P²IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling

Bo Feng, Alejandro Mera, and Long Lu
Northeastern University

USENIX Security 2020
Microcontrollers (MCU) are ubiquitous

- MCU is a single-chip computer
- 28.1 billion MCUs are sold worldwide in 2018*

MCU vulnerabilities

• Consequences
  • Digital damage (e.g., privacy leakage)
  • Physical damage (e.g., human injury)
• Most vulnerabilities are from firmware
MCU firmware

- Whole software stack of the MCU
- Bugs appear in all components
Firmware testing

• Fuzzing can effectively find bugs on desktop programs
• As firmware has similar bugs to desktop programs, we test firmware with fuzzers
• Firmware can be tested either on a device or emulator

Because of limited resources on MCU, on-device fuzzing is not feasible
Emulator-based firmware testing
Emulator-based firmware testing

- Not emulated because peripherals are diverse and hard to emulate
- Firmware cannot boot
Existing solution (1)

Peripheral emulation:
Emulate peripheral hardware by software components in the emulator

Incomplete support for peripherals, significant manual efforts
Existing solution (2)

Hardware-in-the-loop emulation:

Use real peripheral hardware to handle peripheral access in the emulator

Rely on real hardware, slow, unscalable
Existing solution (3)

Partial emulation:

Replace peripheral-dependent firmware code with software stubs that have the same functionalities

Unable to test peripheral-dependent code, significant manual efforts
### Design goals

<table>
<thead>
<tr>
<th>Automatic</th>
<th>Hardware-independent</th>
<th>Peripheral-agnostic</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A great number of MCU devices need to be tested</td>
<td>• Firmware is tested in the emulator</td>
<td>• Peripherals are diverse</td>
<td>• Multiple fuzzer instances can run in parallel</td>
</tr>
<tr>
<td>• Limited time and money budget for testing</td>
<td>• Faster and easier to automate</td>
<td>• Handle peripherals using a uniform approach</td>
<td>• Improve code coverage</td>
</tr>
<tr>
<td>• Human efforts can be minimized</td>
<td></td>
<td>• Given a new peripheral, no extra effort is needed</td>
<td></td>
</tr>
</tbody>
</table>
Observation

- Peripherals are diverse in terms of type and functionality, but interface is not.

<table>
<thead>
<tr>
<th></th>
<th>Peripheral</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Many</td>
<td>2</td>
</tr>
<tr>
<td>Functionality</td>
<td>Many</td>
<td>3</td>
</tr>
<tr>
<td>Diversity</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
Key idea

- Treat peripherals as black box
- Abstract a model to handle register access and interrupt firing for a wide range of peripherals
## Comparison with state-of-the-art

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Automatic</th>
<th>Hardware-independent</th>
<th>Peripheral-agnostic</th>
<th>Scalable</th>
<th>Existing work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral emulation</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>GNU MCU Eclipse QEMU (2015), PartEmu (Usenix ’20)</td>
</tr>
<tr>
<td>Hardware-in-the-loop emulation</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>Avatar (NDSS ’14), Prospect (Asia CCS ’14), Surrogates (WOOT ’15), Charm (Usenix ’18)</td>
</tr>
<tr>
<td>Partial emulation</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Firmadyne (NDSS ’16), HALucinator (Usenix ’20), PartEmu (Usenix ’20)</td>
</tr>
<tr>
<td>P²IM (our work)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Workflow

Processor Emulator \rightarrow P^2IM \rightarrow Firmware Binary

Fuzzer \rightarrow Processor-peripheral Interface Model

Crashing Test Cases
Interface modeling

**How to model an interface?**

**Peripherals** determine register value and interrupt-firing timing, but peripherals are considered as black box.

**Registers** are categorized by their functionalities and handled accordingly.

**Interrupts** can be fired at any time. We use a fixed frequency.
Register categories

- Control register (CR)
- Status register (SR)
- Data register (DR)
- Control-status register (C&SR)
Given a firmware, how to identify the interface needs to be modeled?

**Registers** are identified and categorized by monitoring access to the memory-mapped peripheral region.

**Interrupts** are detected by monitoring the interrupt controller.
Workflow

Existing fuzzers can be used without modification.
Evaluation

- 70 sample firmware for essential peripheral operations
  - E.g., data transmission through USART peripheral
Results

• The majority of firmware boot and perform essential peripheral operations normally
  • 79% test cases pass

• The accuracy of register categorization is between 76% and 92%
Fuzzing

• Fuzz-test 10 real-world firmware
  • Drone, Robot, Gateway, PLC, etc.
Fuzzing performance

- The accuracy of register categorization is between 69.6% and 100%
- Speed and basic block coverage:

<table>
<thead>
<tr>
<th>Firmware</th>
<th>Speed (# tests/s)</th>
<th>Basic block coverage</th>
<th>Coverage improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drone</td>
<td>17.2</td>
<td>58%</td>
<td>7x</td>
</tr>
<tr>
<td>CNC</td>
<td>18.0</td>
<td>70%</td>
<td>26x</td>
</tr>
<tr>
<td>Steering C.</td>
<td>32.3</td>
<td>20%</td>
<td>30x</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fuzzing result

- Detect 7 unique bugs, all of which are
  - Previously unknown
  - Remotely exploitable
  - Reproducible on real device

<table>
<thead>
<tr>
<th>Firmware</th>
<th>Unique bugs</th>
<th>Bug nature</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLC</td>
<td>3</td>
<td>Incorrect Type Cast</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Integer overflow</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Incorrect Conversion between Numeric Types</td>
</tr>
<tr>
<td>Gateway</td>
<td>1</td>
<td>Buffer overflow</td>
</tr>
<tr>
<td>Heat Press</td>
<td>1</td>
<td>Buffer overflow</td>
</tr>
</tbody>
</table>
Summary

• Propose **P²IM**, the first scalable and hardware-independent firmware testing framework
• Design and implement a novel interface modeling mechanism
• Fuzz-test 10 real-world firmware
• Find 7 previously-unknown vulnerabilities

**Code and Tested Firmware at:**

https://github.com/RiS3-Lab/p2im
Thank You

Questions?

feng.bo@northeastern.edu