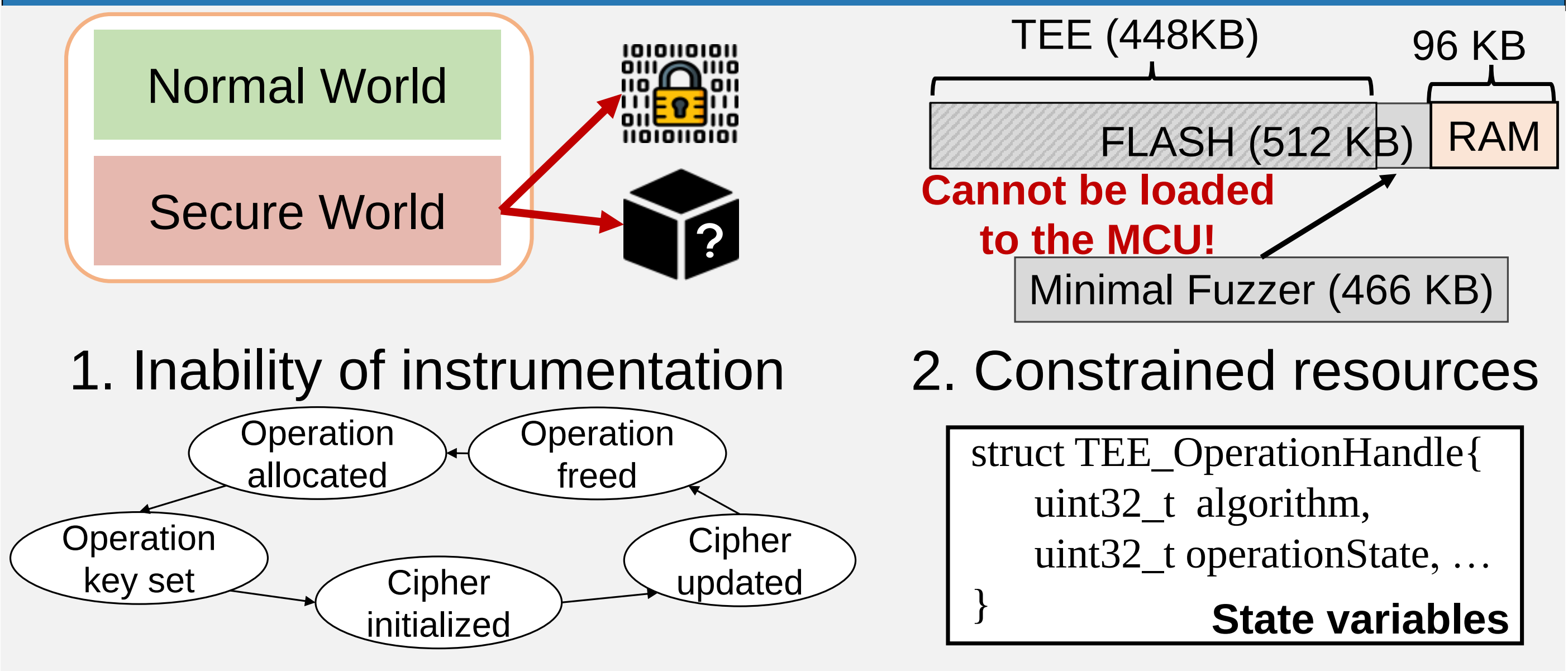


The Security of Trusted OS Matters



Trusted OS is the **primary** component to enable the TEE to use security-based design techniques.

Key Challenges

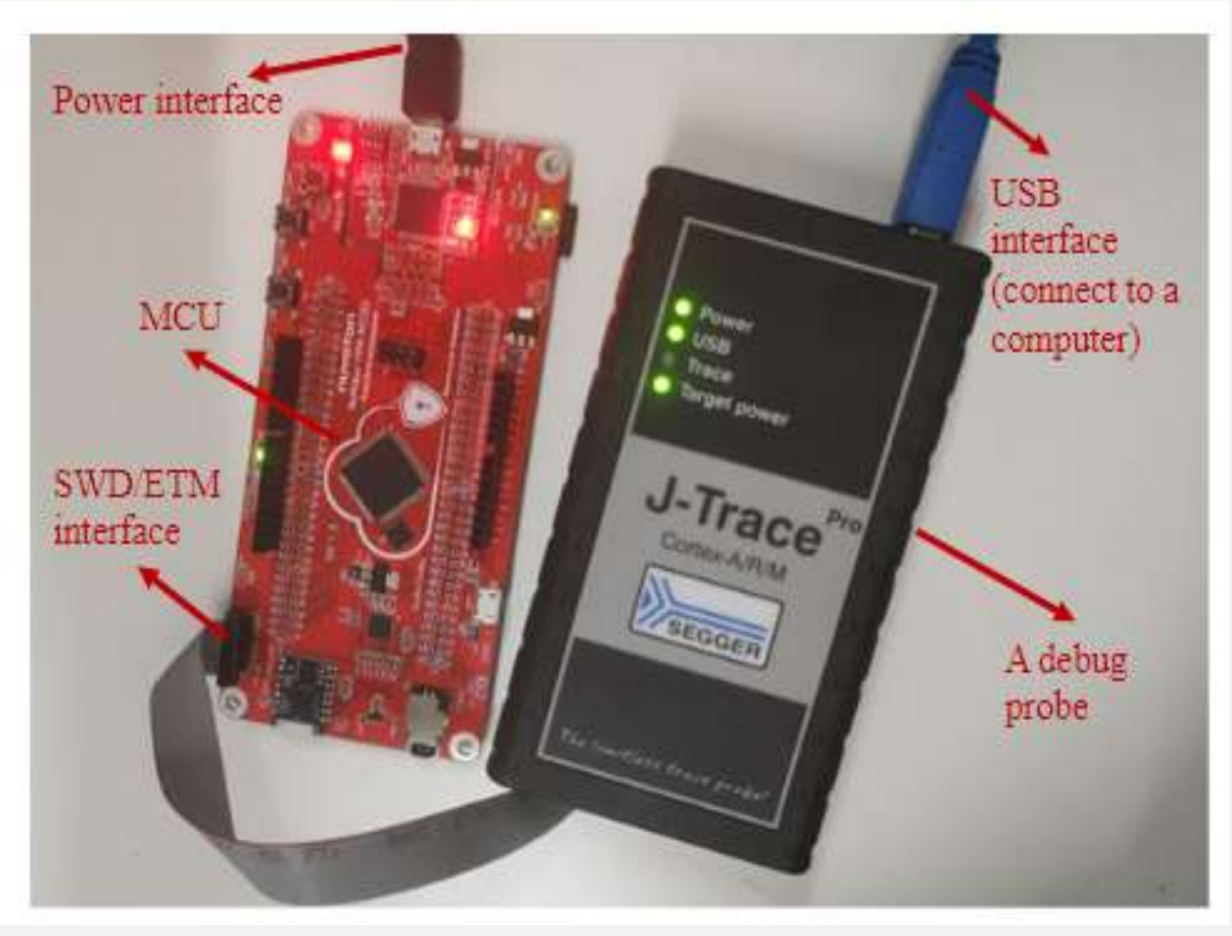


```
struct TEE_OperationHandle{
   uint32_t  algorithm,
   uint32_t operationState, …
}                              State variables
```

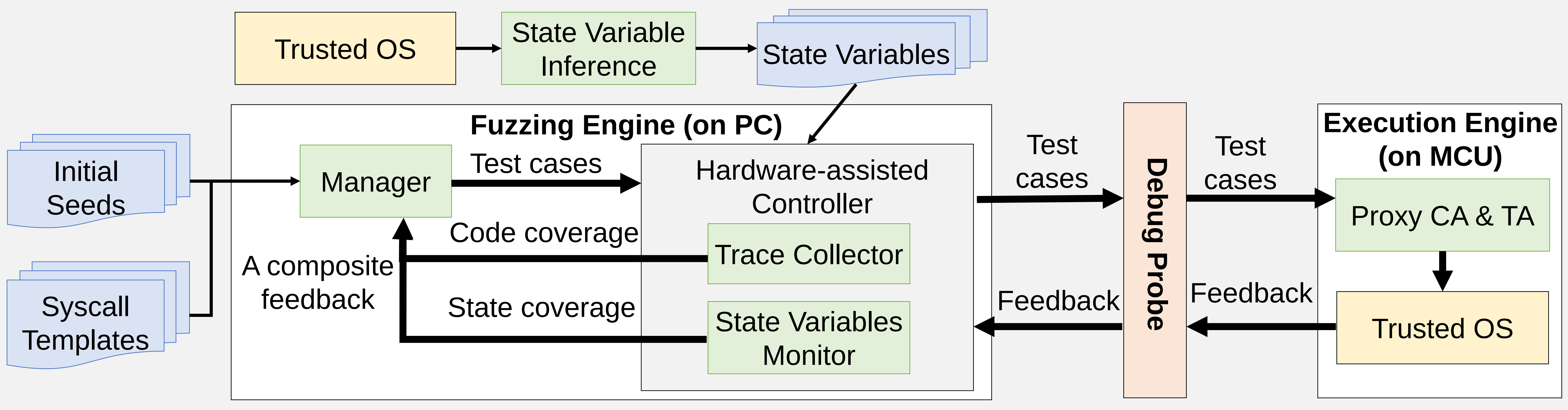
1. Inability of instrumentation
2. Constrained resources
3. Stateful workflow
4. Complex structure

Key Observations towards A Solution

- A **hardware-assisted** fuzzing framework
- **Decouple** execution offload heavy-weight tasks
- Utilize the ETM feature to **track coverage unintrusively**
- An active **state variable inference** algorithm
- A composite feedback mechanism, leveraging **both code and state coverage**

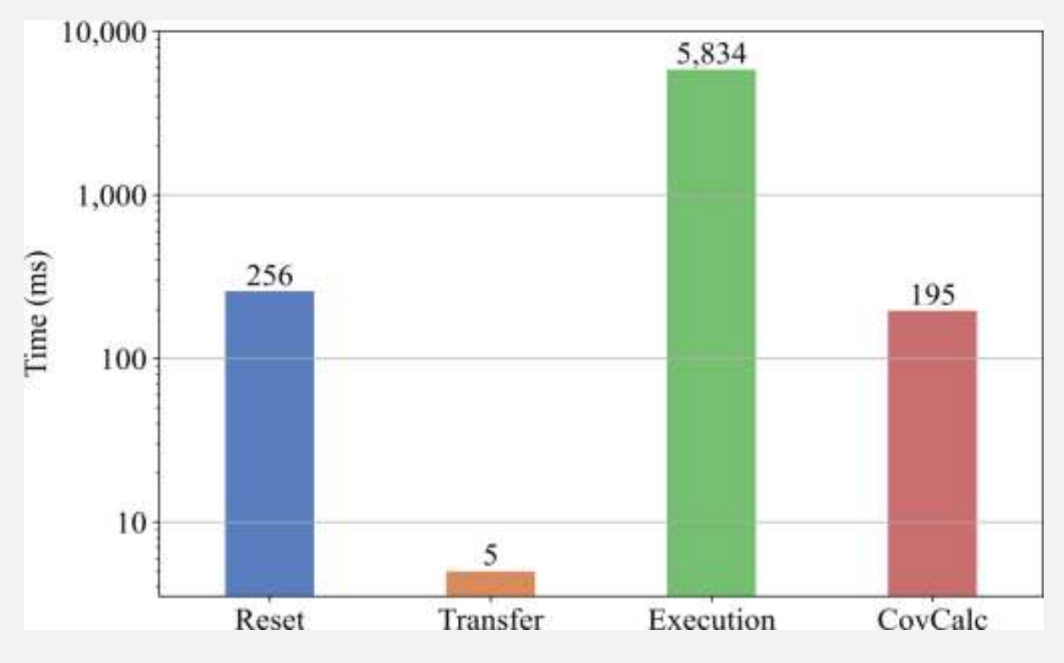


Overview of SyzTrust

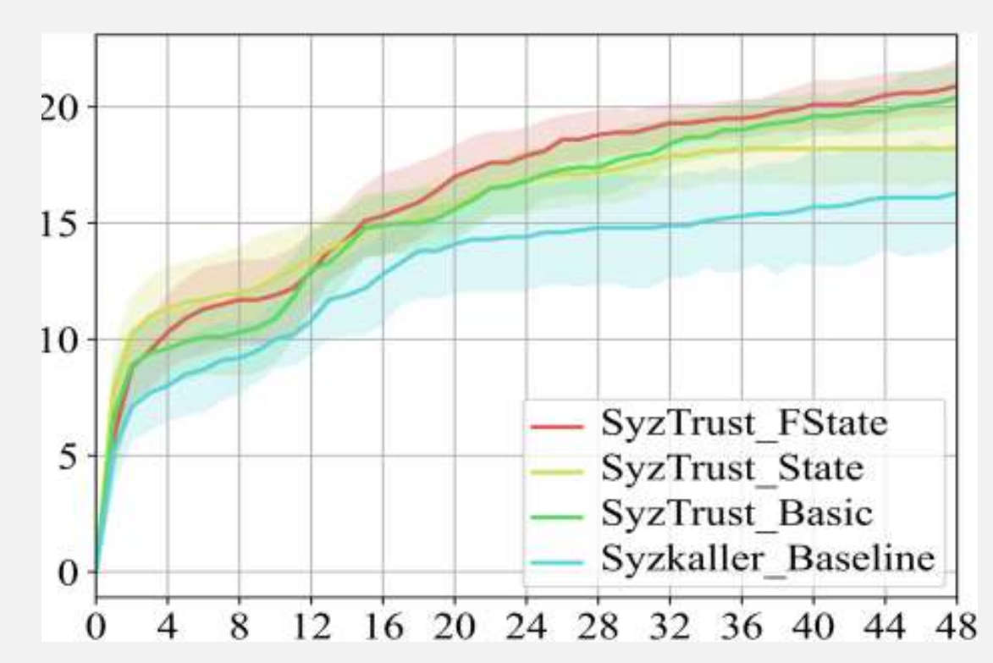


Evaluation

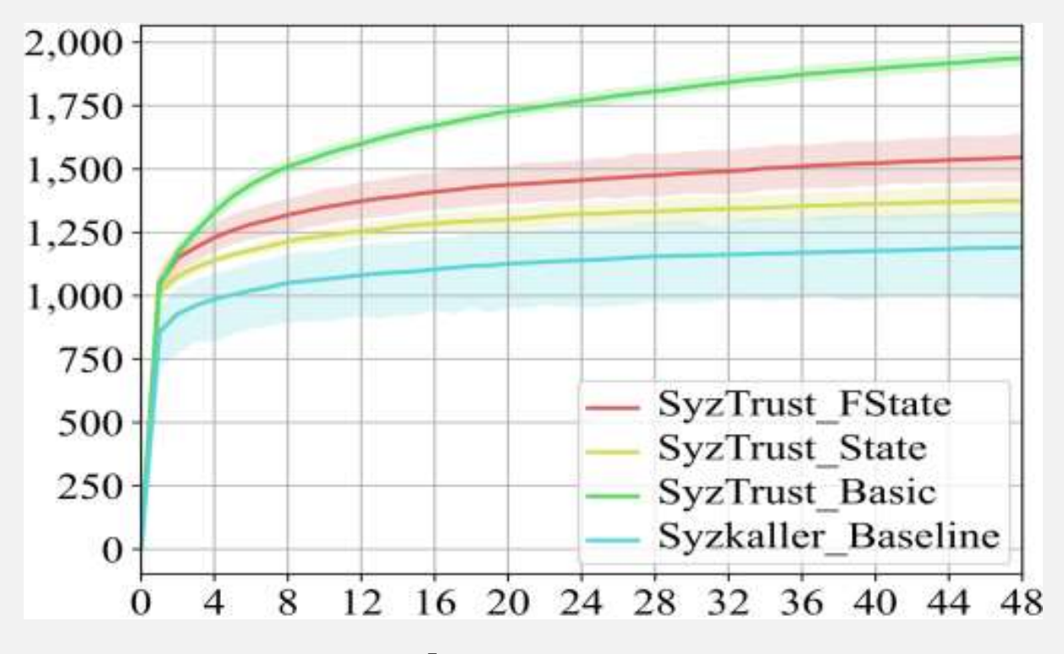
mTower — SAMSUNG
Link TEE Air — Alibaba Cloud
TinyTEE — Tsinglink Cloud



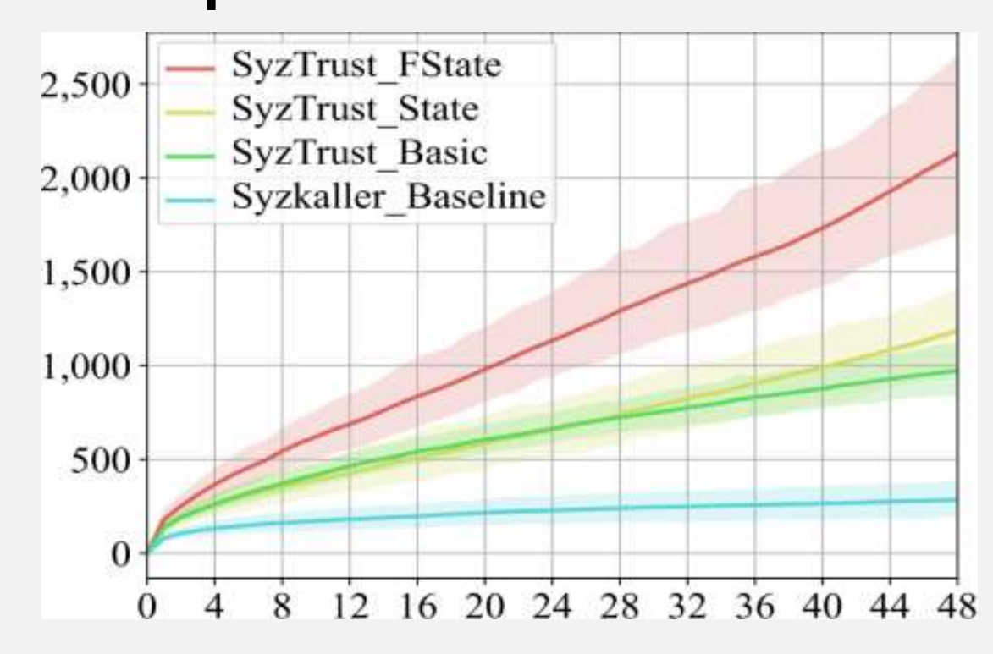
Overhead breakdown



Unique vulnerabilities



Branch coverage



State coverage

Summary

- SyzTrust is the **first** fuzzing framework targeting IoT Trusted OSes.
- With SyzTrust, we discovered **70 unknown vulnerabilities.**

浙江大学网络系统安全与隐私实验室 NETWORK SYSTEM SECURITY & PRIVACY LAB   hexhive