Detecting Malwares Using Dynamic Call Graphs and Opcode Patterns

K.P. Deepta(✉) and A. Salim

College of Engineering Trivandrum, Thiruvananthapuram, Kerala, India
deeptakp6@gmail.com, salim.mangad@gmail.com

Abstract. Classification and detection of malware includes detecting instances and variants of the existing known malwares. Traditional signature based approaches fails when byte level content of the malware undergoes modification. Different static, dynamic and hybrid approaches exist and are classified based on the form in which the executable is analyzed. Static approaches include signature based methods that uses byte or opcode sequences, printable string information, control flow graphs based on code and so on. Dynamic approaches analyze the runtime behavior of the malwares and constructs features. Hybrid methods provide an effective combination of static and dynamic approaches. This work compares the classification accuracy of static approach that employs opcode sequence analysis and dynamic approach that uses the call graph generated from the function calls made by the program and an integrated approach that combines both these approaches. Integrated approach shows an improvement of 2.89% than static and 0.82% than dynamic approach.

Keywords: Dynamic analysis · Integrated approach · Malware · Static analysis

1 Introduction

Malware or malicious software is any software which has the potential to damage the information stored in a computer. There are different categories of malware including viruses, worms, spyware and so on. There has been a profound increase in the number of malicious samples due to the deployment of morphing techniques, which prevents the anti-malware software from detecting them. Researchers and anti-malware vendors face the challenge of how to detect previously unseen malware (also called zero-day attack).

Malware detection involves identification of both instances and variants of the existing malware. For this, it is essential to observe and study the organization of the malicious code and its behavior. This gives an understanding about the nature of infection caused by the code and thus identifies similar ones.

1.1 Basics of Malware Analysis

Malware analysis can be considered as an art of dissecting the malware to find out the way it works, methods to identify it and how to trash and wipe it out. Malware analysis is critical area as it is a threat to security of computer systems.

1.1.1 Static Analysis

Static analysis consists of examining the executable file without viewing the actual instructions. It can be applied on different representations of a program like binary code, source code etc. It can be used to find memory corruption issues and prove the correctness of models if the source code is available.

Some of the techniques used for static malware analysis are file fingerprinting, extracting hard coded strings, file format inspection, disassembling of machine code etc.

Islam et al. presented an automatic malware classification method based on function length frequency and Printable String Information [1]. Their results demonstrated that a combined approach of string and function length features into a single test provides a superior result than they were used individually. Even though these features are easier to collect, the use of static features alone, fails when obfuscation and packing is performed.

Santos et al. proposed a malware detection system that uses semi-supervised learning to detect malicious programs [2]. This method is useful when a limited amount of labeled data exists for benign and malware classes. N-gram distribution is used to represent the executables. The main contribution of this paper is the reduction in the number of required labeled instances while maintaining a good precision of above 90%. However, due to the use of static nature of features, the system will be unable to counter packed malware.

Ye et al. proposed a detection system based on static analysis to generate signatures for clustering malwares [3]. Two features have been considered here, namely Instruction Frequency and Function-based sequences. A Hybrid hierarchical clustering algorithm which combines the advantages of hierarchical clustering and k-medoids algorithm was used to cluster the malwares.

Cesare et al. has proposed a static method of malware detection [4]. Initially the system unpacks the executables and disassembles the code. Malware signature is generated based on the set of control flow graphs (from high level source code) the malware contains. Feature vector is a decomposition of the set of graphs into either fixed size k-subgraphs, or q-gram strings. Similarity with the known malwares is computed using string distance matching methods like edit distance. This method is more efficient than signature based methods as the control flow overcomes the limitations of byte level and instruction level classification but cannot properly handle packing and obfuscation issues. The major drawback of the approach is that it considers the static control flow of the program and this can be easily bypassed by the code obfuscation techniques.

The key advantage of static malware analysis is that it permits a comprehensive analysis of a given binary i.e., it can cover all possible execution paths of a malware sample. Additionally, as the source code is not actually executed, static analysis is generally safer than the dynamic analysis. But the drawback is that static analysis is usually conducted manually and thus consumes time and requires expertise.

1.1.2 Dynamic Analysis

Executing a given malware sample within a controlled environment and monitoring its actions to analyze the malicious behavior is called dynamic malware analysis. Zhao et al. proposed a vector space model where API sequences were translated into features [5].

Another approach is a graph based model where API calls and OS resources are represented as graph nodes and edges representing reference between them [6]. The Graph Edit Distance algorithm was used to find the match between different graphs.

Anderson et al. proposed a malware detection algorithm that analyzes the graphs constructed from dynamically collected instruction traces of that executables [7]. The nodes of the graph represent instructions and the transition probabilities are estimated by the data in the trace. Using the concept of graph kernels, similarity matrices between instances in training set were constructed. Gaussian kernel and spectral kernel were the two measures used to construct kernel matrix.

Borojerdi and Abadi proposed MalHunter, which is a method for generating behavioral signature [8]. Sequence-based clustering algorithm is based on the Basic Sequential Algorithmic Scheme (BSAS). The steps involved in generating the signature were identifying semantic behaviors, clustering the behavior sequences, generalizing these sequences and generating multiple behavioral signatures.

As the analysis and detection is during runtime and malware unpacks itself, dynamic malware analysis avoids the restrictions of static analysis like unpacking and obfuscation issues. The main drawback is the issue of dormant code. Also if the analysis environment is not properly managed, the system itself may get damaged. Furthermore, malware samples may change their behavior or stop the execution when they detect that execution is taking place in a controlled analysis environment.

Observing the runtime-behavior of an application is currently the most promising approach. It is mostly conducted utilizing sandboxing. A sandbox refers to a controlled runtime environment which is partitioned from the rest of the system in order to isolate the malicious process. This partitioning is typically achieved using virtualization mechanisms on a certain level.

1.1.3 Integrated Approach

Integrated approaches are the combination of static and dynamic malware analysis techniques. Sharma proposed a combination of dynamic representation of program calling structures, with a static analysis applied to a region of that structure with observed performance problem [9]. Suspicious behavior showing portions are taken as signatures and searched for similar patterns. This was a theoretical work and does not contain any experimental results for evaluating the actual performance.

Nguyen et al. proposed a method to identify real target of an indirect jump [10]. This work was aimed at reducing the false target identification that could happen when processing such jumps and during CFG construction. The framework consists of two phases: Static and Dynamic. In the static phase, the program was divided into regions and sub-CFGs were generated. During the dynamic analysis Intermediate labeled transition system is generated and test cases are executed and then the CFG is updated, followed by first step and this continues.

2 Proposed Architecture

Figure 1 shows the proposed architecture of the malware detection system. It consists of a static, dynamic and integrated approach for fast and accurate detection of malware.

2.1 Static Feature Extraction

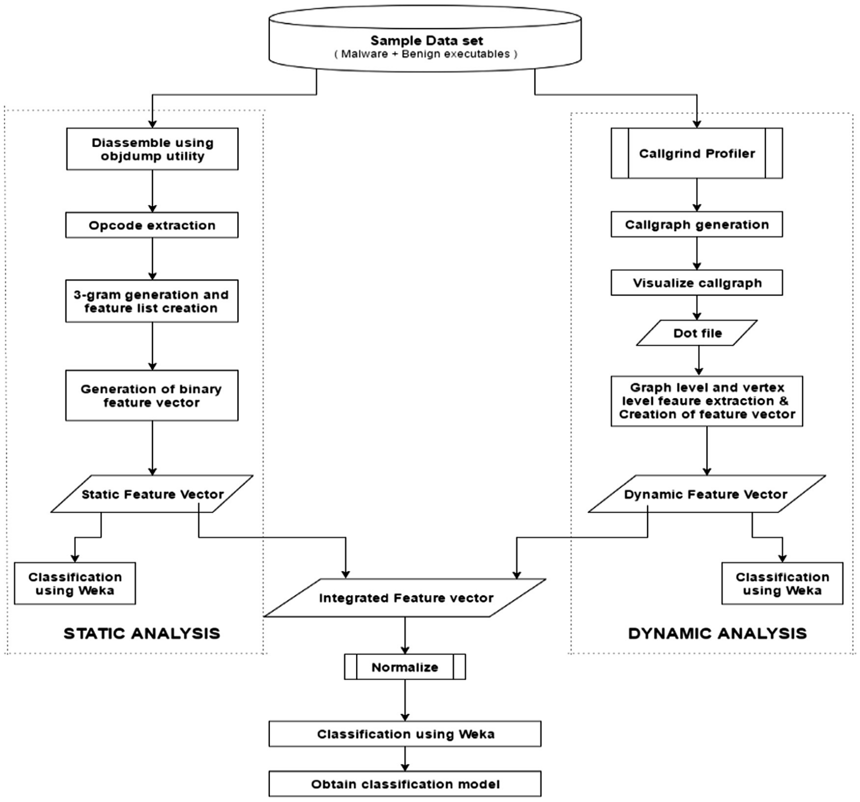
Sample Data set
(Malware + Benign executables)
Diassemble using objdump utility
Opcode extraction
3-gram generation and feature list creation
Generation of binary feature vector
Static Feature Vector
Classification using Weka
STATIC ANALYSIS
Callgrind Profiler
Callgraph generation
Visualize callgraph
Dot file
Graph level and vertex level feaure extraction & Creation of feature vector
Dynamic Feature Vector
Classification using Weka
DYNAMIC ANALYSIS
Integrated Feature vector
Normalize
Classification using Weka
Obtain classification model

Fig. 1. Proposed architecture of malware detection method

2.1.1 Disassemble the Executable

A disassembler is a computer program that translates machine language into assembly language. Objdump command in Linux is used to disassemble the codes.

2.1.2 Feature Vector Formation

The disassembled code contains the opcode sequence of the program. Opcodes from the disassemble input has been separated using a python program and 3-gram patterns were generated using the Linux utility text2ngram.

Initially all the opcodes from the disassembled code were extracted and frequency of every possible 3-gram sequence were computed. The 3-grams with frequency above a threshold were filtered from both malware and benign samples. To get a feature vector that represent malware family more precisely, we extracted all frequent 3-grams and filtered out that are common in both classes. Various length feature vectors (size 10, 20, 50, 80 and 100) were considered. The feature vector consists of strings of 1 s and 0 s depending on the presence of 3-gram patterns. This was done for each pattern and feature vectors corresponding to each sample was generated. Algorithm 1 describes the steps in Static feature extraction.

```
Algorithm 1 static feature extraction
       S : Sample set containing malware and benign files
       B : Binary  feature vector
      begin
      for each $s_i$ in S do
            Extract all the opcodes in $s_i$;
            Generate all 3-grams from opcodes;
            Add each unique 3-gram to a csv file $f_{csv}$;
      for each 3-gram in $f_{csv}$ do
            counter :=0;
            for each $s_i$ in S do
                  Increment counter if 3-gram is present
                  in $s_i$ and store 3-gram and its counter
                  in file $f_c$;
      Sort $f_{csv}$ based on frequency of 3-grams;
      Remove 3-gram having frequency < Threshold
      Store the new list in file $f_c$;
      i=0;
      for each $s_i$ in S do
            i = i+1;
            for each 3-gram in $f_c$ do
                  k = 0;
                  if 3-gram present in $s_i$ then
                        $b_{ik}$ :=1;
                  else
                        $b_{ik}$ :=0;
                  k = k+1;
                  //Each bi is a feature vector
            Tabulate the binary feature vectors of all $s_i$
            in S and Input this to classifiers;
      end.
```

2.2 Dynamic Feature Extraction

The core idea is to make malware detection system more resilient to byte- and instruction-level modifications and obfuscations. A call graph derived from the binary code provides a reasonable approximation of the program's run time behavior. So we translate the problem of finding similar graphs into that of extraction of different features of the graph and analyze the similarities in them.

The executable files were analyzed using *Callgrind*, a profiling tool that records the call history among functions in a program's run as a call-graph. By default, the collected data consists of the number of instructions executed, their relationship to source lines, the caller/callee relationship between functions and the numbers of such calls.

We used Callgrind to analyze the executables. This is a profiling tool which records the call history of functions in a program during execution as a call-graph. The output generated by Callgrind was visualized using KCachegrind. Figure 2 shows a section of a sample call graph. The nodes of the graph contain the function name, its memory location, number of instructions and cost and the edges represent time consumed. The graph needs to be exported to Dot format for further analysis. The following features were extracted from graph corresponding to each of the sample. We ranked the features based on the information gain and Table 1 shows the features in descending order of ranks.

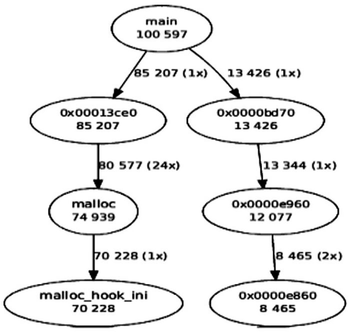

Fig. 2. A section of a sample call graph

2.3 Classifier Model

Dataset consisted of 787 malicious and 425 benign samples. Malware samples were collected from malware repositories Virusshare [13] and AVCaesar [14] and benign samples were obtained from Windows system directory. BayesNet and RandomForest classifiers were chosen to implement the classification model. BayesNet has the capability to find interdependencies between different attributes. Unlike decision tree, in Random forest the best parameter at each node in the tree is selected from a randomly chosen set of features. This helps Random Forest to perform well and makes it less vulnerable to noise in the data. An open source machine learning tool, Weka was used to implement the models [12].

Table 1. Ranking of features extracted from graph

Rank	Feature	Description
1	Network diameter	Longest graph distance between any 2 nodes. ie. How far apart are the two most distant nodes?
2	Average path length	Average number of steps along the shortest paths for all possible pairs of nodes
3	Average weighted degree	Ratio of sum of the in- and out-degree of all nodes to the number of vertices
4	Average degree	Ratio of sum of the in-degree of all nodes to the number of vertices
5	Modularity	Measures the density of links inside communities as compared to links between communities
6	Graph density	How close network is to complete? Ratio of number of edges to the possible number of edges
7	Strongly connected components	A directed graph is strongly connected if there is a path between all pair of nodes. A strongly connected component of a directed graph is a maximal strongly connected subgraph
8	Edge count	Number of edges
9	Node count	Number of nodes
10	Average clustering coefficient	Indicates how nodes are embedded in their neighbourhood
11	Weakly connected components	

3 Experimental Results and Discussion

Performance of the classification models were evaluated using measures like True Positive Rate (TP Rate), False Positive Rate (FP Rate), Precision, Recall, F-Measure (Harmonic mean of precision and recall), ROC Area (ROC curve is the curve created by plotting TPR against FPR at various threshold settings) and Accuracy.

3.1 Static Approach

Tables 2 and 3 show the classification results obtained using RandomForest and BayesNet classifiers respectively. Experiments have been repeated with varying number of static features from 10 to 100. Classification results of static analysis using RF classifier indicate that the features do not show much variation in accuracy between lengths 20 and 50. 95.87% is the best accuracy observed from RF classifier when feature length was 20.

Table 2. Weighted average of classification results with RandomForest classifier

Feature vector length	TP rate	FP rate	Precision	Recall	F-Measure	ROC area	Accuracy (%)
10	0.941	0.092	0.942	0.941	0.941	0.952	94.14
20	0.959	0.06	0.959	0.959	0.959	0.973	95.87
30	0.959	0.06	0.959	0.959	0.959	0.983	95.87
40	0.957	0.063	0.957	0.957	0.957	0.983	95.71
50	0.957	0.062	0.957	0.957	0.957	0.985	95.71
80	0.95	0.075	0.951	0.95	0.95	0.964	95.05
100	0.945	0.086	0.945	0.945	0.944	0.956	94.47

Table 3. Weighted average of classification results with BayesNet classifier

Feature vector length	TP rate	FP rate	Precision	Recall	F-Measure	ROC area	Accuracy (%)
10	0.87	0.145	0.871	0.87	0.87	0.916	86.96
20	0.823	0.183	0.83	0.823	0.825	0.889	82.26
30	0.771	0.227	0.787	0.771	0.775	0.869	77.06
40	0.744	0.237	0.771	0.744	0.75	0.854	74.42
50	0.732	0.24	0.766	0.732	0.738	0.849	73.18
80	0.721	0.253	0.755	0.721	0.727	0.833	72.11
100	0.716	0.258	0.75	0.716	0.723	0.826	71.62

3.2 Dynamic Approach

We conducted experiments with complete set of dynamic features and Tables 4 shows the classification results obtained using RF classifier with dynamic approach. Dynamic feature analysis shows an improvement of 1.65% over static feature analysis with RF classifier.

Table 4. Weighted average of classification results with RFclassifier

Classifier	TP rate	FP rate	Precision	Recall	F-Measure	ROC area	Accuracy (%)
Random Forest	0.975	0.042	0.976	0.975	0.975	0.97	97.52
Bayes Net	0.979	0.038	0.98	0.979	0.979	0.962	97.93

3.3 Integrated Approach

We conducted experiments by taking complete set of dynamic features and varying number of static feature lengths. Tables 5 and 6 shows the classification results obtained using RF and BayesNet classifiers using integrated approach. Figure 3 depicts the variation in ROC curves obtained from RF classifier with change in static feature vector length. Classification results obtained after integrating the features obtained from

Table 5. Weighted average of classification results with RFclassifier

Feature vector length	TP rate	FP rate	Precision	Recall	F-Measure	ROC area	Accuracy (%)
10	0.979	0.034	0.979	0.979	0.978	0.993	97.85
20	0.982	0.027	0.982	0.982	0.982	0.983	98.18
30	0.985	0.026	0.985	0.985	0.985	0.993	98.51
40	0.985	0.024	0.985	0.985	0.985	0.995	98.51
50	*0.988*	*0.019*	*0.988*	*0.988*	*0.988*	*0.997*	*98.76*
80	0.987	0.019	0.987	0.987	0.987	0.998	98.68
100	0.985	0.9024	0.985	0.985	0.985	0.998	98.51

Table 6. Weighted average of classification results with BayesNetclassifier

Feature vector length	TP rate	FP rate	Precision	Recall	F-Measure	ROC area	Accuracy (%)
10	0.978	0.034	0.978	0.978	0.978	0.992	97.77
20	0.971	0.042	0.971	0.971	0.971	0.988	97.11
30	0.975	0.036	0.975	0.975	0.975	0.991	97.52
40	*0.979*	*0.04*	*0.979*	*0.979*	*0.978*	*0.993*	*97.85*
50	0.967	0.041	0.967	0.967	0.967	0.985	96.70
80	0. 912	0.075	0.918	0.912	0.913	0.975	91.17
100	0. 869	0.101	0.886	0.869	0.871	0.969	86.88

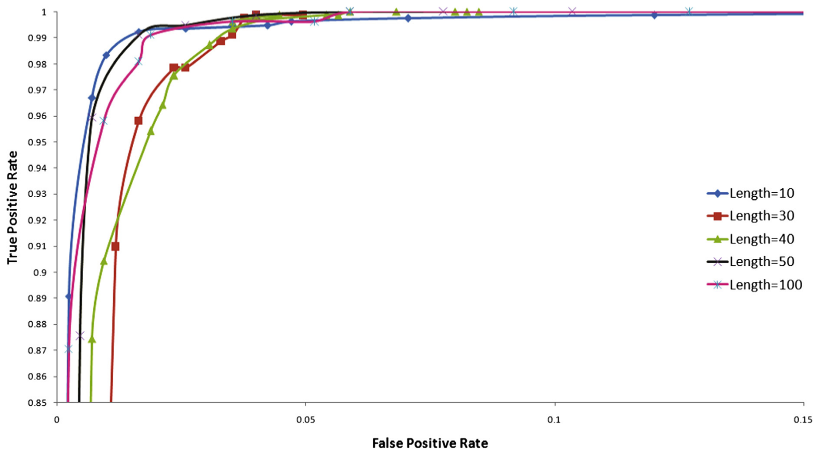

Fig. 3. Variation in ROC curves obtained from RF classifier with change in static feature vector length (Integrated approach)

static and dynamic analysis shows an improvement of 2.89% than static and 0.82% than dynamic approach.

On analyzing the classification results, we observed that the accuracy shows a declining trend on increasing the static feature vector length beyond 50. On an average both classifiers gives best accuracy rates for vector length between 30 and 50. The average time taken to build the model was 0.07 s. This is faster than the Control flow based malware variant detection [4] which took 0.7 s and Hybrid concentration based feature extraction approach for malware detection [11] which took 0.99 s on an average.

4 Conclusion

Dynamic malware analysis is done by watching and logging the behavior of the malware while running on the host. As most of the malwares pack themselves or morph their code during execution, static analysis alone is not efficient in malware detection. So here we have proposed a method that extracts features by analyzing the dynamic behavior of the malware and also static properties of the binaries and identifying whether the new sample possess similar features. The experimental results show that the method is more efficient than the static and dynamic analysis methods alone and also will be able to identify the polymorphic variants of the malware. The detection accuracy of integrated approach is comparatively better than several existing approaches. Integrated approach shows an improvement of 2.89% than static and 0.82% than dynamic approach. A practical extension to this work is to improve the static analysis part by incorporating the concept of function length analysis and also to consider files other than executables while constructing the model.

References

1. Islam, R., Tian, R., Batten, L., Versteeg, S.: Classification of malware based on string and function feature selection. In: IEEE Second Cybercrime and Trustworthy Computing Workshop (2010)
2. Santos, I., Nieves, J., Bringas, P.G.: Semi-supervised learning for unknown malware detection. In: 9th International Conference on Practical Applications of Agents and Multi-agent Systems (PAAMS) (2011)
3. Ye, Y., Li, Y., Chen, Y., Jiang, Q.: Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2010, pp. 95–104. ACM, New York (2010)
4. Cesare, S., Xiang, Y., Zhou, W.: Control flow-based Malware variant detection. IEEE Trans. Dependable Secure Comput. **11**, 307–317 (2014)
5. Zhao, Z., Wang, J., Bai, J.: Malware detection method based on the control flow construct feature of software. IET J. Inf. Secur. **8**, 18–24 (2014)
6. Elhadi, A.A.E., Maarof, M.A., Osman, A.H.: Malware detection based on hybrid signature behavior application programming interface call graph. Am. J. Appl. Sci. **9**, 283 (2012)

7. Anderson, B.H., Quist, D.A., Neil, J.C.: Graph-based Malware Detection Using Dynamic Analysis. Los Alamos National Laboratory Associate Directorate for Theory, Simulation, and Computation (ADTSC) LA-UR 12-20429
8. Borojerdi, H.R., Abadi, M.: MalHunter: automatic generation of multiple behavioral signatures for polymorphic Malware detection. In: 3rd International Conference on Computer and Knowledge Engineering (ICCKE 2013), 31 October–1 November 2013. Ferdowsi University of Mashhad (2013)
9. Sharma, V.: A theoretical implementation of blended program analysis for virus sign extraction. In: IEEE International Carnahan Conference on Security Technology (ICCST), October 2011
10. Nguyen, M.H., Nguyen, T.B., Quan, T.T., Ogawa, M.: A hybrid approach for control flow graph construction from binary code. In: IEEE Software Engineering Conference (APSEC) (2013)
11. Zhang, P., Tan, Y.: Hybrid concentration based feature extraction approach for malware detection. In: 2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (2015)
12. http://www.nilc.icmc.usp.br/elc-ebralc2012/minicursos/WekaManual-3-6-8.pdf
13. https://virusshare.com
14. https://avcaesar.malware.lu