Identification of SQL Injection Security Vulnerabilities in Web applications Based on Binary Code Similarity

Jianhua Wang

Northwest Minzu University, Lanzhou, Gansu 730030, China
E-mail: 13809312825@163.com

Received 31 May 2024; Accepted 16 July 2024

Abstract

The existing SQL injection security vulnerability identification technology for Web applications has inherent flaws, which are relatively passive in defense methods, and cannot deal with increasingly changeable attack methods. In order to improve the accuracy of SQL injection security vulnerability identification of Web applications, this paper uses an improved skip-gram model to realize unsupervised learning of the embedding process, converts the information related to program functions contained in the vertices of the basic block into feature vectors to obtain the ACFG vector of the basic block, and measures the similarity of binary functions by evaluating the similarity of feature vectors. The experimental results show that the technical processing route proposed in this paper can effectively compare binary functions with different architectures and optimization levels, and use the advantages of neural networks to obtain higher accuracy and better analysis efficiency, thereby effectively improving the identification effect of SQL injection security vulnerabilities in Web applications. Therefore, it can play a certain role in the security management of subsequent Web applications.

Keywords: Binary, similarity, SQL, injection type, security breach.

Journal of Cyber Security and Mobility, Vol. 13_6, 1239–1262.
doi: 10.13052/jcsm2245-1439.1361

1 Introduction

Although web application software has been widely used in many fields, it is a relatively new type of application software compared to traditional web network applications. In addition, the current part of web application platform developers are still thinking in the traditional c/s architecture system development routines [1]. In the original c/s architecture, clients often defaulted to trusted users. Moreover, programmers who are accustomed to the traditional development mode tend to ignore the sufficient and necessary verification of the information input from the network client and filter out some user inputs that are easy to cause abnormalities in time when developing software. Therefore, such a stereotype of development thinking is easy to produce some Web vulnerabilities that can be exploited by hackers in the development of network platforms [2].

In this paper, some problems that need to be solved urgently in the current Web vulnerability detection system are deeply discussed and studied. By studying the causes of Web vulnerabilities, various attack approaches and steps against vulnerabilities and combining with the current network security forms, this paper focuses on the combination of efficiency and detection accuracy in the detection process of SQL injection vulnerabilities and the existing problems. Finally, the reasons that affect the efficiency and accuracy of the security detection system and solutions are obtained. Moreover, this paper gives a brief introduction to the background of the subject, and expounds the practical needs and feasibility of the scheme to improve and develop the vulnerability scanning efficiency and detection rate. Then, based on the study of the causes, principles, approaches and main detection technologies of Web vulnerabilities, according to the research status of the detection methods of various Web vulnerabilities including SQL injection vulnerabilities at home and abroad, this paper puts forward a new solution according to the development trend of current emerging technologies.

This paper aims to identify SQL injection security vulnerabilities of Web applications through the similarity of binary codes. Moreover, this paper uses the improved skip-gram model to realize unsupervised learning of the embedding process, converts the information related to program functions contained in the vertices of the basic block into feature vectors to obtain the ACFG vector of the basic block, and evaluates the similarity of the feature vectors. Measure the similarity of binary functions, and use the advantages of neural networks to obtain higher accuracy and better analysis efficiency, thereby effectively improving the identification effect of SQL

injection security vulnerabilities in Web applications. Therefore, it can play a role in the security management of subsequent Web applications.

2 Literature Review

In recent years, research on SQL injection vulnerability scanning has developed rapidly in China. The static analysis tool JDBC aiecker avoids attacks from SQL injection statements from attackers by verifying the string content entered by users in real-time environments [3]. However, if hackers use similar normal types and syntax to disguise attack statements during injection attacks, this method is difficult to effectively pre mine, and in addition, it requires multiple modifications to the web source code. Reference [4] adopts a dynamic debugging technique different from static analysis, constructs a testing environment based on input flow analysis and input validation analysis results, and tests user input to detect the existence of SQL injection vulnerabilities. Compared to static analysis methods, dynamic debugging techniques do not require frequent modifications to the coding structure. However, the disadvantage of this method is that it relies on known SQL injection vulnerability information, which comes from vulnerability libraries previously identified by security experts and developers. For newly emerging SQL injection attack methods that have not yet been added to the vulnerability library, they cannot be detected in advance. Based on the advantages and disadvantages of static analysis methods and dynamic debugging techniques, reference [5] combines the two and designs and implements an SQL injection vulnerability detection system that combines static and dynamic SQL queries. Reference [6] proposes an SQL injection vulnerability detection based on control flow graph comparison technology, where users can perform SQL query detection while analyzing stored procedures and runtime. Based on the study of SQL query statements running in the backend database of a site, reference [7] designed a randomized SQL query statement based on proxy mode between the backend database and the server by randomly inputting instructions. However, this method also has obvious drawbacks, that is, once the randomized selection of SQL query statements is successfully predicted by hackers, the system can no longer be used to detect special statement injection attacks targeting the backend database of web applications. Reference [8] proposes an intrusion detection method based on machine learning for SQL injection attacks. The SQL statements in the database logs will be used for machine learning, and the generated behavior pattern model

will be compared with the SQL statement to be detected. If the difference between the detection result and the predetermined pattern exceeds the predetermined threshold, the system will determine that the SQL statement has injection behavior. Reference [9] studied and analyzed the current types of SQL injection attacks and corresponding detection or protection measures, implemented SQL injection detection technology based on static analysis, and was able to mine easily hidden SQL injection points. Reference [10] proposes a self-learning SQL injection attack filtering method, which can automatically learn legitimate SQL statement structures to build its own security detection knowledge base. When detecting vulnerabilities, it determines the existence of SQL injection vulnerabilities by matching the SQL statements in the knowledge base. Reference [11] proposed a SQL injection vulnerability detection mechanism based on hidden webpage crawling, and implemented a scanning system aimed at improving webpage detection coverage and SQL injection vulnerability detection capability. A network-based vulnerability scanner implementation method was proposed, which provides a better vulnerability coverage and can reduce false positives in the short term.

At the same time, foreign researchers have also made significant contributions to SQL injection protection, such as the encoding mechanism introduced in reference [12], which can be deployed as a universal prevention method in specific practices. The lack of a reasonable, complete, and standardized verification mechanism is an important reason for the emergence of SQL injection attacks, so starting from the verification mechanism is an important aspect of implementing SQL injection attack prevention.

The vulnerability mining scheme for binary code similarity comparison only requires labeling the trigger location of the vulnerability to automatically extract vulnerability features for vulnerability detection. This technique utilizes neural networks to obtain binary code representations containing vulnerabilities and target semantic information, and compares their similarity [13]. According to the different contents of the comparison, existing methods for binary code similarity can be divided into two types: function matching and patch analysis. In patch analysis methods, MVP [14] (Matching Vulnerabilities with Patches) adopts vulnerability signature and patch signature schemes, capturing the generation and repair of vulnerabilities through patch signatures. A set of basic block trajectories within the function are signed and represented, and patch semantics are applied to design trajectory similarity to identify whether the target program has been patched. The patch

based method mainly utilizes patch update information, which requires the use of this method only when the patch is released. As software patches are not released separately, and software version updates contain a large amount of information, the usage scenarios of the above scheme are limited. In function matching methods, to alleviate the issue of instruction differences caused by comparing the same source function code in different architectures, reference [15] uses intermediate representation based on LLVM (Low Level Virtual Machine), The representation method of IR. Reference [16] establishes a mapping between function source code and function binary code based on LLVMIR, effectively alleviating the problem of cross language similarity. In the field of code defect detection, reference [17] is a method that integrates sequence information and semantic features. However, due to only considering the data dependency between basic blocks and ignoring the control dependency between basic blocks, some semantic information is lost. In order to extract semantic information more comprehensively and improve detection accuracy, reference [18] proposed a standardized unsupervised feature extraction method. This method utilizes the attention weights of Control Flow Graph (CFG) nodes, while considering both data flow relationships and control dependencies to extract semantic information, in order to more accurately capture program behavior. Reference [19] proposed the method of introducing instruction attributes to enrich function semantics, but the effectiveness of these methods is limited.

However, the above solutions all have certain limitations. The granularity of analysis based on binary functions is too coarse, and the vulnerabilities are caused by some of the code. The implementation scheme of using modifying dangerous functions as vulnerability patches involves minimal and difficult to capture modifications to the execution logic of binary functions. Based on binary function similarity calculation, it is difficult to describe data dependency constraint relationships and capture vulnerabilities triggered by unconstrained data. In addition, due to the use of different optimization levels to compile binary code with different architectures from a source code, the differences can be very large. Function oriented vulnerability similarity schemes cannot identify security vulnerabilities triggered by the lack of necessary constraint judgments between functions. Deep learning schemes based on intermediate language representation result in a high false alarm rate in vector distance based comparisons. The similarity based vulnerability detection scheme lacks a finer grained filtering mechanism, and vulnerability verification still requires a lot of manual labor.

3 Conceual Framework

This paper mainly solves the problem of vulnerability mining in open-source programs, and is also applicable to programs compiled across architecture platforms. It implements a universally applicable and efficient vulnerability mining tool for binary programs. This tool, combined with anti obfuscation techniques, can bypass some defense measures of the program and restore the true control flowchart of the program by removing control flow flattening techniques, while also reducing path pressure for subsequent symbol execution; Combined with binary program function similarity detection based on neural networks, function similarity detection can detect high-risk vulnerability functions contained in the program, and then implement targeted symbol execution vulnerability mining in the subsequent mining process, to some extent avoiding path explosion and improving the effective coverage of test samples.

Referring to the technical theory of NLP(Natural Language Processing), it solves the difficulty of binary code similarity analysis. The specific application methods are as follows: the instructions of the basic block of the program are regarded as phrases in text processing, and the instructions of the basic block are converted into vector values with function characteristics by using the vector embedding model. Based on the long-short temporal memory network (LSTM), a bi-directional LSTM (BiLSTM) model is introduced to add additional basic block semantic information vectors to the basic block function eigenvectors, forming a basic block embedding vector (ACFG) with basic block attributes. Finally, a set of basic block embedding vectors are formed, it is still usable for programs with different architectures. To sum up, the model used to generate the basic block semantic embedding vector is shown in Figure 1, which mainly includes two parts, one is the instruction embedding model, and the other is the basic block embedding model.

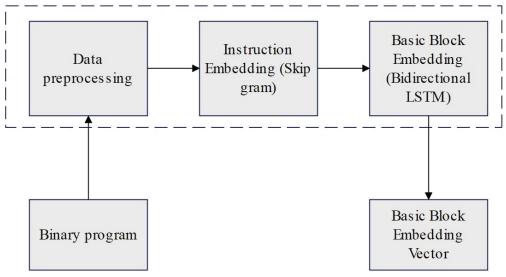

Figure 1 Generation of basic block semantic embedding vector.

Embedded models are a type of machine learning model widely used in fields such as natural language processing (NLP) and computer vision (CV). Its main function is to transform high-dimensional data into low dimensional embedding space, while retaining the features and semantic information of the original data, thereby improving the efficiency and accuracy of the model; Basic block embedding is a fundamental work in machine learning based binary program analysis methods.

3.1 Optimization of Improved Skip-gram Model

Each instruction of the basic block is constructed in the form of h according to a given instruction w_i, where $m+1 \leq i \leq N-m$. For example, if we set the size of the sliding window to 3, then when training, using the sliding window mode on the basic block instruction will overwrite the first 3 and last 3 instructions of the current instruction. The embedding layer of the network is composed of a $V \times D$ matrix, and these two parameters together determine the size of the final eigenvector. Among them, V is the standard of the size of the basic block instruction set, and D is the dimension of the basic block eigenvector. There is a softmax function between the two, the weight is $D \times V$, and the offset dimension is determined by V. The model starts from each randomly selected word vector and is trained while traversing the sliding window. The embedded vector of the middle vector w_i in the current sliding window is calculated by the softmax function. The specific method can refer to Equation (1).

$$g(w_{i+m}, w_i) = \frac{\exp(\delta_{w_{i+m}}^T v_{w_i})}{\sum_{w_{i+m} \in V} \exp(\delta_{w_{i+m}}^T v_{w_i})} \tag{1}$$

The function g also represents the correlation of the position relationship between w_{i+m} and w_i, which can ensure that the direction of the vector δ corresponding to w_{i+m} is opposite to that of wi, so as to ensure that every instruction of the basic block can be completely encoded without omission. Among them, the function g is updated according to rules like this:

$$v_{w_i}^{(new)} = v_{w_i}^{(old)} - \gamma(\sigma(v_{w_i}^T \delta_{w_{i+m}}) - D)\delta_{w_{i+m}} \tag{2}$$

$$v_{w_{i+m}}^{(new)} = v_{w_{i+m}}^{(old)} - \gamma(\sigma(v_{w_i}^T \delta_{w_{i+m}}) - D)\delta_{w_i} \tag{3}$$

In the function, σ is the *sigmoid* function, $\sigma(x) = \frac{1}{(1+\exp(-x))}$, γ represents the decay index of the learning rate, and D represents the label of w_{i+m}

under the premise of w_i, where:

$$D\begin{cases}1; & (i<0)\\ 0; & (i>0)\end{cases} \tag{4}$$

The loss function of the skip-gram model itself is shown in Equation (5):

$$Loss=\frac{1}{|V|}\sum_{i=0}^{|V|}\sum_{0<|m|\le c}\log(p(w_{i+m})w_i) \tag{5}$$

Among them,

$$p(w_{i+m}|w_i)=\frac{\exp(v'^{T}_{w_{i+m}}v_{wi})}{\sum_{w_{i+m}\in V}\exp(v'^{T}_{w_{i+m}}v_{wi})} \tag{6}$$

The loss function of the improved skip-gram model is to combine the two, and calculate the loss according to the loss function formula shown in the following Equation (7):

$$Loss=\frac{1}{|V|}\sum_{i=0}^{|V|}\sum_{0<|m|\le c}\log p(w_{i+m}|w_i)+g(w_{i+m},w_i) \tag{7}$$

Therefore, an instruction embedding model based on skip-gram can be obtained in the end. The model structure is shown in Figure 2.

The model establishes a connection between intermediate vocabulary and the entire text by controlling the size of the sliding window, but this approach increases overhead and significantly increases training time. Due to the fact that this chapter is designed to implement instruction embedding in the program, only the semantic information and contextual connections of the basic blocks need to be considered. Therefore, it is necessary to improve the model. By establishing semantic connections between intermediate vocabulary and context in the model, predicting intermediate vocabulary, context, and key words in the text through intermediate vocabulary, we can increase the contextual connections of basic blocks and utilize semantic information with less training costs.

Each instruction will be embedded in the $V\times D$ matrix E, and each instruction will be embedded in the n-th instruction w_n in the vocabulary, whose instruction embedding e_n is calculated by:

$$e_n=u_nE \tag{8}$$

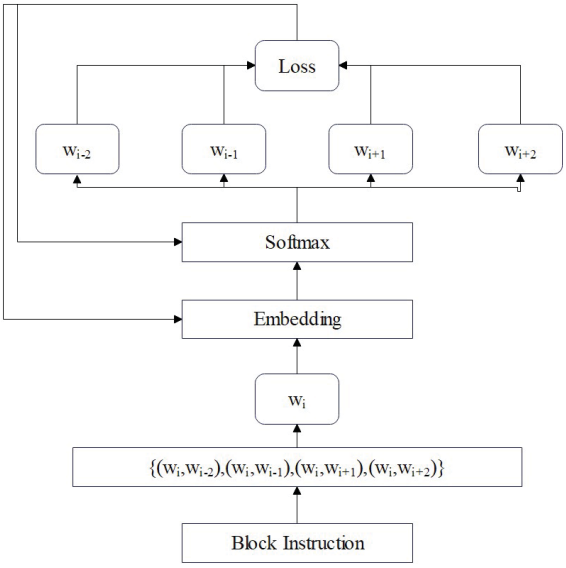

Figure 2 Schematic diagram of skip-gram model.

u_n is a vector of $1 \times V$, and the n-th element is 1, while the rest represent 0. Through the skip-gram model built above, the instruction embedding of the basic blocks can be realized. For example, when a basic block block of a function is given, this block is composed of a list of instructions, and the instruction stream of block is denoted I (b):

$$I(b) = [b_1, \ldots, b_n] \tag{9}$$

Among them, b_n represents an instruction in block. Similarly, different functions can be parsed into corresponding basic block instruction streams according to the same logical thinking, and then these instruction streams can be input into the model for training, and the final result is the instruction embedding matrix. Of course, the embedding dimension of the matrix can be manually adjusted according to needs, which will be mentioned in subsequent experiments.

Before finally generating the instruction embedding vector of the basic block, in order to improve the overall performance of the model, this paper introduces a layer of attention mechanism self-attention network, which can effectively transmit information and effectively reduce redundant embedding vectors. The specific principle of the self-attention mechanism is shown in Figure 3 below, where FC represents the fully connected network layer.

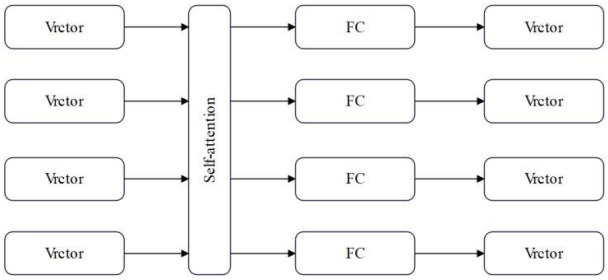

Figure 3 Schematic diagram of the self-attention mechanism.

Attention mechanism is an important concept in computer models, which refers to the ability of the model to focus on certain important parts and ignore other irrelevant parts when processing input data. In computer data processing tasks, attention mechanisms can help models better understand input data, thereby improving model performance.

The self-attention mechanism consists of $Query = Q$, $Key = K$, and $Value = V$, where the vector represented by K and V corresponds one-to-one. K and V represent the vectors generated by the same instruction. The weight value α_t is obtained through the operation of Q and K, and the important features in V are selected to complete the feature vector fusion. Generally, the following calculation formula is selected:

$$\alpha_t = softmax(sim_t(Q, K_t)) \tag{10}$$

After getting the weight value α_t, we can get the value of attention by weighting and summing it with the value represented by V:

$$attention(Q, K, V) = \sum_t \alpha_t V_t \tag{11}$$

The standard attention mechanism is to assign different weights to the input information of the unit, so that the model pays more attention to the key information, while those low-weight information will be filtered out. For the instruction embedding of basic blocks, not only the key instruction information related to the program function is required, but also the acquisition of control flow information without obvious special marks should be given similar attention weight allocation. Therefore, this model needs to adjust the weight distribution of the attention mechanism and modify the algorithm. The schematic diagram of the instruction embedding model after fusion of the attention mechanism is shown in Figure 4. The attention network "screens"

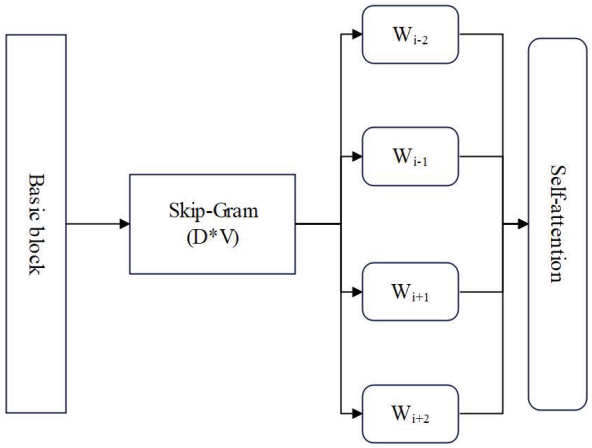

Figure 4 Schematic diagram of instruction embedding model.

the instruction embedding vectors generated by the improved skip-gram model to remove redundant feature vectors.

Combined with the characteristics of the instruction vector of the basic block of the function, the calculation formula of the attention mechanism is finally determined as follows:

$$e_t = u \tanh(W_\alpha m_t + b) \tag{12}$$

$$f(m_t, m_s) = W_\alpha [m_t; m_s] \tag{13}$$

$$\alpha_t = \frac{\exp(f(m_t, m_s))}{\sum_j \exp(f(m_t, m_s))} \tag{14}$$

$$s_t = \sum_{t=1}^{n} e_t \alpha_t \tag{15}$$

m_t represents Q mentioned above, m_s represents that K and b are bias, e_t represents the attention distribution determined by the input vector at moment t, and s_t is the eigenvector of the final output.

3.2 ACFG Vector Similarity Calculation Model Based on Attention Mechanism

By converting a binary program into a control flow graph (ACFG) with basic block attributes, the characteristic information of each node is extracted, so

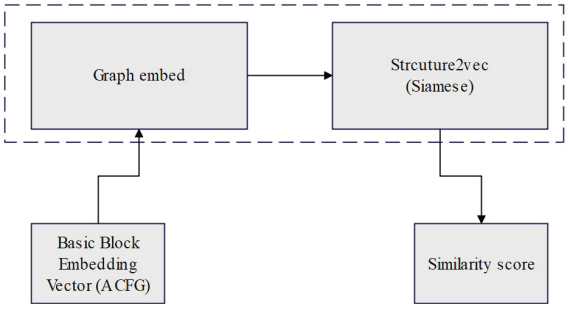

Figure 5 ACFG vector similarity calculation model based on attention mechanism.

that the program can not only obtain the code, but also obtain the relationship between them. Using the struc2vec algorithm, the ACFG vector is transformed into a more complex graph, which better represents the vector characteristics of the basic blocks of each function, as shown in Figure 5.

The vertex eigenvectors of all the basic blocks are fused to finally generate the embedded vector μ_g of the ACFG:

$$\mu_g' = M_{v \in V}(\mu_v) \tag{16}$$

Based on the struc2vec algorithm, this paper constructs the following ACFG embedding vector generation method, and its basic block node vector mapping relationship is as follows:

$$G\left(x_v, \sum_{u \in E(v)} \mu_u\right) = \tanh\left(W_1 x_v + \sigma\left(\sum_{u \in E(v)} \mu_u\right)\right) \tag{17}$$

Among them, x_v is the d-dimensional vector, represents the characteristics of ACFG nodes, W_1 is the parameter matrix of $d \times p$, and p is the embedding dimension of ACFG. The function σ is a fully connected layer of *n* layers:

$$\sigma(l) = P_1 \times Relu(P_2 \times \ldots Relu(P_n l)) \tag{18}$$

Among them, $P_i(1, 2, \ldots, n)$ is the $p \times p$ parameter matrix, n is the ACFG embedded depth, $Relu$ is the linear rectifier unit, and meet the condition $Relu(x) = \max(0, x)$.

If the set of neighbor base block vertices of the base block vertices v in a graph g is represented as $E(v)$, the first step in building a struc2vec algorithm is to set the eigenvectors of all base block vertices to 0, and in

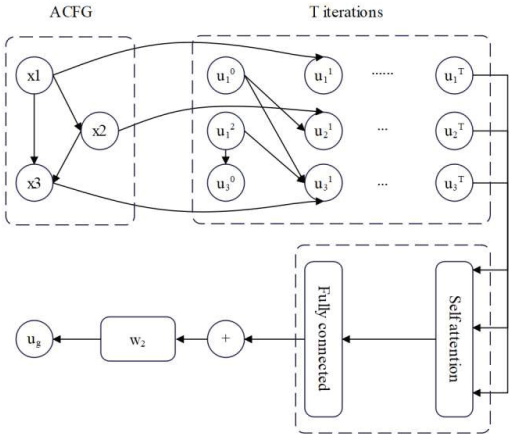

Figure 6 ACFG vector embedding model diagram.

the next iteration process, this step is repeated continuously using the same strategy.

$$u_v^{(t+1)} = G(x_v, \sum_{u \in E(v)} \mu_u^{(t)}), \forall v \in V \tag{19}$$

Among them, x_v is the characteristic attribute of the vertex v of the basic block, V is the set of vertices of the graph g, $\mu_u^{(t)}$ is the embedding of the eigenvector of vertex μ at time t, $u_v^{(t+1)}$ is the embedding of the eigenvector of vertex v at time $t+1$, and G is a vector mapping function.

The struc2vec model used is written using the RNN network, so the attention mechanism in the RNN network is considered. The purpose of the attention network is to further fuse the feature vectors generated by the previous network element, further reduce the redundancy in the feature vectors, reduce the calculation overhead, and obtain the final feature vector u_g. The final ACFG vector embedding model structure is shown in Figure 6, and w_2 is a $p \times p$ dimension matrix with the same embedding dimension.

In view of struc2vec algorithm is mostly used for classification and not for calculating similarity, the network is designed as a Siamese structure, and the similarity of the left and right parts of the vector is calculated through the way of parameter sharing, so as to obtain the function similarity score of the program. The Siamese architecture model is shown in Figure 7.

The Siamese network structures have exactly the same characteristics, and they all contain the same parameters, which allow them to coordinate with each other.

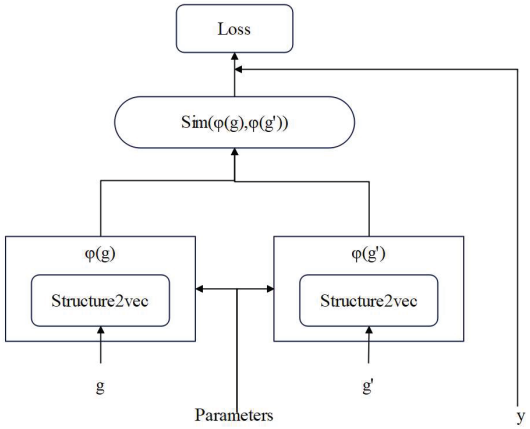

Figure 7 Siamese architecture model diagram.

We are given a dataset containing N ACFG pairs $< g_i, g_i' >$ and their similarity information $y_i \in \{+1, -1\}$, where $y_i = +1$ means that two basic blocks have similar functions, and conversely $y_i = -1$ means that these two basic blocks have no similar functions. Then, the Siamese network will calculate each ACFG according to the following algorithm Theoretical similarity between pairs:

$$Sim(g, g') = \cos(\phi(g), \phi(g')) \tag{20}$$

Among them, n represents the dimension of the embedded vector and $\phi(g)$ is the ACFG embedded vector.

In order to optimize the parameters, the embedded model updates the parameters through back-propagation and stochastic gradient descent techniques, so that the cross-entropy loss function reaches a minimum value. The calculation process is as follows:

$$Loss = \sum_{i=1}^{N} (Sim(g, g') - \pi Sim(g, g'))^2 \tag{21}$$

4 Experimental Results and Analysis

4.1 Experimental Methods

The graph embedding model introduced in this paper is built by Python language on the Keras platform. Keras relies on TensorFlow as the backend.

Hardware configuration: Intel ® Core^{TM} i5-10400 CPU @ 2.90 GHz 2.90 GHz processor and running memory of 16GB, and Operating environment: Ubuntu 20.04 64bit.

Manual analysis of binary programs typically requires a combination of static and dynamic analysis. For binary programs with source code, first identify the vulnerability points in the program through source code analysis, and then confirm the program vulnerabilities and exploitation methods through dynamic debugging. For programs without source code, reverse programming is usually required, which involves analyzing assembly code or pseudocode, observing software behavior through dynamic debugging, and confirming analysis based on the personal experience of the analyst.

The source of the first dataset of this paper: combining the data of the second part with the help of crawlers on the network, a complete dataset can be built to effectively train, verify and test the model.

The basic principle of a web crawler is based on the website network protocol, and the process of obtaining information from web pages in bulk according to the website address. Simply put, it refers to using computer programs to simulate the process of manually clicking on web pages to obtain data.

1. Firstly, select a carefully selected subset of seed URLs (uniform resource locators);
2. Put these URLs into the URL queue to be crawled;
3. Retrieve the URL to be crawled from the queue of URL to be crawled, resolve DNS, and obtain the host's IP. Download the corresponding webpage from the URL and store it in the downloaded webpage library. In addition, put these URLs into the crawled URL queue;
4. Analyze the URLs in the crawled URL queue, analyze other URLs within it, and place the URLs in the URL queue to be crawled, thus entering the next loop.

The source of the second dataset: the dataset corresponding to the original model that has not been processed by the technical route of this paper. Among them, the data set of the original model is constructed in the following way: some functions of OpenSSL (v1.0. 1a and v1.0. 1f) and Linux packages are compiled to implement the compilation of MIPS and ARM architectures, and gcc and clang compilers are used to ensure the accuracy and reliability of compilation. Three optimization levels of O1-O3 are adopted respectively, and the data is further divided into three parts, as shown in Tables 1 and 2.

Table 1 Dataset 1

	Training set			Validation set			Test connection			TOTAL		
O1	33814	31913	65727	2999	3613	6611	4324	4302	8626	41115	39827	80964
O2	44115	42330	86446	4963	5018	9981	5017	5534	10551	54096	52883	106978
O3	45677	44334	90011	5336	5325	10661	5940	5840	11780	56953	55499	112452
TOTAL	123606	118577	242184	13298	13956	27254	15282	15676	30957	152163	148209	300395

Table 2 Dataset 2

	Training set			Validation set			Test connection			TOTAL		
O1	10362	10246	20608	1519	1719	3237	1422	1623	3044	13303	13587	26889
O2	9977	10178	20155	1369	1312	2681	1024	1421	2444	12370	12911	25281
O3	10331	10251	20582	1905	1128	3032	1932	1191	3123	14168	12570	26738
TOTAL	30670	30675	61345	4793	4158	8951	4378	4234	8612	39841	39067	78908

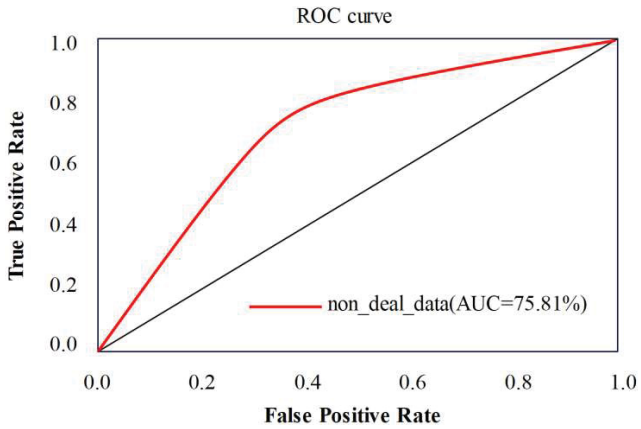

Figure 8 ROC curve of ACFG embedding model without data processing.

The extracted-ACFG tool is formed by using Angr framework to extract ACFG automatically. This tool can perform code loading on different binary functions and control flow graph extraction to generate basic block embedding vectors.

4.2 Results

First, we set the epoch of model training to 100, and then use the verification set to test the performance of the model, and save the model corresponding to the largest AUC value as the basic model. Then, we test the accuracy of the model with the test set data. Finally, the unprocessed data set is used as input for comparison, and the results are shown in Figures 8 and 9.

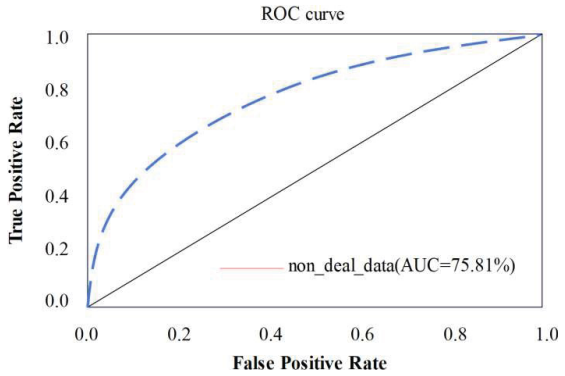

Figure 9 ROC curve of data-processed ACFG embedding model.

Table 3 Changes of TPR and FPR under different threshold

FPR(%)	TPR(%)	Threshold
100.00	100.00	0.00
57.42	97.02	0.20
44.55	95.04	0.40
41.58	92.07	0.59
39.60	83.16	0.79
17.82	17.82	1.00

True Positive Rate (TPR) is the number of true positive samples detected divided by the number of all true positive samples;False Positive Rate (FPR) is the number of false positive samples detected divided by the number of all true negative samples.

As shown in Table 3, the true and false positive rates of the model at different thresholds are recorded.

For the epoch number of the model, this paper sets different epoch numbers, and observes the change of the AUC value during the training process, and then stops the training when the AUC value tends to be stable, as shown in Figure 10.

According to the vertex feature dimension of the model, the features of ACFG structure can be effectively constructed by the basic block embedding technology. In order to study the impact of different basic block feature dimension embedding on the performance of ACFG embedding model, the basic block embedding is adjusted to 4, 8, 10, 12 and 14 dimensions in sequence in the experiment, and their AUC values are recorded. The test results are shown in Figure 11.

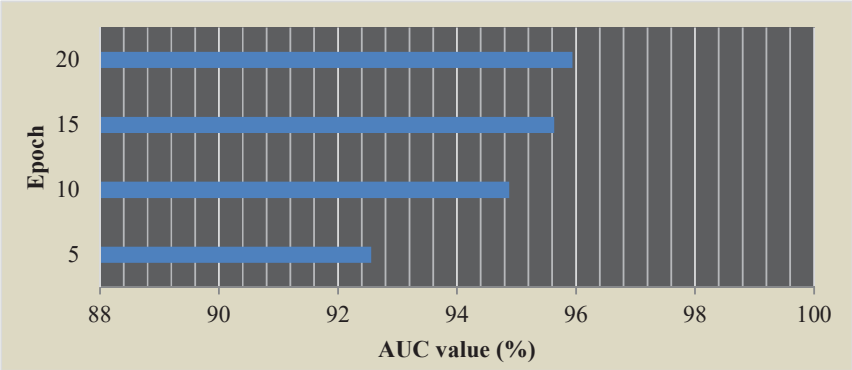

Figure 10 AUC of the model at different Epoch numbers.

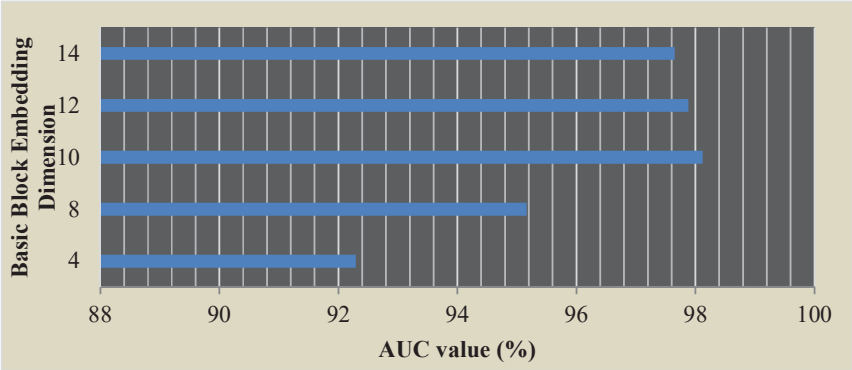

Figure 11 AUC of the model under different basic block embedding dimensions.

For the ACFG embedding dimension of the model, the ACFG embedding dimension is adjusted according to 20, 40, 60, 80 and 100 dimensions in turn, and the AUC value of each model is recorded. According to the experimental data, the model AUC under different ACFG embedding dimensions is shown in Figure 12.

The embedding depth of ACFG can significantly improve the performance of the model. The embedding depth of the ACFG can be reflected by calculating the number of network layers n for s. In the simulation process, ACFG needs to be embedded into different layers, including layers 1, 2, 3, and 4. The AUC values for each level are recorded, and this information is shown in Figure 13.

In order to reflect the effectiveness of the model and to see that the ACFG embedding vector satisfies the condition that the eigenvectors compiled from

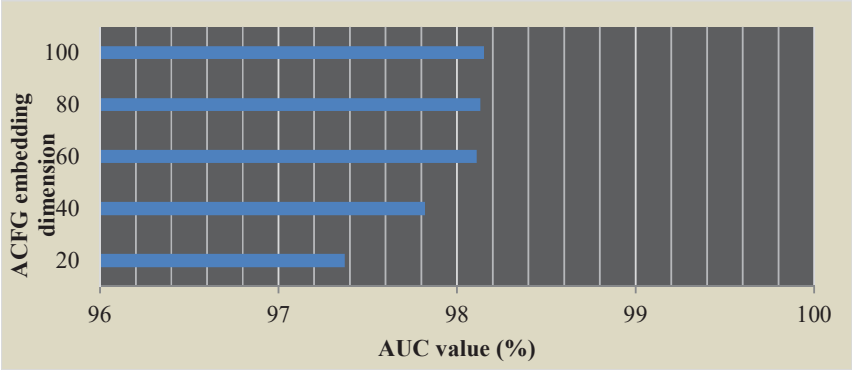

Figure 12 AUC of the model under different ACFG embedding dimensions.

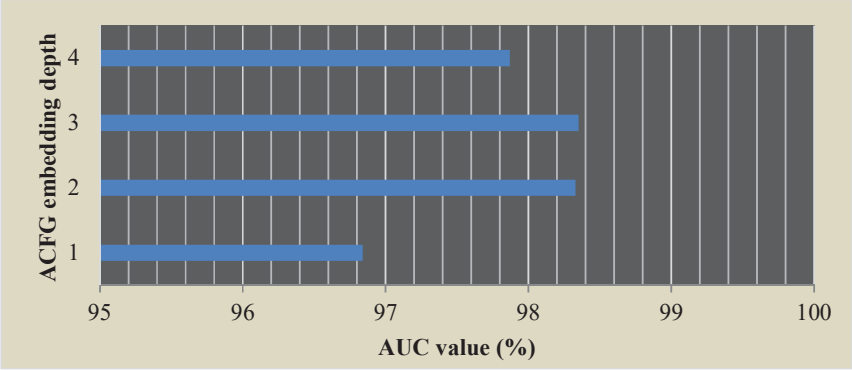

Figure 13 AUC of the model at different ACFG embedding depths.

the same source functions belong to similar block pairs, and the vectors compiled from different source functions belong to dissimilar block pairs, this paper conducts model tests on 6 source functions (alloc _ procps _ scan, cat _ main, fgetgrent, find _ mount _ point, get _ linux _ version _ code, modprobe _ main) from the vulnerability sample library, and then the high-dimensional embedding vectors generated by the model are visualized using t-SNE. The results are shown in Figure 14.

In order to test the performance advantages of this model, this paper uses other similar models to conduct comparative experiments, and records the AUC values of the model respectively, so as to verify the accuracy of the model for data processing. This section of the experiment compares the performance of the three methods of reference [10], reference [11] and the

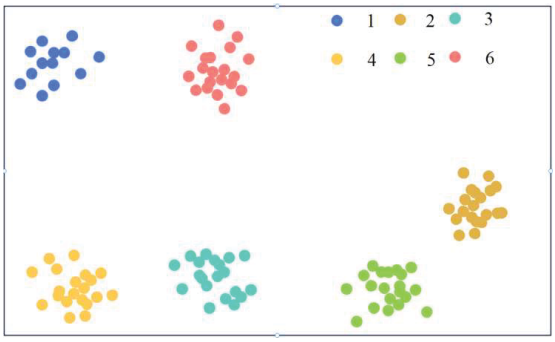

Figure 14 Visualization results of high-dimensional embedded vectors.

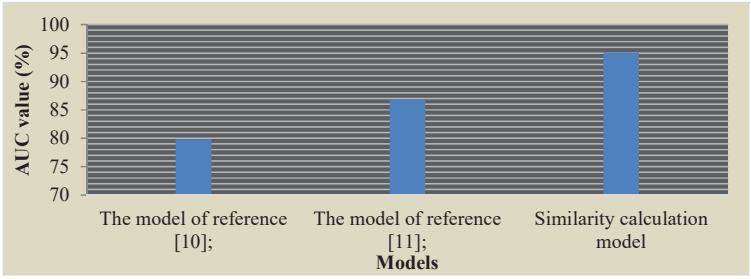

Figure 15 AUC of different models.

model proposed in this paper on the verification set, and records the AUC value on the verification set, as shown in Figure 15.

4.3 Analysis and Discussion

As shown in Figures 7 and 8, the red curve is the ROC curve of the data set without corresponding processing on the model. It can be concluded that the data processing technology proposed in this paper has a higher AUC value, which can make the model show better performance. Among them, a low threshold will lead to a higher positive case rate in the model, and a higher threshold will lead to a higher false positive case rate in the model.

It can be seen from the results in Table 3 that with the increase of the threshold value of the model, the TPR and FPR of the model will continue to decline, but the decline rate of FPR is faster than that of TPR. When the model is at a high threshold, TPR is at a high level and FPR is at a low level. Therefore, it can be concluded that the model in this paper has good performance.

As shown in Figure 9, increasing the epoch number will increase the value of AUC, but the value of AUC is not only determined by the epoch number. However, it can still be observed that the pre-processing scheme of the data set proposed by this technology can make the model complete training quickly and achieve a relatively good performance state. Finally, according to the training record, the number of Epoch is determined to be 10.

As shown in Figure 10, increasing the feature dimension of vertices will introduce more low-impact factors, which will actually reduce the value of AUC. Therefore, the best basic block embedding dimension of this model is 10 dimensions.

Figure 11 shows the relevant conclusion: increasing the ACFG dimension will increase the value of AUC, but the value of AUC is not only determined by the ACFG embedding dimension, and excessive increase will introduce more low-impact factors. Therefore, the embedding dimension suitable for this model is 60 dimensions.

As shown in Figure 12, it has been experimentally proved that when the model has 2 layers and 3 layers, the AUC value of the ACFG embedded model is the highest, and as the number of layers increases, while the AUC value of the embedded model decreases. According to the experimental results, the optimal number of ACFG embedding layers is 2 layers, which can meet the requirements of the model.

According to the distribution between the vectors in Figure 13, the distance between the functions can be seen, which proves that the semantic information of the functions is effectively preserved, while also eliminating the differences caused by different architectures and compilation optimization options. From Figure 14, it can be seen that the performance of the model proposed in this paper is higher than that of the model proposed in reference [10], and its performance advantage is more obvious than that of the unsupervised learning model proposed in reference [11].

Finally, through experiments, it is proved that the technical processing route proposed in this paper can effectively compare binary functions with different architectures and optimization levels, and use the advantages of neural networks to obtain higher accuracy and better analysis efficiency.

5 Conclusion

For SQL injection security vulnerability identification for Web applications, this paper discusses how to map high-risk functions to other functions to calculate the similarity between them. Moreover, through in-depth study of

graph embedding problem and struc2vec algorithm, this paper combines basic block embedding technology to convert the information related to program functions contained in the vertices of basic blocks into eigenvectors, and then obtain the ACFG vector of basic blocks. In addition, this paper aggregates the ACFG embedding vectors of each vertex to form the feature vectors of the program functions represented by the basic blocks of the function, and measures the similarity of binary functions by evaluating the similarity of the feature vectors. Finally, through experiments, it is proved that the technical processing route proposed in this paper can effectively compare binary functions with different architectures and optimization levels, and use the advantages of neural networks to obtain higher accuracy and better analysis efficiency.

However, although the vulnerability function sample library in this paper involves the mainstream architectures ARM and MIPS, the specific number of vulnerability functions involved is still not enough. Therefore, in order to better reflect the detection and mining capabilities of the tool, the type and quantity of the sample library can continue to be expanded in the future. The method of removing control flow flattening in this article to achieve anti obfuscation will be limited when faced with other obfuscation measures. Other anti obfuscation algorithms can be added to the anti obfuscation algorithm, such as the data flow anti obfuscation algorithm, to resist program obfuscation of the data flow.

References

[1] Humayun, Mamoona, Mahmood Niazi, N. Z. Jhanjhi, Mohammad Alshayeb, and Sajjad Mahmood. “Cyber security threats and vulnerabilities: a systematic map study.” *Arabian Journal for Science and Engineering* 45 (2020): 3171–3189.

[2] Jimmy, F. N. U. (2024). Cyber security Vulnerabilities and Remediation Through Cloud Security Tools. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 129–171.

[3] Jiang, X., Lora, M., and Chattopadhyay, S. (2020). An experimental analysis of security vulnerabilities in industrial IoT devices. *ACM Transactions on Internet Technology (TOIT)*, *20*(2), 1–24.

[4] Yu, M., Zhuge, J., Cao, M., Shi, Z., and Jiang, L. (2020). A survey of security vulnerability analysis, discovery, detection, and mitigation on IoT devices. *Future Internet*, *12*(2), 27–36.

[5] Yaacoub, J. P. A., Noura, H. N., Salman, O., and Chehab, A. (2022). Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations. *International Journal of Information Security*, *21*(1), 115–158.

[6] Ponta, S. E., Plate, H., and Sabetta, A. (2020). Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering*, *25*(5), 3175–3215.

[7] Kim, D. W., Choi, J. Y., and Han, K. H. (2020). Risk management-based security evaluation model for telemedicine systems. *BMC medical informatics and decision making*, *20(1)*, 1–14.

[8] Tawalbeh, L. A., Muheidat, F., Tawalbeh, M., and Quwaider, M. (2020). IoT Privacy and security: Challenges and solutions. *Applied Sciences*, *10*(12), 4102–4112.

[9] Vyas, B. (2023). Security Challenges and Solutions in Java Application Development. *Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal*, *12*(2), 268–275.

[10] Somasundaram, R., and Thirugnanam, M. (2021). Review of security challenges in healthcare internet of things. *Wireless Networks*, *27*(8), 5503–5509.

[11] Khan, M., and Ghafoor, L. (2024). Adversarial Machine Learning in the Context of Network Security: Challenges and Solutions. *Journal of Computational Intelligence and Robotics*, *4*(1), 51–63.

[12] Mishra, N., and Pandya, S. (2021). Internet of things applications, security challenges, attacks, intrusion detection, and future visions: A systematic review. *IEEE Access*, *9(1)*, 59353–59377.

[13] Harbi, Y., Aliouat, Z., Refoufi, A., and Harous, S. (2021). Recent security trends in internet of things: A comprehensive survey. *IEEE Access*, *9(1)*, 113292–113314.

[14] Mughal, A. A. (2022). Building and Securing the Modern Security Operations Center (SOC). *International Journal of Business Intelligence and Big Data Analytics*, *5*(1), 1–15.

[15] Rawat, R., Chakrawarti, R. K., Vyas, P., Gonzáles, J. L. A., Sikarwar, R., and Bhardwaj, R. (2023). Intelligent fog computing surveillance system for crime and vulnerability identification and tracing. *International Journal of Information Security and Privacy (IJISP)*, *17*(1), 1–25.

[16] Alfadel, M., Costa, D. E., and Shihab, E. (2023). Empirical analysis of security vulnerabilities in python packages. *Empirical Software Engineering*, *28*(3), 59–70.

[17] Tabrizchi, H., and Kuchaki Rafsanjani, M. (2020). A survey on security challenges in cloud computing: issues, threats, and solutions. *The journal of supercomputing*, *76*(12), 9493–9532.
[18] Mrabet, H., Belguith, S., Alhomoud, A., and Jemai, A. (2020). A survey of IoT security based on a layered architecture of sensing and data analysis. *Sensors*, *20*(13), 3625–3637.
[19] Ranaweera, P., Jurcut, A., and Liyanage, M. (2021). MEC-enabled 5G use cases: a survey on security vulnerabilities and countermeasures. *ACM Computing Surveys (CSUR)*, *54*(9), 1–37.

Biography

Jianhua Wang, date of birth: July 12, 1977, male, Han nationality. Native place: Langfang City, Hebei Province, Master's degree, lecturer, Research interests: Information and signal processing.