Binary semantic Similarity Comparison based on software gene

Wenjie Sun¹, Yihang Chen¹, Zheng Shan¹*, Fudong liu¹*, Qiuna Wang²

¹ State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China
² Materials branch of State Grid Henan Electric Power Company, Zhengzhou, China
*Corresponding author’s e-mail: zzzhengming@163.com, lwfydy@126.com

Abstract. The similarity of binary code is widely used in code copyright protection, vulnerability mining, malicious code analysis and etc. In this paper, we proposed a method for measuring/evaluating the similarity of two binary files based on software genes. Some of Natural language processing methods were introduced into program semantics analysis, including word2vec and doc2vec models to generate assembly instruction embeddings and gene semantic embeddings. Then the longest common subsequence method was applied to evaluate the software similarity. Experiments show that our method can effectively evaluate the similarity of binary files.

1. Introduction

The analysis of binary files without source code is widely used in the fields of software copyright protection, vulnerability mining, malware detection and etc. Due to the reuse and sharing of source code, open source projects provide convenience and cause a great hidden danger to software security issues[1]. Black Duck, in their Open Source Software Security and Threat Analysis Report[2], pointed out that more than 67% of commercial software using open source components exist vulnerabilities as they known. Therefore, evaluating the local similarity of binary files, analysing their mutual inclusions, code reuse and other issues, has important significance in the field of software security analysis.

This paper presents a method for measuring semantic similarity of binary files based on software genes. On the basis of the software gene proposed by Chen et al[3], this paper uses word2vec and PV-DM model to understand the semantics of software genes, and the gene semantic embedding model is also obtained. Furthermore, the semantic similarity evaluation of different genes for Euclidean distance is used to obtain the semantic equivalence genes, and the software common similarity score is obtained by using the largest common subsequence of the two semantic equivalence genes. Experiments have shown that our method has achieved good results. The main contributions of this paper are as follows:

- Natural language processing methods are used to analyse program semantics. Word2vec and PV-DM algorithms are introduced to realize the understanding of assembly instruction embedding and software gene semantic embedding.
- Software similarity measurements based on software gene slicing method has been improved and has achieved good results in the experiments.
2. Related work

2.1. Software gene
The software gene is inspired by biological genes, and draws on the research and analysis software of genetic analysis in bioinformatics to solve the analytical problems in the field of software engineering. Inspired the research ideas of bioinformatics, Chen et al.[3] proposed the concept of software genes, and improved the clustering algorithm of biological system development model to measure the difference between genes, which was applied to the detection of malware with an accuracy 96.14%. Meng et al.[4] proposed that the software genes are code fragments carrying functional information in the program, and the software genes are applied to the malware code classification. The MCSMGS system was designed to classify malware families by using malware gene sequences, with an accuracy of 98%. At present, the related research on software genes mainly focuses on the malware detection, malware family classification, but does not extend to the analysis of common software.

2.2. Binary similarity
There are many different ideas and methods for clone detection or similarity measurement of software, including string matching-based detection methods, token-based matching methods, tree-based methods and so on. Some methods from the perspective of source code can only solve the similarity comparison of software with source code, but cannot handle the comparison of binary files, which is not practical in practice. Some researchers use symbolic execution and theorem prover to compare the semantic similarities of binary code, such as BinHunt[5] and CoP[6], but their methods are not suitable for large scale code bases because of their high space-time complexity, large computational resources and time-consuming.

Recent researches have demonstrated the usefulness of applying machine learning and deep learning techniques to code analysis [7]. Zheng et al. [8] independently proposed the use of word embedding to represent instructions, but they only use the word embedding model to understand assembly instructions, thus losing the overall semantic information of the program.

3. Methodology

![Fig. 1. Architecture](image)

The structure of our system is shown in Figure 1. First, we extract the software gene from the input binary files. Then the gene is embedded into vectors by the word2vec algorithm. After that, we use the sentence vector model PV-DM to extract the semantic information of the software gene, which can convert the software gene into a vector contains semantic information. After the gene semantic embedding, we use the Euclidean distance to compare the gene semantics, and further find the largest common subsequence of the semantic equivalent gene, so as to obtain the software similarity score. This section is divided into four parts, and introduces our proposed gene-based software semantic similarity comparison method. Each part introduces the definition and extraction methods of software genes, the semantic similarity analysis of software genes, the largest common subsequence of semantic equivalent genes, and the software local similarity comparison methods.
3.1. software gene
Inspired by biological ideas, Chen and other researchers proposed the concept of software genes, and applied this method to malware detection, achieving an accuracy rate of 96.14%. In Chen's method, the software gene is defined as a binary fragment that satisfies the following conditions:

- a. The end of this code fragment must be a conditional jump instruction, a return instruction, or a call instruction of a user-defined function;
- b. there is not any instruction in the contexts to merge with the software gene to form a larger gene.

According to their definition, the software gene allows the inclusion of the system API call, and any instruction in the software gene is executed, all of instructions in the entire gene will be executed. There is no case where the gene is only executed part and the other part is not executed. Therefore, the software gene can reflect the atomicity and indivisibility of software functions. It can be regarded as the smallest functional unit of software analysis, which can best characterize the behaviour and semantics of programs.

We refer to Chen's algorithm for software gene extraction for binary files to be compared.

![Gene extraction process](image)

Fig 2. Software gene extraction process

3.2. gene semantic similarity
The software gene consists of several assembly instructions, which are arranged and combined in a certain logic and order to realize the overall function of the gene, which is similar to natural language. for natural language, a sentence consists of several words or phrases, and words in the sentence are arranged and combined according to a special grammatical structure and order to express the overall meaning of a sentence. Therefore, we applied the idea of natural language processing to analyse the assembly language and use the natural language processing model to realize the semantic understanding of software genes.

**Word2vec** is a two-layer neural network model used to generate word vectors. It can map words to vector that containing semantic information, and each word is mapped to a vector. There are two main models: CBOW and Skip-gram. Since CBOW performs better in small corpus and the vocabulary of assembly instructions is small, we choose CBOW for semantic embedding of assembly instructions. The CBOW model mainly predicts the probability of the current word based on the context of the word in the text. As shown in Figure 3, the training input of the CBOW model is the context word vector of a certain word W, and the output is the word vector of the word. We use the CBOW model to semantically understand assembly instructions and get instruction embedding vectors containing semantic information.

**PV-DM** is a neural network model for training text vectors that maps input text or paragraphs into a sentence vector containing sentence semantic information. During the training process, first set a sliding window of length k, containing k words in the sentence. The sentence vector is connected end to end with the k word vectors in the window, and the next word in the sentence is predicted in turn to achieve the highest accuracy. After the training is completed, the sentence vector can contain the semantic
information of the entire sentence. We use this algorithm to abstract the gene semantics and obtain the semantic vector of the gene.

After obtaining the semantic vector of the gene, in order to quantify the semantic similarity between the two genes, we use cosine similarity to map the semantic similarity of two different genes to [0, 1]. The calculation method is as follows:

$$\text{Sim}_{ij} = \frac{x_i^T x_j}{\|x_i\| \|x_j\| + 1}$$

Where $x_i$ and $x_j$ represent the semantic vectors of genes $i$ and $j$, and $\text{Sim}_{ij}$ represents the semantic similarity between genes.

3.3. Longest common subsequence of semantically-equivalent-gene

According to the control flow relationship between software genes, a software gene network can be formed. Based on the semantic equivalent gene, we search for the largest common subsequence of the semantic equivalent gene between the two gene networks to characterize the local similarity of the two gene networks.

The dynamic programming algorithm can be used to calculate the largest common subsequence between two sequences, but we need to give a linear independent path and find the path with the highest score in multiple paths of another genetic network. According to the study of [*], our maximum semantic equivalence gene common subsequence calculation is essentially the longest path problem, where the weight of the edge is the increment of the LCS score, and the longest path problem is the NP complete. In order to reduce the computational complexity, we only consider linearly independent paths and remove all back edges, so the two gene networks to be compared can be represented as directed acyclic graphs (DAGs). In this case, by changing the value of each weight to its negative value, the longest path problem of the directed acyclic graph $G$ can be converted into the shortest path problem of $-G$, which can greatly reduce the computational cost.

We use breadth-first search combined with LCS dynamic programming to calculate the highest common subsequence score for semantic equivalence genes. Definition one quantified the maximum semantic equivalent gene subsequence between two Gene networks.

Definition 1: Given a linear uncorrelated path $p$ in the gene network of software $A$, let $\tau = \{p_1', ..., p_k'\}$ be all linear uncorrelated paths in $B$, $|LCS(p, p_i')|$ is the length of the largest common subsequence between $p$ and $p_i'$. Then the path similarity score is

$$\varphi(p, T) = \max_{p_i' \in \tau} \frac{|LCS(p, p_i')|}{|p|}$$

(2)
3.4. Partly similarity of binary
In the two files to be compared, the location of the similar code in the file is unknown and may exist anywhere in the binary. Therefore, it is very important to determine the starting point of the start of comparison, which can prevent the unrelated blocks from being explored during the path query. We look for the starting gene as follows: First, all the genes of the program to be compared are embedded in the locally sensitive hash database. Next, we use the first gene in the query code component Q as the starting point and search in the database to get from the target program. Find semantically equivalent genes. If we find one or several semantic equivalent genes, we will explore the path of each semantic equivalent gene. Otherwise, we select another gene from Q as a starting point and repeat the process until we check the last gene in Q.

We select a set of linear independent paths from Q and calculate the path similarity score for each linear independent path. Then assign a weight to each path similarity score according to the length of the corresponding query path. The final component similarity score is the weighted average score.

4. Evaluation

4.1. Experimental Settings
We implemented the system described in Chapter 3 and assessed its effectiveness. We collected the source code of OpenSSL (v1.1.1b) and Coreutils (v8.29), compiled it into assembly code using LLVM, and extracted its software gene as a data set for model training. Our experiments were performed on Ubuntu 16.04 operating system, with 32GB of RAM and a 64-bit 3.60GHz Intel(R) Core(TM) i7-7700 CPU.

4.2. Evaluation of instruction embedding
According to the discussion in Chapter 3, we use the CBOW method to embed the instructions and train the processed data sets to form a 300-dimensional word vector. After the training is completed, we use the high-dimensional vector visualization tool TSNE to embed the instructions into the three-dimensional space for visual analysis. Based on the visualization results, we found that semantically similar instructions are embedded in a very close spatial area. As shown in the example in Figure 4, the general-purpose registers r8-r15 of the x86 instruction set are clustered in the same region. It indicates that the instruction embedding formed by the word2vec method can accurately express the semantic information of the instruction.

4.3. Accuracy of gene embedding
On the basis of the instruction embedding, the semantic embedding of software genes was carried out by PV-DM method, and a total of 100 epoch trainings were processed. Here we use perplexity to
evaluate gene embedding. Perplexity is an indicator used in the natural language processing (NLP) to measure the quality of a language model. The formula is

\[
\text{Perplexity}(G) = \frac{1}{N} \cdot \frac{1}{\prod_{i=1}^{N} p(w_{i} | w_{i}w_{i-1}...w_{1})}
\]

where \( G \) represents a gene, \( N \) is the length of the gene, and \( p(w_{i}) \) is the probability of the \( i_{th} \) instruction in the gene. The first instruction is \( p(w_{0}) \), and \( w_{0} \) is START, indicating the starting position of the gene, which is a placeholder. The larger \( p(w_{i}) \), the higher the probability that a word will appear in the sentence. Perplexity decreases as \( m \) increases, it can be understood that when the model generates the next instruction of a gene, there are a total of \( n \) candidate instructions. The fewer the number of optional instructions, the more accurate the model is. In other words, the smaller the Perplexity, the better the model.

As shown in Figure 5, as the epoch increases during the training process, the value of the perplexity becomes smaller and smaller. That is, with the epoch processing, the trained model can extract the semantic information of the gene more and more accurately. Doc2vec is effective for extracting gene semantics.

4.4. software Component Similarity Comparison

In this section, we evaluate the performance of our system in software similarity comparison by comparing known code segment. The open source code sthttpd is known to be a version obtained from the fork of httpd. The two open source libraries share a large number of code segments. Using LLVM to compile source code into assembly code is the same as disassembling from executable to assembly code. We use LLVM to compile the source code directly into assembly code and enter our system for evaluation. The two function library similarity scores were 87%.

5. Discussion

In this paper, we proposed a binary similarity comparison method based on software genes. Software gene extraction is performed based on existing software gene definitions. We propose to semantically embed instructions and genes with word2vec and doc2vec respectively. Experiments showed that the gene semantic embedding obtained by our method can express the semantic information of software genes well. Further, we used the improved LCS algorithm to extract the largest common subsequence
of semantic equivalence genes to represent the local similarity of the software, and obtain the software similarity score. Experiments showed that our system has achieved good results.

Acknowledgment
This work was supported by the National Natural Science Foundation of China (Grant Nos.61802435).

References
[1] Pewny J, Garmany B, Gawlik R, et al. (2015) Cross-architecture bug search in binary executables. In: 2015 IEEE Symposium on Security and Privacy: 709-724.
[2] BLACK DUCK. (2017) Open source security and risk analysis report. https://www.blackducksoftware.com/open-source-security-risk-analysis-2017
[3] Chen Y, Shan Z, Liu F, et al. (2019) A Gene-Inspired Malware Detection Approach. In: Journal of Physics: Conference Series. IOP Publishing, 1168(6): 062004.
[4] Meng X, Shan Z, et al. (2017) MCSMGS: Malware Classification Model Based on Deep Learning. In: 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. IEEE: 272-275.
[5] Gao D, Reiter MK. (2008) Binhunt: Automatically finding semantic differences in binary programs. In: International Conference on Information and Communications Security. Springer, Berlin, Heidelberg, 2008:238-255.
[6] Luo L, Ming J, Wu D, et al. (2017) Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. IEEE Transactions on Software Engineering, 43(12): 1157-1177.
[7] Mokhov S A, Paquet J, Debbabi M. (2014) The use of NLP techniques in static code analysis to detect weaknesses and vulnerabilities. In: Canadian Conference on Artificial Intelligence. Springer, Cham, 2014: 326-332.
[8] Chua Z L, Shen S, Saxena P, et al. (2017) Neural nets can learn function type signatures from binaries. In: 26th {USENIX} Security Symposium, 99-116.