Searching for similar code sequences in executable files using siamese neural network

Alexander Yumaganov
Geoinformatics and Information Security Department
Samara National Research University
Samara, Russia
yumagan@gmail.com

***Abstract*—This work is dedicated to solving the problem of finding similar code sequences (functions) in executable files. The description of functions obtained using the proposed method for solving this problem is based on the mutual spatial position of processor instructions and the corresponding operands in the function body. The word embedding model word2vec is used to form an intermediate description of the executable file functions. The final description of the functions is formed using the siamese long short-term memory network (siamese-LSTM). Then it description directly used to search for similar functions. The results of experimental studies of the developed method are presented in comparison with some previously known methods.**

Keywords—searching, code sequence, siamese neural network

I. INTRODUCTION

Nowadays, developers of new software often use code that was developed earlier to solve other problems. Reusing the code can improve the development time of new software, but it can also cause errors and vulnerabilities in the created products. According to studies presented in [1], 69 vulnerable fragments of C ++ code found from StackOverflow.com were used in 2859 projects on the GitHub service. In addition, using someone else's code may be illegal. According to the Synopsys report "Open source security and risk analysis" [2] for 2019, it was found that out of more than 1200 audited software products, 67% contained components with license conflicts.

Most software developers do not share their source code, so it is difficult to perform the analysis of their software. In this case, the analysis of executable files is the only way to analyze the code of that kind of software.

Thus, solving the problem of finding similar code sequences in executable files allow us to solve such problems such as finding known vulnerabilities and plagiarism detection without using the source code of the analyzed software.

There are a lot of methods of similar code sequences search in executable files. In [3], a method finding similar code sequences of code was presented. This method is based on comparing limited sequences of processor instructions (k-gram). To identify library functions in the IDA disassembler, the Fast Track Identification and Recognition Technology (FLIRT) algorithm [4] is used. This algorithm is based on a comparison of function templates. The methods mentioned above are sensitive to syntactic changes of the program code (for example, replacing a command with an equivalent command or group of commands). There are also methods based on the analysis of function's control flow graph (CFG). The authors of [5] proposed a search method in which the vector representation of function is formed by taking into account the structure of its CFG, and the similarity rate is estimated using a neural network. The methods of similar code sequences search presented in [6, 7, 8] are based on comparing subgraphs of function's CFG. The authors of [9] presented an intrusion detection system for distributed data processing systems based on a comparison of process signatures obtained from their control flow graphs. This group of methods also has disadvantages: high sensitivity to structural changes in functions, inapplicability to functions with a small number of basic blocks CFG. Thus, despite the large number of works in the field of similar code sequences search in executable files, there is no universal method or approach to solve this problem, which is devoid of all the shortcomings.

This paper presents a method of similar code sequences search in which the initial description of functions is formed on the basis of the spatial position of processor instructions and their corresponding operands. The siamese neural network is used to obtain the final description of the function.

The paper is structured as follows. The first section presents the basic definitions and a brief description of the proposed method. In the second section, the process of obtaining the initial description of functions is considered. The third section presents the descriptions of intermediate function representation. The fourth section describes the siamese neural network, which is used to obtain the final representation of functions. The fifth section is devoted to the description of the similar functions search algorithm. The sixth section provides a method for evaluating the effectiveness of the method and the results of the experiments. In the final part of the paper, the conclusions and a list of references are presented.

II. BASIC CONCEPTS AND PRINCIPLE OF OPERATION

The following definitions are used in this paper:

- current library - the set of the investigated executable functions;
- archived data - the set of the known functions.

Taking into account the above definitions, the problem solved by the proposed method is formulated as follows: for a given (or each) function of the current library, find the most similar function from the archive data.

The method of similar code sequences search proposed in this work includes several stages. At the first stage, the process of training the siamese neural network is carried out. At the second stage a description of the archive data functions is formed using a trained neural network. At the third stage, a description of the current library functions is formed in a similar way. At the final stage, the search for similar functions is performed.

III. Construction of the initial and intermediate descriptions of functions

A. Construction of the initial descriptions of functions

After disassembly analysis of the executable code performed by disassembler, we can get an assembler of every function in the executable file. Each function consists of a sequence of processor instructions and corresponding operands.

All processor instructions are divided into K functional groups according to the type of operations they perform (for example, a group of logical operations, a group of arithmetic operations). All operands are also divided into N groups according to the types of operands presented in the IDA disassembler [10] (for example, base register, FPP register, the value itself).

Each command and its corresponding operands are converted into lexemes (words) as follows:

$$Lexeme(k, n_0, n_1) = Concatenate(k, n_0, n_1),\ k \in [0, K-1],\ n_0 \in [0, N-1], n_1 \in [0, N-1],$$

where k is a index number of the functional group of the considered processor instruction, n_0 and n_1 are index numbers of the operands groups to which the first and second operands belong respectively.

For example, let us consider the command "mov rax, 1": $k = 0$ (Data Transfer Instructions), $n_0 = 1$ (General Register), $n_1 = 5$ (Immediate):

$$Lexeme(0,1,5) = '000105'$$

Thus, each function can be represented as an ordered set of lexemes.

B. Construction of the intermediate description of the function

The word2vec model [11] is used to obtain a vector representation of the function's initial description. The vector representations obtained by this model are based on the contextual proximity of lexemes inside functions. About a dozen different binary files were used to train this model. As a result of the training process, we have a dictionary in which a vector of a given dimension is assigned to each lexeme (obtained from training binary files).

Thus, a function consisting of m commands describes as a two-dimensional vector I of dimension $m \times s$. Let us call this vector as an intermediate description.

IV. Construction of the final description of the function

It is proposed to use the siamese neural network [12] to construct the final description of the functions. This network consists of two identical long short-term memory (LSTM) neural networks with the same weights.

The LSTM network is a special type of recurrent neural network (RNN). In contrast to RNN, it is able to work with long-term dependencies. This is achieved due to its ability to transfer the state of the cell from the previous time step to the next step, as well as to control the information flow in the LSTM cell.

The input of each siamese network's LSTM network is a function in the form of previously obtained intermediate description. Then the outputs of the last layers of each network are sent to the input of a function that evaluates the similarity between these outputs. In this paper, the following metric is used as such a function:

$$D(\overline{a}, \overline{b}) = \exp(-\sum_{i=0}^{J-1} |a_i - b_i|), \quad (1)$$

where a_i is the value of the i-th element of the last layer of the first LSTM network, b_i is the value of the i-th element of the last layer of the second LSTM network. Two functions are considered to be same if $D = 1$. In the contrary case, if $D = 0$ two functions are considered to be totally different.

Structural chart of siamese neural network is shown in Fig. 1.

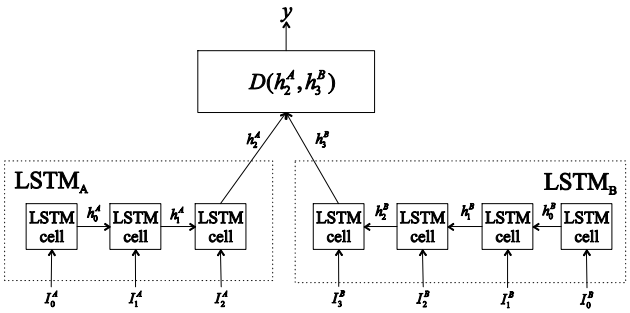

Fig. 1. Structural chart of siamese neural network

Here, D is the metric (1), I_0^A, I_1^A, I_2^A are the components of the intermediate description vector of the first function, corresponding to its lexemes, $I_0^B, I_1^B, I_2^B, I_3^B$ are the components of the intermediate description vector of the second function, corresponding to its lexemes, y is the output value of the siamese neural network, taking the value 0 if the functions are different, otherwise 1.

Contrastive loss [13] is used as a loss function, which is given by:

$$L(y, D) = \frac{1}{2}(1-y)D^2 + \frac{1}{2}y\{\max(0, 1-D)\}^2.$$

This loss function maximizes and minimizes the value of (1) between similar ($y = 1$) and different ($y = 0$) functions, respectively.

Two “archives” of functions were used to train siamese neural network : the first one contained the library functions libtiff 4.0.3 [14] and proj 4.9.1 [15], and the second one contained the library functions libtiff 4.0.8 and proj 5.0.1.

In the learning process, for each function of the first archive, two pairs were formed: current function and the function having the same name from the second archive; current function, and a random function from the second archive, which has a name different from the first function. The resulting pairs of vectors are fed to the input of the neural network, the output value for the first pair is 1 (similar), and for the second pair is 0 (different).

The trained model of the siamese neural network is used to form the final description of the functions by passing an intermediate description of the analyzed function to the input of one of the LSTM networks. The output vector of this LSTM network, in this case, is the final description of the function. The dimension of the output layer is $J = 32$.

Thus, any function of the analyzed binary file can be represented as a vector of dimension J.

V. Search for similar functions

The final stage of the proposed method is the search for similar functions using the final description of the functions of archive data and the current library.

Let $\bar{a}$ be the final description vector of the current library function, $\bar{b}$ be the final description vector of the archive data library. For feature vector comparison we use the Euclidean metric (distance):

$$d(\bar{a}, \bar{b}) = \sqrt{\sum_{i=0}^{J-1} (a_i - b_i)^2}, \quad (2)$$

where a_i is the value of the i-th component of the current library function feature vector, b_i is the value of the i-component of the archive data library function feature vector. If $d = 0$ the functions are considered equal.

The final description of the given function of the current library is compared with each function of archive data using Euclidian metric. Then, the obtained results are sorted by increasing the distance (2). As a result, we get the list of archival functions, sorted by similarity in descending order, for the analyzed current library function.

VI. Experiments

To evaluate the efficiency of the proposed method of similar code sequences search in executable files, the functions of one dynamic library are used as archive data, and the functions of the same library, but of a different version, are used as the current library. It was considered that in the process of switching from one version of the dynamic library to another, the names of the functions did not change and there are no functions with the same name among the functions of the archive data.

Using the search algorithm described in the fifth section, for a given function of the current library, we obtain an ordered list of archive data functions. Let us assign a binary sequence $\beta = (\beta_1, \beta_2, \ldots, \beta_L)$ to this list, the i-th element of which is equal to one, if the name of the function at the i-th position of the list is identical to the name of the function being checked, and the i-th element of which is equal to zero otherwise. The following criteria to evaluate the quality of information retrieval [17,18] are used:

- Precision for the k-th position of the list:

$$P_k = \frac{\sum_{l=1}^{k} \beta_l}{k}$$

- Recall for the k-th position of the list:

$$R_k = \frac{\sum_{l=1}^{k} \beta_l}{K}$$

- The average precision of the list:

$$AveP = \sum_{k=1}^{L} P_k (R_k - R_{k-1}), R_0 = 0$$

.

The average precision for all functions included in the current library is calculated by the formula:

$$P = \frac{1}{S} \sum_{s=0}^{S-1} AveP_s, \quad (3)$$

where S is a number of functions in the current library.

The archive data was represented by the curl 7.6.3 [16], and the current library functions was represented by other versions of the curl library.

The average precision of search of the proposed method was compared with some previously known methods: a method based on the analysis of the spatial position of processor functional groups [19] and a method based on k-gram comparison [3].As a comparison object for the first method, the comparison object recommended by the authors was used (the spatial distribution of instructions in the function body in the integral form). For the second method, the value of the parameter k = 5 was also chosen based on the recommendations of the authors. The results are presented in table 1.

TABLE I. Comparison of similar functions search methods

Current library	Average precision of search P		
	Using proposed method	*Using k-gram based method*	*Using method presented in [15]*
curl 7.5.4	0.7830	**0.8208**	0.7828
curl 7.5.6	**0.8631**	0.8543	0.8550
curl 7.5.9	**0.8886**	0.8768	0.8851
curl 7.6.0	0.8960	0.8830	**0.9036**

The analysis of the obtained results shows that the method of similar code sequences search presented in this work is highly competitive with known methods, and in some cases surpasses them. Since the training of the siamese neural network was carried out on a small data set, the results obtained for this method can be further improved by increasing the amount of data for training the neural network.

VII. Conclusion

The paper presents a method of similar code sequences search in executable files based on the siamese neural network. The results of experimental studies demonstrating the efficiency of the developed method (average precision 0.78 - 0.89). Further research will be aimed at tuning the parameters of the siamese neural network in order to increase the average precision of the search for this method.

Acknowledgment

This work was supported by RFBR research project № 18-01-00748 A.

References

[1] M. Verdi, A. Sami, J. Akhondali, F. Khomh, G. Uddin and A.K. Motlagh, "An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples," arXiv: 1910.01321, 2019.

[2] Synopsys Cybersecurity Research Center: Open source security and risk analysis, 2020 [Online]. URL: https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2020-ossra-report.pdf

[3] G. Myles and C. Collberg, “K-gram based software birthmarks,” Proceedings of the ACM symposium on Applied computing, 2005.

[4] IDA Pro. F.L.I.R.T, 2019 [Online]. URL: https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml.

[5] N. Shalev and N. Partush, “Binary Similarity Detection Using Machine Learning,” Proceedings of the 13th Workshop on Programming Languages and Analysis for Security - PLAS, 2018.

[6] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, “Polymorphic Worm Detection Using Structural Information of Executables,” Lecture Notes in Computer Science, Springer Berlin, Heidelberg, pp. 207-226, 2006.

[7] Y. David and E. Yahav, “Tracelet-based code search in executables,” ACM SIGPLAN Notices, vol. 49, no. 6, pp. 349-360, 2014.

[8] S. Eschweiler, K. Yakdan and E. Gerhards-Padilla, “DiscovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code,” Proceedings Network and Distributed System Security Symposium, 2016.

[9] S. Aditham and N. Ranganathan, “A Novel Control-flow based Intrusion Detection Technique for Big Data Systems,” arXiv: 1611.07649, 2016.

[10] Hex-Rays IDA: About, 2019 [Online]. URL: http://hex-rays.com/products/ida/.

[11] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient estimation of word representations in vector space", Proc. of ICLR Workshop, 2013.

[12] J. Bromley, I. Guyon, Y. LeCun, E. Siickinger and R. Shah, “Signature verification using a “siamese” time delay neural network,” Proceedings of the 6th International Conference on Neural Information Processing Systems, pp. 737-744, 1994.

[13] R. Hadsell, S. Chopra and Y. LeCun, “Dimensionality Reduction by Learning an Invariant Mapping,” IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, 2006.

[14] TIFF Library and Utilities, 2019 [Online]. URL: http://www.libtiff.org/.

[15] PROJ coordinate transformation software library, 2019 [Online]. URL: https://proj.org/about.html.

[16] Libcurl - the multiprotocol file transfer library, 2019 [Online]. URL: https://curl.haxx.se/libcurl/.

[17] M. Buckland and F. Gey, “The relationship between Recall and Precision,” Journal of the American Society for Information Science, vol. 45, no. 1, pp. 12-19, 1994.

[18] D.M.W. Powers, “Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation,” Journal of Machine Learning Technologies, vol. 2, no. 1, pp. 37-63, 2011.

[19] A.S. Yumaganov and V.V. Myasnikov, “A method of searching for similar code sequences in executable binary files using a featureless approach,” Computer Optics, vol. 41, no. 5, pp. 756-764, 2017. DOI: 10.18287/2412-6179-2017-41-5-756-764.