Poster: Discovering Authentication Bypass Vulnerabilities in IoT Devices through Guided Concolic Execution

Jr-Wei Huang, Nien-Jen Tsai, Shin-Ming Cheng
National Taiwan University of Science and Technology
{m10815007, m11115009, smcheng}@mail.ntust.edu.tw

Abstract—The severity of attacks on IoT devices underscores the pressing need for efficient and effective vulnerability discovery methods. Specifically, authentication-related vulnerabilities consistently cause significant security breaches in IoT devices, allowing attackers to seize control through these vulnerabilities.

We propose a novel concolic execution framework to uncover vulnerabilities in IoT devices with HTTP services. By integrating the extensive testing coverage of symbolic execution with the rapid execution speed on emulated IoT devices, execution paths can be explored both efficiently and effectively. Furthermore, with the assistance of offline graph-based static analysis, unresolved functions in symbolic execution can be identified and redirected to concrete execution on emulated devices, maintaining the genuine behavior of the program and thereby enhancing analysis accuracy. In our experiments, we assessed the performance and efficacy by comparing to the state-of-the-art system.

I. INTRODUCTION

Malicious users might leverage backdoors of web service to bypass the user verification and to perform privileged operations illegally [5]. The hard-coded credentials or hidden authentication interface for maintenance and upgrade are easily found in vulnerable firmware [7].

Automated firmware vulnerability analysis is generally achieved in static and dynamic manners. By using symbols as the inputs of the target binary and representing all the operations and branches in the form of symbols, symbolic execution could get thesymbolic constraints of the arbitrary path in the binary [9]. However, when infinite loops exist in the program or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to alleviate the problem is to concretely execute the unresolved functions or the program scope is too large, symbolic execution will encounter the path explosion problem [10]. One innovative approach to allevi
such as system(), reboot() functions. The unresolved functions can be identified by using the ELF file structure. Our next step is to generate a control flow graph (CFG) for program slicing, which we modified Ghidra for less time consuming.

For the concolic execution part, we firstly emulate with full system emulator, and then execute the GDB agent to attach the service and wait for angr to connect. When angr is connected, we set breakpoints for synchronizing the symbolic execution environment.

TABLE I. TEST CASES

<table>
<thead>
<tr>
<th>Service</th>
<th>Efficiency Test</th>
<th>Effectiveness Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyHTTPd</td>
<td>TinyHTTPd</td>
<td></td>
</tr>
<tr>
<td>uHTTPd</td>
<td>NETGEAR R7800</td>
<td></td>
</tr>
<tr>
<td>Firmware</td>
<td>OpenWRT</td>
<td></td>
</tr>
<tr>
<td>Firmware Version</td>
<td>1.02.52</td>
<td></td>
</tr>
<tr>
<td>Authentication Bypass</td>
<td>Self-injection</td>
<td>PSV-2019/0076</td>
</tr>
</tbody>
</table>

IV. RESULT

Table I shows our test cases, and all of our experiments are conducted on Ubuntu 18.04 LTS system with Intel Core-i5 CPU and 16GB RAM. We push the boundaries of our system to also support the ARM architecture, in addition to the x86 architecture firmware.

Table II is our system efficacy by comparing to SYMBION. Total Path means the number of paths that the symbolic execution tool totally generates. 1st Vul. Found means the time that the system spends for finding the first vulnerable path. Vul. Path stands for the number of vulnerable paths that the system found. SYMBION suffers from path explosion, which is shown as “(not finish)” in Table II. Our system found 24 vulnerable paths, but SYMBION could only find 11 of them. Most importantly, we only spent 37 seconds noticing the first vulnerable path, which is much less than 230 seconds (3m 50s).

TABLE II. EFFICACY RESULTS

<table>
<thead>
<tr>
<th></th>
<th>Total Path</th>
<th>Time Cost</th>
<th>1st Vul. Found</th>
<th>Vul. Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYMBION</td>
<td>1168 (not finish)</td>
<td>8m 10s</td>
<td>3m 50s</td>
<td>11</td>
</tr>
<tr>
<td>Ours</td>
<td>445</td>
<td>1m 23s</td>
<td>37s</td>
<td>24</td>
</tr>
</tbody>
</table>

V. CONCLUSION

Our framework leverages symbolic execution and concolic execution, which is helpful for finding authentication bypass vulnerabilities. We improve the efficiency of symbolic execution and program runtime slicing, and this is significant for solving the path explosion problem. Our system holds a promising future as a concolic execution tool, and our preliminary results are quite remarkable.

ACKNOWLEDGMENT

Thank you our anonymous reviewers. The research funding was supported by the TACC project from the National Science and Technology Council of Taiwan.

REFERENCES

Problem Overview and Contributions

- Attacks related to bypassing user verifications or conducting privileged operations are common.
- The symbolic execution method has shown great promise for security tools.
- State-of-the-art systems can either focus solely on symbolic execution or limit themselves to find authentication bypasses caused by missing peripherals.
- We are the first to propose a “guided” concolic execution tool with symbolic execution within a full system emulation environment.
- We push the boundaries of our system to support more CPU architectures compared to the previous research.
- Our preliminary results show significant improvements for the path explosion problem.

System Design

- Emulate with full system emulator.
- Model the unresolved function in firmware binaries to substitute the original function during concolic execution.
- Set the keywords as source and sink points for static analysis.
- Attach GDB agent to the service and set breakpoints for synchronizing the symbolic execution environment.

Experimental Results and Conclusion

Future Work

- Make source and sink points more precisely.
- Try to evaluate by running more test cases.
- Add more supported target architectures.