Metamorphic Malware Detection using Control Flow Graph Mining

Mojtaba Eskandari[†] and Sattar Hashemi[††],

[†]APA Malware Research Center, Shiraz University, Iran
[††] Department of Computer Science & Engineering, Shiraz University, Iran

Summary
Metamorphic malware propagation has persuaded the security society to consider about new approaches to confront this generation of malware with novel solutions. Control Flow Graph, CFG, has been successful in detection of simple malwares. By now, it needs to improve the CFG based detection methods to detect metamorphic malwares efficiently. Our Approach has improved the simple CFG with beneficial information by assuming called APIs on the CFG. Converting the resulted sparse graph to a vector to decrease the complexity of graph mining algorithms, a specific feature selection is utilized and different classification approaches has been qualified. The experimental results show the contribution of this approach in both accuracy and false detection rate measurements in comparison with the other simple graph modifications. Among different classifiers on our approach the best results were attained by random forest. On the computation complexity side also this work has decreased the elaboration regarding to the simple feature selection conducted before decision making.

Key words:
malware, detection, metamorphic, obfuscated, PE-file, CFG, API, ACFG.

1. Introduction

In recent years, many researchers have focused on data mining methods for detecting unknown malwares. Data mining methods use statistical tools and machine learning algorithms, and by applying them on a set of features, can determine the malicious programs from benign programs. Most of data mining methods start their process by generating a feature set. These features, sometimes, can be n-grams, instruction sequences, API call sequences etc. Signature based approaches need signature of each seen malware to detect it. Therefore, detecting unknown malwares is not possible with classic static signature based methods, so we need semantic based detection methods, to detect them. One of the semantic based approaches uses control flow graphs to understand the semantic of malwares, whereas, we use control flow graph and API calls to have more semantical aspect of new malwares, hence can detect them.

The structure of this paper as follows. Section 2 describes related works in semantic based malware detection. Section 3 discusses the proposed method, an overview of our malware detection system, feature selection methods and classification algorithms used in experiments. The system evaluation is presented in Section 4 that results are interpreted and commented in the same section. Section 5 concludes our achievements, future works and summarizes the results.

2. Related Work

Gao et al. represented a new method to find semantic differences in binary programs. Syntactic differences indeed have the potentiality to cause noise, so finding the semantic differences would be challenging. They utilized a new graph isomorphism technique and symbolic execution to analyze the control flow graph of PE-files by identifying the maximum common sub graph. A PE-file is the standard executable (EXE) file format used by the Microsoft Windows operating systems. Their method found the semantic difference between a PE-file and its patched version, but with significantly worse execution speed [1].

Cesare and Xiang proposed a new classification method that used flow graphs to detect polymorphic malwares. They applied a heuristic algorithm for the flow graph matching to find graph isomorphisms, therefore, they can estimate similarity between PE-files, and finally they represented a classification algorithm based on their method [12].

Jeong and Lee, extract the control flow graph by the instructions with API calls as code graph. They did not use any disassemble tools; instead, they worked on binary files directly and fetched the jump and call instructions from that PE-file and created the code graph. Their method can detect 67% of unknown malwares [6].

Dullien and Rolles, assumed each function as a flow graph, and then drew call relation between them (i.e. from caller node to callee); consequently each PE-file is represented as a graph of graphs. Under this trend, they could have an improved isomorphism between the sets of basic blocks and sets of functions in two disassembled PE-files; therefore, they could detect code theft [13].

Manuscript received December 5, 2011
Manuscript revised December 20, 2011

Abadi et al. formalized the semantic of Control Flow Integrity (CFI) by machine code rewriting. CFI represents a method to verify the execution proceeds within a given control flow graph which is derived from static program analysis. In the other words, CFI relies on dynamic checks for enforcing control flow integrity and has an efficient implementation of a program shadow call stack with high protection [9].

3. The Proposed Approach

First, the system disassembles the PE-file then extracts the CFG from assembly file. After that, the system generates the API call graph with CFG and called APIs. In the final phase, system uses a classification algorithm to make a decision whether the PE-file is malicious or not.

3.1 System overview

Our system consists of three principal components, PE-file disassembler, API call graph generator and classification module that is illustrated in Figure 1. In the first step, the system disassembles the PE-file. After that, the system performs a preprocessing algorithm on assembly files, removing not necessary statements, and generates the control flow graph (CFG) from these instructions. A CFG is a connected and directed graph consisting of a set of vertices corresponds to the lines of disassembled file, a set of directed edges that corresponds to the execution sequence of program (e.g., normal sequence, conditional jump, unconditional jump, function call and return instruction), concurrently in the API repository, each API name is mapped to a global unique number. In the next steps, system performs the labeling algorithm on CFG and consequently API call graph would be generated. The graph is then converted into a feature vector. In the next step, we used a feature selection algorithm, because the number of generated features is very high and with this step we can reduce the number of features, therefore, we have a faster classification process. After feature selection, we get some of data items as training set and use those for the training phase, then the classifier generates a set of rules which are saved in the rule database. Finally, the testing set is given to decision module. This module utilizes rule database to decide whether a sample is malware or not. Therefore, it generates a report for each PE-file.

3.2 Generating API call graph

Based on our motivation, feature generation and feature selection are most significant phases in malware detection systems. Therefore, the key step of our presented approach focuses on these phases. According to Figure 1, the disassembling step uses PE-Explorer [10], for disassembling the PE-files, this tool cannot disassemble a number of files in batch mode, hence, the user must disassemble the PE-files one by one, so we made an auxiliary tool, that follows users' interactions with PE-Explorer and then disassembling process for the huge dataset with PE-Explorer becomes more simplified. After disassembling step, the assembly file preprocessing starts, this step removes unnecessary assembly instructions from input assembly file, the necessary instructions are as followed: jump instructions, procedure calls, API calls and all lines which are targets of jump instructions. This step makes two lists, first, the list of needed instructions for making the control flow graph and the second, a list of called APIs, for API repository. In the next step, the system extracts the control flow graph of preprocessed assembly file. After that system creates the API call graph. In this step each API call graph consists of a control flow graph with a set of labels on its edges. Each label represent called API number that called on that situation.

3.3 Selecting Appropriate Features

The isomorphism problem of graphs is a well known NP-complete problem, so we present a method to convert the API call graph to a feature vector; in this method we use a suitable property of API call graph. The API call graph is a sparse graph, and can be shown as a sparse matrix, on the other hand, we can convert a sparse matrix to a vector, saving only situation of nonzero items, we can obtain situation of nonzero items with Equation (1).

$$S_m = (i-1) * n + j \tag{1}$$

where i is the row number, j is column number, n is the number of nodes in API call graph and S_m is situation number of item $[i , j]$ in sparse matrix that is saved in m^{th} situation of list S.

So we can convert each API call graph to a feature vector, as illustrated in Figure 2. In this feature vector, each item represents an edge and possibly an API call. We have a problem, we cannot use data mining methods for these feature vectors right now, because the number of nodes in different API call graphs are different, therefore, obtains different S_m for same $[i , j]$ in different API call graphs. To solve this problem, we must resize all API call graphs to a fix same size, hence, we use the size of largest graph to satisfy this constraint. This is called padding. When we use a general n, in the Equation (1), the results are meaningful, for example, in Figure 2(c) number *20*, in all graphs has same semantic.

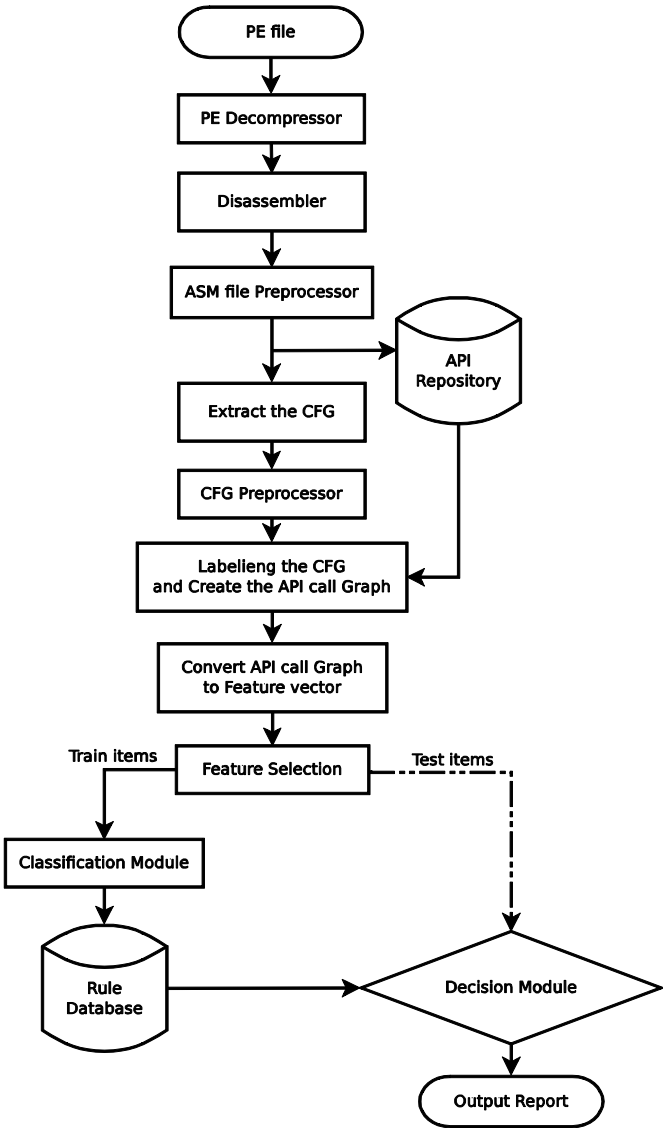


Fig. 1 System perspective. The detection process which is shown here, includes the following phases, control flow graph extraction, API call graph generation, converting graph to feature vector, feature selection and in the final phase, system can performs any classification algorithms to decide whether the input file is malware or not.

This process is represented in Algorithm 1. After this process, the system finds all distinct edges in API call graphs and use them as feature, then generates a nominal dataset, consists of API number as data-item and edge number as feature. The feature selection step of system uses best-first weight features the weight of each feature obtains with Equation (2).

$$W_k = \sum_{i=0}^{n} (\xi_{k,i} + \delta_{k,i}.\eta) \qquad (2)$$

where W_k is k^{th} feature weight, n is the number of data items, η is API weight coefficient that would be obtained in experiments, $\xi_{k,i}$ denotes the profitable function for k^{th} feature that shows in Equation (3) and $\delta_{k,i}$ represent the API exist function that shows in Equation (4).

$$\xi_i = \begin{cases} +1 & \text{if } \theta = \gamma \\ -1 & \text{if } \theta \neq \gamma \end{cases} \qquad (3)$$

where θ is the value of the feature and γ is the class label of i^{th} data item. Notice that, in our system, the class label of malwares is *1* and the class label of benign files is *0*.

$$\delta_i = \begin{cases} 1 \text{ if have an API on this feature of i}^{\text{th}} \text{ data item} \\ 0 \text{ Otherwise} \end{cases} \qquad (4)$$

Algorithm 1. Convert API call graph to feature vector.

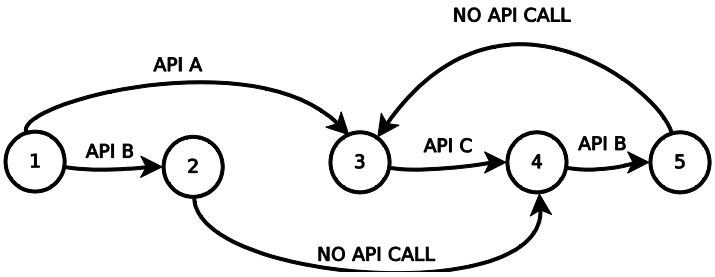
```
INPUT:
   APIGraph an API call graph;
   N is an integer number that represents
   the number of vertices of the biggest
   graph;
OUTPUT:
   Vector is an empty queue;

1: For each vector V_i in APIGraph do
2:   For each outcome edge E_ij of V_i do
3:     Value_ij = (label of V_i - 1) × N + (label
       of target vertex of E_ij );
4:     API_ij = label of E_ij ;
5:     Add pair <Value_ij , API_ij > to Vector;
6:   End for;
7: End for;
```

3.4 Malware detection models

A decision stump is a machine learning model consisting of a one-level decision tree [15]. That is, it is a decision tree with one internal node (the root) which is immediately connected to the terminal nodes. A decision stump makes a prediction based on the value of just a single input feature.

Sequential minimal optimization (SMO) is an algorithm for solving large quadratic programming (QP)



(a) An API call graph.

	1	2	3	4	5
1	0	B	A	0	0
2	0	0	0	1	0
3	0	0	0	C	0
4	0	0	0	0	B
5	0	0	1	0	0

(b) Adjacency matrix of the graph.

(2, B), (3, A), (9, No), (14, C), (20, B), (23, No)

(c) Feature vector of that graph represents as sorted pair of edges and API calls.

Fig. 2 Converting API call graph to feature vector by Algorithm 1.

optimization problems, widely used for the training of support vector machines. SMO breaks up large QP problems into a series of smallest possible QP problems, which are then solved analytically [4].

A naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature.

Random forest is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees [7].

A random tree is a tree or arborescence that is formed by a stochastic process. Different types of random trees include uniform spanning tree, random minimal spanning tree, random binary tree, random recursive tree, treap or randomized binary search tree, rapidly exploring random tree, Brownian tree, random forest and branching process [14].

K-Star is an instance-based classifier, which is the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function. It differs from other instance-based learners in that it uses an entropy based distance function [5].

4. Experimental Results

4.1 Datasets in brief

The intent of most malwares is to infect PE-files, therefore, this study focuses on PE-files and analyzing malware that infect them. Essentially malware researchers used an imbalanced dataset in their experiments in which number of malwares is much more than the number of benign files [2, 16, 3]. We thought, however, that the above assumption does not stand in the real world domain, in which numbers of malicious binaries are at most as much that of benign files. Clearly, it can be conclude that those methods, in which we have a considerable amount of malicious binaries compared to the benign ones, provide better accuracy. According to this thought we collect 2,140 benign Windows PE-files and select 2,305 Windows 32-bit network worms, randomly, from malware repository of APA malware research center at Shiraz University.

4.2 Evaluation measures

We define "Detection Rate" as the percentage of all PE-files labeled "malicious" that can receive correct label by the system, as illustrated in Equation (5).

$$DetectionRate = \frac{TP}{TP + FN} \quad (5)$$

The "False Alarm Rate" is the percentage labeled "normal" that likewise receive the wrong label by the system, as illustrate in Equation (6).

$$FalseAlarmRate = \frac{FP}{TN + FP} \quad (6)$$

The "Accuracy" is the overall accuracy of the system to detect malwares and benign files, as illustrate in Equation (7).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

The "cross validation", is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. One round of cross validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset that called the training set, and validating the analysis on the other subset that called the testing set. To reduce variability, multiple rounds of cross validation are performed using different partitions, and the validation results are averaged over the rounds, we called each round as a fold [11].

The "ROC curve", is a graphical plot of true positive rate, versus false positive rate, for a binary classifier system as its discrimination threshold is varied. The area under the "ROC curve", called "AUC". The "AUC" is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one [8].

4.3 Observations and analysis

We illustrate the different detection rates and false alarm rates over different classification models in tabular form, as illustrate in Table 1. These results obtained with 10 fold cross validation test. The results indicate that the method based on API call graph classification provides excellent detection rate and lower false alarm rates than rival approaches proposed for unknown malwares. The "AUC" helps us to choose the best classifier in the classification phase. We find out with the "Random Forest" can have the best detection rate and the lowest false alarm rate among other experimented models.

Table 1. Experimental results on mentioned dataset with different classifiers.

	Detection rate	False alarm rate	Accuracy	AUC
Decision Stump	93.81%	04.95%	94.42%	92.29%
SMO	89.74%	08.20%	90.76%	87.23%
Naive Bayes	95.23%	03.81%	95.70%	94.06%
Random Tree	91.11%	07.11%	91.99%	88.94%
Lazy K-Star	96.61%	02.71%	96.94%	95.78%
Random Forest	97.53%	01.97%	97.77%	96.92%

In the last experiment, we can detect 97.53% of unknown malwares, whiles Jeong and Lee in [6] can detect only 67% of unknown malwares by their method.

5. Conclusions and Future Work

By using control flow graph, we are able to have a semantic aspect of the obfuscated or mutated PE-files, which can detect them with mining in these graphs. The CFG based approach improves the current approaches for detecting malwares by adding a behavioral attribute of malicious files into the detection model. On the other hand, as mentioned, the isomorphic problem in graphs is NP-complete and hence a time consuming problem, therefore, we proposed a method to simplifying the control flow graphs. The proposed method converts each control flow graph to a feature vector, to have simpler data items and to have easier process in vector space. The number of edges in control flow graph are often too large, accordingly the number of features in vector space model, would be too large. To alleviate the problem, we utilized a feature selection in our approach. As a future work, we intent to study on imbalanced datasets that number of malwares are much less than the number of benign files.

References

[1] D. Gao, M. K. Reiter, and D. Song, 2008, "BinHunt: Automatically Finding Semantic Differences in Binary Programs," Technical report, School of Information Sciences, Singapore Management University.

[2] G. Bonfante, M. Kaczmarek, and J. Marion, 2007, "Control Flow to Detect Malware," Inter-Regional Workshop on Rigorous System Development and Analysis.

[3] G. Bonfante, M. Kaczmarek, and J. Marion, Sep 2008, "Architecture of a morphological malware detector," Journal of Computer Virology.

[4] J.C. Platt, 1999, "Fast training of support vector machines using sequential minimal optimization," In Advances in kernel methods, pp. 185-208, MIT Press, Cambridge, MA, USA.

[5] J.G. Cleary and L. E. Trigg, 1995, "K*: An Instance-based Learner Using an Entropic Distance Measure," In 12th International Conference on Machine Learning, 108-114.

[6] K. Jeong and H. Lee, 2008, "Code Graph for Malware Detection," Information Networking, ICOIN 2008, International Conference on, pp.1-5.
[7] L. Breiman, 2001, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5-32.
[8] L.E. Dodd and M.S. Pepe, 2003, "Partial AUC Estimation and Regression," Biometrics, pp. 614-623.
[9] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, Nov 2005, "Control-Flow Integrity: Principles, Implementations, and Applications," In Proceedings of the ACM Conference on Computer and Communications Security (CCS).
[10] PE-Explorer, 2011, www.pe-explorer.com/peexplorer-download.htm (version 1.99 R6).
[11] R. Picard and D. Cook, 1984, "Cross-Validation of Regression Models," Journal of the American Statistical Association, pp. 575-583.
[12] S. Cesare, and Y. Xiang, 2010, "A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost," Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, pp. 721-728.
[13] T. Dullien and R. Rolles, Jun 2005, "Graph-based comparison of Executable Objects," In Symposium sur la Securite des Technologies de l'Information et des Communications (SSTIC).
[14] T.G. Dietterich, 2000, "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," Machine Learning, vol. 40, no. 2, pp. 139-157.
[15] W. Iba and P. Langley, 1992, "Induction of One-Level Decision Trees," Proceedings of the Ninth International Conference on Machine Learning.
[16] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, Feb 2008, "An intelligent PE-malware detection system based on association mining," Journal of Computer Virology.