Wei Zhou, Le Guan, Peng Liu and Yuqing Zhang

Automatic Firmware Emulation through Invalidation-guided Knowledge Inference --μEmu
Bugs in MCU-based devices are being exploited to jeopardize our cyber-physical systems.

Multiple Vulnerabilities in Treck TCP/IP Stack

Arbitrary Code Execution

MS-ISAC ADVISORY NUMBER: 2020-171

DATE(S) ISSUED: 12/21/2020

OVERVIEW:

Multiple vulnerabilities have been discovered in code execution. Treck TCP/IP Stack is widely used. Successful exploitation of arbitrary code in the context of the Treck Stack could install programs; view, change, or destroy data; impersonate other users; and have less impact than if it was configured to have fewer user account restrictions.

THREAT INTELLIGENCE:

There are currently no reports of the vulnerabilities being actively exploited.

The hack attack led to failures in plant equipment and forced the fast shut down of a furnace.

Hacking risk leads to recall of 500,000 pacemakers due to patient death fears

FDA overseeing crucial firmware update in US to patch security holes and prevent hijacking of pacemakers implanted in half a million people.
Dynamic analysis is popular in bug finding

Dynamic Taint Analysis

Dynamic Symbolic Execution

Fuzz Testing
Existing dynamic approaches fail to test MCU firmware

- Real hardware does not provide rich run-time information.
  - WYCNWYC [NDSS’18]

- Emulating firmware is hard
  - QEMU only supports 10+ MCU devices
  - Problem: peripheral diversity
Existing solution: manual peripheral modeling

- Idea: develop peripherals models based on chip manuals like QEMU.
  - Can be useful dealing with a particular device model

- Disadvantages
  - Unscalable
  - Significant manual efforts
Existing solution: hardware-in-the-loop

- Idea: forward the interactions with peripherals to the real hardware.
  - Avatar [NDSS’14]

- Disadvantages
  - Rely on real hardware
  - Slow
  - Unscalable
Existing solution: hardware abstraction

- Idea: replace the hardware abstraction layer (HAL) with software stubs that provide the same functions
  - HALucinator[SEC’20]

- Disadvantages
  - Unable to test peripheral-dependent code
  - HAL does not always available for all firmware
Existing solution: automatic peripheral modeling

- Idea: observe peripheral-firmware interactions and infer peripheral models
  - P2IM [SEC’20]
    - Categorize the peripheral registers based on access pattern
    - Respond to peripheral access requests based on register category and heuristics

- Disadvantages
  - Register mis-categorization
  - Blindly guess the response for some registers is low accuracy
Existing solution: automatic peripheral modeling

- Idea: observe peripheral-firmware interactions and infer peripheral models
  - P2IM [SEC'20]
  - Categorize the peripheral registers based on access pattern
  - Respond to peripheral access requests based on register category and heuristics

Disadvantages:
- Register mis-categorization
- Blindly guess the response for some registers is low accuracy

We propose a new automatic peripheral modeling approach with less heuristics and higher accuracy
Observation and high-level idea

- Observation 1: If a response is incorrectly fed to the firmware, the error will eventually be reflected as an invalid execution state (e.g., a dead loop).
- Observation 2: A response can be reused when the same peripheral is accessed again under the same context.
- Idea
  - Represent responses to peripheral accesses as symbolic values and then use symbolic execution to explore the firmware.
  - Detect potential invalid paths.
  - Build a knowledge base (KB) that guides the execution to avoid invalid paths.
    - The knowledge base is a tiered cache system.
Invalid States

- **Infinite Loop** - If the firmware execution encounters an unrecoverable error, it will halt itself by running a simple infinite loop.

- **Long Loop** - It is common that firmware waits for a certain value in a peripheral register, which is implemented via a long loop that pulls the value. A wrong response leads the execution to finish the long loop with an error code returned.

- **Invalid Memory Access** - Firmware uses response from peripheral to address memory. A wrong response lead to unmapped memory access.

- **User-defined Invalid Program Points** - μEmu is interactive. It allows analysts to specify program points to be avoided.
Context matching of the tiered cache system

**Storage Model**

- **Entry**: T0_addr_{written value}
- **Value**: The most recent value written to it.

**PC-based Matching**

- **Entry**: T1_addr_pc_value
- **Value**: 1. Peripheral register address 2. Reading PC

**Context-based Matching**

- **Entry**: T2_addr_pc_Hash_value
- **Value**: 1. Peripheral register address 2. Reading PC 3. contextHash: call stack and function arguments.

**Replay-based Matching**

- **Entry**: T3_addr_vec{v1,v2,..}
- **Value**: 1. Peripheral register address 2. Vector of value sorted by reading order
In the example, two branches both correspond to reading the peripheral register mapped at 0x40064006 at PC 0x1a9a.
Implementation - ARM support for S2E

- S2E
  - Full system concolic execution engine based on QEMU and KLEE
  - Useful plugins and APIs
  - No ARM support
- ARM CPU Emulation
  - ARM TCG front-end
  - Low-level plugin hook
- KVM interface emulation
  - Register Initialization
  - Memory Registration
  - System Peripheral State Synchronization
Implementations - μEmu plugins

- Invalid States Detection
  - Invalid states notification
  - Path Switch
- Knowledge Extraction
  - Build and use KB
- Interrupt Control
  - Trigger external IRQs
Implementation - Fuzzer Integration

- AFL as a drop-in fuzzer
- Fuzzer Helper
  - Coverage feedback
  - Fork sever
  - Crash/hang detection
  - Data input channel identification
Unit tests

• Reuse the same 66 firmware samples in the P2IM experiment.
  – These samples cover most popular MCU peripherals, MCU chips, MCU OS/system.

• For each unit test, we first run the knowledge extraction phase. During dynamic analysis, we monitor whether its output and execution flow as expected.
### Unit test results

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Functional Operations</th>
<th>F103/Arduino</th>
<th>F103/RIOT</th>
<th>F103/NUTTX</th>
<th>K64F/RIOT</th>
<th>SAM3/Arduino</th>
<th>SAM3/RIOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Read an analog-to-digital conversion</td>
<td>Pass</td>
<td>N/A</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>DAC</td>
<td>Write a value for digital-to-analog conversion</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>GPIO</td>
<td>Execute callback after pin interrupt</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>Read status of a pin</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>Set/ Clear a pin</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>PWM</td>
<td>Configure PWM as an autonomous</td>
<td>Pass</td>
<td>N/A</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>I2C</td>
<td>Read a byte from a slave</td>
<td>Pass</td>
<td>N/A</td>
<td>Fail</td>
<td>Fail</td>
<td>Pass</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Write a byte to a slave</td>
<td>Pass</td>
<td>N/A</td>
<td>N/A</td>
<td>Fail</td>
<td>Pass</td>
<td>N/A</td>
</tr>
<tr>
<td>UART</td>
<td>Receive a byte</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>Transmit a byte</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>SPI</td>
<td>Receive a byte</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>Transmit a byte</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>Timer</td>
<td>Execute callback after interrupt</td>
<td>N/A</td>
<td>Pass</td>
<td>N/A</td>
<td>Pass</td>
<td>N/A</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>Read counter value</td>
<td>N/A</td>
<td>Pass</td>
<td>N/A</td>
<td>Pass</td>
<td>N/A</td>
<td>Pass</td>
</tr>
</tbody>
</table>

Compared with the passing rate of **79% achieved by state of the art tool P2IM**, µEmu achieves **95% without any manual assistance**. With little manual assistance, all unit tests can be passed.
Fuzzing

- Fuzzed 21 real-world firmware
  - 10 from P2IM [https://github.com/RiS3-Lab/p2im-real_firmware](https://github.com/RiS3-Lab/p2im-real_firmware)
  - 3 from HALucinator [https://github.com/ucsb-seclab/hal-fuzz](https://github.com/ucsb-seclab/hal-fuzz)
  - 2 from Pretender [https://github.com/ucsb-seclab/pretender](https://github.com/ucsb-seclab/pretender)
  - 6 from more complicate real devices collected by us.
Basic block coverage improvement

- The code coverage increases 10x to 140x compared to that in the normal QEMU without peripheral emulation.

<table>
<thead>
<tr>
<th>Firmware</th>
<th>BB Cov. Qemu</th>
<th>BB Cov. μEmu</th>
<th>Improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNC</td>
<td>2.68%</td>
<td>67.96%</td>
<td>24.96x</td>
</tr>
<tr>
<td>Console</td>
<td>2.19%</td>
<td>35.90%</td>
<td>16.42x</td>
</tr>
<tr>
<td>Drone</td>
<td>8.40%</td>
<td>89.74%</td>
<td>10.69x</td>
</tr>
<tr>
<td>Gateway</td>
<td>1.70%</td>
<td>52.71%</td>
<td>30.94x</td>
</tr>
<tr>
<td>Heat Press</td>
<td>1.11%</td>
<td>30.21%</td>
<td>27.22x</td>
</tr>
<tr>
<td>Reflow Oven</td>
<td>3.57%</td>
<td>40.53%</td>
<td>11.3x</td>
</tr>
<tr>
<td>Robot</td>
<td>2.47%</td>
<td>43.25%</td>
<td>17.51x</td>
</tr>
<tr>
<td>Soldering Iron</td>
<td>4.21%</td>
<td>62.01%</td>
<td>14.73x</td>
</tr>
<tr>
<td>Steering Control</td>
<td>0.68%</td>
<td>32.59%</td>
<td>48.09x</td>
</tr>
<tr>
<td>6LoWPAN Sender</td>
<td>0.88%</td>
<td>48.30%</td>
<td>55.17x</td>
</tr>
<tr>
<td>6LoWPAN Receiver</td>
<td>0.88%</td>
<td>47.36%</td>
<td>54.08x</td>
</tr>
<tr>
<td>RF Door Lock</td>
<td>0.25%</td>
<td>24.37%</td>
<td>97.57x</td>
</tr>
<tr>
<td>XML Parser</td>
<td>0.72%</td>
<td>26.31%</td>
<td>36.45x</td>
</tr>
<tr>
<td>GPS Tracker</td>
<td>0.49%</td>
<td>23.90%</td>
<td>48.81x</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fuzzing results

- Known Bugs Reproduce
- Two previously unknown bugs
  - Steering C: Double Free
  - µtasker_USB: Buffer overflow
- False Crashes/Hangs
  - Due lack of DMA support

<table>
<thead>
<tr>
<th>Firmware</th>
<th>False Crashes/Hangs</th>
<th>True Crashes/Hang</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNC</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Drone</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Gateway</td>
<td>0/0</td>
<td>6/0</td>
</tr>
<tr>
<td>Heat_Press</td>
<td>0/0</td>
<td>2/0</td>
</tr>
<tr>
<td>PLC</td>
<td>0/0</td>
<td>139/0</td>
</tr>
<tr>
<td>Reflow_Oven</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Soldering_Iron</td>
<td>32/4</td>
<td>0/0</td>
</tr>
<tr>
<td>Steering_Control</td>
<td>0/0</td>
<td>12/0</td>
</tr>
<tr>
<td>6LoWPAN_Sender</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>6LoWPAN_Receiver</td>
<td>0/0</td>
<td>2/0</td>
</tr>
<tr>
<td>RF_Door_Lock</td>
<td>0/0</td>
<td>98/0</td>
</tr>
<tr>
<td>Thermostat</td>
<td>0/0</td>
<td>76/0</td>
</tr>
<tr>
<td>GPS_Tracker</td>
<td>0/29</td>
<td>29/0</td>
</tr>
<tr>
<td>µtasker_USB</td>
<td>0/0</td>
<td>47/0</td>
</tr>
</tbody>
</table>
...
Future work

• Our current prototype is limited to fuzzing. In the future, we will explore other dynamic approaches on top of uEmu, such as taint analysis.

• We will also integrate DICE with uEMU to handle DMA.
We present μEmu, a security-oriented dynamic analysis platform with the capability to emulate firmware for previously-unseen hardware.

We developed μEmu on top of S2E using plugins and reimplementation of ARM support.

We demonstrate the emulation capability by integrating it with a fuzzer plugin.

We fuzzed 21 real-world firmware and previously-unknown vulnerabilities were found.

Open source at [https://github.com/MCUSec/uEmu](https://github.com/MCUSec/uEmu)
Thank You

Questions?

weizhoulightning@gmail.com