To Incorporate Sequential Dynamic Features in Malware Detection Engines

Mojtaba Eskandari, Zeinab Khorshidpur, Sattar Hashemi
Department of Computer Science and Engineering
Shiraz University
Shiraz, Iran
Email: {m_eskandari, zkhorshidpur, s_hashemi}@cse.shirazu.ac.ir

Abstract—Currently, signature-based detection is a widely used method within commercial antivirus. Although this method is still used by the most commercial antivirus softwares and is capable of detecting specific malwares quickly, it fails to detect new malwares. Therefore, antivirus engines are not limited to static signature based detection, their intelligent detection subsystems can detect unknown malwares more accurately than before. It utilizes an analyzer to extract appropriate features from executable files. It, then, applies a data mining technique on these features to learn behavior of benign programs and malicious ones. Consequently, it is able to detect unknown malwares according to their behavior. Application Programming Interface (API) call sequences are commonly used features in intelligent malware detection systems. An API call sequence captures the activities of a program and, hence, it is an excellent candidate for mining of any malicious behavior. Different order of each API in sequence infers different behavior model. Therefore, ordering of called APIs is an important issue to analyze malwares' behavior. In this paper we propose a novel feature extraction approach for modeling malwares' behavior. The presented approach extracts called API's sequence by dynamic analysis method which is executing programs and capturing their called API's. This approach utilizes N-grams method to preserve call ordering sequence of API’s. The experimental results show promising accuracy of the presented approach for analyzing malwares.

Index Terms—Malware detection; API call sequence; N-grams; Dynamic analysis;

I. INTRODUCTION

Security threats identification is one of the most challenging in our nowadays human practice. Malware, stands for malicious software, is every computer software that has destructive purposes, such as viruses, Trojan horses, Spyware or Internet worms.

The classic method to detect malware is based on a signature database. Signature-based detection methods model the behavior of malware and use this model in the detection of malware. The collection of all of these models represents signature-based detection’s knowledge that is referred to as the signature. When a new signature is created, it should be added to the signature dataset. Generally extracting signatures is a tedious task and needs great deal of time and expertise. Major drawback of this approach is disability in detection of new threats because there is no corresponding signature stored in the signature database. Regarding dramatic growth of malwares, the said process seems imposable. Therefore, the existence of an automated process is felt. In fact, malwares use self protection techniques. Based on this techniques malwares have two forms: polymorphic and metamorphic. In these methods malwares change the source code while they save their functionality.

These two types of malwares change their body for each infection. Polymorphic malware has an embedded encryption engine which use it with a random key to generate a new copy of itself. The content of this new copy is pretty different from its original version. Obviously, it is so hard to detect this type of malware by classic signature based method. Metamorphic malware can generate a copy of itself with a different internal structure that doesn’t resemble to its parent, but behaves like its ancestors. It uses some obfuscation techniques to change its code. To overcome the main drawback of classic signature-based method, behavior-based intelligent (or heuristic) methods are developed. Such methods try to find the behavior of malwares for further recognitions of similar malwares. Behavior-based methods can fulfill the malicious activity detection during code execution by trying to trace any suspicious stir [1]. Even if a code is obfuscated, its behaviors and aims remain resolute.

Heuristic method essentially uses data mining and machine learning techniques hence requires feature extraction phase to extract information from executable files. Process of feature extraction phase is called malware analysis. This process contains several preprocessing steps to construct a behavior model of executable input files. Most of malware detection approaches use these features: DLL usage information, n-grams, assembly instructions sequence, application programming interface (API) call sequences, dependency graph, control flow graph etc.

An n-gram is a sequence of bytes with fixed or variable length. N-gram features are extracted from the hexadecimal dump of an executable program [2]. Traditional data mining based malware detection methods apply n-grams on hexadecimal dump of a program to extract its features. Since n-grams fail to capture the semantics of a program, assembly instruction sequences should be used instead. Instructions need disassembling the executable files. Since almost all programs use application programming interface (API) calls to send their
requests to the Operating System, one of the most attractive ways to find the behavior of a program such as malware is recognizing the pattern of its API call sequences. API is an interface provided by the operating system for programs through which they invoke the operating system to get a service [3]. The best representation of executable files behavior is Application Programming Interface (API) call sequences. API call sequence gives us valuable information about malicious activities [4] by indicating the flow of called API’s using a program which can be either a malicious program or a benign one. Each program should be analyzed to extract this sequence.

API call extraction method can be divided into two categories: static and dynamic methods. Static method also is known as reverse engineering method. This method offers information about programs’ API sequences without actually running them while dynamic analysis extracts these information from the executable files at runtime.

The paper is organized as follows: in Section II, a variety of data-mining based malware detection approaches are presented. Section III presents our proposed approach. The experimental results are described and discussed in Section IV and, finally we will come to the conclusion.

II. RELATED WORK

Several studies have been focused on presenting promising methods for malware modeling, analysis and detection. [5], [6] and [7] introduce detection method which were mentioned in previous section.

The most common method to describe the activity of a program is to monitor the extracted API/system calls. Wagner and Dean proposed a technique that created a control flow graph for a program representing its system call trace [8]. At run time this CFG was compared with the system call sequences to check for any violation. Bergeron et al. propose a static detection of malicious code in executable programs [9]. This approach is done in three steps: generating an intermediate representation, analyzing the control and data flows, and doing static verification. This last step consists of comparing a security policy to the output of the analysis phase.

Lo et al. propose the idea of tell-tale signs which were heuristic signatures of malicious program behaviors [10]. They created an intermediate representation of the program under investigation in the form of control flow graph. The CFG was verified against the tell-tale signs to detect any malicious activity. The approach was implemented in their system called Malicious Code Filter (MCF).

Sekar’s approach use Finite State Automata (FSA) to represent system call sequences [11]. The FSA’s were created by executing the programs multiple times and recording the system calls. This approach builds compact FSAs in a fully automatic and efficient manner.

Heuristic method added data mining as an identification way in malware detection category. Data Mining involves the application of a full suite of statistical and machine learning algorithms on a set of features derived from malicious and clean programs. Feature type describes the input data to the malware detection system. These features include hexadecimal byte sequences, instruction sequences, API/system call sequences etc.

Yang and Pedersen introduce five methods to select the best features for text classification task [12]. Printable strings are extracted from the files and some operating system dependent features such as DLL information are used as feature in this work.

Li et al. propose a method to analyze files and to categorize their type by using efficient 1-gram analysis of their binary contents [13]. The method represents each file type by a compact representation that is called a file-print. These file-prints represented the normal profile of the files and were compared against file prints taken at a later time by using simplified Mahalanobis distance. A large distance indicated a different n-gram distribution and hence maliciousness.

Malan and Smith propose a host-based, runtime defense against worms with insignificant false alarm rate through peer-to-peer cooperation [14]. They view correlation among otherwise independent peers’ behavior as anomalous behavior, indication of a fast-spreading worm and detect correlation by exploiting worms’ temporal consistency, similarity (low temporal variance) in worms’ vocations of system calls.

Sung et al. introduce a signature based detection system called Static Analyzer of Vicious Executables (SAVE) that compares API sequences extracted from programs to sequences from a signature database [15]. This work emphasizes on detecting obfuscated (or polymorphic) malware and mutated (or metamorphic) malware.

III. PROPOSED APPROACH

Our approach focuses on mining the API call sequences for detecting unknown malware detection. Mining API call sequences needs to extract them from executable files. There are two categories of methods to extract called API’s. The first one is static analysis and the second one is dynamic analysis.

Static analysis extracts API call sequence according to executable file’s source code. First of all, it constructs control flow graph (CFG) with disassembled source code. Second, API call sequence is obtained by traversing the CFG. This method is able to deal with decision making points and generally makes random decision in these points. This issue is the major drawback of static methods. For example, as shown in Figure 1, suppose an API function is called in a conditional jump. At the run time it is clear that API is called or not, but the static analyzer may be confused at this point and there might be an incorrect API presence or absence in the output sequence. Another drawback of this method is encountering with obfuscated an encrypted malware.

An obfuscated malware is developed to avoid the static analysis methods. The malware author uses a set of dirty codes to develop it. The complexity of these codes is high, so, static analysis usually fails to trace them. Figure 2 represents an example of these codes.

The body of an encrypted malware is encrypted by a specific encryption algorithm and a particular key. This malware has a
1. mov esi, eax
2. cmp esi, ebx
3. jnz short loc_12DF6
4. mov ecx, [ esp + 1Ch ]
5. mov edx, [ esp + 18h + var_10 ]
6. lea eax, [ esp + 0Ch ]
7. push eax
8. push 18h
9. push ecx
10. push ebx
11. push edx
12. mov [ esp + 20h ], ebx
13. call ds:ZwQueryInformationProcess
14. mov esi, eax
15. loc_12DF6:
16. mov eax, [ esp + 18h + var_10 ]

Fig. 1. Static analysis method will fail in analyzing this code extracted from Backdoor:W32/Duqu. There is an API call instruction at line 13 which calls the "ZwQueryInformationProcess" API function. Considering this API call in extracted API call sequence depends on comparison instruction at line 2. The conditional jump at line 3 and the target label of that conditional jump placed at line 15. If jnz instruction jumps to line 15, the call instruction will be ignored and its API function does not add to API call sequence. Otherwise, this API must be added into output sequence at correct situation. Hence, the static analysis will be confused at line 2 and consequently its output API call sequence is not reliable.

Fig. 2. This obfuscated code was extracted from Spam-Mailbot.c!Rootkit:W32.Rustock. The malware author uses a nice trick. At line 3 the stack pointer (esp) is decreased by 4. Instead of using "sub esp, 4" instruction, he/she makes this code to confuse the analyzer. At line 4. a mov instruction is used which seems to be simple but this instruction and line 3 do push operation. Finally, at line 5 a ret instruction is appeared. This instruction pops the target address from stack and jumps to it. Consequently, these three instructions play the role of an unconditional jump instruction.

Decryption engine. At the runtime this decryption engine decrypts the body codes and executes them. Since, the discovery of decryption algorithm and its key is difficult, it is impossible to use an automated system to deal with them. Therefore, static analysis is not useful for analyzing this type of malware.

Dynamic analysis captures effects of programs on the environment and ignores analyzing their source codes. This method extract called API’s by running them in the controlled environment which is simulated operating system. Sandbox is a powerful tool which simulates a real environment for the executable files, so malwares rely on it and request their system calls via API calls. Sandbox replies the request in virtual environment and logs them.

Extracted API’s from this log file are accurate and actual. These sequences use as feature set in heuristic malware detection methods. Our proposed approach is based on mining these features. Applying data mining methods on API sequences encounters with several problems. API sequence fragmentation helps us to handle these problems.

1) API Sequence Fragmentation: To use data mining techniques we need to construct a feature set according to extracted API call sequence set. Since the number of instructions in variant executable files are different, length of the extracted API call sequences are different. A common solution to construct feature set is identifying distinct API’s from this sequence set and put them into a new set named as feature set. This feature set construction method is simple and fast, but, this feature set misses an important portion of behavioral information of programs. Therefore, detection method based on this feature set is not accurate enough.

Although appearance of each API indicates a part of program’s behavior, order of API calls in sequence plays an important role in program’s behavior. Therefore, different order of each API call in sequence infers different behavioral model. Preserving order of called API’s has significant information for analyzing malware’s behavior. The former feature set construction method misses the ordering of called API’s. Thus, its accuracy is low. We need to consider the ordering information of API calls and store them into the feature set.

This paper proposes a new approach to handle this issue. We use a method named “N-grams” to fragment the API call sequences without missing significant information about call ordering. An N-gram is a sequence of bytes of fixed or variable length, extracted from the hexadecimal dump of an executable program [2]. Traditional data mining based malware detection methods apply n-grams on hexadecimal dump of a program to extract its features. We utilize n-grams on API call sequences to construct a perfect feature set.

The presented approach utilizes a fixed window by length $N$ and moves it on an API call sequence to construct the feature set. Each move is represented as a feature. Algorithm 1 shows this trend. This algorithm takes two parameters, sequence set and $N$ and constructs the feature set. This feature set has two outstanding attributes. The first one is keeping called API’s within itself, and the second special attribute of this feature set is preserving call order of each program while the feature vector for each data-item (each executable file) has fixed length. This approach helps us to fragment all API call sequences and put their information into a tabular dataset with fixed size data-items.

The tabular form of dataset is generated by Algorithm 2. An example of this tabular form is shown in Figure 4. This algorithm takes two parameters which are API call sequence set and a feature set that generated by Algorithm 1. It is obvious that the number of features for the generated dataset is pretty high. Therefore, it is hard to perform data mining algorithms on it. So, an appropriate feature selection need to be applied in order to reduce the size of feature set and consequently reduced the size of overall dataset. Here, we use best first selection method due to its simplicity and speed.

An overall structure of our presented method is shown in Figure 3. First API calls extracts from each program by
dynamic approach. Sandbox is employed for this reason and after that API sequence is fragmented by N-gram window and feature set is constructed. For reducing the number of these feature we utilize feature selection techniques to feature reduction.

An interesting question that might arise is that since \( N \) (the window length) is fixed in our approach, the call ordering information might be missed on cut points. To answer this question, it should be considered that each called API is related to near called API’s. Therefore, the designed method must preserve call ordering information for each API call regarding to its near API calls. The next question is that how many API calls near to the current called API should be considered? To solve this problem we must find the best value for parameter \( N \). If \( N = 1 \), the length of window becomes 1. In this case, no call ordering information is preserved. In other words, if window length is one, we use the original feature set construction method. So, the parameter \( N \) must take a value greater than 1. It should be noted that the best value for \( N \) is obtained by experiment.

IV. EXPERIMENTAL SETUP

In this section the presented approach is evaluated. First of all, the used dataset is described in detail. Second, performance measures are described and at the end of this section the results are shown and discussed.

A. Dataset Description

To generate data sets, we collected 956 executable files consisting of 501 Massive Mailing Worms as malware set and 455 benign programs. A mailing worm is a worm that spread itself by email. It typically searches the system for email addresses and then sends copies of itself to those addresses, using either the system’s email client or a self-contained mailer built into the worm itself. Besides overwhelming servers and

![Algorithm 1: FeatureSetConstructor( \( S_{in} \), \( N \))](image)

**Algorithm 1:** FeatureSetConstructor( \( S_{in} \), \( N \))

- **Input:** \( S_{in} \): Set of API call sequences; // Each sequence corresponds to a PE-file
- **Output:** \( F_{out} \): Constructed feature set;

1. \( F_{tmp} \): an empty set;
2. foreach sequence \( s \) in \( S_{in} \) do
3.   foreach API \( i \) in \( s \) do
4.     \( f = \) get a set of \( i^{th} \) API up to \((N+i)^{th}\) item;
5.     Add \( f \) to \( F_{tmp} \);
6.   end
7. end
8. \( F_{out} = \) fetch unique features from \( F_{tmp} \);
9. return \( F_{out} \); // A set of subsequences

![Algorithm 2: DataSetGenerator( \( S_{in} \), \( F_{in} \))](image)

**Algorithm 2:** DataSetGenerator( \( S_{in} \), \( F_{in} \))

- **Input:** \( S_{in} \): Set of API call sequences;
- **Output:** \( D_{out} \): Output dataset;

1. \( D_{out} = \) an empty table;
2. foreach sequence \( s \) in \( S_{in} \) do
3.   Add \( s \) to the first column of \( D_{out} \);
4.   foreach feature \( f \) in \( F_{in} \) do
5.     if \( f \) is found in \( s \) then
6.       Add 1 to next column of \( D_{out} \);
7.     else
8.       Add 0 to next column of \( D_{out} \);
9.   end
10. end
11. Add Label of \( s \) to last column of \( D_{out} \);
12. end
13. return \( D_{out} \);

![Fig. 4. An example of tabular form of dataset is generated by Algorithm 2.](image)

In this table each feature is represented by a subsequence of API calls and each data-item (PE-file) is represented by a vector of absence/presence of API call subsequences.
networks with massive volumes of e-mails, mass mailing worms often incur serious performance issues on infected systems. These malware are selected randomly from malware repository of APA, the malware research center at Shiraz University. The benign programs are gathered from a fresh installed Microsoft Windows XP SP2 on a virtual machine. Some popular user applications are installed on that machine such as Microsoft Office, Adobe Photoshop, Music players and so on.

B. Performance Measure

In this section we present some commonly used measures for evaluating our approach. There are typical objective functions to evaluate classification algorithm. We use a quantitative measure of classification performance.

For classification tasks, the terms true positives, true negatives, false positives, and false negatives, compare the results of the classifier under test with trusted external judgments. The terms positive and negative refer to the classifier’s prediction (sometimes known as the observation), and the terms true and false refer to whether that prediction corresponds to the external judgment (sometimes known as the expectation). Precision and recall are defined as Equation (1) and (2) respectively.

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{1}
\]

where \( TP \) indicates true positive rate and \( FP \) refers to false positive rate.

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{2}
\]

where \( FN \) is false negative rate. We define recall as the percentage of all PE-files labeled “malicious” that can receive correct label by the system.

A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional F-measure or balanced F-score, as illustrates in Equation (3).

\[
F\text{-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3}
\]

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 [16].

C. Discussion

The presented analysis approach can be used in a detection system based on data mining techniques. Most of these systems use called API’s as feature to detect malwares. Feature set is constructed based on two methods: traditional API based and sequence fragmentation method. For more exploration we employed several classifiers to evaluate accuracy of detection method based on API mining. These API’s were extracted by dynamic approach. In the following we briefly introduce classifiers that used in this experiment.

A Naïve 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. It is a kind of dependent constraint modeling [17].

Sequential minimal optimization (SMO) is an algorithm for solving large quadratic programming (QP) 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 [18].

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, randomized binary search tree, rapidly exploring random tree, Brownian tree, random forest and branching process [19].

Random forest is an ensemble classifier that contains many decision trees and outputs the class that is the mode of the class’s output by individual trees [20].

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 [21].

A logistic regression classifier provides conditional probability classifications of input objects using an underlying logistic regression model and feature extractor. Logistic regression is a discriminative classifier which operates over arbitrary feature vectors extracted from items [22].

In this paper we proposed a novel approach to construct feature set for malware detection based on data mining techniques. This approach modifies traditional API based techniques and handle main drawback of previous method. Our approach considers ordering of API sequence in generated feature set. Since the ordering of calls API represent malware behavior, this issue is most important in the malware description.

Fragmentation of API sequence employs N-gram window of length \( N \) to generate feature set while the ordering of API being preserved. This process is done via Algorithm 2.

After feature construction, we need to generate a data set to evaluate malware detection methods. This data set is generated based on algorithm 2. As mentioned for each input file a sequence of called API is extracted by dynamic method and based on absence or presence of each feature in feature set rows of data set are filled.
There are lots of classifiers with high accuracy in the world. So, the major step in designing an intelligent malware detection system is feature extraction. In this paper, we offer a novel approach to extract appropriate features to model programs’ behavior. Since these features can well represent the behavior of executable files, we gain high level of accuracy for several classifiers.

As a future work, we will explore on variable length of window to develop an adaptive malware detection system. Window with variable length helps us to determine the window length at extracting phase for each group of API’s. This method can deal with a wide range of malwares.

V. CONCLUSION AND FUTURE WORK

Feature extraction method plays a superb role in designing and developing a desirable malware detection system. Finding a proper classifier for detecting malicious program is a complementary process to the former task.

As you can see in Figure 5, mean of F-Measure is increasing when length of window expanded by 1 to 4. The main reason for this improvement is using ordering information of called API’s. Experiment indicates that it is not necessary to preserve ordering of all API in the sequence. When length of window is more than 4 performances is dropped. Therefore ordering of a specific number of called API is important in the feature construction and modeling of malware behavior. Increasing this number will have no affect in malware detection even if it is backfire.

Figure 7 illustrates performance of various classifiers among different length of data sets. According to this result logistic regression classifier has best mean F-Measure among data sets with different length of feature set.

As mentioned classifiers have higher detection accuracy on generated dataset with \( N = 4 \). Table I compares accuracy of our proposed approach with window length \( N = 4 \) and traditional approach. Results show that our approach achieves best performance per different classifiers.

**Table I**

<table>
<thead>
<tr>
<th></th>
<th>Traditional Approach ((N = 1))</th>
<th>Our Approach ((N = 4))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.970</td>
<td>0.976</td>
</tr>
<tr>
<td>SMO</td>
<td>0.984</td>
<td>0.983</td>
</tr>
<tr>
<td>Random Tree</td>
<td>0.968</td>
<td>0.979</td>
</tr>
<tr>
<td>K-Star</td>
<td>0.968</td>
<td>0.974</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.971</td>
<td>0.983</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.971</td>
<td>0.983</td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT

This work is supported in part by APA, malware research center at Shiraz University.

REFERENCES

Fig. 6. Mean F-measures of rival approaches across datasets generated by different N-gram window length. Data set with $N = 4$ achieve best mean F-Measure.


