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
Contents

• 1. Motivation and Challenges
• 2. Methodology
• 3. Evaluation
• 4. Summary
Motivation

- Trust Execution Environments (TEEs) are essential to securing important data and operation in IoT devices.
Motivation

- Trusted OS is the **primary** component to enable the TEE to use security techniques.
- The flaws in Trusted OS result in **sensitive data leakage** and **code execution**.

---

![Diagram of TEE Client APIs and Trusted OS](image)

- Gaining control
- Extracting confidential data
- Causing system-wide crashes

... Trusted OS or other TAs
Challenges of Fuzzing Trusted OSes

- **Challenge 1**: Inability of instrumentation and constraint resource

  - **Normal World**
  - **Secure World**

  [Diagram showing RAM: 96 KB, MCU FLASH: 512 KB, TEE: 448 KB, Minimal Fuzzer: 466 KB]

  - Cannot be loaded to the MCU!

- **Challenge 2**: Stateful workflow and complex structure

  - Operation allocated
  - Operation freed
  - Operation key set
  - Cipher initialized
  - Cipher updated

  ```c
  struct TEE_OperationHandle{
    uint32_t algorithm,
    uint32_t operationState,
    TEE_ObjectHandle key...
  }
  ```

  Control the execution context
Methodology
Observations and Intuitions

- **Inability of instrumentation:** ARM Coresight ETM provides real-time instruction tracing, which can be utilized to calculate code coverage.

- **Constraint resource:** we can **decouple** execution to offload heavy-weight tasks to the PC (e.g., seed scheduling).

A *hardware-in-the-loop* framework
Observations and Intuitions

- Several variables in **handle structures** determine the Trusted OS’ internal state.
- **State coverage** can be calculated based on the combination values of the variables, which supplement code coverage.
SyzTrust End-to-End

- The fuzzing engine generates and sends test cases to the MCU via a debug probe.
- The execution engine executes the received test case on the target Trusted OS.
SyzTrust End-to-End

- The fuzzing engine generates and sends test cases to the MCU via a debug probe.
- The execution engine executes the received test case on the target Trusted OS.

![Diagram of fuzzing engine and execution engine](image)

- **Fuzzing Engine (on PC)**:
  - Initial Seeds
  - Manager
  - A composite feedback

- **Execution Engine (on MCU)**:
  - Proxy CA & TA
  - Trusted OS

- **Fuzzing Loop**
  - Test cases
  - Feedback
  - Hardware-assisted Controller
    - Trace Collector
    - State Variables Monitor
  - Code coverage
  - State coverage

- **Trusted OS**
- State Variable Inference
- State Variables

**State Variables**
- Initial Seeds
- Syscall Templates

**Test cases**
- Debug Probe

**Feedback**
- Manager
- Execution Engine (on MCU)
SyzTrust – Trace Collector

- **Problem:** the ETM component records all instruction traces generated by the CA, rich OS, the TA, and the Trusted OS, which contain noisy trace packets.

- **Solution:** collect instruction traces only when Trusted OS executes a syscall.

![Diagram](image)

An event-based filter via the Data Watchpoint and Trace Unit
SyzTrust – Trace Collector

- **Problem:** aligning decoded ETM packets to disassembled instruction sequences is hard and time-consuming.

- **Solution:** directly calculate the coverage via ETM branch and P-header packets.

---

**Branch Coverage**

PC addresses

ETM packets

LCSAJ Block

LCSAJ\_i

LCSAJ\_(i+1)
SyzTrust End-to-End

- The fuzzing engine generates and sends test cases to the MCU via a debug probe.
- The execution engine executes the received test case on the target Trusted OS.
SyzTrust – State Variable Inference and Monitor

- **Goal:** infer the address ranges of state variables before fuzzing
  track the values of the address ranges during fuzzing

```
TEE_AllocateOperation
(TEE_OperationHandle *operation,
 uint32_t algorithm, uint32_t mode,
 uint32_t maxKeySize)
TEE_ResetOperation(...)...
```

```
struct __TEE_OperationHandle{  
    [0:3] algorithm: 0000 0000 00c0 0000 ...
    ...
    [40:43] operationState: 0001
    ...
}
```

Test Cases

Test Harness

Output Buffer

Trusted OS

State Variable Inferer

Hardware-assisted State Variables Monitor
SyzTrust – State Variable Inference and Monitor

• **Goal:**
  - infer the address ranges of state variables before fuzzing
  - track the values of the address ranges during fuzzing

\[
\text{StateHash} = \text{Hash}([0:3] \ldots [40:43])
\]
SyzTrust End-to-End

- The fuzzing engine generates and sends test cases to the MCU via a debug probe.
- The execution engine executes the received test case on the target Trusted OS.
SyzTrust – Fuzzing Loop and Composite Feedback Mechanism

- **Goal**: state and code coverage guided seed preservation.

---

**Test cases**

- New code or state coverage is triggered

---

**Corpus**

- Hit Times: $Hit_1, Hit_2, \ldots, Hit_N$
- State Hashes: $St_1, St_2, \ldots, St_N$
- Seed Buckets: $Sd_1, Sd_2, Sd_3, \ldots, Sd_M$

- **A Seed Node**
  - A Syscall Sequence with Arguments
  - Branch Coverage

---

**Seed Preservation**
SyzTrust – Fuzzing Loop and Composite Feedback Mechanism

- **Goal**: state and code coverage guided seed collection.

---

**Seed Selection**

1. **Choose the state that rarely hit**
2. **Choose the seed that achieves higher branch coverage**
Evaluation
Evaluation – Effectiveness of SyzTrust

- SyzTrust outperforms Syzkaller in terms of code and state coverage and detected vulnerabilities on mTower from Samsung.
Evaluation – Effectiveness of State Variable Inference

- On average, our active state variable inference method achieves \textit{83.3\% precision}. From semantics perspective, the inferred state variables are meaningful.
Evaluation – Real World Vulnerabilities

- SyzTrust identifies **70 vulnerabilities** on Trusted OSes from Samsung, Alibaba and Tsinglink Cloud, resulting in **10 CVEs**.

<table>
<thead>
<tr>
<th>Target</th>
<th>Unique bugs</th>
<th>Branches</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTower</td>
<td>38</td>
<td>2,105</td>
<td>3,994</td>
</tr>
<tr>
<td>TinyTEE</td>
<td>13</td>
<td>1,072</td>
<td>2,908</td>
</tr>
<tr>
<td>Link TEE Air</td>
<td>19</td>
<td>10,710</td>
<td>182,324</td>
</tr>
</tbody>
</table>
Evaluation – Overhead Breakdown

- The subprocess of executing a test case on the MCU takes the most time, while the orchestration and analysis take only roughly 1% of the overall time.
Extend SyzTrust to Other Trusted OSes

- **Prerequisites**: (1) a TA can be installed in the Trusted OS; (2) target devices have ETM enabled.

<table>
<thead>
<tr>
<th>Extend to Trusted OS implementing standard APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) update MCU configurations;</td>
</tr>
<tr>
<td>(2) slightly adjustment on our designed TA and CA.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extend to Proprietary Trusted OSes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Update MCU configurations;</td>
</tr>
<tr>
<td>(2) augment the syscall templates</td>
</tr>
<tr>
<td>and the API declarations in our</td>
</tr>
<tr>
<td>designed TA;</td>
</tr>
<tr>
<td>(3) extract the state-related</td>
</tr>
<tr>
<td>structures (e.g., context).</td>
</tr>
</tbody>
</table>
Summary
SyzTrust: State-aware Fuzzing on Trusted OS Designed for IoT Devices

• Inability of instrumentation, constrained resource, and stateful workflow make testing IoT Trusted OS challenging.
• SyzTrust is the first fuzzing framework for IoT Trusted OSes.
  (1) A branch coverage collection utilizing ARM Coresight ETM.
  (2) A composite feedback mechanism including code and state coverage.
• SyzTrust found 70 new bugs in Trusted OSes from Samsung, Alibaba and Tsinglink Cloud.
Thanks
Backup Slides
## IoT Trusted OSes in Real World

An overview of the major Trusted OS implementations provided by leading IoT vendors

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Trusted OS</th>
<th>Standards</th>
<th>Support (installing TA)</th>
<th>Some of supported devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung</td>
<td>mTower</td>
<td>GP Standards</td>
<td>●</td>
<td>NuMaker-PFM-M2351</td>
</tr>
<tr>
<td>Alibaba</td>
<td>Link TEE Air</td>
<td>Proprietary</td>
<td>●</td>
<td>NuMaker-PFM-M2351</td>
</tr>
<tr>
<td>TsingLink Cloud</td>
<td>TinyTEE</td>
<td>GP Standards</td>
<td>●</td>
<td>NuMaker-PFM-M2351/LPC55S69/STM32L562</td>
</tr>
<tr>
<td>Beanpod</td>
<td>ISEE-M</td>
<td>GP Standards</td>
<td>●</td>
<td>LPC55S series/GD32W515/STM32L5 series</td>
</tr>
<tr>
<td>Trustonic</td>
<td>Kinibi-M</td>
<td>PSA Certified APIs</td>
<td>●</td>
<td>MicroChip SAML11</td>
</tr>
<tr>
<td>ARM</td>
<td>TF-M</td>
<td>PSA Certified APIs</td>
<td>●</td>
<td>NuMaker-PFM-M2351, STM32L5, ...</td>
</tr>
</tbody>
</table>
## IoT Trusted OSes in Real World

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Device</th>
<th>Privilege Secure Debug (including ETM)</th>
<th>Debug Authentication Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuvoton</td>
<td>NuMaker-PFM-M2351</td>
<td>Enable in default</td>
<td>ICP programming tool</td>
</tr>
<tr>
<td>NXP Semiconductors</td>
<td>LPC55S69</td>
<td>Enable in default</td>
<td>Debug credential certificate</td>
</tr>
<tr>
<td>STMicroelectronics</td>
<td>STM32L562</td>
<td>Enable in default</td>
<td>STM32CubeProgrammer</td>
</tr>
<tr>
<td>GigaDevice</td>
<td>GD32W515</td>
<td>Enable in default</td>
<td>Efuse</td>
</tr>
<tr>
<td>MicroChip</td>
<td>SAML11</td>
<td>Enable in default</td>
<td>Extern debugger</td>
</tr>
</tbody>
</table>

**ETM feature on IoT devices**