SIMILARITY SEARCH OVER PROGRAM CODE SEQUENCES USING FEATURELESS PATTERN RECOGNITION TECHNIQUES

A.S. Yumaganov, V.V. Myasnikov

Samara National Research University, Samara, Russia

Abstract. In this paper, we describe a problem of searching similar code sequences over binary executable program files. A proposed method is based on function opcode partitioning into functional groups, forming the function descriptions by comparing the executable code with the code of "base" library functions, and the dimensionality reduction of the resulting indirect description, which is used to run a search at the final step. We conducted an experimental study to show, that the proposed method is applicable for the problem solution, and to demonstrate its operability and efficiency.

Keywords: search, code sequence, featureless pattern recognition.

Citation: Yumaganov AS, Myasnikov VV. Similarity search over program code sequences using featureless pattern recognition techniques. CEUR Workshop Proceedings, 2016; 1638: 437-443. DOI: 10.18287/1613-0073-2016-1638-437-443

1 Introduction

With each passing day the amount of malicious software increases rapidly [1,2]. So, the malware detecting method, based solely on comprasion of the signatures for verifiable executable module, seems to be insufficient [3]. Due to the fact that most new viruses are a modification of the old ones, the effective method of analysis is the comparison of executable code with previously met [4]. Unfortunately, the direct comparison of code sequences turned out to be ineffective due to various reasons (various code compilation/optimization options, the virus code alteration and improvement and etc.). Thus, the development of the high-level code analysis methods is becoming an urgent problem, and the detection means, based on recognition techniques and data analysis, become more popular.
In this paper, the method of binary executable file functions search, similar to the known functions from some software "archive", is proposed. This method can be used directly for malware detection, by using the set of existing programs as software "archive". Due to the extreme complexity of obtaining the primary comprehensive executable code (graph) description, the proposed search method is based on the

principles of featureless pattern recognition: the function, we are interested in, is represented by its relationships (similarity/proximity) with some base library functions. Thus, the obtained redundant description is reduced using Principal Component Analysis (PCA) for the next search implementation.
The paper is organized as follows. In the first section we give a brief description of the proposed method. The second section is devoted to the method of constructing the function description via set of auxiliary functions (base library). The third section briefly describes the dimensionality reduction method, used to derive an ultimate function representation. In the fourth section we provide a similar function search algorithm based on the derived description. In the fifth section we provide the method effectiveness evaluation technique and the results of the experiments. The final part of the paper comprises the findings and the list of references.

2 Basic concepts and the proposed method

In this paper, the following definitions are used:

- *current library* – the set of the analyzed executable file functions;
- *archival data* – the set of known functions and their descriptions via base library;
- *base library* – an auxiliary set of functions, used to compare archival data functions with current library functions.

The problem can be informally stated as follows: to find the most similar archival data function for a given (or each) current library function. In general, the concrete definition of similarity (measures) can vary depending on different problems and solutions. In this papper, the proposed similarity measure is specified in the second section.
The process of solving the described problem can be divided into several stages. At the first stage, we represent the current library function(s) via (similarity) relationships with functions of a particular base library. At the second stage, the resulting description is reduced via PCA method. A reduced description is entered into the description database, which also stores archival data function descriptions. At the third stage, the search is actually implemented (via database tools), performing ordering of archival data function descriptions by the Euclidean distance criterion.
The presented method, consisting of the three stages, has a number of tunable parameters, which can be used to enhance the effectiveness of the proposed solution.

3 Base library function representation

After disassembly analysis of the executable code, we can get an assembler code partition into functions. Each function consist of set of processor instructions.
For convenience, we divide all processor instructions into K=45 functional groups [5]. Each group contains a set of instructions that perform one type of action. A set of data transfer instructions, a group of control transfer instructions are the examples of such

groups. Such partition lays down the requirements for the used base library: each functional group of instructions must have its representative in at least one base library function.

Let us consider the processor instructions inside of the given function. For each group of instructions of this function we build a (spatial) distribution of instructions, belonging to this group, via function length, as follows.

Let $n_0^k,...,n_{N_k-1}^k$ be the absolute offsets (positions) with respect to the start point of a function, belonging to the *k* group, N_k be the total number of instructions, be-longing to the group, inside the given function, and $N=\sum_{k=0}^{K-1} N_k$ be the total number of instructions inside the function, called the *function's length*. We define the spatial distribution of the *k* instruction type as absolute frequency of this type instruction occurrences 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 I(.) is an event indicator, taking values "0" or "1" depending on whether the corresponding argument is true or false.

To eliminate the effect of the small offsets of instructions in the code on the result of the search, we additionally use a kernel [6]. Then code instruction distribution estimation has the following form:

$$f_i^k=\sum_{j=1}^{N} f_j^k\,\frac{1}{h}K\left(\frac{i-j}{h}\right),\quad i=\overline{0,I}\,,$$

where $K(r)$ is a kernel function, *h* – window height. For example, we can use the Gaussian kernel as a kernel function:

$$K(r)=\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}r^2\right).$$

As a result, we get *K* mutually characterizing a particular function vectors of the following form:

$$\bar{a}_k=\left(f_1^k,f_2^k,\ldots,f_I^k\right)^T,k=\overline{0,K-1}\,.$$

The set of vectors mutually forms the description matrix:

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

A similar matrix, denoted as *C*, can be constructed for each function, including the base library functions.

The *similarity between two functions*, defined by matrices *A* and *C*, is measured as follows:

$$\mu(A,C)=\sum_{k=0}^{K-1}\alpha_k\mu_{\cos}(\bar{a}_k,\bar{c}_k)\ ,$$

where

$$\mu_{\cos}(\bar{a},\bar{c})=\frac{\sum_{i=1}^{I}a_ic_i}{\sqrt{\sum_{i=1}^{I}{a_i}^2}\sqrt{\sum_{i=1}^{I}{c_i}^2}}\ ,$$

and the values $\alpha_k \geq 0$, satisfying a condition

$$\sum_{k=0}^{K-1}\alpha_k=1,$$

are the method parameters.
In the case of complete similarity between functions the value of similarity measure is "1", in case of complete dissimilarity it is "0".
Let *J* be the number of the base library functions, each of which has a description in a form of a matrix B_j. Thus, we compare the investigated function of the current library with each base library function to get the following description vector of the function:

$$\bar{x}_A=\left(\mu(A,C_0),\mu(A,C_1),\dots,\mu(A,C_{J-1})\right)^T\ .$$

4 The description dimensionality reduction method

Since $J>>1$, we apply the *dimensionality reduction method*, namely, Principle Component Analysis [7]. This method include calculation of eigenvectors and eigenvalues of the covariance matrix of the corresponding vectors:

$$B=\mathrm{E}\{(\bar{x}-\mathrm{E}(\bar{x}))(\bar{x}-\mathrm{E}(\bar{x}))^T\}\ .$$

After sorting the eigenvalues of the covariance matrix in descending order, we arrange the corresponding eigenvectors in a similar manner. If we select only the $I<J$ largest eigenvalues, we will get the transition matrix *Y*, consisting of *I* eigenvectors. Thus, to obtain feature vector of size $I<J$ we use the following expression (an index, containing a reference to the original description, hereinafter omitted):

$$\bar{z}=Y\bar{x}\ . \tag{2}$$

Vector z serves *as the final representation of the investigated function via the set of the base library functions* and does not contain any elements of the primary description. This allows us to associate the proposed approach with featureless pattern recognition techniques [8,9].
The described method of the data description via the base library functions is applied for archival data function description, as well as for current library function description. The transition matrix *Y* is precalculated and remains unchanged.

5 Search for similar functions

The final stage of the proposed method is the search for similar functions via derived descriptions, represented as vectors (2).
Let $\bar{z}$ be the vector, describing the current library function representation via the base library, $\bar{z}^*$ be the vector, describing archival data function representation via the base library. For feature vector comparison we use the Euclidean metric (distance):

$$d\left(\bar{z},\bar{z}^*\right)=\sqrt{\sum_{i=0}^{I-1}(z_i-z_i^*)^2}\ , \qquad (3)$$

where z_i is the i th component of the first function feature vector, z_i^* is the i th component of the second function feature vector. If d = 0, the functions are considered equal.
Assume that, when function is modified, its size varies by no more than 25%. In this case, we use the following functions search algorithm, similar to the one under consideration:

1. First we determine the function length *N*.
2. Then we get the list of archival data functions, which size differs from the investigated function size by no more than 25%.
3. For each function from the list, obtained on the second step, we find the Euclidean distance to the investigated function.
4. Finally, we sort the results, obtained in the previous step, by ascending the *d* value.

As a result, we get the list of archival data functions, sorted by similarity in descending order (by distance in ascending order (3)), for the investigated function.

6 Results

To test the effectiveness of the proposed method of similar functions search we take functions of one library as archival data and functions of another version of the same library as a current library. To determine a priori similar (identical) functions we assume, that, when the library code is being modified, function names do not change, and there are no functions with the same name among archival data functions.

Under these assumptions, it is possible to evaluate the quality of recognition by getting the list of similar functions $\{F_l\}_{l=1,...,L}$, sorted by similarity in descending order. To do this, we associate this list of functions with binary sequence $\beta = (\beta_1, \beta_2,..., \beta_L)$, such that if there is a function with the same name on the l th position, then $\beta_l = 1$, else $\beta_l = 0$. Let us introduce the following measures, which are commonly used as quality criteria of information retrieval [10, 11].

1) Precision P_k for the k th position of the list :

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

is the number of functions with the same name as the investigated function on the first k positions of the ordered list as a proportion of the total number of functions in the list.

2) Recall (completeness) R_k for the k th position of the list:

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

is the number of functions with the same name as the investigated function on the first positions of the ordered list as a proportion of the total number of functions with the same name among archival data functions.

3) The average precision for the list:

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

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

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

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

To carry out research we put parameters (1) of the method equal to "1": $\alpha_k = 1 \left(k = \overline{0, K-1}\right)$.

The archival data are represented by the libtiff 4.0.3 library [12], and current library is represented by one of the following versions of the libtiff library: libtiff 4.0.4, libtiff 4.0.5, libtiff 4.0.6. Each library contains about 1200 functions in total; the code size is 498 Kb.

The results of the experimental study on the evaluation of the proposed method quality are shown in Table 1.

Table 1. The average recognition precision for different versions of the libtiff library

	libtiff 4.0.4	**libtiff 4.0.5**	**libtiff 4.0.6**
P	**0.8496**	**0.8479**	**0.8391**

On the basis of these results it can be concluded that the proposed method of search similar code sequences over executable files is efficient, as well as quite effective.

7 Conclusion

In this paper, the method of search similar code sequences over binary executable program files is proposed. The method is based on function description construction by comparing the executable code with the code of a “base library” functions, dimensionality reduction of the resulting indirect description, which is used in the final step to perform the search. The results of the experimental study demonstrate the efficiency and effectiveness of the proposed method.
Further research will be carried out in:

– parametric optimization of the proposed method;
– development of proximity measures, based on structural joint analysis of function performance graphs.

References

1. 2015 Internet Security Threat Report: Attackers are bigger, bolder, and faster. URL: http://www.symantec.com/connect/blogs/2015-internet-security-threat-report-attackers-are-bigger-bolder-and-faster.
2. PandaLabs identifies 227,000 malware samples per day in the first quarter of 2016. URL: http://www.pandasecurity.com/mediacenter/news/pandalabs-study-q1/.
3. Umakant Mishra, Methods of Virus Detection and Their Limitations. URL: http://ssrn.com/abstract=1916708.
4. IT threat evolution in Q1 2016. URL: https://securelist.com/analysis/quarterly-malware-reports/74640/it-threat-evolution-in-q1-2016/.
5. x86 Assembly Language Reference Manual. URL: https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf .
6. Fukunaga K. Introduction to statistical pattern recognition. New York and London: Academic Press, 1972.
7. Alpaydin E. Introduction to Machine Learning. MIT, 2nd edn., 2010.
8. Vapnik V. An Overview of Statistical Learning Theory. Neural Networks, IEEE Transactionson, 1999; 10(5): 988-999.
9. Kuznetsov AV, Myasnikov VV. A comparison of algorithms for supervised classification using hyperspectral data. Computer Optics, 2014; 38(3): 494-502. [in Russian]
10. Buckland MK, Gey FC. The relationship between recall and precision. JASIS, 1994; 45(1): 12-19.
11. Powers DMW. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. Journal of Machine Learning Technologies, 2011; 2(1): 37-63.
12. LibTIFF – TIFF Library and Utilities. URL: http://www.libtiff.org/.