Searching for similar code sequences in executable files based on the structural analysis of functions

A S Yumaganov¹ and V V Myasnikov¹,²

¹Samara National Research University, Moskovskoe Shosse 34, Samara, Russia, 443086
²Image Processing Systems Institute - Branch of the Federal Scientific Research Centre “Crystallography and Photonics” of Russian Academy of Sciences, Molodogvardeyskaya str. 151, Samara, Russia, 443001

e-mail: yumagan@gmail.com

Abstract. This work is devoted to the development of a method of similar code sequences search in executable files, based on functions control flow graph analysis. This method is also based on featureless approach, where functions are represented by their relationships with predefined basis library functions. The results of experimental studies demonstrate the applicability of proposed method and its efficiency in comparison with another method of similar code sequences search.

1. Introduction

Nowadays, in the process of development new software products, developers often use different library functions that were implemented earlier by other people. According to the study presented in [1] more than 10% of files found in open source Java projects are clones of other files. This approach of software development may cause transferring vulnerabilities and errors from reused code. It also may cause the software license agreement violations. Searching for similar code sequences can be used for finding both errors or vulnerabilities in programs and plagiarism detection in program code. Moreover, most new viruses are a modification of the known ones. So, it is possible to use similar code sequences search methods for their detection.

There are a lot of known methods of similar code sequences search in executable files. The authors of [2] proposed a method of similar code sequences search based on a comparison of processor fixed-size instruction sequences (k-grams) in functions. These sequences are obtained using a sliding window. Method of similar code sequences search presented in [3] is based on static variables comparison. This approach allows to analyze obfuscated files, but it has a low precision of search results. The above methods are based on syntactic code analysis. These methods are sensitive to code changes: instructions replacement, instructions rearrangement, new instructions insertion, static variables changes and etc. Methods based on structural analysis of functions are free from this problem. For example, methods of similar code sequences search presented in [4, 5] are based on function's CFG (control flow graph) analysis. These types of methods are sensitive to structural code changes.
This paper is devoted to the development of the method of executable file's functions search, which are similar to the known functions from some software "archive". The function description in the proposed method is formed through its similarity to the base library functions. This approach was used in previous authors works [6, 7, 8], where the comparison of functions was based on the syntactic analysis of the code. However, the method presented in this paper is based on the control flow graph structure analysis.

This paper is organized as follows. Section 2 introduces a brief description of the proposed method of similar code sequences search. Section 3 presents the description of the process of obtaining an executable file's functions initial description on the basis of CFG analysis. Section 4 presents the description of the intermediate and final function representations via the base library functions. Section 5 presents the results of experimental studies. The final part of the paper comprises conclusions and a list of references.

2. Basic concepts and principle of operation

In this paper, the following basic notations are used:

- **archival data** denotes a set of known functions and their descriptions via base library;
- **current library** denotes a set of functions to be examined;
- **base library** denotes an auxiliary set of functions that are used in constructing descriptions of function from archival data and current library.

The problem solved by this method is formulated as follows: for the given function of current library find the most similar function from the archival data. There are many ways to specify a measure of similarity between two functions. For example, in authors' previous work [8] the proposed measure of similarity was based on the location of the functional groups of instructions in the body of the function. Within the scope of this paper, the measure of similarity based on comparing subgraphs of function's CFG is considered. A detailed description of this measure of similarity is presented in the fourth section.

The proposed method of finding similar functions includes three stages. On the first stage, functions from archival data are described using the base library functions, and these descriptions are stored in the database. On the second stage, same operations are performed for the current library functions. On the final stage, the search for similar functions is performed using descriptions obtained on the previous stages. The first and the second stages are performed only once for fixed archival and current data, and the third stage can be performed the required number of times.

3. Analysis of function's CFG and construction of it's the initial description

After disassembly analysis of the executable code performed by IDA disassembler, we can get an assembler code and control flow graph of every function in the executable file.

A control flow graph (CFG) is a directed graph which nodes are function's basic blocks, and edges determine the order of the basic blocks in the control flow of the function. A basic block is a sequence of processor instructions for which there are no transfer control instructions (last instruction of basic block and "call" instructions are the exceptions) and instructions to which control is transferred (first instruction of basic block is an exception).

For a given function the set of subgraphs of fixed size $k$ ($k$-subgraphs) are extracted from CFG. This is done by choosing each basic block as a starting node and traversing all edges beginning from that node until $k$ nodes are encountered. Next, for each $k$-subgraph the adjacency matrix is obtained.

Thus, the structure of each $k$-subgraph can be represented as a binary vector $\vec{a}$ of dimension $k*k$, which is obtained by concatenating the rows of its adjacency matrix. The value of $k$ is one of the method parameters. Our approach to structural subgraph's description is similar to the one presented in [10].

It is also necessary to take into account the content of basic blocks to increase the precision of the search. All $k$-subgraphs of a function are described by a binary vector $\vec{b}$ which characterizes the presence or absence of read /write operations of various data types. The descriptions of each vector's
The position of vector's element | Description of feature
---|---
1 | Reading from register
2 | Writing to register
3 | Reading from memory reference
4 | Writing to the memory reference
5 | Reading from indirect memory reference
6 | Writing to the indirect memory reference
7 | Immediate value read
8 | Immediate address read

Pairs of vectors $\vec{a}$ and $\vec{b}$ are obtained for every $k$-subgraph of given function. A set of pairs for given function represents its initial description. The function description obtained in this way is stored in the corresponding database and used later to construct an intermediate function description using the base library.

### 4. Construction of the intermediate and final description of the function

To obtain an intermediate description of the given function, it is necessary to obtain the value of some similarity measure of each of the base library functions. We suggest using generalized Jaccard index as such measure of similarity [11]:

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

(1)

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 $x \cup y$. In the case of complete similarity between functions the value of similarity measure is "1", in case of complete dissimilarity it is "0".

Let $N$ be the number of functions in the base library, each of which has a description in the form of the set of vector pairs $y_i$, $x$ is a set of vector pairs of given function. We compare the analyzed function of the current library with each of the base library functions using generalized Jaccard index (1) to obtain the following vector of this function's intermediate description:

$$\vec{z} = (J(x, y_0), J(x, y_1), \ldots, J(x, y_{N-1}))^T$$

Since the number of functions in the base library $N \gg 1$, we apply dimension reduction method, called Principle Component Analysis (PCA) [4]. As a result, we obtain a new vector $\vec{f}$ of dimension $I < N$:

$$\vec{f} = Y\vec{z}$$

where $Y$ is the transition matrix, consisting of $I$ eigenvectors.

The obtained vector $\vec{f}$ acts as a final representation of the analyzed function through the set of the base library functions. Then, vector $f$ is stored in the corresponding database (archival or current library).
5. Experiments

To evaluate the efficiency of the proposed method of similar code sequences search, we took functions of the certain dynamic library (libtiff[12], libcurl[13]) as an archival data, and functions of another version of the same dynamic library as a current library. We assumed that modification of the library code did not affect on the function names.

For the analyzed function of the current library, we got a list of similar functions of the archival data that was sorted by the similarity in descending order using the algorithm presented in the previous authors’ work [6]. We associate this list of functions with binary sequence $\beta = (\beta_1, \beta_2, \ldots, \beta_L)$, such that if there is a function with the same name on the $i$-th position, then $\beta_i = 1$, else $\beta_i = 0$.

We used the following criteria to evaluate the quality of information retrieval [14] [15]:

- Precision for the $k$-th position of the list: $P_k = \frac{\sum\beta_i}{k}$
- Recall for the $k$-th position of the list: $R_k = \frac{\sum\beta_i}{K}$
- The average precision of the list: $\text{Ave}P = \sum_{i=1}^{L} P_i (R_i - R_{i-1})$, $R_0 = 0$

Thus, the average precision for all current library functions was calculated by the following formula:

$$P = \frac{1}{S \sum_{i=0}^{L} \text{Ave}P_i}$$  (2)

where $S$ denotes the number of functions in the current library.

The archived data was represented by the libtiff 4.0.3 and libcurl 7.53.1 libraries, and the current library was represented by different versions of the libtiff library (3.9.2, 4.0.8) or libcurl library (7.49.0, 7.51.0) respectively. The base library contained $N = 50$ functions selected manually from different executable and dynamic library files.

We establish the minimum number of basic blocks in the CFG of the functions of the archive data and the current library $bb_{\text{min}} = 20$ in order to exclude the "small" functions, the intermediate description of which due to their size may be very similar to the intermediate description of other such functions. The presence of "small" functions may cause low precision of the search.

The results of comparing using different values of the proposed method's parameter $k$ (subgraphs order) are presented in Table 2.

| Archive data | Current library | Average precision of search P |
|--------------|-----------------|-----------------------------|
|              |                 | $k=3$ | $k=4$ | $k=5$ |
| libtiff 4.0.3 | libtiff 3.9.2   | 0.8313 | 0.8143 | 0.7862 |
| libtiff 4.0.3 | libtiff 4.0.8   | 0.9117 | 0.8966 | 0.8674 |
| libcurl 7.53.1 | libcurl 7.49.0  | 0.9281 | **0.9308** | 0.8766 |
| libcurl 7.53.1 | libcurl 7.51.0  | **0.9579** | 0.9576 | 0.9078 |

From this table, we can conclude, that the highest average precision of the search in most cases is obtained when $k=3$. This is due to the fact that for a larger value of $k$ vector $\vec{b}$ (which describes the content of basic blocks) characterize a larger number of basic blocks. Thus, the probability of the presence of each feature described by vector $\vec{b}$ increases and the uniqueness of this vector decrease respectively. As a result, the average precision of the search is reduced.
The results of comparing the efficiency of the proposed method of finding similar functions and another method presented in previous authors’ work [8] are presented in Table 3. As noted above, the method presented in [8] is based on a comparison of the spatial position of the processor instructions in the body of the function. As a comparison object (method parameter) we use the object recommended by the authors - the spatial distribution of instructions in the function body in integral form.

| Archive data | Current library | Average precision of similar functions search $P$ using proposed method $(k = 3)$ | Average precision of similar functions search $P$ using method presented in [8] |
|--------------|-----------------|---------------------------------|---------------------------------|
| libtiff 4.0.3 | libtiff 3.9.2   | 0.8313                          | 0.8039                          |
| libtiff 4.0.3 | libtiff 4.0.8   | 0.9117                          | **0.9286**                      |
| libcurl 7.53.1| libcurl 7.49.0  | **0.9281**                      | 0.8950                          |
| libcurl 7.53.1| libcurl 7.51.0  | **0.9579**                      | 0.9342                          |

The presented results of the experiments indicate that the proposed method in most cases exceeds the method presented in [8] by the criterion of the maximum average precision of the search with a given constraint $bb_{min} = 20$. However, it should be noted that when the value of $bb_{min}$ is reduced to 5, the average precision of the search for the original method is significantly reduced and the best results are shown by the method described in [8]. The results of comparing two methods with mentioned above constraint are presented in Table 4.

| Archive data | Current library | Average precision of similar functions search $P$ using proposed method $(k = 3)$ | Average precision of similar functions search $P$ using method presented in [8] |
|--------------|-----------------|---------------------------------|---------------------------------|
| libtiff 4.0.3 | libtiff 3.9.2   | 0.7990                          | **0.8218**                      |
| libtiff 4.0.3 | libtiff 4.0.8   | 0.8446                          | **0.8995**                      |
| libcurl 7.53.1| libcurl 7.49.0  | 0.8973                          | **0.9116**                      |
| libcurl 7.53.1| libcurl 7.51.0  | 0.9320                          | **0.9453**                      |

All the results of the experimental studies presented above were obtained using libraries compiled with the optimization flag /Od (optimization is disabled). The assembler code of these libraries significantly differs from the code of the corresponding versions of the same libraries compiled with the enabled optimization (flags /O1, /O2, /Ox). If the archive data is represented by a library compiled with disabled optimization and the current library is presented by a library compiled with enabled optimization, it is impossible to find similar functions using the method presented in this work. However, for example, the library functions compiled with the /O1 flag can be used to find similar functions in the library compiled with the /Ox optimization flag. Nevertheless, if the source code of the program is available, it is possible to compile it several times with all the possible options for optimizing and put the functions of each received instance of this program in the archive data library. Thus, similar functions can be found regardless of the compiler optimization settings of current library functions.

6. Conclusion
In this paper, the method of similar code sequences search based on functions control flow graph analysis is proposed. The results of experimental studies showed the proposed method's superiority over another previously known method. Further research will be carried out in improving the
efficiency of the presented method for functions with a small number of basic blocks in control flow graph.

7. References
[1] Ossher J, Sajnani H and Lopes C 2011 File cloning in open source Java projects: the good, the bad, and the ugly Proceedings of the International Conference in Software Maintenance (Williamsburg) VI 283-292
[2] Myles G and Collberg C 2005 K-gram based software birthmarks Proceedings of the 2005 ACM symposium on Applied computing 314-318
[3] Hemel A, Kalleberg K, Vermaas R and Dolstra E 2011 Finding software license violations through binary code clone detection Proceedings of the 8th International Working Conference on Mining Software Repositories 63-72
[4] Flake H 2004 Structural comparison of executable objects Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment 161-173
[5] Kruegel C, Kirda E, Mutz D, Robertson W and Vigna G 2005 Polymorphic worm detection using structural information of executables RAID’05 207-226
[6] Yumaganov A and Myasnikov V 2016 Similarity search over program code sequences using featureless pattern recognition techniques CEUR Workshop Proceedings 1638 437-443
[7] Yumaganov A and Myasnikov V 2017 Comparison of the ways of the program’s code initial description in the problem of similar code sequences search Procedia Engineering 201 445-452
[8] 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
[9] Hex-Rays IDA: About (Access mode: http://hex-rays.com/products/ida/) (09.11.2017)
[10] Khoo W, Mycroft A and Anderson R 2013 Rendezvous: a search engine for binary code Proc. of MSR’13
[11] Spath H 1981 The minisum location problem for the Jaccard metric Operations Research-Spektrum 3 91-94
[12] TIFF Library and Utilities (Access mode: http://www.libtiff.org/) (09.11.2017)
[13] Curl: command line tool and library (Access mode: http://www.curl.haxx.se/) (09.11.2017)
[14] Buckland M and Gey F 1994 The relationship between recall and precision JASIS 45(1) 12-19
[15] Powers D 2011 Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation Journal of Machine Learning Technologies 2(1) 37-63

Acknowledgments
This work was supported by the Federal Agency of scientific organization (Agreement 007-GZ/43363/26).