FirmSV: Detecting Stored Vulnerabilities in IoT Firmware using Static Taint Analysis

Qingqi Liu[a,b], Haoran Yang[a,b], Shuangning Yang[c], Guoli Zhao[a,b], Jiaqian Peng[a,b], Jiaming Guo[a,b], Weijuan Zhang[a,b*]
[a] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[b] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[c] School of Internet, Anhui University, China
{liuqingqi, yanghaoran, zhaoguoli, pengjiaqian, guojiaming, zhangweijuan}@iie.ac.cn
realyangshuangning@gmail.com

***Abstract*—The rapid proliferation of Internet of Things (IoT) devices has introduced severe security vulnerability challenges. While extensive research utilizes static taint analysis for firmware vulnerability discovery, it predominantly focuses on volatile memory, neglecting the security risks associated with configuration items in non-volatile memory (NVM). To address this gap, this paper proposes FirmSV, a taint analysis tool targeting stored vulnerabilities. FirmSV leverages static taint analysis to effectively trace inter-procedural data flows originating from configuration items, enabling the detection of these vulnerabilities. Experimental results demonstrate that FirmSV successfully discovered 41 vulnerabilities in mainstream IoT firmware, 29 of which were assigned CVE IDs. FirmSV detected 6.3 times more vulnerabilities than the state-of-the-art (SOTA) tool HermeScan while achieving a 7.4 faster analysis time, further validating its effectiveness in stored vulnerability detection.**

***Index Terms*—IoT Security, Firmware Security, Static Taint Analysis, Stored Vulnerabilities.**

I. Introduction

The number of IoT devices has grown rapidly in recent years, seeing widespread application in both personal and industrial environments. The global count of active IoT devices is projected to surge to approximately 40 billion by the end of 2030 [1]. However, the security landscape of IoT devices is increasingly severe, marked by frequent attack incidents [2]–[4]. Static taint analysis, a mainstream approach for IoT vulnerability discovery, operates by tracking the propagation paths of untrusted or sensitive data. It aims to identify risks where this data reaches sensitive locations without proper sanitization.

However, existing static analysis tools largely lack the capability to detect stored vulnerabilities in IoT firmware. Specifically, during device operation, users can control the inputs for many critical configuration items, such as Wi-Fi names or DNS addresses. These configurations are typically stored in corresponding non-volatile memory (NVM) partitions. Once a configuration item is set to a malicious payload, vulnerabilities such as stack overflows and command injections can be readily triggered when the program subsequently reads these non-volatile configuration items [5], [6]. Moreover, unlike attacks confined to volatile memory, malicious configurations written to NVM are persistent. This persistence enables covert and long-term control through a "store-and-trigger" separation strategy.

To efficiently discover stored vulnerabilities, we propose FirmSV: an automated analysis tool targeting stored vulnerabilities, designed to construct configuration item-based exploitation chains efficiently. First, FirmSV employs static taint analysis to identify the mapping relationship between user inputs (e.g., a Common Gateway Interface (CGI) parameter name) and configuration items, thereby identifying all user-controllable configuration items. Next, FirmSV enumerates the call sites of configuration item reading functions and performs a reachable analysis between CGI handler functions and these reading functions. This analysis ensures that the reading of the configuration item can be triggered (e.g., by accessing the corresponding CGI). Finally, by matching the configuration item names, the system connects the complete exploitation chain, tracing the path from the initial user input to its storage as a configuration item, and ultimately to the read operation from that item. This methodology significantly improves the accuracy and coverage of discovering exploitation paths for backend stored vulnerabilities.

To evaluate FirmSV, we conducted experiments on five IoT devices from two mainstream vendors. The results demonstrate that FirmSV discovered 6.3 times more vulnerabilities than the current SOTA static taint analysis tool, HermeScan [7]. Among the discovered vulnerabilities, 29 were assigned CVE IDs [8]. Furthermore, FirmSV achieved a 7.4 faster analysis time than HermeScan [7].

In summary, the main contributions of this paper are as follows:

- We design a novel analysis methodology specifically targeting stored vulnerabilities in IoT devices.
- We implement FirmSV, a taint analysis tool for stored vulnerabilities. FirmSV effectively detects these vulnerabilities, and we demonstrate its capability on multiple real-world IoT firmwares.
- In our evaluation, FirmSV discovered 41 buffer overflow vulnerabilities, representing a 6.3 increase in detected

* Corresponding author: Weijuan Zhang, Email: zhangweijuan@iie.ac.cn

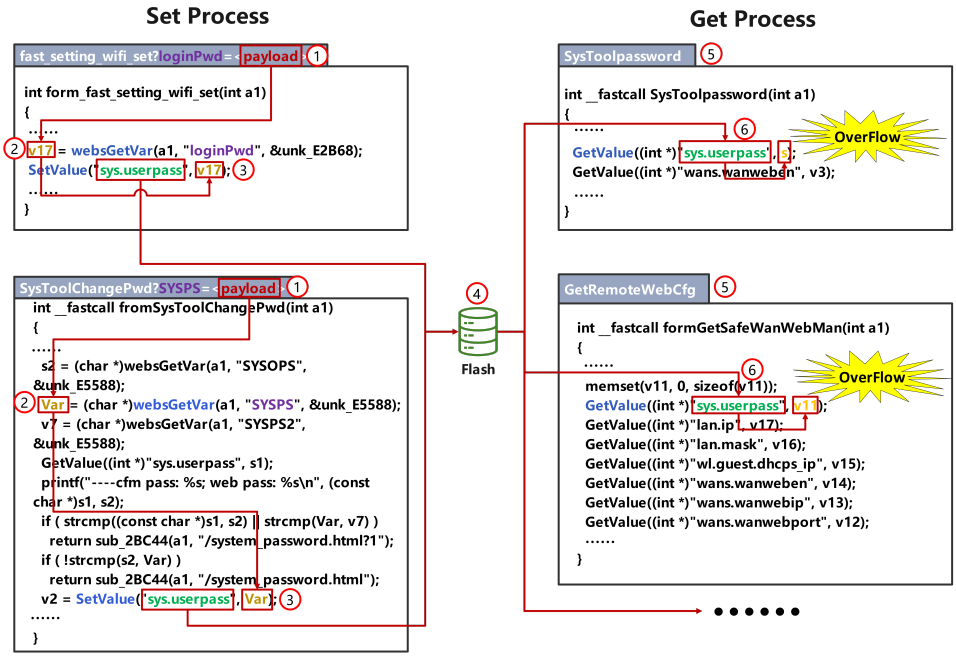

Fig. 1: Motivating Example

vulnerabilities compared to the SOTA tool HermeScan [7], while achieving a 7.4 faster analysis time.

II. Background and Motivation

A. Configuration items in IoT devices

Nowadays, IoT network devices rely on a large number of configuration items at runtime to control functional behavior, security policies, and network communication. These configurations are typically stored as key-value pairs in NVM, such as flash, and are loaded during firmware initialization [9]. To facilitate reading and modifying these parameters, firmware often provides a set of unified management interfaces (e.g., *apmib_get()* / *apmib_set()*), which offer an abstract API for upper-layer applications to access the underlying configuration data. Furthermore, many of these configuration items can be modified externally via CGI endpoints (e.g., */goform/SetDDNSCfg*) [10].

If this parameter-writing process lacks sufficient input validation, an attacker can exploit these interfaces to write malicious data into NVM, thereby triggering severe vulnerabilities such as buffer overflows, command injections, or configuration corruption.

B. Observation

Figure 1 illustrates a typical example of a storage-based vulnerability discovered in an IoT device [11]. For the same configuration item *sys.userpass*, the backend program implements multiple CGI functions capable of writing to the item and multiple functions capable of reading from it. Using the *fast_setting_wifi_set* interface (write) and the *SysToolpassword* interface (read) as an example, the attack proceeds as follows.①The attacker invokes *fast_setting_wifi_set* and supplies a malicious payload via the *loginPwd* parameter.②The backend program retrieves the payload through *websGetVar* and stores it in the variable *v17*.③The backend program calls *SetValue* to persist *v17*.④The payload is written to the Flash partition associated with the configuration item *sys.userpass*.⑤The attacker requests the *SysToolpassword* interface.⑥the backend program reads the stored payload from the Flash partition into the buffer *s*; due to insufficient validation at the read-side, this operation results in a buffer overflow.

C. Motivation

Our investigation reveals that existing static taint analysis tools struggle to analyze stored vulnerabilities effectively. We tested this firmware using the current SOTA tools, SaTC [12],Karonte [13] and HermeScan [7], but none of the tools raised an alert for this vulnerability. The primary reason for this failure is that the storing and reading of the configuration item are decoupled, often occurring in entirely different functions. This separation breaks the data flow path for traditional static taint analysis, which struggles to trace propagation across NVM. Consequently, stored vulnerabilities remain undetected. Therefore, we designed FirmSV, a static analysis framework based on taint analysis, specifically engineered to bridge this gap and efficiently discover this class of stored vulnerabilities.

III. Methodology

A. Overview

To discover stored vulnerabilities efficiently and accurately, FirmSV processes firmware using a four-step pipeline (de-

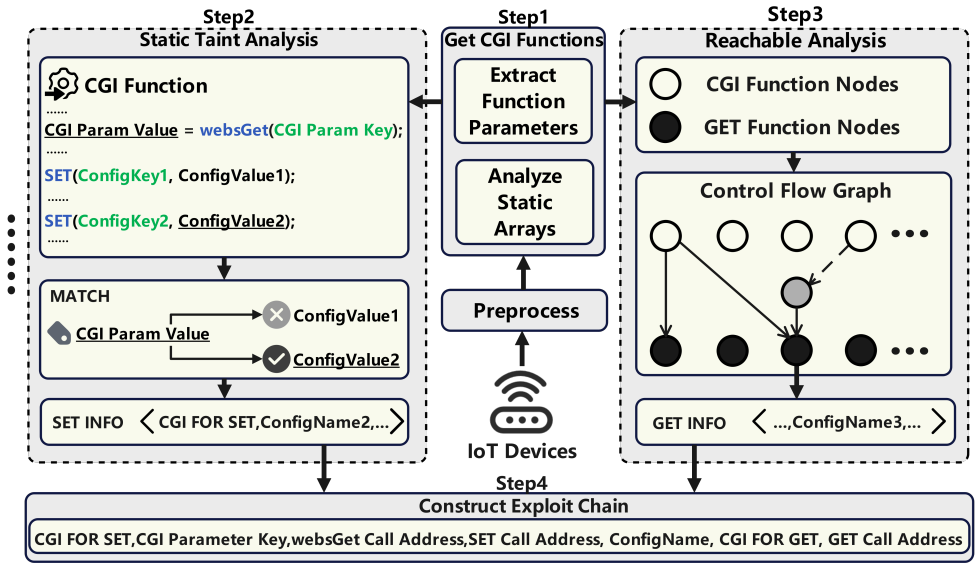

Fig. 2: FirmSV Workflow

scribed in Figure 2). This pipeline is designed to first identify the two decoupled parts of vulnerability—storage and retrieval—and then connect them.

1) CGI Function Identification: The firmware is unpacked to extract the backend program. Subsequently, all CGI functions are located and enumerated by analyzing function parameters or static arrays.
2) Taint Analysis of CGI Parameters and Configuration Items: Static taint analysis is performed on each CGI function. The analysis determines if the return value of websget-type function, which retrieves CGI parameters, can propagate to a configuration-setting function (a "set-type" function, e.g., *apmib_set*). This process identifies the mapping between CGI parameter keys and configuration items, constructing a "set collection" of all externally controllable configuration items.
3) Reachable Analysis for get-type Functions: Using the Control Flow Graph (CFG), a reachable analysis is conducted from all CGI functions to all configuration-reading functions (a "get-type" function, e.g., *apmib_get*). This analysis builds a "get collection" of configuration items that can be triggered and read by an external CGI call.
4) Exploit Chain Reconstruction: Finally, the "set collection" and the "get collection" are matched using their configuration item names. This process connects the complete exploitation chain, from the initial writing of a configuration item to its subsequent read operation.

B. CGI Function Identification

CGI endpoints typically map one-to-one with CGI functions in IoT devices, which are responsible for processing different user requests. To identify all CGI endpoints and their corresponding functions within the firmware, FirmSV implements and combines two identification strategies on the backend program:

- Runtime Binding: This strategy involves statically traversing potential binding API call sites (e.g., a function that registers handlers at runtime) and extracting their arguments to reconstruct the mapping between a CGI endpoint and its handler function.
- Static Table/Array Binding: This strategy focuses on locating and parsing static tables or arrays within the firmware to bulk-extract CGI strings and their corresponding function pointers.

By combining these two strategies, FirmSV identifies the vast majority of CGI endpoints and their backend handler relationships. This provides a reliable starting point for the subsequent taint propagation analysis, which originates from these handler functions.

C. Taint Analysis of CGI Parameters and Configuration Items

In a CGI function, the program typically retrieves CGI parameters and then, within the same function body, calls a set-type interface (e.g., *SetValue*) to write that data into NVM. Based on this observation, we use the CGI function as the starting point for our taint analysis to trace the propagation from the parameter to the configuration item. Furthermore, to identify set-type interface functions, get-type interface functions, and websGet-like functions, we first perform a manual cross-reference analysis of configuration item strings. By tracing their access and invocation paths throughout the program, we systematically summarize the corresponding read and write interface function sets. This process provides a solid foundation for constructing propagation rules in the subsequent taint analysis.

During analysis, when the execution path reaches a websget-type function, FirmSV extracts the CGI parameter name from

the function's argument list. The return value of this function is then marked as tainted data. Crucially, the call address of this websget-type function is recorded in the taint tag.If the analysis path subsequently reaches a set-type function call, FirmSV checks if the value being written is tainted. If it is, the analysis extracts two pieces of information: (1) the *websGet* call address stored in the taint tag, and (2) the configuration item name from the set-type function's argument list. Finally, this entire sequence is recorded as a "set pair", containing the following tuple: (*CGI_for_set*, *websGet_Call_Address*, *CGI_Parameter_Key*, *SET_Call_Address*, *ConfigName*). This taint analysis continues until the end of the current CGI function, achieving a complete identification of the mapping relationship between *CGI_Parameter_Key* and *ConfigName*.

D. Reachable analysis for get-type Functions

To enable the triggering of configuration item reads via CGI access, we must determine whether a given CGI function can reach and execute a target get-type function. To achieve this, we first construct the backend program's Control Flow Graph (CFG) and collect the call addresses of all get-type functions. Next, by analyzing the CFG, we determine if the node containing the CGI function can reach the node containing the get-type function. If reachability is confirmed, we consider the CGI function capable of triggering the corresponding get-type function. At this point, we extract the configuration item name passed as an argument to the get-type function and record a "get pair" tuple: (*CGI_for_get*, *GET_Call_Address*, *ConfigName*).

Based on this analysis, we successfully establish the mapping between CGI functions and the get-type functions that they can trigger.

E. Exploit Chain Construction

By matching the configuration item names recorded in the "set pair" and "get pair" tuples, we can reconstruct the complete exploitation chain. Each successful match is represented as a complete record: (*CGI_for_set*, *CGI_Parameter_Key*, *websGet_Call_Address*, *SET_Call_Address*, *ConfigName*, *CGI_for_get*, *GET_Call_Address*). Specifically, a successful match implies the following attack chain:

1) The attacker sends a request with a malicious payload to the write-enabling POST endpoint: `POST https://<ip>/<CGI_for_set>` (with parameters: `<CGI_Parameter_Key>=<payload>`).
2) The CGI function retrieves the CGI parameter value and calls the `set-type` function to store it: `Set(ConfigName, <payload>)`. This action persists the malicious payload into the target configuration item in NVM.
3) Subsequently, the attacker accesses the trigger-enabling endpoint, activating the read logic: `POST https://<ip>/<CGI_for_get>`. The program calls `Get(ConfigName)` and returns the content of that configuration item.
4) If this retrieved value is used in subsequent sensitive operations without sufficient validation, it can trigger severe consequences, such as remote command execution, buffer overflows.

IV. Evaluation

Based on methodology, we implemented FirmSV and performed vulnerability discovery and comparison experiments to validate its effectiveness.

Implementation:We implemented the prototype system of FirmSV in approximately 2,000 lines of Python code and conducted vulnerability discovery experiments on real-world IoT devices.We used IDA [14] (version 9.0.241217) as our static analysis tool and the angr [15] (version 9.2.174) analysis framework.We then used binwalk [16] to unpack the firmware and extract its file system, obtaining the backend program binaries for analysis.

A. Experiment Setup

To evaluate the effectiveness, accuracy, and efficiency of our vulnerability discovery approach,we thoroughly assessed five devices on a Windows 11 system equipped with Intel Core i7-10750H CPU (6 cores and 12 threads) and 32 GB of RAM.The execution time limit for each tool was set to 4 hours.

We selected five IoT devices from two mainstream vendors, Tenda and TOTOLINK, covering two processor architectures: ARM and MIPS. To ensure consistent firmware versions across the experimental environment, we downloaded the firmware images from the vendors' official websites and updated each device. Details are summarized in Table I.

B. Comparison Experiment

To evaluate FirmSV's real-world vulnerability discovery capabilities, We conducted a comparison with HermeScan [7],Mango [17],SaTC [12] and Karonte [13].Although several recent systems have been proposed, some are not publicly available or rely on subjective components, making fair reproduction and direct comparison challenging; therefore, they were excluded from our evaluation.The evaluation criteria are as follows: each exploitable CGI parameter is regarded as one alert. In our 5 target devices, the return value of get-type functions is written directly to the stack.

The results are summarized in Table II.Because the SaTC [12] tool encountered errors on three MIPS devices, only two ARM devices were counted in its evaluation.Experimental results show that FirmSV consistently surpasses the other tools in effectiveness, accuracy, and efficiency.Compared with the SOTA tool HermeScan [7], FirmSV identified 6.3 times more vulnerabilities while achieving a 7.4 improvement in speed.

FirmSV is able to detect a large number of vulnerabilities because user-supplied data is often initially stored on disk. Other tools performing static taint analysis frequently overlook such persistent data flows, whereas FirmSV can effectively track and analyze them. Its superior execution speed stems from focusing specifically on all CGI functions in backend programs, while other tools examine a much broader code scope.

TABLE I: Device Set. `Vendor`: device manufacturer. `Product`: product model. `Type`: device type. `Firmware`: firmware version. `Binary`: backend program name. `Arch`: CPU architecture.

Device Name	Vendor	Product	Type	Firmware	Binary	Arch
TendaAC18	Tenda	AC18	Router	V15.03.05.19(6318)	httpd	ARM
TendaAC15	Tenda	AC15	Router	V15.03.05.18	httpd	ARM
TendaAC7	Tenda	AC7	Router	V15.03.06.44	httpd	MIPS
TendaO3V2	Tenda	O3V2	Bridge	V1.0.0.10(2478)	httpd	MIPS
TOTOlinkA3300R	TOTOlink	A3300R	Router	V17.0.0cu.557_B20221024	shttpd	MIPS

TABLE II: Comparison of vulnerability detection results. `Alerts`: the number of alerts generated by each tool. `TP`: the number of true positives. `TPR`: the true positive rate. `Time`: the execution time of each tool.

Tool	Metric	Device Name					Total	Average
		TendaAC18	TendaAC15	TendaAC7	TendaO3V2	TOTOlinkA3300R		
FirmSV	**Alerts**	**80**	**80**	**77**	**51**	**40**	**328**	**65.6**
	TP	**75**	**74**	**71**	**47**	**37**	**304**	**60.8**
	TPR	**0.938**	**0.925**	**0.922**	**0.922**	**0.925**	**/**	**0.926**
	Time	**3min**	**3min**	**6min**	**2min**	**1min**	**15min**	**3min**
HermeScan	**Alerts**	8	5	27	23	12	75	15
	TP	3	2	21	13	9	48	9.6
	TPR	0.375	0.400	0.778	0.565	0.750	/	0.522
	Time	33min	19min	4h	17min	8min	5h17min	1h3min
Mango	**Alerts**	12	123	100	21	/	256	64
	TP	4	21	19	5	/	49	12.25
	TPR	0.333	0.171	0.190	0.238	/	/	0.191
	Time	50min	4h	4h	1h50min	/	10h40min	2h40min
Karonte	**Alerts**	0	0	0	0	0	0	0
	TP	0	0	0	0	0	0	0
	TPR	/	/	/	/	/	/	/
	Time	1h14min	1h48min	27min	1h23min	8min	5h	1h
SaTC	**Alerts**	23	0	/	/	/	23	11.5
	TP	5	0	/	/	/	5	2.25
	TPR	0.217	/	/	/	/	/	0.217
	Time	4h	4h	/	/	/	8h	4h

TABLE III: Set-Get Pair Analysis Results. `Num.CGI`: the number of CGI functions extracted by FirmSV. `Num.Set`: the number of configuration items with set operations. `Num.Get`: the number of configuration items with get operations. `Num.Matched`: the number of configuration items matching set/get operations. `Num.Exploit Chains`: the number of exploit chains. `Time`: the execution time. `Avg.Time`: average execution time per CGI function.

Device Name	Num.CGI	Num.Set	Num.Get	Num.Matched	Num.Exploit Chains	Time (s)	Avg.Time (s)
TendaAC18	188	191	226	113	651	222.18	1.18
TendaAC15	188	190	229	111	628	222.94	1.19
TendaAC7	180	166	221	92	535	295.03	1.64
TendaO3V2	84	58	92	38	174	102.63	1.22
TOTOlinkA3300R	115	98	77	34	79	64.28	0.56

C. Vulnerability discovery

We conducted a systematic analysis of our target firmware dataset and submitted the discovered vulnerabilities. To date, 29 previously undisclosed stored vulnerabilities have been confirmed and assigned CVE IDs.All confirmed CVE cases are publicly disclosed on our GitHub repository https://github.com/noahze01/IoT-vulnerable.

These vulnerabilities received an average CVSS 4.0 [18] score of 7.4, with the highest score of 8.7, indicating a HIGH severity risk. It reflects that an attacker can exploit these controllable configuration items to achieve persistent privilege escalation, configuration tampering, or sensitive information leakage. Consequently, these vulnerabilities pose long-term and stealthy security risks to the affected devices and their networks.

D. Configuration Item Exploit Chains

We evaluated FirmSV on the selected IoT devices and measured several representative metrics, with the results detailed in Table III. The experiment demonstrates that FirmSV can efficiently and accurately analyze the CGI functions within the web applications. In terms of performance, FirmSV demonstrates high efficiency. The average time to analyze a single CGI function is 1.20 seconds, and the average total time to analyze one firmware image is 181.41 seconds. This makes FirmSV suitable for large-scale batch scanning and vulnerability discovery on extensive firmware collections.

V. Discussion

Our prototype system may still be subject to a certain degree of false negatives and false positives:

False Negatives: Our tool currently only performs taint analysis on CGI functions and their call chains. If a set-type function is not located within this scope (i.e., not in the call chain of a CGI function), it will not be identified. This could lead to potential false negatives. However, such set-type functions are typically not controllable via CGI parameters, meaning their exploitability is low, and the impact on our overall detection results is limited.

False Positives: FirmSV currently performs vulnerability chain reconstruction based on reachability analysis, but it does not fully model path conditions. As a result, some reconstructed chains may be infeasible in practice, potentially leading to false positives. In future work, we plan to incorporate lightweight symbolic execution and sanitization-aware analysis to better reason about path feasibility and further improve precision.

VI. Related Work

Static taint analysis is one of the key methods for IoT firmware vulnerability detection and can be divided into two categories: traditional taint analysis and semantic or learning-based approaches.Below, we discuss representative works in each category.

A. Taint Analysis for Firmware Vulnerabilities

DTaint [19] was the first to apply dataflow analysis to trace the propagation of insecure inputs within firmware. Subsequently, EmTaint [20] introduced demand-driven, context-sensitive alias analysis and indirect call resolution mechanisms, effectively improving the completeness and accuracy of taint propagation. To address the challenge of source identification in stripped binaries, Karonte [13] modeled multi-binary interactions and tracked cross-module data flows, while SaTC [12] leveraged shared keywords between front-end and back-end components to locate input entry points, thereby detecting insecure interaction vulnerabilities. Furthermore, HermeScan [7] built an efficient taint analysis framework based on reachable analysis, enhancing control-flow recovery and employing on-demand path merging strategies to achieve high precision with significantly reduced time costs. Mango [17] proposed a scalable static data-flow analysis framework that combines value analysis with data-dependency tracking, enabling more precise and efficient reasoning about input propagation to vulnerability sinks.

B. Semantic and Learning-based Enhancements

In recent years, researchers have begun integrating deep learning and large language models (LLMs) with static analysis. Works such as LLIFT [21] and Lara [22] leverage the semantic understanding capabilities of LLMs to assist in path reasoning and taint source identification, achieving higher accuracy in detecting complex vulnerabilities. Collectively, these studies demonstrate the continuous evolution of firmware static analysis—from traditional dataflow tracking toward semantic enhancement and cross-binary modeling. However, systematic approaches targeting storage-type vulnerabilities in IoT firmware remain largely unexplored.

VII. Conclusion

In this paper, We design a novel analysis methodology specifically targeting stored vulnerabilities in IoT devices and proposed FirmSV, a detection method that specifically targets configuration items. We evaluated FirmSV on five real-world IoT products, demonstrating its effectiveness. Our prototype generated 328 alerts, the vast majority of which were previously unknown vulnerabilities; 29 of these have been assigned CVE IDs. Compared to the SOTA tool Hermescan [7], FirmSV discovered 6.3 times more vulnerabilities and was 7.4 times faster.

VIII. Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 62202465).

References

[1] S. Sinha. (2025) State of iot 2025: Number of connected iot devices growing 14% to 21.1 billion globally. IoT Analytics. Accessed: Oct. 31, 2025. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[2] CE Pro. (2025) Securing the smart home network: The cybersecurity risks of iot. CE Pro. Accessed: Oct. 31, 2025. [Online]. Available: https://www.cepro.com/news/securing-the-smart-home-network-the-cybersecurity-risks-of-iot/121547/

[3] Number Analytics. (2025) The ultimate guide to iot vulnerabilities and network security. Number Analytics Blog. Accessed: Oct. 31, 2025. [Online]. Available: https://www.numberanalytics.com/blog/ultimate-guide-iot-vulnerabilities-network-security

[4] Threatpost. (2025) Half of iot devices vulnerable to severe attacks. Threatpost. Accessed: Oct. 31, 2025. [Online]. Available: https://threatpost.com/half-iot-devices-vulnerable-severe-attacks/153609/

[5] CVE. (2023) CVE-2023-51958 Detail. The MITRE Corporation. Accessed: Oct. 31, 2025. [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2023-51958

[6] CVE. (2023) CVE-2023-51972 Detail. The MITRE Corporation. Accessed: Oct. 31, 2025. [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2023-51972

[7] Z. Gao, C. Zhang, H. Liu, W. Sun, Z. Tang, L. Jiang, J. Chen, and Y. Xie, "Faster and better: Detecting vulnerabilities in linux-based iot firmware with optimized reaching definition analysis," *Proceedings 2024 Network and Distributed System Security Symposium*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267617879

[8] CVE. (2025) Cve® program. The MITRE Corporation. Accessed: Oct. 31, 2025. [Online]. Available: https://www.cve.org/

[9] S. Ul Haq, Y. Singh, A. Sharma, R. Gupta, and D. Gupta, "A survey on iot embedded device firmware security: architecture, extraction techniques, and vulnerability analysis frameworks," *Discover Internet of Things*, vol. 3, no. 1, p. 17, Oct 2023. [Online]. Available: https://doi.org/10.1007/s43926-023-00045-2

[10] W3C Web Security Interest Group. (2025) Web security faq. Accessed: Oct. 31, 2025. [Online]. Available: https://www.w3.org/Security/Faq/wwwsf4.html

[11] Tenda. (2025) Tenda official website. Accessed: Oct. 31, 2025. [Online]. Available: https://www.tenda.com.cn/

[12] L. Chen, Y. Wang, Q. Cai, Y. Zhan, H. Hu, J. Linghu, Q. Hou, C. Zhang, H. Duan, and Z. Xue, "Sharing more and checking less: Leveraging common input keywords to detect bugs in embedded systems," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 303–319. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-libo

[13] N. Redini, A. Machiry, R. Wang, C. Spensky, A. Continella, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Karonte: Detecting insecure multi-binary interactions in embedded firmware," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1544–1561.

[14] Hex-Rays. (2025) Ida pro disassembler and debugger. Accessed: Oct. 31, 2025. [Online]. Available: https://hex-rays.com/ida-pro/

[15] angr developers. (2025) angr/angr: The next generation binary analysis platform. GitHub. Accessed: Oct. 31, 2025. [Online]. Available: https://github.com/angr/angr
[16] ReFirm Labs. (2025) Refirmlabs/binwalk: Firmware analysis tool. GitHub. Accessed: Oct. 31, 2025. [Online]. Available: https://github.com/ReFirmLabs/binwalk
[17] W. Gibbs, A. S. Raj, J. M. Vadayath, H. J. Tay, J. Miller, A. Ajayan, Z. L. Basque, A. Dutcher, F. Dong, X. Maso, G. Vigna, C. Kruegel, A. Doupé, Y. Shoshitaishvili, and R. Wang, "Operation mango: Scalable discovery of Taint-Style vulnerabilities in binary firmware services," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 7123–7139. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/gibbs
[18] NVD. (2025) Cvss v4.0 calculator. National Institute of Standards and Technology. Accessed: Oct. 31, 2025. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v4-calculator
[19] K. Cheng, Q. Li, L. Wang, Q. Chen, Y. Zheng, L. Sun, and Z. Liang, "Dtaint: Detecting the taint-style vulnerability in embedded device firmware," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 430–441.
[20] K. Cheng, Y. Zheng, T. Liu, L. Guan, P. Liu, H. Li, H. Zhu, K. Ye, and L. Sun, "Detecting vulnerabilities in linux-based embedded firmware with sse-based on-demand alias analysis," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 360–372. [Online]. Available: https://doi.org/10.1145/3597926.3598062
[21] H. Li, Y. Hao, Y. Zhai, and Z. Qian, "Enhancing static analysis for practical bug detection: An llm-integrated approach," *Proc. ACM Program. Lang.*, vol. 8, no. OOPSLA1, Apr. 2024. [Online]. Available: https://doi.org/10.1145/3649828
[22] J. Zhao, Y. Li, Y. Zou, Z. Liang, Y. Xiao, Y. Li, B. Peng, N. Zhong, X. Wang, W. Wang, and W. Huo, "Leveraging semantic relations in code and data to enhance taint analysis of embedded systems," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 7067–7084. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/zhao