A combined method of similar code sequences search in executable files

A S Yumaganov[1]

[1]Samara National Research University, Moskovskoye shosse, 34, Samara, Russia, 443086

e-mail: yumagan@gmail.com

Abstract. The article is devoted to the development of a method of similar code sequences search in executable files, which is based on both syntax analysis of the code and function's control flow graphs analysis. The syntax analysis method used in this paper is based on a comparison of the spatial distribution of processor instructions in the function body. The analysis of function control flow graph is used a structural description of fixed-order subgraphs of the function control flow graph. The results of experimental studies, including the comparison of the proposed method and previously known methods of searching for similar code sequences, are presented.

1. Introduction

The problem of finding similar code sequences in executable files is very relevant today. According to the studies presented in [1, 2], developers often use previously created program code during the process of the new software development. This approach to software development is called code reuse. Despite the obvious advantages of this approach, it can also cause errors and vulnerabilities in the software being developed. In addition, third-party code reuse may be illegal. This approach is also used in the development of various malicious programs [3]. Thus, solving the problem of finding similar code sequences in executable files allows us to solve several problems: finding known vulnerabilities, finding plagiarism in software, and searching for malware.

Currently, there are a large number of methods of similar code sequences search in executable files. All known algorithms and methods for solving the above problem are usually based either on the syntactic analysis of the assembler program code or on the analysis of the structure of control flow graphs of the executable file functions.

Let us consider the methods based on the syntax analysis of the code. In [4, 5], similar methods of similar code sequences search was presented. These methods are based on comparing sequences or permutations of processor instructions of a fixed length (k-grams or n-perms) in functions. The IDA disassembler uses the IDA FLIRT algorithm [6] to identify library functions, which is based on a comparison of function's patterns. The author of [7] presented a method for detecting malware, based on the analysis of the frequency of processor instructions occurrence inside the examined file. Significant impact on the quality of the similar code sequences search for this group of methods is made by syntactic code changes: replacing processor instructions with equivalent ones, rearranging instructions, inserting new ones, and deleting old instructions. The methods based on the analysis of the functions control flow graph allow to overcome this

drawback. The authors of the method presented in [8] used the information of basic blocks (the vertices of the control flow graph) to detect and classify malware. The authors of [9] used a method based on determining the isomorphism of control flow graphs to identify malware. A similar approach was described in [10], where the detection of malware was based on determining the isomorphism of fixed length function's subgraphs and comparing the signatures (fingerprints) of their basic blocks. However, this group of methods also has several disadvantages: low search accuracy for functions with a small number of basic blocks, high sensitivity to structural changes in functions.

In this paper, a combined method for searching functions of an executable file, which are similar to the known functions from some software "archive" is proposed. The basis of the proposed method is to use of both the syntactic analysis of the code and the analysis of functions control flow graph. The description of the function in the presented method is formed by its similarity with the functions of the basis library.

The paper is structured as follows. The first section presents the basic definitions and a brief description of the proposed method. The second section discusses the process of getting a syntactic description; the third one describes the process of getting the structural description of functions. The fourth section is devoted to the description of the similar functions search algorithm. The fifth section provides the effectiveness evaluation technique of the proposed method and the results of the experiments. In the final part of the paper, the conclusions and a list of references are presented.

2. Basic concepts and principle of operation

The following definitions are used in this paper:

- current library - the set of the investigated executable file functions;
- archival data - the set of the known functions;
- basis library - an auxiliary set of functions used to compare the functions of archived data and the current library.

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. In this paper, we use two definitions of the functions similarity measure. The first one is based on the position of the functional groups of processor instructions in the body of functions. The second one is based on the analysis of the structure of the functions control flow graph. In the previously published works of the author [11, 12], two methods of similar code sequences search were presented, each of which is based on one of the above definitions of the function similarity measure, respectively. This paper presents a combined method of similar code sequences search. In this method, the description of function is formed through its similarity with the functions of the basis library, using two different definitions of the functions similarity measure described in [11, 12].

The proposed method of similar code sequences search includes several stages. At the first stage, the description of the archive data functions is formed through the library of basis functions. In addition, for each function, two descriptions are obtained (syntactic and structural) corresponding to different measures of functions similarity. At the second stage, the description of the current library functions is formed similarly. At the final stage, the search for similar functions is performed. The algorithm of search is described in detail in the fifth section.

3. The syntactic description of the function

Using the IDA [13] disassembler, we can obtain a partition of the assembler code of the analysed executable file into functions. The assembler code consists of a sequence of processor instructions and associated operands. All processor instructions can be divided into K functional groups

according to the type of operations they perform. Examples of such groups are: a group of arithmetic instructions, a group of logical instructions, a group of data transfer instructions. For each of the K functional groups for a given function, we obtain a list of offsets relative to the beginning of the function, on which the instructions of this group are located.

Let us determine the spatial distribution of the k instruction type as the absolute frequency of the instructions of this type entering in a relative normalized i-th interval ($I = 100$):

$$\tilde{f}_i^k = \sum_{j=0}^{N_k-1} I\left(\frac{n_j^k}{N} \cdot 100 \in (i-1, i]\right), \quad i = \overline{1, I}, \tag{1}$$

where $n_0^k, ..., n_{N_k-1}^k$ are absolute offsets relative to the beginning of the function of instructions of group k, N_k is a total number of instructions of this group in this function, N is the length of the function, $I(.)$ is the event indicator, which takes the values "0" or "1" depending on the truth of the corresponding argument.

To obtain the spatial distribution of instructions in the integral form, we use the following formula:

$$\hat{f}_i^k = \frac{\sum_{y=0}^{i} \tilde{f}_y^k}{\sum_{j=0}^{I} \tilde{f}_j^k}, i = \overline{1, I}. \tag{2}$$

Then, the spatial position of the k-th group of processor instructions in the body of the function is described by the vector:

$$\bar{a}_k = \left(\hat{f}_1^k, \hat{f}_2^k, \ldots, \hat{f}_I^k\right)^T, k = \overline{0, K-1}. \tag{3}$$

As a result, the description of the considered function has the following form:

$$A = \left(\bar{a}_0, \bar{a}_1, \ldots, \bar{a}_{K-1}\right). \tag{4}$$

The matrix B, which represents the description of the basis library functions, is formed in a similar way. The measure of functions similarity, which description is given by the matrices A and B, has the following form:

$$\mu\left(A, B\right) = \sum_{k=0}^{K-1} \alpha_k \mu_{\cos}(\bar{a}_k, \bar{b}_k), \quad \sum_{k=0}^{K-1} \alpha_k = 1, \tag{5}$$

where

$$\mu_{\cos}$$

is the cosine distance. If two functions are identical, the measure of similarity takes the value "1", otherwise "0".

Let a library of basis functions contains J functions, each of which has a description in the form of a matrix B_j. Then the description of the considered function through the library of basis functions will have the following form:

$$\bar{x}_A = \left(\mu\left(A, {}_0\right), \mu\left(A, {}_1\right), \ldots, \mu\left(A, {}_{J-1}\right)\right)^T. \tag{6}$$

Further, using the obtained intermediate description of the function, its final description is formed using the PCA (principal component analysis) method for reducing the dimensionality of the data. The process of forming the final description is described in detail in [11]. The resulting description of the function is stored in the corresponding database (archive or current).

4. The structural description of the function

IDA disassembler allows to obtain a control flow graph of the analysed executable file functions. The control flow graph of a function is a directed graph which vertices are the basic blocks of function. The basic block of function is a sequence of processor instructions. The first instruction of the basic block receives the control from some processor instruction, and the last one is an instruction, which passes the control to another basic block. The edges of the control flow graph determine the order of the basic blocks in the control flow of the function.

The control flow graph of the analysed function is divided into subgraphs of fixed order k (k-subgraphs) as follows: each basic block is chosen as a starting node and all edges beginning from that node are traversed until k nodes are encountered. In this paper, $k = 3$ is used.

The description of each of the k-subgraphs of the function consists of a pair of vectors: $\bar{a}$ and $\bar{b}$. The $\bar{a}$ vector is a binary vector obtained by combining the rows of the adjacency matrix of a given k-subgraph. The $\bar{b}$ vector characterizes the presence or absence of reading or writing operations in operands of various types in this k-subgraph. A detailed description of the vector $\bar{b}$ is presented in [12]. Thus, the initial description of a function consists of a set of pairs of vectors $\bar{a}$ and $\bar{b}$ describing each k-subgraph of the function.

The intermediate description of the function is formed on the basis of its similarity with the functions of the basis library. As a measure of similarity, we use the generalized Jaccard index [14]:

$$J(x, y) = \frac{\sum\limits_i \min(x_i, y_i)}{\sum\limits_i \max(x_i, y_i)}, \tag{7}$$

where x is a set of vector pairs which described the first function, y is a set of vector pairs which described the second function,x_i is a number of pairs i in set x,y_i is a number of pairs i in set y, i passed through all unique pairs of vectors in the combined set $x \bigcup y$. In the case of complete similarity between functions the value of similarity measure is "1", in case of complete dissimilarity it is "0".

Let x be a set of vector pairs described the analysed function, y_i is a set of vector pairs described the i-th function of the basis library, I is a number of functions in the basis library, then the intermediate description of the function under investigation has the following form:

$$\bar{z} = (J(x, y_0), J(x, y_1), ..., J(x, y_{I-1}))^T. \tag{8}$$

The final description of the function is obtained after applying the PCA dimension reduction method and is stored in the appropriate database (archive or current).

5. Search for similar functions

The final stage of the proposed method is the search for similar functions based on the previously obtained function description vectors.

In the previous works of the author, a search algorithm with the following assumption was used: the size of the changed functions differs from the original by no more than 30% [11]. This assumption was used to increase the quality of the search. Thus, at the first stage of the search, the archive functions were filtered by their size, then the Euclidean distance to each function was calculated from the filtered list of archive functions, and the result was sorted by increasing the Euclidean distance.

This paper presents a combined method of similar functions search, which uses a syntactic and structural description of functions. However, if the number of basic blocks of the function being studied is small ($bb_{\min} \leq 5$), only the syntactic description of the function and the search algorithm described above are used. This condition allows to increase the search accuracy since

the structural description of small functions can be very similar to the structural description of the similar size functions due to their small size.

The search method presented in this paper uses the following algorithm for similar functions search (if $bb_{\min} > 5$):

- At the first stage, preliminary filtering of archive functions is performed. For the function under study and all the functions of the archival data, the Euclidean distance between the vectors of the syntactic (or structural) description of the functions is calculated and sorted by ascending distance. The first $top_{th} = 30$ elements of the list of the most similar archive functions are used in the second stage of the search.
- At the second stage of the search, the Euclidean distance from the function under investigation to the list of functions obtained above is calculated. However, in this case, we use another type of vectors as the description vectors. In other words, if at the first stage the functions were compared by the vectors corresponding to their syntactic description, then at the second stage functions will be compared by the vectors corresponding to their structural description and vice versa. Then, the obtained result is sorted by increasing the Euclidean distance.

As a result, for the analysed function, a list of archive data functions is obtained, sorted by similarity in descending order.

6. 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 $\beta = (\beta_1, \beta_2, ..., \beta_L)$ 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 [15, 16] are used:

- Precision for the k-th position of the list: $P_k = \frac{\sum\limits_{l=1}^{k} \beta_l}{k}$
- Recall for the k-th position of the list: $R_k = \frac{\sum\limits_{l=1}^{k} \beta_l}{K}$
- The average precision of the list: $AveP = \sum\limits_{k=1}^{L} P_k(R_k - R_{k-1}), R_0 = 0$

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

$$P = \frac{1}{S} \sum_{s=0}^{S-1} AveP_s, \tag{9}$$

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

Several versions of the libtiff library [17] were used for experiments. The functions of the library libtiff 4.0.8 were used as functions of the archived data. These libraries were compiled with the optimization flag /Od (optimization disabled).

The results of comparing two methods of preliminary filtering of the archival data functions are presented in table 1.

Table 1. Comparing preliminary filtering methods of the archival data functions.

Current library	Average precision of search P, filtering by syntactic description	Average precision of search P, filtering by structural description
libtiff 3.9.2	0.7424	0.7592
libtiff 3.9.7	0.7551	0.7707
libtiff 4.0.3	0.7835	0.8290
libtiff 4.0.5	0.7943	0.8432

The average precision of the search for the considered libraries is higher when preliminary filtering of the archival data functions by structural description is used. Therefore, in further experiments, this method of preliminary filtering of archive functions will be used.

In next experiment, 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 [11] and a method based on k-gram comparison [4].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 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 2.

Table 2. Comparison of similar functions search methods.

Current library	Average precision of search P,using proposed method	Average precision of search P,using k-gramm based method	Average precision of search P,using method presented in [11]
libtiff 3.9.2	0.7592	0.7528	0.7370
libtiff 3.9.7	0.7707	0.7681	0.7524
libtiff 4.0.3	0.8290	0.7970	0.8257
libtiff 4.0.5	0.8432	0.8119	0.8427

The analysis of the obtained results shows that the method of similar code sequences search presented in this paper is superior to the previously known methods in each of the used current libraries. Moreover, the "closer" the version of the current library to the version of the archive data library, the less advantage the presented method has over the method [11]. This is explained by the fact that for libraries used in experimental studies, some functions of older versions of the current library have significant syntactic changes with respect to archive functions. And preliminary filtering of archival data by structural description can significantly improve the precision of the search.

7. Conclusion

The paper presents a combined method of similar code sequences search in executable files using both syntactic and structural descriptions of functions. The results of experiments demonstrating the superiority of the developed method over some previously known methods have been presented. Further studies will be carried out in improving the accuracy of similar functions search and studying the efficiency of the proposed method using executable files compiled with different compilation settings.

8. References

[1] Abdalkareem R, Shihab E and Rilling J 2017 On code reuse from StackOverow: An exploratory study on Android apps *Information and Software Technology* **88** 148-158
[2] Gharehyazie M, Ray B and Filkov V 2017 Some from here, some from there: cross-project code reuse in GitHub *Proc. of the 14th International Conference on Mining Software Repositories* **1** 291-301
[3] Examining Code Reuse Reveals Undiscovered Links Among North Koreas Malware Families URL:https://securingtomorrow.mcafee.com/mcafee-labs/examining-code-reuse-reveals-undiscovered-links-among-north-koreas-malware-families/ (05.11.2018)
[4] Myles G and Collberg C 2005 K-gram based software birthmarks *Proc. of the ACM symposium on Applied computing* 1 314-318
[5] Karim M, Walenstein A, Lakhotia A and Parida L 2005 Malware phylogeny generation using permutations of code *Journal in Computer Virology* **1** 13-23
[6] IDA F.L.I.R.T Technology: In-Depth URL: https://www.hex-rays.com/products/ida/tech/irt/indepth.shtml (05.11.2018)
[7] Bilar D 2007 Opcodes as predictor for malware *International Journal of Electronic Security and Digital Forensics* **1** 156-168
[8] Gheorghescu M 2005 An automated virus classi_cation system *Virus Bulletin Conference* **1** 294-300
[9] Bruschi D, Martignoni L and Monga M 2006 Detecting selfmutating malware using control-ow graph matching *Proc. of the Third international conference on Detection of Intrusions and Malware & Vulnerability Assessment* **1** 129-143
[10] Kruegel C, Kirda E, Mutz D, Robertson W and Vigna G 2005 Polymorphic worm detection using structural information of executables *Recent Advances in Intrusion Detection* **1** 207-226
[11] Yumaganov A and Myasnikov V 2017 A method of searching for similar code sequences in executable binary files using a featureless approach *Computer Optics* **41(5)** 756-764 DOI: 10.18287/2412-6179-2017-41-5-756-764
[12] Yumaganov A and Myasnikov V 2018 Searching for similar code sequences in executable files based on the structural analysis of functions *J. Phys.: Conf. Ser.* **1096** 012093
[13] Hex-Rays IDA: About URL: http://hex-rays.com/products/ida/ (05.11.2018)
[14] Spath H 1981 The minisum location problem for the Jaccard metric *Operations-Research-Spektrum* **3** 91-94
[15] Buckland M and Gey F 1994 The relationship between recall and precision *JASIS* **45** 12-19
[16] Powers D 2011 Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation *Journal of Machine Learning Technologies* **2** 37-63
[17] TIFF Library and Utilities URL: http://www.libti_.org/ (05.11.2018)