Identifying Vulnerable Third-party Components in IoT Firmware Using Deep Learning

Chia-Mei Chen, Sheng-Hao Lin and Zheng-Xun Cai
Department of Information Management
National Sun Yat-Sen University
Kaohsiung, Taiwan

Gu-Hsin Lai
Department of Technology Crime Investigation
Taiwan Police College
Taipei, Taiwan

Ya-Hui Ou
General Competency Center
National Penghu University
Penghu, Taiwan

***Abstract*—To reduce the development time and the cost of Internet of Thing (IoT) products, vendors leverage Third-Party Components (TPCs) to manufacture various types of IoT products. However, such third-party software might not be validated with proper software testing or might contain vulnerabilities. Furthermore, existing research rarely proposed a cross-architecture solution for detecting both top IoT vulnerabilities. Therefore, this study proposes a cross-architecture IoT vulnerability detection method that identifies vulnerable third-party components used in IoT firmware. This study leverages a Siamese Neural Network (SNN) architecture and designs a similarity algorithm to identify vulnerable functions on different processor architectures. The evaluation results demonstrate that the proposed method can identify vulnerable TPCs effectively.**

Keywords- IoT attacks; vulnerability detection; deep learning.

I. INTRODUCTION

With the prevalence of the Internet of Things (IoT) and its flourishing development, about 83% of organizations rely on IoT technologies to boost their productivity [1]. IoT devices are becoming increasingly ubiquitous. In 2025, the IoT market is expected to reach 27 billion active connections [2], and the IoT-related services will grow to 58 billion dollars [3]. Such network-connected devices in enterprise networks are hard for security teams to properly identify and monitor them, which may become security blind spots for the organizations.

To reduce development costs and to shorten the time to market for a new device, the functionalities of the IoT device typically are provided through previously developed software. Furthermore, the current development leverages third-party software heavily to improve development efficiency. The use of TPCs in IoT products tripled during the recent years [4]. Including TPCs in an IoT device implies that the device inherits the vulnerabilities existing within the TPCs. Such external components mostly are not secure, and their vulnerabilities influence IoT security [5]. TPCs play an imperative role in IoT firmware development.

IoT devices commonly adopt embedded Linux systems. Examining such embedded systems requires a comprehensive understanding of the operating systems and the experience of reverse engineering. In addition, manufacturers adopt various processor architectures.

Based on the literature review, past research paid little attention to the vulnerabilities caused by TPCs, so an automatic and effective solution to identify vulnerable TPCs used in IoT firmware is desired. In addition, existing work mostly focused on a single architecture and rarely provided a solution for multiple architectures. Even though the recent work proposed solutions for cross-architecture, their detection performance needed to be improved.

To fill in the aforementioned research gaps, this study designs an automatic firmware analysis and vulnerability detection approach for multi-architecture IoT devices, which integrates several Open-Source Software (OSS) solutions to automate the firmware analysis process and facilitates the state-of-the-art machine learning technologies to extract features of function codes and to identify vulnerable components.

The remainder of this paper is constructed as follows. Section 2 reviews the related research. Section 3 presents the proposed detection method, followed by the performance evaluation in Section 4. The last section draws the conclusion remark and the future directions of this study.

II. LITERATURE REVIEW

To enhance detection performance, some research adopted ensemble approach which aggregates the multiple classifiers. Essa and Bhaya [6] applied two feature selection approaches: mean and hard-voting schemes, with ensemble soft voting classifier. According to the evaluation, their method achieved better results than other ensemble and individual classifiers.

Zhao et al. [7] conducted a large-scale analysis of TPC usage in IoT firmware. During the analysis process, their approach requires several stages of manual work to facilitate the detection. It applied the tool Binwalk [8] for file system decomposition, extracted the features from Control Flow Graphs (CFGs), and utilized the version check to determine if a target TPC contains vulnerabilities. Even though this past work involved intensive human analysis effort, it provided statistical analysis results and highlighted the IoT cyber risk.

Ngo et al. [9] reviewed the existing IoT malware detection work based on static analysis and pointed out that most existing solutions only detect malware in a single architecture. They summarized commonly used static features including function call graphs, CFGs, operation codes (opcodes), strings, and file headers.

A control flow graph is a directed graph *G(V, E)* that represents all the possible execution paths of a code piece, where *V* is a set of basic execution blocks and *E* is a set of

edges representing the connections between these basic blocks. Recent work adopted a CFG variation that contains extra information about a code piece. An Attributed Control Flow Graph (ACFG), *G(V, E, M)*, is an extended version of CFG, where *M* is the labeling function that maps a basic execution block in *V* to a set of attributes in *A*. The attribute set *A* can be tailored to capture the semantic meaning of the basic blocks or to characterize the blocks.

Feng et al. [10] adopted ACFGs to represent binary functions and extracted their statistical and structural attributes, such as the number of calls, the number of instructions, and the number of offspring. They utilized the bipartite graph matching algorithm to measure the similarity of two ACFGs.

Xu et al. [11] improved the previous work by employing the structured data embedding technique Structure2Vec to transfer ACFGs into feature spaces. They utilized a large-scale training dataset obtained from compiling the same source code on different architectures and with different compiler optimization techniques to train an SNN for code similarity detection. Sun et al. [12] also concluded that ACFGs can capture relevant features of binary codes and SNNs measure cross-architecture code similarity efficiently.

Feature selection plays an important role in the construction of efficient detection classifiers. The literature review concluded that most existing solutions only detect malware in a single architecture and commonly adopted static features. An improved CFG, ACFG, can extract better semantic features to represent the algorithm, instead of platform-dependent features. Therefore, it is suitable for detecting cross-platform TPCs.

III. METHODOLOGY

The proposed solution is outlined in Figure 1. As mentioned above, the tools for firmware image decomposition and binary analysis are available, but some are unstable. After a preliminary investigation, this study selects reliable tools (Binwalk [8] and Angr [13]) to automate the binary code extraction process.

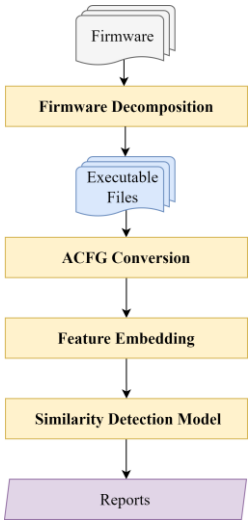

Figure 1. The proposed system architecture.

The Firmware Decomposition module applies Binwalk to decompose firmware images. The module identifies file types through the file header and then extracts executable files for vulnerable TPC inspection. The executable files serve as inputs for the ACFG Conversion module are converted into ACFGs, where the conversion applies the binary analysis tool Angr to convert binary codes into ACFGs. The Feature Embedding module adopts Struc2Vec [14] to encode the graphic structure features of ACFGs, and the detection module applies a SNN to compare the similarity of two embedded ACFGs: a known vulnerable TPC and a function code to be examined.

A. Firmware Decomposition

The Firmware Decomposition module utilizes Binwalk to disassemble firmware and retrieves the file system and image files. IoT firmware mostly uses the file system SquashFS to compress a Linux operating system, where most research applied Binwalk to perform the decompression. This module searches for all the file directories recursively, starting from the root directory. The file format Executable Linkable Format (ELF) is a common standard file format for executable files, object codes, and shared libraries. By using the ELF information, this module identifies the above executable files to be inspected.

B. ACFG Conversion

The ACFG Conversion module applies the tool Angr [13] to convert executables into ACFGs. Angr decomposes a target executable file into binary functions and then converts each binary function into its corresponding ACFG, where a basic block in ACFGs uses program control-related instructions, such as Jump, Call, or Return, as an edge connecting to another basic block. In other words, the ACFG represents the target binary function. Angr translates binary codes of different processor architectures in a set of intermediate instructions so that opcodes from different architectures map to the same or similar intermediate instructions. Based on Angr's instruction set, this study defines two types of relevant instructions: data movement and arithmetic & logic, to represent the functionaflity of a basic block.

C. Feature Embedding

This study applies Struc2Vec [14] to embedding the structural feature of ACFG and the features of a basic block in the graph, namely to encode the ACFG features. Struc2Vec learns latent representations that capture the structural identity of nodes in a graph. It measures node similarity hierarchically at different scales, constructs a multilayer graph to encode structural similarities, and generates the embeddings for nodes.

Struc2Vec consists of the following main steps to learn latent representations for a graph *G(V, E)*. (1) It determines the structural similarity between each node pair in the graph for different neighborhood sizes, where the structural distance is explained in the following paragraph. In this way, it measures node similarity hierarchically at different scales. (2)

TABLE I. THE DETECTION RESULTS OF THE OPENSSL VULNERABILITY.

Firmware	Executable file	Vulnerable function	Similarity
DAP-1562	wpa_supplicant	ssl3_get_new_session_ticket	0.8913
E1550USB-NVRAM60K	libssl.so.1	ssl3_get_new_session_ticket	0.8602
E4200USB-NVRAM60K	libssl.so.1	ssl3_get_new_session_ticket	0.8602
ddwrt.v24-23838	openvpn	ssl3_get_message	0.8491
E3000USB-NVRAM60K	libssl.so.1	ssl3_get_message	0.8133
K26-1.28.RT-MIPSR1	libssl.so.1	ssl3_get_new_session_ticket	0.8084
Tomato-K26USB-1.28RT-N5X	libssl.so.1	ssl3_get_new_session_ticket	0.8084

It constructs a weighted multi-layer graph hierarchically, where each layer corresponds to a level of the hierarchy in measuring structural similarity. (3) it applies a biased random walk to generate node sequences, where the sequences include more structural similarity of the graph. (4) It learns the latent representation from the node sequences.

D. Similarity Detection

The above feature embedding module encodes ACFGs into graph-embedded networks. This study employs a SNN to learn the similarity of two ACFGs, where the two networks in the SNN are designed as ACFG graph embedded networks and convert the input, a pair of ACFGs, into vectors.

To detect binary codes of cross-architecture, the binary functions of the same category have similar control flows and similar ACFGs. Hence, the loss function is defined as the cosine distance of the two vectors.

IV. SYSTEM EVALUATION

This study designs the experiment that investigates the performance of the proposed system in two aspects: the success rate of firmware decomposition and detection efficiency, as automatic decomposition is as important as vulnerability detection. It collects 50 firmware samples from two sources (FirmAE [15] and DD-WRT [16]) in order to examine diversified IoT devices.

This experiment employs AUC to evaluate the model efficiency, as it counts as a measure of the ability of the detection model to distinguish between the classes. A high AUC implies that the model distinguishes between the positive and negative classes efficiently. The AUC values between 0.9~1 are considered excellent; those between 0.8-0.9 are good; those between 0.7-0.8 are fair; those between 0.6-0.7 are poor. Figure 2 illustrates the ROC curves. The evaluation results show that the proposed method yields AUC-ROC value of 0.971 and achieves efficient detection performance.

For inspecting the detection performance on cross-platform, this experiment targets OpenSSL vulnerabilities in multiple architecture environments. The proposed system can automatically decompose firmware images with a high success rate of 94%. Table 1 lists the detected vulnerable firmware and functions that contain the OpenSSL vulnerabilities. The proposed detection method identifies the vulnerabilities with high similarity scores over multiple architecture environments, which implies that the proposed system can identify vulnerable functions effectively.

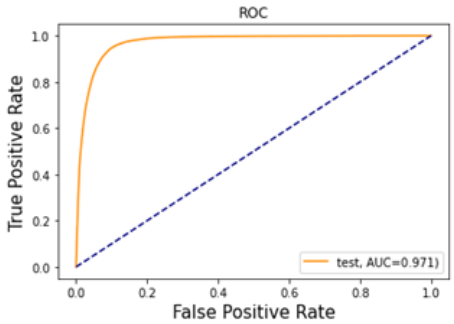

Figure 2. AUC-ROC

V. CONCLUSION

This study proposes a cross-architecture vulnerability detection method for IoT devices that discovers vulnerable TPC functions. Most past studies address one of the above IoT cyber risks, which motivates this research to discover these two critical IoT vulnerabilities. By evaluating the proposed system with real-world firmware images, the experimental results show that the proposed method can identify efficiently the aforementioned IoT vulnerabilities.

The proposed method employs the graphical representation ACFG to represent the semantic meaning of a binary function and the graph embedding technique Struc2Vec to extract the structural and code features of a function code. It applies a Siamese network to identify a function. The evaluation demonstrates that the proposed solution can identify the top IoT vulnerabilities mentioned above effectively. The real-world case studies demonstrate that the proposed system can automatically decompose firmware effectively.

Future work directions can investigate hybrid analysis approaches or extract more features from function codes to

improve detection performance. Furthermore, investigating other similarity detection models is another possible research direction as well. IoT devices are prevailing. Hence, reducing IoT cyber risks can improve network security for businesses and users.

REFERENCES

[1] S. Bennett. "IoT security statistics 2023 - Everything you need to know." [Online]. Available from: https://webinarcare.com/best-iot-security-software/iot-security-statistics/ [retrieved: May, 2024].

[2] M. Hasan. "State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally." [Online]. Available from: https://iot-analytics.com/number-connected-iot-devices/ [retrieved: May, 2024].

[3] Gartner. "Forecast: IT services for IoT, worldwide, 2019-2025." [Online]. Available from: https://www.gartner.com/en/documents/4004741 [retrieved: May, 2024]

[4] F. Baldassari. "IoT developers can't afford to ignore third-party code." [Online]. Available from: https://www.techtarget.com/iotagenda/post/IoT-developers-cant-afford-to-ignore-third-party-code [retrieved: May, 2024].

[5] Micro.ai. "The risk of using third-party components in IoT devices." [Online]. Available from: https://micro.ai/blog/the-risk-of-using-third-party-components-in-iot-devices [retrieved: May, 2024].

[6] H. A. Al Essa and W. S. Bhaya, "Ensemble learning classifiers hybrid feature selection for enhancing performance of intrusion detection system," Bulletin of Electrical Engineering and Informatics, vol. 13, no. 1, pp. 665-676, 2024.

[7] B. Zhao et al., "A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware," Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, pp. 442-454.

[8] ReFirm Labs. "Binwalk." [Online]. Available from: https://github.com/ReFirmLabs/binwalk [retrieved: May, 2024].

[9] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of IoT malware and detection methods based on static features," ICT Express, vol. 6, no. 4, pp. 280-286, 2020/12/01/ 2020, doi: https://doi.org/10.1016/j.icte.2020.04.005.

[10] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable graph-based bug search for firmware images," Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 480-491.

[11] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 363-376.

[12] H. Sun, Y. Tong, J. Zhao, and Z. Gu, "DVul-WLG: Graph Embedding Network Based on Code Similarity for Cross-Architecture Firmware Vulnerability Detection," Cham, 2021: Springer International Publishing, in Information Security, pp. 320-337.

[13] Y. Shoshitaishvili et al., "Sok:(state of) the art of war: Offensive techniques in binary analysis," 2016 IEEE Symposium on Security and Privacy (SP), 2016: IEEE, pp. 138-157.

[14] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 385-394.

[15] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "Firmae: Towards large-scale emulation of iot firmware for dynamic analysis," Annual Computer Security Applications Conference, 2020, pp. 733-745.

[16] DD-WRT. "Router Database." [Online]. Available from: https://dd-wrt.com/support/router-database/ [retrieved: May, 2024].