Similarity Based Binary Backdoor Detection via Attributed Control Flow Graph

Tianchen Zhang*, Haixiang Wang†, Huan Ying†, Jiyuan Li‡
*College of Electrical Engineering, Zhejiang University, Zhejiang, China
Graduate School of China Electric Power Research Institute, Beijing, China
†China Electric Power Research Institute, Information & Communication Department, Beijing, China
‡State Grid Zhejiang Electric Power Research Institute, Zhejiang, China
Email: {tianchen-zhang,xji}@zju.edu.cn, {wanghx,yinghuan}@epri.sgcc.com.cn, li_jiyuan@zj.sgcc.com.cn

***Abstract*—The problem of backdoor detection aims at detecting whether binary backdoor functions coming from different embedded end devices are similar to some known backdoors. Existing approaches are using backdoor signature or approximate graph-matching algorithms. They all meet a problem that they are hard to adapt to a new task. Also, approximate graph-matching algorithms are inevitably slow and sometimes inaccurate. To address these issues, in this work, we propose a novel neural network-based approach to compute the embedding, i.e., a numeric vector, based on the control flow graph of each binary function, then the backdoor detection can be done efficiently by measuring the distance between the embeddings for functions in firmware and typical backdoor functions. We evaluate this method can achieve a F-1 score of 0.75 and detect backdoor in firmware significantly faster than graph-matching method.**

***Index Terms*—Firmware, Backdoor Detection, Neural Network**

I. Introduction

mart grid is a national infrastructure that provides reliable, efficient electricity transmission and distribution with information and communication technology[1, 2]. With the integration of cyber space and physical space in smart grid, it is easy for an adversary to affect the normal operation of the embedded end device by adding a backdoor. On December 23, 2015, Black Energy attacked the Ukrainian power grid, leading nearly 700,000 user power outages[3]. Therefore, timely response to attacks is especially critical for the stable operation of smart grid.

Given a binary file of an embedded end device, we would like to detect whether its firmware has a backdoor. This problem one application of binary code similarity detection which has many people researching on[4, 5]. Nowadays, backdoors detection in firmware images of embedded end devices is particularly critical and more crucial than ever because the development of smart grid. The backdoor will implement some basic function. They may be quite the same in source code. However even the same backdoor at source code level may spread across hundreds or more devices that have diverse hardware architectures and software platforms.

Due to the increasing security requirements, security practitioners need a tool for fast backdoor detection of binary code on multiple platforms such as x86, ARM or MIPS. Reseachers are detecting backdoors by extracting some backdoor signatures and comparing binaries with these pre-collected backdoor signature[6]. However, this method need a lot of experts' effort and may fail to find a valid backdoor signature. Recently, researchers have proposed to extract some platform-independent features in the malicious backdoor code or backdoor control flow graph to represent a function, and then detected whether a binary file contains a malicious backdoor with these features using the graph matching algorithm[**?**]. When detecting the backdoor in a binary file, these methods need to calculate the embedding amount of a binary function and a typical backdoor, but also depend on the graph matching algorithm[7] to calculate the similarity between the target function and the binary backdoor codebook. The efficiency of all similarity detection methods based on graph matching is limited by the efficiency of graph matching algorithms (such as bipartite graph matching[8]) which are slow. In addition, if the effective features of the back door cannot be extracted, this method cannot effectively detect the variant of the back door.

In recent years, deep learning has been applied to many applications and has shown stronger effects on binary analysis than other methods. Binary code data is characterized by continuity, density, and local correlation, making it ideal applications to use deep learning. A deep neural network can find a high quality representation of binary code based on domain knowledge by training a neural network in en end-to-end way. Because neural networks have the ability of feature combining and generalization, we are dealing with more complex binary code security issues without the tedious feature engineering.

Inspired by these advantages, in this work, we propose a deep neural network-based approach to generate embeddings of binary functions for backdoor detection. In particular, assuming a binary backdoor function is represented as a control-flow graph with attributes attached to each node, we use a LSTM neural network to convert the graph into an embedding. We summarize our contributions as follow:

1) We propose the first neural network-based approach to generating embeddings for binary backdoor functions

2) We propose a novel approach to train the embedding network using a Siamese network so that a pre-trained model can generate embedding to be used for backdoor detection.

3) We evaluate our method can achieve a F1 score of 0.75 and detect backdoor in firmware significantly faster than graph

978-1-7281-4390-3/20/$31.00 ©2020 IEEE

match method.

II. Background

A widely recognized definition of a backdoor is a piece of software code or function that allows an adversary to gain unauthorized access to the system without the end user's knowledge[9, 10]. Backdoor as a form of malware which is different from other malware families, such as viruses and worms, has a open definition. Adversaries can implement them in a variety of ways. Therefore, it is more challenging to create a generic methods for detecting them. However, in this article, we will divide the way backdoors into the following common categories:

The first category is the so-called authentication bypass vulnerability, which refer to the security vulnerabilities introduced by standard authentication mechanism of a program or system. Adversary can obtain a higher level of access by exploiting this category of backdoor. There are multiple implementations of this vulnerability and the most common implementations are those that caused by static hard-coded data.

A standard authentication usually includes the interaction of username, password and other authentication attributes. Then the authentication system grants the user the corresponding access privilege accordding to the specific credential logic. One of the characteristics of embedded devices is that they may not always be connected to the Internet, so some authentication needs to be done locally. The adversary can analyse the static data stored in the program or the device's non-volatile memory to check the hard-coded credential logic whether there is a backdoor can be exploit to pass the authentication. The most naive backdoor is that the program compare the authentication password with the static correct string and act as a valid credential if they are the same. Also, there are many more complicate examples. For instance, CVE-2013-6026 is reported "Joel's Backdoor" which is elaborated in a series of D-link routers[11]. All HTTP requests without authentication are available once an attacker sets the user-agent header as "xmlset_roodkcableoj28840ybtide" which means "edit by 04882 joel backdoor" in reverse order. A notable feature of this backdoor is the comparison with static strings in the program binary code.

A more skillful authentication bypass vulnerability is caused by traditional programming vulnerability. In this case, magic strings can not be found in the program or the device's non-volatile memory. The adversary need to construct an input which can cause incorrect handle of the program and bypass the standard authentication process with a method not considered by a badly-designed authentication logic. Finding this kind of backdoor is equivalent to searching vulnerability, such as buffer overflows and format string vulnerability. So most vulnerability searching work can be help in backdoor detection.

Another type of backdoor is not included in the standard certification process, but a convenient debug interface for developers to facilitate development. It is usually not documented in the user manual of the embedded devices and will be disabled before the embedded devices release. However, some developer may leave the backdoor available intentionally or unintentionally. Different implements of this type of backdoor vary greatly in function. Some backdoor will provide a service that anyone can directly execute commands on the embedded device with an unauthenticated protocol, while others provide a service that only leak some privileged information. A real-world example is the TCP-32764 backdoor[12] which listens for the TCP packets from port 32764 and leak the configuration values of the devices. The severity of the backdoor depends on the information leaked. To detect this kind of backdoor need to focus on control flow (which control branch will lead to a command line) and data flow (which data branch will leak the privileged data) in the embedded device firmwares.

There is also a type of back door called encrypted backdoor. They can destroy devices by weakening cryptographic primitives such as random number generators[13].

III. Design

In this section, we start with a brief overview of the firmware detection method, and then introduce the main technical point we use in the method.

A. Overview

To detect backdoor in the firmware, we propose a method consisting of graph embedding and nueral network. First we extract the attributed control flow graph feature of the code block in the firmware. Then construct a neural embedding network to get the representation of the whole function in the firmware. By training the embedding network in Siamese architecture, the network will learn to get a better representation of a function with or without backdoor and distinguish whether two representations contain the same backdoor at the same time. Finally, we extract the representations of firmwares with backdoors and construct a representation codebook of function with backdoor. We can quickly detect whether there is a backdoor in the firmware by doing a similarity calculation between target firmware and the nearest representations in the codebook.

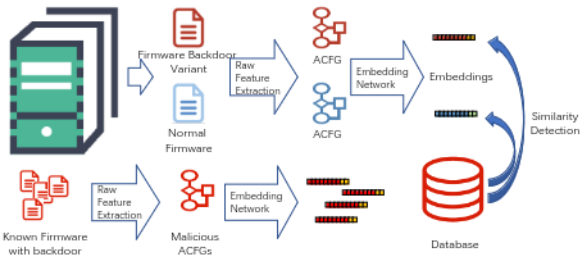

Fig. 1. System overview of the approach.

TABLE I
THE ATTRIBUTE USED IN ACFG

Type	Attribute name
Block-level attributes	String Constants Numeric Constants No. of Transfer Instructions No. of Calls No. of Instructions No. of Arithmetic Instructions
Inter-block attributes	No. of offspring Betweenness

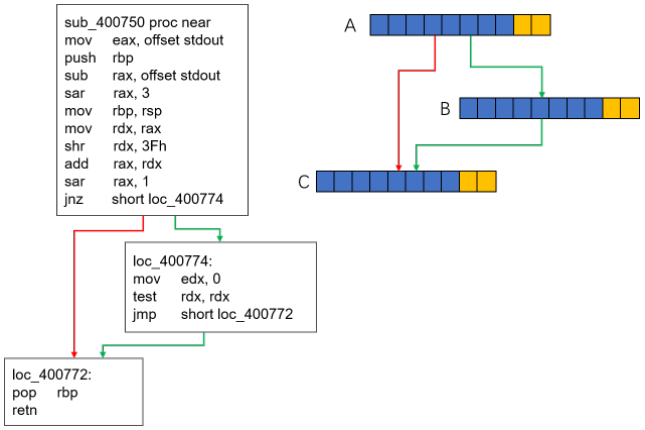

Fig. 2. A example of extracting ACFG from code section. The gray part is the corresponding ACFG.

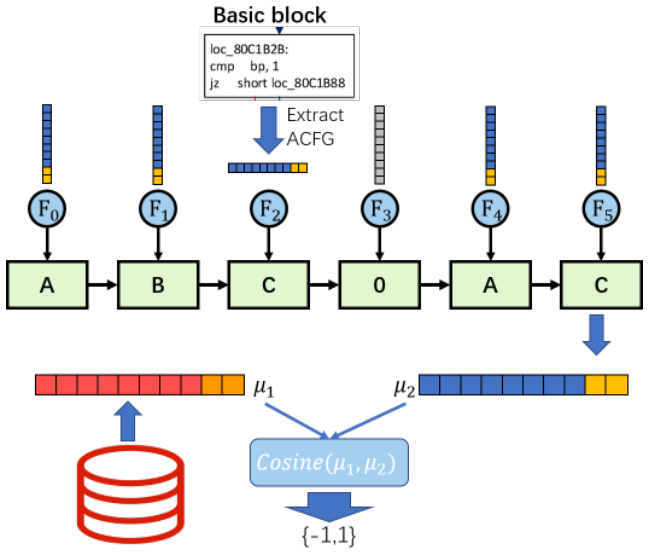

Fig. 3. A example of extracting ACFG from code section. The gray part is the corresponding ACFG.

B. Graph Embedding

Since the length of different embedded end device firmware are different, we need to find a way to encode the firmware into an unified representation so that we can easily compare an unknown firmware with existing firmwares with backdoor to detect whether the firmware contains a backdoor.

The binary text is a low-level representation of a firmware which contains whole intrinsic information but few intuitive information for human or neural network. So we encodes the control flow graph which more comprehensible than binary text into a numeric feature vector. There are many methods to evaluate the similarity of two numeric feature vector, so we can easily find a similarity function to compute the distance of two control flow graph. Another advantage is we can find the most similiar feature vectors with the faiss algorithm and check the similarity between the two feature vectors.

Usually, the backdoor will appear as a function or as a few lines of instructions oin a function. Each binary function from a firmware image , We first extracts raw features in the form of an attributed control flow graph (ACFG). In an ACFG, each vertex is a basic block labeled with a set of attributes. Table 1 lists six block-level attributes and two inter-block attributes used in this paper. Figure 2 illustrates an ACFG for a example function. Consequently, to identify a set of binary functions that have a known backdoor, we just need to find the corresponding embedding of the query function and check whether it is close to the embedding of the known backdoor.

C. Neural Network Model

In this section, we present the neural network model used in our solution to solve the embedding problem. In this work, we will train a embedding network that encode the function f in the target firmware into its representation g via attributed control flow graph.

We design the neural network based on Long Short-Term Memory(LSTM) ϕ [14]. Since the input is an sequence of ACFG, we will leverage recurrent neural network to address the problem. Since the scale of the ACFG may be quite huge, LSTM can avoid the gradient disappearance and gradient explosion when trainning the neural network.

It is worth mentioning that our backdoor detection neural network will be trained based on code similarity instead of a classification way. Thus we use a Siamese architecture which is common now in face recognition network trainning to train the embedding network. A Siamese architecture takes two functions as its input, and they will be encoded into two feature vector by the same embedding network. The embedding network is trained to get a high similarity score when the two functions have the same backdoor and receive a negetive similarity score when the two function have the different backdoor or one function has a backdoor while the other doesn't. Training a Siamese architecture has a huge advantage that we can obtain a large number of pairs of similar functions, as well as pairs of dissimilar functions to train the model instead of suffering from labeling data.

IV. EVALUATION

In this section, we first describe the experiment setup and the evaluation metrics, and then evaluate the performance of

TABLE II
THE PARAMETERS OF THE DETECTION MODEL.

Hyperparameter	Value	Hyperparameter	Value
Optimizer	RMSProp	Learning rate	0.001
Batch size	32	Training epochs	20
Number of lstm cells	40	Input units	8
Hidden layers units of LSTM	128	Hidden layers units of DNN	64,32
Activation	tanh	Dropout	0.5

our method to detect backdoors in firmwares.

A. Experiment setup

We wrote the plugin to the disassembler tool IDA Pro for ACFG extraction. We implemented codebook generation, feature encoding in python, and adopted faiss for search. We utilized mongoDB to store the backdoor id and encoded features. Our experiments were conducted on a server with 64 GB memory, 72 cores at 2.3 GHz and 4 TB hard drives. All the evaluations were conducted based on four types of datasets: 1) baseline evaluation dataset; 2) two public firmware images; 3) 12,413 firmware images and 4) the vulnerability dataset.

B. Evaluation metrics

We measure the robustness and generalization performance of the model by standard metrics: F1 score and time cost to detect a firmware. F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant. In general, precision ,recall and F1 score are defined as below:

$$precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F1\ score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

C. Evaluation Results

In order to compare the performance of the different method. We also implement codeook with other methods, such as signiture matching algorism, graph matching algorism. From the results shown in Fig.4 we can see that, the F1 score is 0.4274 using signiture based method because signiture based method is strict in matching and have poor generalization ability, so it has a poor recall rate which cause the bad preformance in F1 score. The performance of our method and graph matching based method is much better than the signiture based method because it can take the code control flow into consideration, it has a certain degree of generalization ability. However, as Fig.5 shows, as the binary function size getting large, graph matching method suffers from a matching

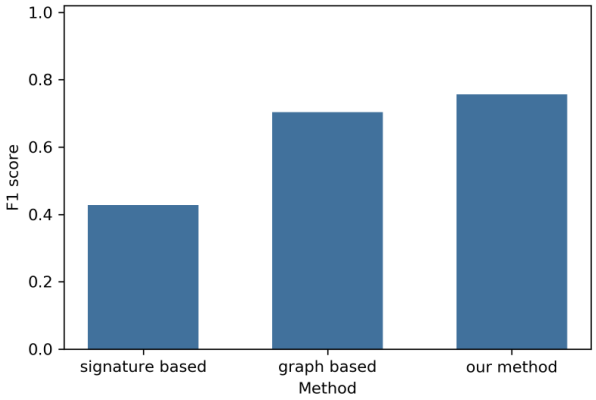

Fig. 4. F1 score of different methods

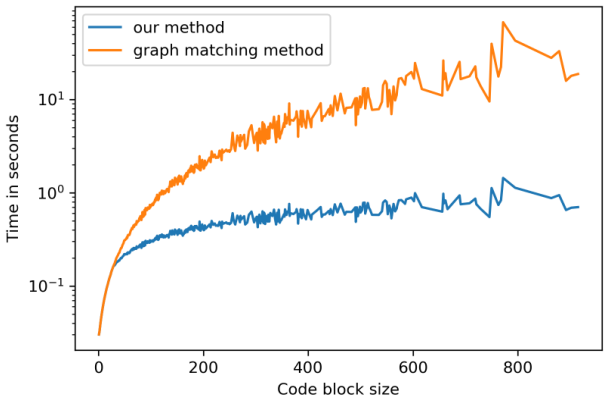

Fig. 5. time costed to detect backdoor

efficiency problems. Our method will encode backdoor feature into a vector, so it has great advantages in storage and matching efficiency.

V. CONCLUSION

In this paper, we present a similarity based binary backdoor detection approach to detect backdoor in firmwares. We trained a model to generate the representation of code gadget. Our extensive evaluation shows that our method outperforms the the state-of-the-art approaches in accuracy and time efficiency.

ACKNOWLEDGMENT

Supported by the science and technology project of State Grid Corporation of China(No.52110418001K, Key Technologies of Vulnerability Mining and Attack Detection for Embedded Terminal in Power Grid)

REFERENCES

[1] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "Smart grid technologies: Communication technologies and standards," *IEEE*

transactions on Industrial informatics, vol. 7, no. 4, pp. 529–539, 2011.

[2] T. Flick and J. Morehouse, *Securing the smart grid: next generation power grid security*. Elsevier, 2010.

[3] H. Ying, X. Ouyang, S. Miao, and Y. Cheng, "Power message generation in smart grid via generative adversarial network," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 790–793.

[4] M. Chilowicz, E. Duris, and G. Roussel, "Syntax tree fingerprinting for source code similarity detection," in *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 2009, pp. 243–247.

[5] B. Liu, W. Huo, C. Zhang, W. Li, F. Li, A. Piao, and W. Zou, "αdiff: cross-version binary code similarity detection with dnn," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 667–678.

[6] B. Kang, H. S. Kim, T. Kim, H. Kwon, and E. G. Im, "Fast malware family detection method using control flow graphs," in *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*. ACM, 2011, pp. 287–292.

[7] S. Shang, N. Zheng, J. Xu, M. Xu, and H. Zhang, "Detecting malware variants via function-call graph similarity," in *2010 5th International Conference on Malicious and Unwanted Software*. IEEE, 2010, pp. 113–120.

[8] S. Bhagwani, S. Satapathy, and H. Karnick, "Semantic textual similarity using maximal weighted bipartite graph matching," in *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics, 2012, pp. 579–585.

[9] C. Hu, Y.-b. XUE, L. ZHAO, and Z.-j. LI, "Backdoor detection in embedded system firmware without file system," *Journal on Communications*, vol. 34, no. 8, pp. 140–145, 2013.

[10] E. P. Vermeulen, "Rules on backdoor listings: a global survey," *Indonesia-OECD Corporate Governance Policy Dialogue*, pp. 2015–2, 2014.

[11] W. Xie, Y. Jiang, Y. Tang, N. Ding, and Y. Gao, "Vulnerability detection in iot firmware: A survey," in *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2017, pp. 769–772.

[12] S. L. Thomas, "Backdoor detection systems for embedded devices," Ph.D. dissertation, University of Birmingham, 2018.

[13] S. Ruhault, "Sok: security models for pseudo-random number generators," *IACR Transactions on Symmetric Cryptology*, pp. 506–544, 2017.

[14] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.