DVul-WLG: Graph Embedding Network Based on Code Similarity for Cross-architecture Firmware Vulnerability Detection

Hao Sun[1], Yanjun Tong[1], Jing Zhao[1(✉)], and Zhaoquan Gu[2]

[1] Dalian University of Technology, Dalian, China
zhaoj9988@dlut.edu.cn
[2] Guangzhou University, Guangzhou, China

Abstract. Vulnerabilities in the firmware of embedded devices have led to many IoT security incidents. Embedded devices have multiple architectures and the firmware source code of embedded devices is difficult to obtain, which makes it difficult to detect firmware vulnerabilities. In this paper, we propose a neural network model called DVul-WLG for cross-architecture firmware vulnerability detection. This model analyzes the similarity between the binary function of the vulnerability and the binary function of the firmware to determine whether the firmware contains the vulnerability. The similarity between functions is calculated by comparing the features of the attribute control flow graph (ACFG) of the functions. DVul-WLG uses Word2vec, LSTM (Long Short-Term Memory) and an improved graph convolutional neural network (GCN) to extract the features of ACFG. This model embeds instructions of different architectures into the same space through canonical correlation analysis (CCA), and expresses instructions of different architectures in the form of intermediate vectors. In this way, the heterogeneity of architectures can be ignored when comparing cross-architecture similarity. We compared DVul-WLG with the advanced method FIT and the basic method Gemini through experiments. Experiments show that DVul-WLG has a higher AUC (Area Under the Curve) value. We also detected vulnerabilities in the real firmware. The accuracy of DVul-WLG is 89%, while FIT and Gemini are 78% and 73%, respectively.

Keywords: Vulnerability detection · Binary code similarity · Graph embedding.

1 Introcudtion

In the era of the Internet of Everything, embedded devices exist in all aspects of daily life. Security issues caused by embedded devices have aroused widespread concern. An embedded device is a closed system that boots into a unified software package called firmware. The lack of security considerations at the beginning of the firmware design and the reuse of a large amount of code have resulted in many vulnerabilities in the firmware. In addition, vulnerabilities in the firmware

can be easily exploited [1]. In July 2020, a research team discovered many serious security vulnerabilities in three different home hubs Fibaro Home Center Lite, Homematic, and eLAN-RF-003. These vulnerabilities can lead to sensitive data leakage, remote code execution, and man-in-the-middle attacks. In December 2020, a hacker used an undiscovered vulnerability to forcibly open the door of a third of PickPoint's lockers, causing thousands of packages throughout Moscow to be at risk of being stolen. From the above-mentioned network attack incidents, it can be found that the vulnerabilities in the firmware have brought great security risks. What's worse is that we cannot use traditional vulnerability scanning tools on PCs and mobile devices to detect firmware vulnerabilities.

The detection of firmware vulnerabilities has become increasingly important. In order to solve this problem, some security researchers have proposed technologies to dynamically detect firmware vulnerabilities [2, 3]. However, dynamic detection technology has great limitations. Usually, the firmware is customized for a specific embedded device, so that the detection method of a certain device cannot be universal. The dynamic detection usually adopts the method of firmware simulation. However, the parameters of NVRAM (Non-Volatile Random Access Memory) are usually not available, causing security analysts to repeatedly hijack certain functions to bypass exceptions so that the program can be executed. This process is not always feasible and very time-consuming. Therefore, for large-scale firmware vulnerability detection, static detection methods are more advantageous. The static detection method for firmware vulnerabilities must be universal and lightweight. Traditional static detection techniques such as symbolic execution and stain analysis are not suitable. At present, many static detection methods have solved the problem of detecting vulnerabilities at the source code level [4, 5]. However, it is difficult to obtain the source code of the firmware, so these detection methods are not suitable. The detection method of binary code similarity does not require firmware source code, and it is universal and lightweight. Therefore, for firmware vulnerability detection, the detection method of binary code similarity is advantageous and efficient.

As shown in Fig. 1, the binary function can be converted into an attribute control flow graph (ACFG) by the IDA pro disassembly tool [6, 7]. When performing binary code similarity detection, first extract the features of ACFG, these features can be used to represent ACFG, thereby representing the binary function. Then the ACFG features are converted into feature vectors through the pre-trained neural network. Finally, the vector distance formula is used to calculate the distance between the feature vectors, and the vector distance is used to express the similarity of the binary function. This paper divides the features of ACFG into three categories: the semantic features of instructions, statistical features and structural features of graphs. FIT [20] uses the method of word embedding in natural language processing to extract the semantic information of instructions. But the traditional word embedding models CBOW (Continuous Bag-of-Words) [8] and Skip-Gram [9] can only consider the semantic relationship of instructions under the same architecture. However, firmwares with different architectures often have the same vulnerabilities, so cross-architecture situations

should be considered when comparing similarities with vulnerable functions. When comparing cross-architecture function similarity, it is not only necessary to maintain the semantic association of instructions in the same architecture, but also to maintain similar embeddings for instructions with the same semantics between different architectures, which cannot be achieved by traditional word embedding models. Regarding the structural features of the graph, Gemini [10] designed an aggregation algorithm inspired by Struc2vec, which can aggregate the features of the basic blocks to represent the graphical features of ACFG. However, this method allows the adjacent nodes of each basic block in the graph to have the same influence factor, and then attaches the attributes of the adjacent nodes to the basic block itself through nonlinear changes. In fact, the influence factors of adjacent nodes of the basic block are different, so Gemini's extraction of the structural features of ACFG is not accurate.

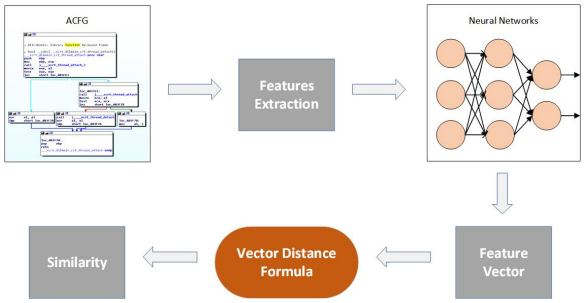


Fig. 1. Schematic diagram of binary code similarity detection.

In view of the above problems, the main challenges of this paper are in two aspects: One is the semantic feature of ACFG. When comparing binary functions of cross-architecture firmware, it is necessary to ensure the similarity of instruction semantics within the same architecture. At the same time, it is necessary to ensure the relevance of instruction semantics under different architectures. The second is the structural feature of ACFG graphics. The traditional GCN cannot extract the structure information of the directed graph, while the ACFG is a directed graph. We need to improve the GCN to be able to accurately extract the structure information of the ACFG.

The main contributions of this paper are as follows:

1 This paper uses code similarity analysis to design a cross-architecture firmware vulnerability detection model. The model combined with deep neural network can accurately extract the semantic and structural features of ACFG.
2 In the process of cross-architecture instruction embedding, this paper compares two classic word embedding models, CBOW and Skip-Gram. At the same time, ARM instructions and MIPS instructions are embedded in the same space through the canonical correlation analysis (CCA) method. When

comparing cross-architecture function similarity, the heterogeneity between architectures can be ignored, so that the semantic features of instructions of different architectures are compared in the same dimension, which improves the accuracy of the comparison.

3 This paper uses DGCN [23] to improve the graph embedding aggregation algorithm proposed by Gemini. According to the principle of DGCN, we assign different influence factors to the adjacent nodes of the basic block. Through experimental verification, the method in this paper can better extract the structural features of ACFG.

The remaining organizational structure of this paper is as follows: In the second section, we review more related work. In the third section, we describe the method that Siamese Network embeds the features of ACFG to compare the similarity. In the fourth section, we evaluate the effectiveness of our proposed method through experimental analysis. Finally, summarize all the work of this paper.

2 Relate Work

For our related work, this paper only discusses related technologies for binary vulnerability detection. In 2008, Gao et al. proposed BinHunt [12], a new technique for discovering semantic differences in binary programs. They use techniques such as graph isomorphism and symbolic execution to analyze the control flow graph of the binary program, and can identify the semantic difference between the original program and the patch program, thereby revealing the vulnerabilities eliminated by the patch program. On this basis, Jiang et al. proposed that the semantic differences between binary programs are easily interfered by others using simple obfuscating functions. Therefore, they used deep pollution and automatic input generation techniques to discover the semantic differences of CFG [13]. However, this method of capturing binary vulnerabilities through semantic differences relies on instruction semantics and is only suitable for a single architecture.

In 2013, Martial et al. proposed a polynomial algorithm by fusing the BinDiff algorithm with the Hungarian algorithm of bipartite graph matching [14]. The graph-based edit distance calculates a meaningful similarity measure, which significantly improves the matching accuracy between binary files. Flake proposed a heuristic method of constructing isomorphism between function sets in the same executable file but in different versions [15]. Pewny et al. observe the IO behavior of basic blocks and obtain their semantics, thereby effectively revealing the bugs in the binary code [16]. These methods all rely on accurate graphic matching technology, and have high time complexity, and are not suitable for large-scale binary vulnerability detection. DiscovRE [17] pre-filtered function pairs through digital features in order to reduce the costly calculation of graph matching. However, this method is not reliable and will produce a large number of false negatives.

In order to reduce the expensive cost of graph matching, the method of graph embedding has become a good choice. Graph embedding refers to the mapping of high-dimensional features in a graph to low-dimensional vector representations. The embedding vector can accurately represent the structural features in the graph, the attribute features of each vertex, and the interactive information between vertices and vertices [18]. For graph embedding vectors, we can use distance formulas between vectors, such as cos distance, Euclidean distance etc. to compare the similarity between graphics more easily. In 2016, Feng et al. [6] first used a codebook-based method to convert the ACFG of a binary function into a numerical vector, which makes it easier to calculate the similarity between graphs. After that, CVSSA [19] accurately extracts the features of ACFG at the binary function level through SVM. Gemini [10] proposed by Xu et al. uses a neural network to calculate the embedding, which extracts features at the basic block level, and then expresses the embedding of ACFG through an aggregate function, which further improves the accuracy of graph embedding. However, Gemini only expresses the embedding of the basic block through the statistical features of the basic block, completely ignoring the semantic features in the basic block, which will have a great limitation. When two basic blocks with completely different semantics have similar statistical features, Gemini will consider the two basic blocks to be similar. FIT [20] extracts the semantic features of instructions through the Word2vec, but instructions of different architectures are embedded in different spaces. FIT ignores the relevance of semantically equivalent instructions under different architectures, which leads to inaccurate comparisons of semantic features of functions under different architectures.

3 Embedded Network

This section will introduce how to convert the ACFG of the binary function into a graph embedding. For the embedded vector, the distance of the vector is calculated by the *cos* distance formula, and then the similarity between the binary functions is obtained. Here we introduce the theoretical model of the Siamese Network, which can better explain how the graph embedding network works.

3.1 Siamese Network

Siamese Network is a new type of neural network architecture. Siamese Network can learn a similarity metric from training data, which is often used to evaluate the similarity of input sample pairs. It has shown better capabilities in certain fields, such as face recognition and signature verification etc. As shown in Fig. 2, the Siamese Network architecture contains two identical sub-networks (the sub-networks have the same configuration and parameters). In this paper, these two networks are designed as ACFG graph embedded networks. These two sub-networks can convert the input ACFG sample pair into a vector, and then judge the similarity of the sample pair through the distance formula of the vector.

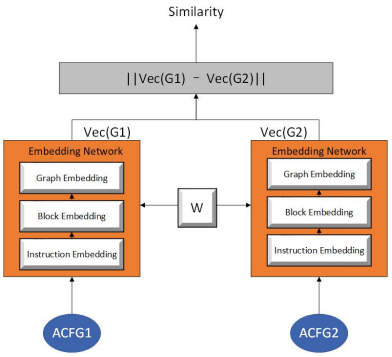


Fig. 2. Siamese Network.

The training goal of Siamese Network is to maximize similarity when a given pair of ACFG samples belong to the same category. The similarity should be minimized when the sample pairs belong to different categories. Whether the sample pairs belong to the same category depends on whether they are compiled from the same source function. As shown in Fig. 2, the input sample pair ACFG1 and ACFG2 are converted into vectors Vec_1 and Vec_2 through the graph embedding network. The similarity is measured by the *cos* distance of the vector, the measurement formula is as follows:

$$cos(Vec_1, Vec_2) = \frac{Vec_1 \cdot Vec_2}{||Vec_1||||Vec_2||} \tag{1}$$

For the input sample pair, we will mark it, if the input is the same category, mark it as +1, otherwise mark it as -1. Therefore, when training the Siamese Network, the input is in the form of triples $\langle ACFG_1, ACFG_2, Label \rangle$. In this paper, the loss function is only considered related to the parameters and input, so the loss function is defined as follows:

$$L = (Label - cos(Vec_1, Vec_2))^2 \tag{2}$$

When the sample pair belongs to the same category, the closer the *cos* similarity value is to 1, the smaller the loss value L. When the sample pairs belong to different categories, the closer the *cos* similarity value is to 0, the smaller the loss value L. Therefore, reducing the loss value L in the iterative process can meet the training goal of Siamese Network.

3.2 Embedding of Instruction Semantic Features

In Section 3.1, the overall architecture of Siamese Network is introduced. The most important part is the graph embedding sub-network. How to convert ACFG into vector representation is also the core work of this paper. As shown in Fig. 2,

the embedded network is mainly divided into three parts, namely instruction embedding, block embedding and graph embedding. This section mainly introduces instruction embedding.

Analogous to Word2vec of natural language processing, we regard each basic block as a sentence, and the instructions in the basic block as words. The classic word embedding models include CBOW and Skip-Gram. Both are composed of three layers of feedforward neural networks, which are input layer, hidden layer and output layer. The input of CBOW is the context of the word, and the context is used to predict the word. The input of Skip-Gram is the word itself, and the word is used to predict its context. The basic principles of the two are the same. This paper takes CBOW as an example to introduce its working principle.

For a given word sequence$w_1, w_2, ..., w_n$, w_k is the word to be predicted, and the sliding window size is c. The input layer is the context of w_k in the sliding window, and these words are represented by One-hot encoding. The weight matrix from the input layer to the hidden layer is W_1, and the word vector of the input layer is multiplied by the weight matrix and averaged to obtain the vector of the hidden layer. The weight matrix from the hidden layer to the output layer is W_2, and the vector of the hidden layer is multiplied by W_2 to get the vector of the output layer. The vector of the output layer is normalized by the softmax function and the value with the largest corresponding position in the vector is the predicted word. The objective function is to maximize the maximum likelihood estimation:

$$\frac{1}{n}\sum_{t=1}^{n}\sum_{-c<j<c} logp(w_k|w_{k+j}) \tag{3}$$

Whether using the CBOW model or the Skip-Gram model will cause a problem, the instruction embedding of the MIPS architecture and the instruction embedding of the ARM architecture are not in the same space. This ignores the semantic association of equivalent instructions between the two architectures, resulting in inaccurate comparisons of cross-architecture similarity. Inspired by [21] cross-language embedding, this paper uses CCA to embed MIPS and ARM instructions into the same space. First, the MIPS and ARM instructions are embedded in different spaces using the Word2vec model, and let $\Sigma \in \mathbb{R}^{n_1 \times d_1}$ and $\Omega \in \mathbb{R}^{n_2 \times d_2}$ respectively denote the vector spaces of the instructions of the two architectures. Instructions with equal semantics under the two architectures are mapped to the same space, which is not as easy as multilingual embedding in natural language. Because in natural language, the semantically equivalent words in different languages can be obtained through the dictionary. The instructions are different, and there is no dictionary-like translation between instructions of different architectures. At the same time, instructions are not atomically structured like words. Instructions are composed of mnemonics and operands, and different operands generate a large number of different instructions. In order to solve this problem, we artificially regard instructions with the same mnemonic as the same type of instructions, because most of the operations performed by instructions with the same mnemonic are similar. Based on prior knowledge, this

paper uses mnemonics to map MIPS and ARM instructions. For example, 'move' in MIPS and 'MOV' in ARM are considered equivalent. Through the instruction dictionary, let the instructions in the two subsets of $\Sigma' \subseteq \Sigma$ and $\Omega' \subseteq \Omega$ map one by one. x and y denote a pair of equivalent instructions from Σ' and Ω' respectively. a and b represent the projection direction, then the vector of x and y after the projection is expressed as:

$$x' = a^{\mathrm{T}}x, y' = b^{\mathrm{T}}y \tag{4}$$

The correlation between the projection vectors x' and y' is expressed as:

$$\rho(x', y') = \frac{E\left[x'y'\right]}{\sqrt{E\left[x'^2\right]E\left[y'^2\right]}} \tag{5}$$

The goal of our optimization is to maximize the correlation $\rho(x', y')$ and output two projection vectors a and b. Using these two projection vectors, all instructions of MIPS and ARM can be projected, which can be summarized as:

$$A, B = CCA(\Sigma', \Omega') \tag{6}$$

$$\Sigma^* = A^{\mathrm{T}}\Sigma, \Omega^* = B^{\mathrm{T}}\Omega \tag{7}$$

3.3 Embedding of Structural Features of ACFG

After the instruction embedding is generated, the instruction sequence in the basic block needs to be aggregated to generate the embedding of the basic block. Considering that in natural language processing, word embedding is used to represent sentence embedding. For an ordered sequence of instructions, the RNN model can effectively mine its semantic information and timing information. However, the instruction sequence in some basic blocks is too long. If the RNN model is used in training, the problem of gradient disappearance and gradient explosion will occur. Therefore, this paper chooses the LSTM model that performs better in long sequences. The LSTM model can summarize the instruction sequence in the basic block, and finally express all the instruction sequences with internal correlation through a vector. At the same time, the statistical features of the combined basic block are shown in Table 1. The combined vector represents the embedding of the basic block. The basic block embedding formula is as follows:

$$B_{fea} = W_{b1}B_{emb} + W_{b2}B_{sta} \tag{8}$$

W_{b1} and W_{b2} represent the weight matrix of instruction semantic feature and statistical feature, respectively.

After the feature of each basic block is generated, the features of all basic blocks need to be aggregated as the feature of ACFG. A simple method is to add the features of all basic blocks to represent the features of ACFG. However, this method cannot extract the structure of the graph, resulting in insufficient accuracy of feature extraction. Inspired by Structure2vec, Gemini recursively

aggregates the features of basic blocks through the topological structure of graph. After a few steps of recursion, the graph embedding network will calculate a new vector representation for each basic block. This vector includes the features of the basic block and the structural features of the graph. Gemini trains the interaction between nodes through a fully connected neural network. The formula is as follows:

$$\mu_v^{(t)} = tanh(W_1 x_v + \sigma(\sum_{u \in N_{(v)}} \mu_u^{(t-1)})) \tag{9}$$

x_v represents the feature vector of the basic block; W_1 is the matrix coefficient of the basic block feature; μ_v represents the new vector representation calculated by the graph embedding network for node v; $N_{(v)}$ represents the adjacent node of the basic block v; σ represents the fully connected neural network. The above formula can be understood as for any basic block v, the graph embedding network calculates a new feature vector for it. This feature vector is obtained by summing the features of all adjacent nodes of the basic block v and then undergoing nonlinear changes, and finally adding to the feature vector of the basic block v. This formula does consider the features of the basic block itself and the topological features of the graph. But let all adjacent nodes of the basic block have the same influence factor for summation. Although σ has a very strong nonlinear transformation, it is still not accurate enough to represent the structural features of the graph.

This paper has made improvements to this. Using GCN to extract the structural features of the topological graph has become one of the most effective methods. GCN uses the Laplacian matrix of the graph to implement the convolution operation of the topological graph, and its propagation rules are as follows:

$$Z_F = H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W) \tag{10}$$

Where $\widetilde{A} = A + I_N$; A are the adjacency matrix of the graph; I_N is the identity matrix; $\widetilde{D}$ is the degree matrix of $\widetilde{A}$; H is the feature of each layer, for the input layer H is the feature of the basic block; σ is the nonlinear activation function; and W is the training parameter. This propagation formula can extract the features of undirected graphs better. Unfortunately, ACFG is a directed graph. If the propagation formula of GCN is used, the direction information of the directed graph will inevitably be lost, which will have a great impact on ACFG. In order to be able to use the powerful ability of GCN to extract graphic features, we are inspired by the DGCN proposed by [11], and retain the direction information of the graphic when using the GCN propagation formula.

As shown in Fig. 3, we add two matrices to ACFG, the in-degree matrix A_{Sin} and the out-degree matrix A_{Sout}. In-degree matrix means that there is a node k, and two nodes i and j point to node k at the same time $\{i \Rightarrow k \Leftarrow j\}$, let $A_{Sin} += 1$. On the contrary, if there is node k pointing to node i and j at the same time $\{i \Leftarrow k \Rightarrow j\}$, let $A_{Sout} += 1$. These two matrices are symmetric because $A_{Sin}(i,j) = A_{Sin}(j,i)$ and $A_{Sout}(i,j) = A_{Sout}(j,i)$. Therefore, these

two matrices can be constructed similar to undirected graph convolution, the formula is as follows:

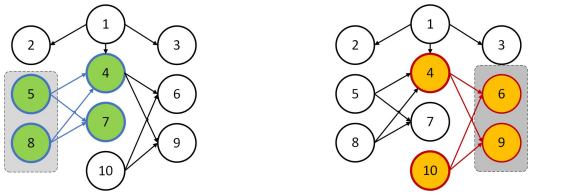
$$Z_{Sin} = H^{(l+1)} = \sigma(\widetilde{D}_{Sin}^{-\frac{1}{2}}\widetilde{A}_{Sin}\widetilde{D}_{Sin}^{-\frac{1}{2}}H^{(l)}W) \tag{11}$$

$$Z_{Sout} = H^{(l+1)} = \sigma(\widetilde{D}_{Sout}^{-\frac{1}{2}}\widetilde{A}_{Sout}\widetilde{D}_{Sout}^{-\frac{1}{2}}H^{(l)}W) \tag{12}$$

Through these two auxiliary formulas, the directionality of the graph can be effectively expressed, and then the convolution formula of the undirected graph is merged. The fusion method used in this paper is splicing, and the fusion formula is as follows:

$$Z = Concat(Z_F, \alpha Z_{Sin}, \beta Z_{Sout}) \tag{13}$$

α and β represent the different weights of in-degree convolution and out-degree convolution, and this weight is obtained through learning. In this way, the structure of the directed graph ACFG can be extracted through the fused convolution formula.



(a) In-degree Proximity (b) Out-degree Proximity

Fig. 3. DGCN second-order proximity.

Similar to the representation of basic block features, we combine graph embedding and graph statistical features, as shown in the Function-level of Table 1. The features of the final ACFG are expressed as follows:

$$F_{fea} = W_{f1}Z + W_{f2}F_{sta} \tag{14}$$

W_{f1} and W_{f2} are the matrix coefficients of the graph embedding feature and the graph statistical feature, and F_{sta} is the graph statistical feature.

4 Evaluation

This section mainly introduces the details of the experiment, as well as evaluating the effectiveness of instruction embedding for spatial projection and evaluating the effectiveness of using DGCN to extract ACFG graph structures. This paper compares the most advanced methods such as Gemini and FIT to prove the

Table 1. Statistical Features

Type	Attribute Name	Type	Attribute Name
Block-Level	No.of String Constants	Function- level	No.of Arithmetic Instructions
	No.of Numeric Constants		No.of Logic Instructions
	No.of Arithmetic Instructions		No.of Transfer Instructions
	No.of Logic Instructions		No.of Transmit Instructions
	No.of Transfer Instructions		No.of Basic Blocks
	No.of Transmit Instructions		No.of Edges
	No.of Instructions		No.of Function Calls
	No.of Calls		No.of Incoming Calls
	No.of Offspring		No.of Instructions
	Betweeness		No.of Variables

effectiveness of the improved method. Finally, this paper detects real firmware vulnerabilities and proves that the method proposed in this paper can be applied to real firmware vulnerabilities detection.

4.1 Implementation

The experiment in this paper is deployed on a server with a 16-core CPU, 128GB RAM, and 1TB SSD.This paper has established 3 data sets: (1) Data set I is used to train the graph embedding model. As shown in Table 2, this papaer compiles different versions of OpenSSL, BusyBox, and FindUtils into binary files of MIPS and ARM architectures, and opens four different optimization levels: O0, O1, O2, and O3. The data in the table represents the number of functions under different architectures of different programs, each function represents an ACFG, a total of 59410 ACFGs. The extraction of ACFG uses IDA pro script written by Gemini, which can effectively extract the features of ACFG. (2) Data set II is used to verify the effectiveness of the graph embedding model. As shown in Table 3, we compile multiple Unix Shell programs such as cat, shown, and cp into binary programs under the two architectures of ARM and MIPS, and open O0 to O3 four optimization levels. There are 13587 ACFGs in total. (3) Data set III is the firmware image obtained from real manufacturers, including manufacturers such as D-Link, TP-Link, Netgear and Buffalo. This paper mainly obtains firmware with corresponding vulnerabilities from various manufacturers, and is mainly used for the detection of three vulnerabilities: CVE-2020-1967, CVE-2020-1971 and CVE-2017-15873. For each of these three vulnerabilities, 50 firmware images are selected for detection.

This paper uses data set I to train Siamese Network, the Batch Size is 10, and 5 sets of similar sample pairs and 5 sets of dissimilar sample pairs are selected each time. Similarity means that ACFG sample pairs are derived from the same original function. Similar sample pairs are marked as <ACFG1,ACFG2,+1>, and dissimilar sample pairs are marked as <ACFG1,ACFG2,-1>. The iterative principle of the model training process can refer to the third section of this paper. The learning rate during training is 0.001, the embedding depth of the model is

128, and the maximum number of iterations is 100. The trained model is tested on data set II. The Batch Size is also set to 10 during the test, and 5 groups of similar sample pairs and 5 groups of dissimilar sample pairs are selected each time. Finally, the TPR (true positive) and FPR (false positive) under different test sets are obtained, and the ROC curve is obtained.

Table 2. Data set I

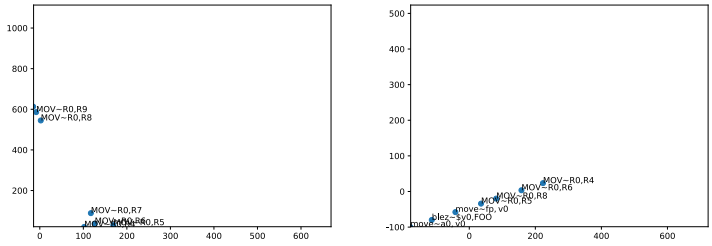
	OpenSSL	BusyBox	FindUtils
MIPS	21085	6700	2360
ARM	20513	6512	2240
Total	41598	13212	4600

Table 3. Data set II

	cat	chown	cp	dd	ls	rm
MIPS	528	1062	1466	788	2092	1048
ARM	480	980	1411	716	2039	977
Total	1008	2042	2877	1504	4131	2025

4.2 Effectiveness of instruction embedding projection

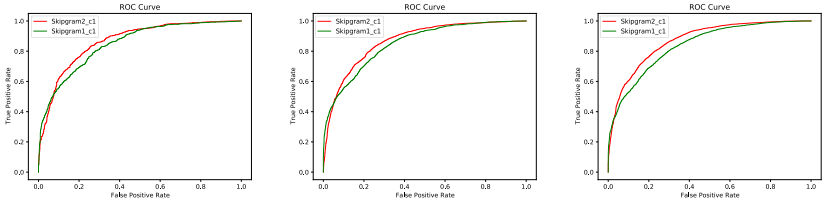
As shown in Fig. 4(a), this paper takes the 'MOV R0, R8' instruction in ARM as an example. It can be seen that due to the heterogeneity of the two architectures, only the instruction with the mnemonic 'MOV' is close to the embedding space of 'MOV R0, R8'. Although Skip-Gram does embed instructions with similar semantics in the same architecture into similar spaces. But for instructions with similar semantics under different architectures, they are not in a similar embedding space. In this regard, this paper constructs equivalent translations of MIPS and ARM instructions, and uses CCA to project instructions in different spaces into the same space. This allows instructions with similar semantics under different architectures to have similar spatial embeddings. As shown in Fig. 4(b), the similar embedding of the 'MOV R0, R8' instruction is no longer only the instruction with 'MOV' as the mnemonic in ARM, but includes the instruction with 'move' as the mnemonic in MIPS. This is in line with the expectation that similar instructions in the same architecture have similar embeddings, and instructions with similar semantics in different architectures also have similar embeddings.



(a) Non-projection embedding (b) Projection embedding

Fig. 4. Instruction embedding space.

In order to prove that the effect of instruction embedding after projection is better than that of instruction embedding without projection, this paper makes a comparison. The embedded instruction after projection is represented by Skipgram2, and the embedded instruction without projection is represented by Skipgram1. The block embedding of the two methods adopts the LSTM model, and the graph embedding adopts the aggregation algorithm of Gemini. The result is shown in Fig. 5. In the three different test sets, the model using Skipgram2 has a higher AUC value. It can be proved that the effect of instruction embedding after projection is better.



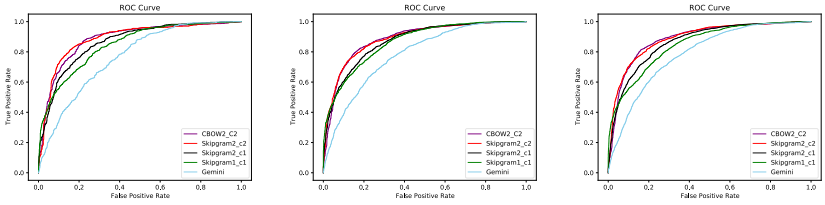
(a) ROC on cat (b) ROC on cp (c) ROC on ls

Fig. 5. Comparison of the effectiveness of instruction projection

4.3 Evaluation of graph embedding

The above has proved that the projected instructions have better performance. But instruction projection is only optimized at the level of instruction embedding. As described in Section 3.4, the graph embedding aggregation algorithm proposed by Gemini is not accurate enough to extract the graph features of ACFG. Therefore, we have improved the algorithm. We use c2 to represent the improved aggregation algorithm, and c1 to represent the original algorithm of

Gemini. At the same time, because FIT uses SkipGram and Gemini's aggregation algorithm, we use Skipgram1_c1 to represent FIT. As shown in Fig. 6, the ROC curves of CBOW2_c2 and Skipgram2_c2 are basically similar, where CBOW2 indicates that the original instruction is embedded using CBOW, and then the instruction is embedded in the re-projection. The effect of instruction embedding using CBOW and Skipgram is similar, which is also easy to understand, because the two models are the same in principle. In addition, we can see that the improved effect of the aggregation algorithm is stronger than Skipgram2_c1, which further proves that the aggregation algorithm we proposed can extract the features of the graph more effectively.



(a) ROC on cat (b) ROC on chown (c) ROC on cp

(d) ROC on ls (e) ROC on dd (f) ROC on rm

Fig. 6. Comparison of the effectiveness of instruction projection

It can be seen from the ROC curve that Gemini's performance is not good. We found that the data set used by Gemini, although the same source code has been optimized by different compilers, has different optimization levels. But most functions of the same origin have the same statistical features, which cause the statistical features to occupy a large proportion in the learning process of the graph embedding network. This will lead to a defect that the graph embedding network ignores the semantic features of instructions in the learning process. As shown in Fig. 7, Gemini will mistakenly regard basic blocks with similar statistical features but completely different semantic information as similar, resulting in a high number of false positives. In response to this problem, this paper modified the data set. We also compiled the same source code into binary codes with different optimization levels and different architectures. But we will try to select similar functions with large differences in statistical features. The similar func-

tions defined here are the same as Gemini. Different binary functions compiled from the same source function are similar functions.

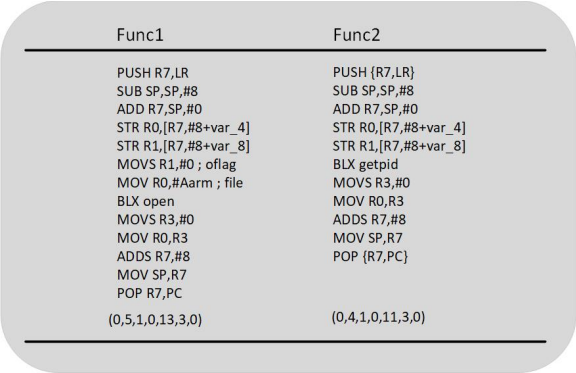
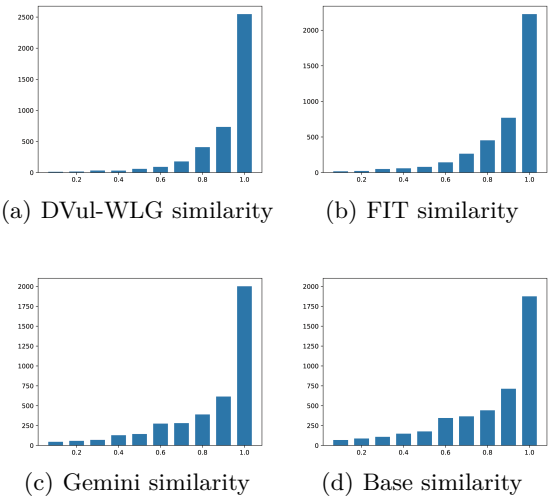


Fig. 7. An example of a binary function with similar statistical features but different semantics.

4.4 Vulnerability detection of real firmware

In this section, we will detect vulnerabilities in real firmware and compare and analyze the effectiveness of different methods. At the same time, it proves that the three features of ACFG proposed in this paper are necessary for similarity detection.As shown in Fig. 8, the four graphs are the statistics of the similarity scores of DVul-WLG, FIT, Gemini and Base. Among them, Base means that the model only uses the semantic features of instructions and the structural features of graphics during the training process, and does not use statistical features, so as to compare with other methods. We randomly selected 4000 similar sample pairs from the data set (compiled by the same original function), among which the top 80% with the highest DVul-WLG similarity score were in the interval of [0.797, 1.0]. Therefore, the threshold of DVul-WLG is selected as 0.797. For firmware functions that use DVul-WLG for similarity detection, if the similarity score is higher than 0.797, it is considered that there are vulnerabilities in the firmware function. Similarly, the threshold selection for FIT, Gemini and Base is 0.741, 0.628 and 0.584 respectively. This also reflects that the model proposed in this paper has a higher similarity score for similar function pairs.

As shown in Table 4, there are three types of vulnerabilities to detecte: (1) CVE-2020-1967 is a high-risk vulnerability in OpenSSL. This vulnerability is caused by the incorrect use of TLS and will lead to a null pointer reference. Cause the server or client to crash when calling the SSL_check_chain() function. This vulnerability mainly affect OpenSSL 1.1.1d, 1.1.1e and 1.1.1f versions. This paper selects 50 firmwares with these three versions for testing. DVul-WLG



(a) DVul-WLG similarity (b) FIT similarity

(c) Gemini similarity (d) Base similarity

Fig. 8. Similarity scores of similar samples.

successfully identified 47 (94%), FIT successfully identified 40 (80%), ,Gemini successfully identified 33 (66%) and Base successfully identified 25 (50%). (2) CVE-2020-1971 is a denial of service vulnerability in OpenSSL. The failure to properly handle the GENERAL_NAME_cmp function results in a null pointer reference, which may lead to a denial of service. The main affected versions are OpenSSL 1.1.1~1.1.1h and OpenSSL 1.0.2~1.0.2w. Among 50 selected firmwares, DVul-WLG successfully identified 42 (84%), FIT successfully identified 38 (76%), Gemini successfully identified 35 (70%), and Base successfully identified 23 (46%). (3) CVE-2017-15873 is an integer overflow vulnerability in BusyBox, which can cause write access violations. The mainly affects the version of BusyBox 1.27.2. Among 50 selected firmwares, DVul-WLG successfully identified 45 (90%), FIT successfully identified 39 (78%), Gemini successfully identified 42 (84%), and Base successfully identified 30 (60%).

The above results can prove that the model DVul-WLG proposed in this paper has higher accuracy than FIT. This is because this paper improves the accuracy of extracting the structural features of ACFG graphics through the improved GCN method, which proves that the structural features of graphics are necessary when comparing the similarity of ACFG. The accuracy of DVul-WLG and FIT is higher than that of Gemini. This is because Gemini did not consider the semantic features of instructions, which can prove that the semantic features of instructions are necessary when comparing ACFG similarities. Finally, the accuracy of Base is the lowest. This is because Base does not use the statistical features of ACFG. Therefore, statistical features are also necessary when comparing the similarity of ACFG. In summary, the three features of ACFG proposed in this paper are all necessary for the comparison of ACFG similarity.

Table 4. Real firmware vulnerability detection

CVE Number	Vulnerability	DVul-WLG	FIT	Gemini	Base
CVE-2020-1967	SSL_check_chain	47	40	33	25
CVE-2020-1971	GENERAL_NAME_cmp	42	38	35	23
CVE-2017-15873	get_next_block	45	39	42	30

5 Conclusion

This paper proposes an ACFG embedding model based on code similarity detection, which can be used for firmware vulnerability detection. This paper uses the method of instruction embedding to improve the accuracy of extracting semantic information of instructions in ACFG. At the same time, in order to better compare the similarity of instructions across architectures, this paper uses the canonical correlation analysis (CCA) method to project instructions in different spaces to the same space. Regarding the extraction of structural features of ACFG graphics, because ACFG is a directed graph, the traditional GCN method cannot be used to extract structural features. Therefore, this paper uses the improved GCN method DGCN to extract the structural features of ACFG. The model proposed in this paper can be used for actual firmware vulnerability detection and has practical significance.

References

1. Davis D B.: ISTR 2019: Internet of Things Cyber Attacks Grow More Diverse. Symatec. Blogs/Expert Perspectives, 2019, 9.
2. Chen, Daming D., et al.: Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. NDSS. Vol. 1. (2016)
3. Shoshitaishvili, Yan, et al.: Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. NDSS. Vol. 1. (2015)
4. Gauthier, François, Thierry Lavoie, and Ettore Merlo.: Uncovering access control weaknesses and flaws with security-discordant software clones. Proceedings of the 29th Annual Computer Security Applications Conference. (2013)
5. Jang, Jiyong, Abeer Agrawal, and David Brumley.: ReDeBug: finding unpatched code clones in entire os distributions. 2012 IEEE Symposium on Security and Privacy. IEEE, (2012)
6. Feng, Qian, et al.: Scalable graph-based bug search for firmware images. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. (2016)
7. Hex Rays,https://hex-rays.com/
8. Kenter, Tom, Alexey Borisov, and Maarten De Rijke.: Siamese cbow: Optimizing word embeddings for sentence representations. arXiv preprint arXiv:1606.04640 (2016)
9. Song, Yan, et al.: Directional skip-gram: Explicitly distinguishing left and right context for word embeddings. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). (2018)

10. Xu, Xiaojun, et al.: 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)
11. Tong, Zekun, et al.: Directed graph convolutional network. arXiv preprint arXiv:2004.13970 (2020)
12. Gao, Debin, Michael K. Reiter, and Dawn Song.: Binhunt: Automatically finding semantic differences in binary programs. International Conference on Information and Communications Security. Springer, Berlin, Heidelberg, (2008)
13. Ming, Jiang, Meng Pan, and Debin Gao.: iBinHunt: Binary hunting with inter-procedural control flow. International Conference on Information Security and Cryptology. Springer, Berlin, Heidelberg, (2012)
14. Bourquin, Martial, Andy King, and Edward Robbins.: Binslayer: accurate comparison of binary executables. Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop. (2013)
15. Flake, Halvar.: Structural comparison of executable objects. Detection of intrusions and malware & vulnerability assessment, GI SIG SIDAR workshop, DIMVA 2004. Gesellschaft für Informatik eV, (2004)
16. Pewny, Jannik, et al.: Cross-architecture bug search in binary executables. 2015 IEEE Symposium on Security and Privacy. IEEE, (2015)
17. Eschweiler, Sebastian, Khaled Yakdan, and Elmar Gerhards-Padilla.: discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code. NDSS. Vol. 52. (2016)
18. Goyal, Palash, and Emilio Ferrara.: Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems 151: 78-94.(2018)
19. Lin, Hong, et al.: Cvssa: Cross-architecture vulnerability search in firmware based on support vector machine and attributed control flow graph. 2017 International Conference on Dependable Systems and Their Applications (DSA). IEEE, (2017)
20. Liang, Hongliang, et al.: FIT: Inspect vulnerabilities in cross-architecture firmware by deep learning and bipartite matching. Computers & Security 99: 102032.(2020)
21. Faruqui, Manaal, and Chris Dyer.: Improving vector space word representations using multilingual correlation. Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics. (2014)