iFIZZ: Deep-State and Efficient Fault-Scenario Generation to Test IoT Firmware

Peiyu Liu  Shouling Ji  Qinming Dai  Kangjie Lu  Lirong Fu  Wenzhi Chen  Peng Cheng  Wenhai Wang  Raheem Beyah

2021
IoT devices are being widely adopted in real-world industries and living environments.
IoT Devices are Vulnerable

IoT devices have become attractive targets for attackers.
Various detection systems appear to discover vulnerabilities in IoT firmware

- **Static Analysis**
  - Taint analysis
  - Symbolic execution
  - Graph matching
  - Inaccurate

- **Dynamic Analysis**
  - Proof of concept
  - Fuzzing
  - Cannot effectively test error-handling code
Motivation
An example of error-handling code in IoT firmware.

```c
FILE *open_memstream( ... ) {
    register __oms_cookie *cookie;  
    ...  
    if (((cookie->buf = malloc(...)) == NULL) {
        goto EXIT_cookie;  
    }  
    EXIT_cookie:  
    free(cookie);  
    return NULL;  
}
```

Error-handling code is intended in erroneous situations where security or reliability issues may potentially occur.
Error-handling code in IoT firmware tends to be error-prone

- Developers may make mistakes when handling complex nested errors.
- More than 28% of IoT patches fix bugs in the error-handling code.
- The patched bug is just the tip of the iceberg.

<table>
<thead>
<tr>
<th>Program</th>
<th>OpenWRT Patches</th>
<th>Error-handling</th>
<th>DD-WRT Patches</th>
<th>Error-handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>busybox</td>
<td>43</td>
<td>8 (18.6%)</td>
<td>148</td>
<td>38 (25.7%)</td>
</tr>
<tr>
<td>dnsmasq</td>
<td>66</td>
<td>27 (40.9%)</td>
<td>81</td>
<td>39 (35.8%)</td>
</tr>
<tr>
<td>dropbear</td>
<td>27</td>
<td>5 (18.5%)</td>
<td>68</td>
<td>23 (33.8%)</td>
</tr>
<tr>
<td>iptables</td>
<td>35</td>
<td>8 (22.9%)</td>
<td>52</td>
<td>16 (30.8%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>171</strong></td>
<td><strong>48 (28.1%)</strong></td>
<td><strong>349</strong></td>
<td><strong>106 (30.4%)</strong></td>
</tr>
</tbody>
</table>

Study result of IoT firmware patches.
Testing Error-handling Code in IoT Firmware is Important

- If error-handling code is incorrect, the intended protection is void.
- Bugs in error-handling code can cause serious security problems, such as DoS and information leakage.
- An attacker could intentionally trigger the errors to exploit the bugs in error-handling code.
- There are still no existing effective approaches for analyzing IoT error-handling code yet.

It is necessary and critical to comprehensively and effectively test the error-handling code of IoT firmware to detect hidden bugs.
Three unique challenges in testing error-handling code in IoT

- C1. Identifying potential runtime errors in IoT firmware
  - Complex hardware dependence and execution environments.
  - The source code of IoT firmware is often not available.

- C2. Effectively covering error-handling code in IoT firmware
  - If an early error stops the execution, the fuzzing will not be able to reach and test deep error paths.
Design of iFIZZ
iFIZZ: a framework for efficiently testing deep error-handling code in IoT firmware

- Automated identification of potential runtime errors
  - Automated binary-based runtime error identification
- Testing of deep error paths
  - State-aware and bounded fault-scenario generation
Two characteristics of runtime errors in IoT firmware

- Error code as the return value
- Input-independent error conditions

Identifying self-defined error codes

Analyzing input-independent error conditions

An example of error-function.
State-aware error producing

- **Observation.** If a runtime error at a specific error stack leads to a crash in a fault-scenario, it is highly possible that the error in the same error stack will trigger the same (redundant) crash in another fault-scenario.

- Reduce redundant fault-scenarios by leveraging the state (defined as runtime context of an error site, i.e., its call stack and its prior error sequences) of error sites.
Bounded faults

- **Observations.** (1) Most crashes are caused by only a small number of errors, generating fault-scenarios with a large number of errors is often unnecessary. (2) Most crashes are caused by neighboring errors.

- The maximum number of errors (ME).

- The maximum distance between the first and the last error (MBE).
Overall Architecture of iFIZZ

Firmware

Error-function Analyzer

Firmware Packer

Emulators & Physical Devices

Fault Scenario

Runtime Monitor

Fault-scenario Generator

Runtime Information

Bug Checker

Bug Reports
Error-function analyzer

- Unpack firmware images to get the IoT programs.
- Analyze the assemble code of the tested program to identify error-functions.
- Leverage automated binary-based runtime error identification method.
Overall Architecture of iFIZZ

Firmware packer

- Repack the tested programs and other necessary tools, e.g., telnet.
- Enable the debug interfaces of the tested firmware.
- Put the fault-scenario generator and the runtime monitor into the tested firmware.
Fault-scenario generator

- Create test cases according to our state-aware and bounded fault-scenario generation method.
- A dynamically linked library.
Overall Architecture of iFIZZ

Runtime monitor

- Obtain the target IoT programs and their corresponding run-commands.
- Produce errors according to fault-scenario by hijacking error-functions.
Bug checker

- Perform an automated analysis of the collected runtime information of detected crashes to generate crash reports.
Evaluation
Experimental Setup

Tested firmware

- 10 IoT firmware produced by 7 vendors are used for evaluation.
- 7 firmware images are tested on emulators, and 3 are tested in physical devices.

<table>
<thead>
<tr>
<th>Model</th>
<th>Vendor</th>
<th>Version</th>
<th>Device</th>
<th>Arch</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR-850L</td>
<td>DLink</td>
<td>1.00B05</td>
<td>Router (E)</td>
<td>Mipseb</td>
</tr>
<tr>
<td>DGS-1210-48</td>
<td>DLink</td>
<td>2.03.001</td>
<td>Switch (E)</td>
<td>Armel</td>
</tr>
<tr>
<td>FW_TV-IP121WN</td>
<td>Trendnet</td>
<td>V2_1.2.1.17</td>
<td>Camera (E)</td>
<td>Mipseb</td>
</tr>
<tr>
<td>K2</td>
<td>Phicomm</td>
<td>v163</td>
<td>Router</td>
<td>Mipsel</td>
</tr>
<tr>
<td>K2</td>
<td>OpenWRT</td>
<td>17.01.0</td>
<td>Router</td>
<td>Mipsel</td>
</tr>
<tr>
<td>TYCAM110</td>
<td>Tuya</td>
<td>V2</td>
<td>Camera</td>
<td>Armel</td>
</tr>
<tr>
<td>WAP200</td>
<td>Cisco</td>
<td>2.0.4.0</td>
<td>AP (E)</td>
<td>Mipseb</td>
</tr>
<tr>
<td>WAP4410N</td>
<td>Cisco</td>
<td>2.0.7.8</td>
<td>AP (E)</td>
<td>Mipseb</td>
</tr>
<tr>
<td>WNAP320</td>
<td>Netgear</td>
<td>v3.0.5.0</td>
<td>AP (E)</td>
<td>Mipseb</td>
</tr>
<tr>
<td>WG103</td>
<td>Netgear</td>
<td>V2.2.5</td>
<td>AP (E)</td>
<td>Mipseb</td>
</tr>
</tbody>
</table>

Basic information of the tested firmware.
Error-Function Extraction

- iFIZZ identifies 140 error-functions out of 3,349 functions.
- 11 false positives in the identified error-functions.

<table>
<thead>
<tr>
<th>Library</th>
<th>Function</th>
<th>Error-function</th>
</tr>
</thead>
<tbody>
<tr>
<td>libclibc-0.9.29.so</td>
<td>937</td>
<td>82</td>
</tr>
<tr>
<td>libclibc-0.9.30.so</td>
<td>1090</td>
<td>11</td>
</tr>
<tr>
<td>libclibc-0.9.30.3.so</td>
<td>1138</td>
<td>44</td>
</tr>
<tr>
<td>libcrypt-0.9.29.so</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>libcrypt-0.9.30.3.so</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>libxtables.so.2.0.0</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3349</strong></td>
<td><strong>140</strong></td>
</tr>
</tbody>
</table>

Result of error-function extraction.
In a certain testing time (24 hours in our test), a set of moderate bounds (ME = 6 and MBE = 12) can improve the efficiency of discovering unique crashes.

Variation of results with respect to different ME and MBE.

Variation of results with respect to different ME and MBE.
Unique crashes

- iFIZZ can find the most unique crashes.

Crashes discovered by different fault-scenario generation approaches.
Error-path coverage

- iFIZZ can cover the most error sites and error stacks.

Code coverage of different fault-scenario generation approaches.
Fault-scenario Generation

Error-path depth

- iFIZZ can trigger deeper error paths than other tools.

Depth of runtime traces covered by different fault-scenario generation approaches.

Depth of error stacks covered by different fault-scenario generation approaches.
Results of Error-handling Testing

Detected bugs

- iFIZZ finds 46 program bugs and 63 library bugs in the tested firmware images.

<table>
<thead>
<tr>
<th>Firmware</th>
<th>Unique Crash</th>
<th>Confirmed Bug</th>
<th>BP</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR-8505</td>
<td>167</td>
<td>9</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>DGS-1210-48</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FW_TV-IPI212WN</td>
<td>21</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>K2</td>
<td>127</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>OpenWRT</td>
<td>45</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>TYCAM110</td>
<td>227</td>
<td>32</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>WAP200</td>
<td>190</td>
<td>11</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>WAP4410N</td>
<td>3079</td>
<td>7</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>WNAP320</td>
<td>2112</td>
<td>23</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>WG103</td>
<td>2270</td>
<td>12</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8244</strong></td>
<td><strong>109</strong></td>
<td><strong>46</strong></td>
<td><strong>63</strong></td>
</tr>
</tbody>
</table>

Detected bugs in IoT firmware.
Comparison with Existing Tools

iFIZZ vs. FirmAFL

- iFIZZ can find significantly more unique crashes than FirmAFL.
- iFIZZ can report unique crashes more efficiently.

<table>
<thead>
<tr>
<th>Program/Lib</th>
<th>iFIZZ Crash</th>
<th>iFIZZ Unique Crash</th>
<th>FirmAFL Crash</th>
<th>FirmAFL Unique Crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzcet</td>
<td>28</td>
<td>12</td>
<td>5.07M</td>
<td>1</td>
</tr>
<tr>
<td>cmp</td>
<td>53</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>wc</td>
<td>56</td>
<td>21</td>
<td>182</td>
<td>19</td>
</tr>
<tr>
<td>uniq</td>
<td>89</td>
<td>23</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>226</strong></td>
<td><strong>69</strong></td>
<td><strong>&gt;5M</strong></td>
<td><strong>20</strong></td>
</tr>
</tbody>
</table>

Results of iFIZZ and FirmAFL.
Case Study

```c
void get_cmdln_options(int argc, char *argv[]) {
    str=(char*)malloc(strlen(...) +14);
    snprintf(str, strlen(...) +14, "%s/...", pwd_entry->pwd_dir);
}
```

Arbitrary null write in bwm-ng.

```c
FILE *open_memstream(...) {
    register __oms_cookie *cookie;
    if (((cookie = malloc(...))) != NULL) {
        if ((cookie->buf = malloc(...)) == NULL) {
            goto EXIT_cookie;
        }
    }
    ...
}
free(cookie->buf);
EXIT_cookie:
    free(cookie);
    return NULL;
}
```

Null pointer dereference in uClibc.
Discussion
False positives and false negatives

- Error-function identification
- Bug detection

Exploitability of error-handling bugs

Manual analysis
Related Work
## Related Work

### Analysis of error-handling code
- Jiang et al., USENIX Security’20
- Bai et al., USENIX ATC’16
- Jana et al., USENIX Security’16
- Kang et al., ASE’16
- Cong et al., ASE’16
- Lawall et al., ISSTA’15
- Zhang et al., ICSE’13
- Saha et al., ICSE’09
- ...

### Vulnerable IoT device discovery and analysis
- Zheng et al., USENIX Security’19
- Muench et al., NDSS’18
- Chen et al., NDSS’18
- Xu et al., CCS’17
- Feng et al., CCS’16
- Chen et al., NDSS’16
- Shoshitaishvili et al., NDSS’15
- Costin et al., USENIX Security’14
- ...

2021/8/28
Conclusion
We presented a novel framework named iFIZZ to effectively test deep error-handling code of IoT firmware.

We propose multiple new techniques in iFIZZ. (1) Automated binary-based error-function identification. (2) State-aware and bounded fault-scenario generation.

We evaluate iFIZZ on 8 widely-used IoT firmware images from leading vendors. It in total finds 59 new bugs. iFIZZ covers 67.3% more error paths than normal execution, and the depth of error-handling code covered by iFIZZ is 15.3 times deeper than that covered by traditional fault injection on average.

We will open-source iFIZZ for facilitating future IoT security research.