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 the symbolic 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 functions of web services or system calls that cannot be handled by symbolic execution, which is known as concolic execution. The previous research proposed multiple approaches to provide real hardware response to symbolic analysis, but emulating firmware with heterogeneous hardware architectures and peripherals are challenging.

We propose a framework by combining symbolic execution and concrete execution to discover authentication. In our framework, we include static analysis on the target httpd binary for symbolic analysis performing on top of angr [11] symbolic execution engine (SEE). In particular, we modify Ghidra [6] to generate control flow graphs (CFGs) of target binary so that the program is sliced and execution flow can be analyzed. The unnecessary program block or heavy-cost loops are labeled to guide the following symbolic execution so that state complexity is significantly reduced.

Regarding the performance and fidelity metric, we compare our preliminary results with SYMBION [1]. Our system is faster and finds more vulnerable paths, which shows promising potential as a security tool for web-based IoT firmware.

II. Background and Related Work

A. Symbolic Execution

Symbolic execution needs to translate the binary machine code to an intermediate representation (IR), which is really time-consuming. Many frameworks only support user-level emulation, which may suffer from the drawbacks of no truthful feedback from peripherals. LEARCH [4] uses an iterative learning procedure to address the path explosion issue. To solve the time-consuming problem, SymCC [8] has proposed a compilation-based approach. However, this approach requires compiler handling and runtime-supported libraries.

B. Concolic Execution

Path explosion problems in symbolic execution can be alleviated by concretely executing the unresolved functions or system calls. SYMBION extends angr from the perspective of synchronization between concrete execution and symbolic execution (like what S2E currently offered). However, SYMBION is designed for malware analysis and only x86-based IoT devices are supported. AnimateDead [2] applies concolic execution method to web applications, but it does not concern firmware cases. Existing concolic executors are much slower than native executors, and SYMSAN [3] addresses the problem by leveraging data-flow analysis.

III. Our Approach

Figure 1 is the architecture of our system. It can be divided into three parts, including static analysis module, symbolic execution engine, and concrete execution module.

We model the unresolved function in firmware binaries to substitute the original function during concolic execution, so we firstly apply static analysis to find source, sink points, and unresolved functions. The source point is the start of the symbolic execution part, and then we use a static analysis tool to find the symbol of input functions, such as `tcp_recv`, `get_line`, and `uh_tcp_recv`. On the other hand, keywords related to administrator permissions are also important,

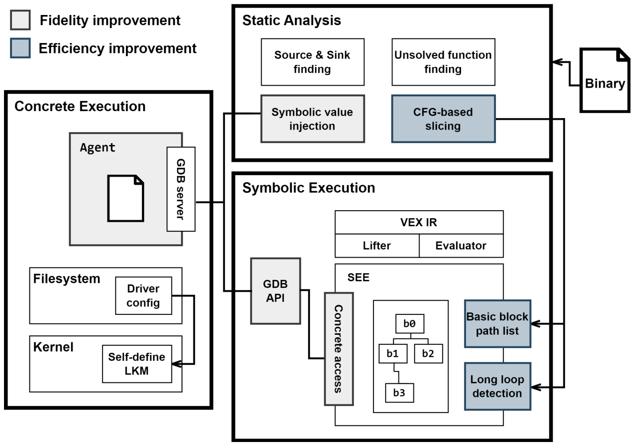


Fig. 1. System Design

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

	Efficiency Test	Effectiveness Test
Service	TinyHTTPd	uHTTPd
Firmware	OpenWRT	NETGEAR R7800
Firmware Version	-	1.0.2.62
Authentication Bypass	Self-injection	PSV-2019-0076

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

	Total Path	Time Cost	1st Vul. Found	Vul. Path
SYMBION	1168 (not finish)	8m 10s	3m 50s	11
Ours	445	1m 27s	37s	24

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

[1] "SYMBION: Interleaving symbolic with concrete execution," in *Proc. IEEE CNS 2020*, Jun. 2020.

[2] B. A. Azad, R. Jahanshahi, C. Tsoukaladelis, M. Egele, and N. Nikiforakis, "AnimateDead: Debloating web applications using concolic execution," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5575–5591.

[3] J. Chen, W. Han, M. Yin, H. Zeng, C. Song, B. Lee, H. Yin, and I. Shin, "SYMSAN: Time and space efficient concolic execution via dynamic data-flow analysis," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2531–2548.

[4] J. He, G. Sivanrupan, P. Tsankov, and M. Vechev, "Learning to explore paths for symbolic execution," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2526–2540.

[5] Y. Jiang, W. Xie, and Y. Tang, "Detecting authentication-bypass flaws in a large scale of IoT embedded web servers," in *Proc. ICCNS 2018*, Nov. 2018, p. 56–63.

[6] NSA, "National security agency. 2019. Ghidra - Software Reverse Engineering Framework," https://github.com/NationalSecurityAgency/ghidra, 2019.

[7] M. O. Ozmen, R. Song, H. Farrukh, and Z. B. Celik, "Evasion attacks and defenses on smart home physical event verification." NDSS, 2023.

[8] S. Poeplau and A. Francillon, "Symbolic execution with SymCC: Don't interpret, compile!" in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 181–198.

[9] Poeplau, Sebastian and Francillon, Aurélien, "SymQEMU: Compilation-based symbolic execution for binaries," in *Proc. NDSS 2021*, Feb. 2021.

[10] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice - Automatic detection of authentication bypass vulnerabilities in binary firmware," in *Proc. NDSS 2015*, Feb. 2015.

[11] F. Wang and Y. Shoshitaishvili, "Angr - The next generation of binary analysis," in *Proc. IEEE SecDev 2017*, 2017, pp. 8–9.

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

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.

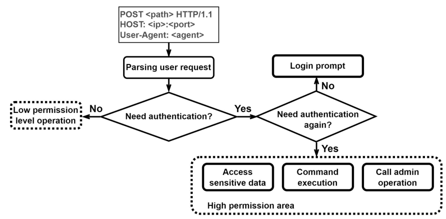
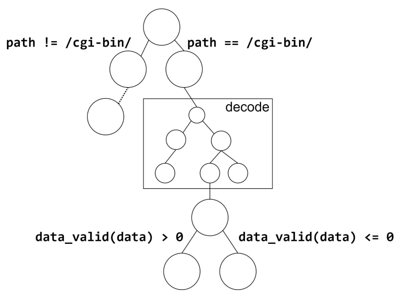


Figure : Normal User Interaction on IoT httpd

- 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.




```
int uh_cgi_auth_check(...) {
    char *auth = header['Authorization'];
    data = base64_decode(auth);
    if (data_valid(data)) {
        sprintf(cmd, "/usr/sbin/hash-data -e %s \
        >/tmp/hash_result", data);
        system(cmd);
    }
}

int handle_request() {
    char request[1024];
    char *path, header, method;
    uh_path_lookup(request);
    ...
    if (path == '/cgi-bin/') {
        if (uh_cgi_auth_check(...)) {
            ...
        }
    }
}
```

Figure : Authentication Bypass Example

System Design

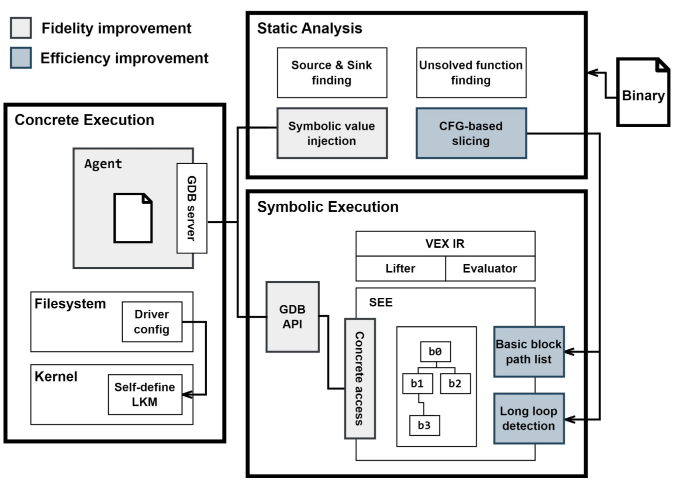


Figure : Our System Design

- 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

Table : Test Cases

	Efficiency Test	Effectiveness Test
Service	TinyHTTPd	uHTTPd
Firmware	OpenWRT	NETGEAR R7800
Firmware Version	-	1.0.2.62
Authentication Bypass	Self-injection	PSV-2019-0076

Table : Results

	Total Path	Time Cost	1st Vul. Found	Vul. Path
SYMBION	1168 (not finish)	8m 10s	3m 50s	11
Ours	445	1m 27s	37s	24

Table : Comparison with Other Systems

	AVATAR (NDSS '14)	Firmalice (NDSS '15)	SYMBION (CNS '20)	uEmu (USENIX '21)	Ours
Goal	Emulator Design	Vul. Discovery	Malware Analysis	Emulator Design	Vul. Discovery
Framework	S2E	angr	angr	S2E	angr
Target Arch.	ARM	ARM, PPC, MIPS	x86	ARM	ARM, x86
Env. Handling	Physical	Modeling	Emulation	Emulation	Emulation
Concolic	O		Partially Supported	O	O
Automation		O	O	O	O
Fidelity	O		O	O	O
Guided Exploration		O			O

Future Work

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