Malware Detection Based on Mining API Calls[*]

Ashkan Sami[†]
CSE&IT Department
Shiraz University
Shiraz; Iran
asami@ieee.org

Babak Yadegari and
Hossein Rahimi
CSE&IT Department
Shiraz University
Shiraz; Iran
{yadegari,rahimy}@cse.shirazu.ac.ir

Naser Peiravian
Soft-Computing Laboratory
Shiraz University
Shiraz; Iran
n.peiravian@gmail.com

Sattar Hashemi
CSE&IT Department
Shiraz University
Shiraz; Iran
s_hashemi@shirazu.ac.ir

Ali Hamze
CSE&IT Department
Shiraz University
Shiraz; Iran
ali@cse.shirazu.ac.ir

ABSTRACT

Financial loss due to malware nearly doubles every two years. For instance in 2006, malware caused near 33.5 Million GBP direct financial losses only to member organizations of banks in UK. Recent malware cannot be detected by traditional signature based anti-malware tools due to their polymorphic and /or metamorphic nature. Malware detection based on its immutable characteristics has been a recent industrial practice. The datasets are not public. Thus the results are not reproducible and conducting research in academic setting is difficult. In this work, we not only have improved a recent method of malware detection based on mining Application Programming Interface (API) calls significantly, but also have created the first public dataset to promote malware research.

Our technique first reads API call sets used in a collection of Portable Executable (PE) files, then generates a set of discriminative and domain interpretable features. These features are then used to train a classifier to detect unseen malware. We have achieved detection rate of 99.7% while keeping accuracy as high as 98.3%. Our method improved state of the art technology in several aspects: accuracy by 5.24%, detection rate by 2.51% and false alarm rate was decreased from 19.86% to 1.51%. This project's data and source code can be found at http://home.shirazu.ac.ir/s̃ami/malware.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining; D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software*

General Terms

Algorithms, Experimentation, Security

Keywords

Malware Detection, API Call, Frequent Pattern Mining, Public dataset, Polymorphic/Metamorphic malware, Portable Executable (PE)

[*]All authors are also members of APA Malware Research and Education Center at Shiraz University.

[†]Institution Phone: +98-711-6474605

1. INTRODUCTION

"Malware" is an abbreviation for 'malicious software' and is typically used as a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network. As malware attacks become more prevalent, attention has begun to shift from viruses and spyware protection, to malware protection, and programs have been developed to specifically combat them.

One association of banks in the United Kingdom estimated the direct losses caused by malware to its member organisations at GBP 12.2 M in 2004, GBP 23.2 M in 2005, and GBP 33.5 M in 2006, an increase of 90% from 2004 and 44% from 2005[1]. These considerations show that protection against malware has become more important [23].

Because of failure of old signature based detection methods on detecting new, unseen and polymorphic/metamorphic malware, in recent years, researchers have begun to study and apply new methods based on unchangeable characteristics of malicious codes. One of these research fields is based on the idea that self modifying malicious codes cannot change specific sections of the portable executable files when trying to modify themselves. Results of these researches showed that applying such methods can be more accurate and precise against unseen and polymorphic/metamorphic malware.

In this paper, we present a framework for analyzing and classifying PE files based on data mining techniques. The framework rests on the idea that calls to Windows Application Programming Interface (API) can be used to extract knowledge describing behavior of executables. Prior to other related works, the data we have collected is much larger and covers various types of either malicious or benign files. We have collected about 32000 malicious files including Viruses, Trojans, Backdoors, Spywares, worms, etc. and near 3000 benign files that include windows system files and many windows portable tools. As a result of the proposed framework, we obtained the detection rate as high as 99.7% which outperforms all previous studies and exposes a more reliable method to protect against malicious destructive activities. Also we have made the data publicly available for further analysis.

Proposed framework consists of three major parts. The first component is a PE analyzer that aims to analyze PE files and extract the API calls imported by PE file. Second component does feature generation and feature selection. Features included in this study, are discriminative and domain relevant features that are produced from the first component's output.

Third part includes a classifier that classifies PE files based on features selected in the second part. We examined several classification methods such as Naïve Bayes [7], J48 [25], and Random Forest [3]. Considering the highly imbalanced class distribution, we will show that Random Forest has the best performance. The rest of paper is organized as follows: section 2 briefly describes related works. Data collection and dataset characteristics are introduced in section 3. Feature generation and selection are described in section 4 respectively. Experiments and results are discussed in section 5, and section 6 concludes the paper.

2. RELATED WORKS

In mid 90's techniques for signature based malware detection were introduced [11, 16]. The major weakness of this type of approach is its weakness to detect polymorphic/metamorphic and unseen malware.

Afterwards [4, 13, 20, 22] introduced methods to improve the signature based detection of unknown and polymorphic malware. One of the best approaches in this category, SAVE [22], focused on detection of polymorphic malware using a similarity measure between the known virus and the suspicious code. Although this work improved the traditional signature-based detection of polymorphic malware, it fails to detect unknown malware.

A few attempted to apply data mining and machine learning techniques [12, 24, 21, 9]. One of these works [12] have used $n-grams$ of 1971 benign and 1651 malicious executables as features. But $n-grams$ is not a suitable feature for detecting polymorphic and metamorphic malware, because of its sensitivity to order. The other two works try to use DLL file names as features [21]. Later in section 5.2, we will also show effectiveness of this feature set. But our method significantly outperforms those approaches. Another problem with these researches is a small number of data which does not exceed 3,000 PE files.

A recent work by Ye et al. [27], a system named IMDS, is the first attempt to use API calls. They have used data mining techniques and features generated from API calls. Their data set consist of 12,214 benign executables and 17,366 malware. They used about 2,000 PE files, which is claimed to be randomly selected from training data. Although experiments presented good results, due to industrial nature of research there is no access to their data set to regenerate their results for further investigations.

Although the detection rate and accuracy is generally good, the problem is that the studied data was either small or the range of API calls extracted from PE files were not sufficient and not more than 12000 in any previous work. Another problem with IMDS, is unbalanced test data in contrast to fairly balanced train data.

Our work is based on a large collection of executables which covers a wide range of portable software kinds and malware that results in extraction of more than 44,000 distinct API calls imported by 34,820 PE files, from 890 different DLLs. And introduces novel features for malware detection, the API call categories.

Regarding large number of Application Programming Interfaces available at the software market and used by software producing companies, it is more rational to consider a wide variety of API calls in the study. We have made our collected data public to researchers to make the first academic data set which is aimed for malware detection research.

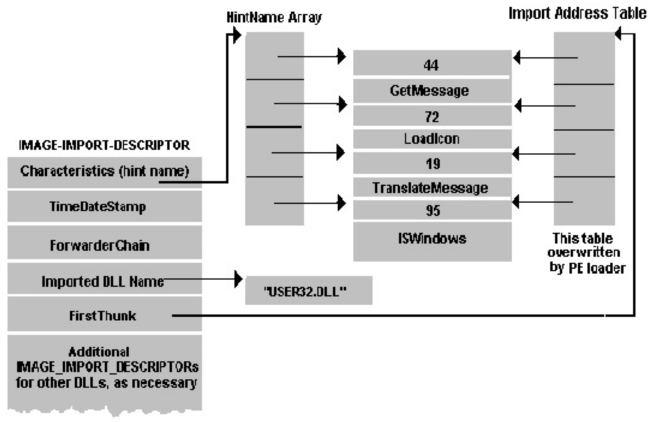


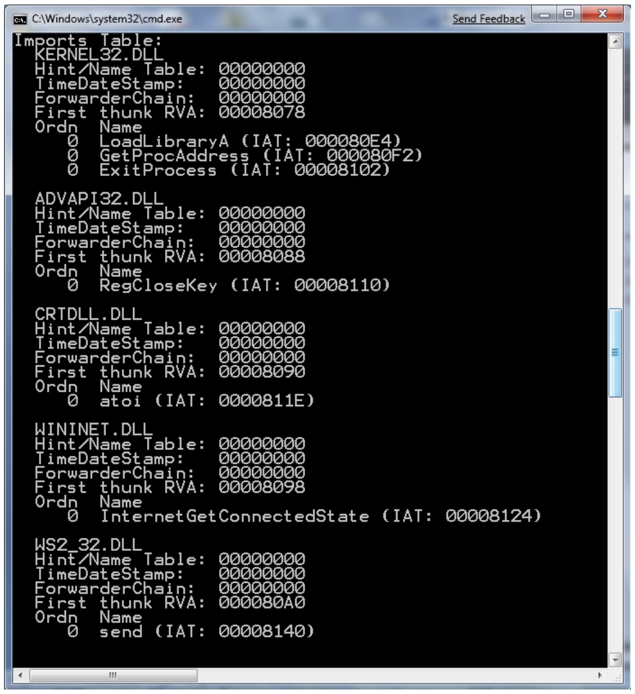
Figure 1: Portable Executable's IMAGE_IMPORT_DESCRIPTOR structure [18].

3. DATA COLLECTION

We have gathered 34,820 PE files where 31,869 were malicious and 2951 were benign windows PE files. Malicious PE's includes all varieties of PE malware including Viruses, Backdoors, Worms, Trojans, Spywares, etc. gathered from various "underground" sources such as [10]. The benign files were gathered from various versions of Microsoft Windows family including Windows system files and a wide range of portable benign tools.

"When any PE file loads, one of the jobs of the Windows loader is to locate all the imported functions and data and make those addresses available to the file being loaded. Within a PE file, there's an array of data structures, one per imported DLL. Each of these structures gives the name of the imported DLL and points to an array of function pointers. The array of function pointers is known as the import address table (IAT). Each imported API has its own reserved spot in the IAT where the address of the imported function is written by the Windows loader. This last point is particularly important: once a module is loaded, the IAT contains the address that is invoked when calling imported APIs" [19].

There is no field indicating the number of structures in this array. Instead, the last element of the array is indicated by



```
Imports Table:
  KERNEL32.DLL
  Hint/Name Table: 00000000
  TimeDateStamp:   00000000
  ForwarderChain:  00000000
  First thunk RVA: 00008078
  Ordn  Name
     0  LoadLibraryA (IAT: 000080E4)
     0  GetProcAddress (IAT: 000080F2)
     0  ExitProcess (IAT: 00008102)

  ADVAPI32.DLL
  Hint/Name Table: 00000000
  TimeDateStamp:   00000000
  ForwarderChain:  00000000
  First thunk RVA: 00008088
  Ordn  Name
     0  RegCloseKey (IAT: 00008110)

  CRTDLL.DLL
  Hint/Name Table: 00000000
  TimeDateStamp:   00000000
  ForwarderChain:  00000000
  First thunk RVA: 00008090
  Ordn  Name
     0  atoi (IAT: 0000811E)

  WININET.DLL
  Hint/Name Table: 00000000
  TimeDateStamp:   00000000
  ForwarderChain:  00000000
  First thunk RVA: 00008098
  Ordn  Name
     0  InternetGetConnectedState (IAT: 00008124)

  WS2_32.DLL
  Hint/Name Table: 00000000
  TimeDateStamp:   00000000
  ForwarderChain:  00000000
  First thunk RVA: 000080A0
  Ordn  Name
     0  send (IAT: 00008140)
```

Figure 2: A sample PE analyzer output for Blaster.exe.

an structure that has fields filled with NULLs. The format of the structure is shown in Figure 1. The important parts of the structure are the imported DLL name and two arrays of IMAGE_IMPORT_BY_NAME pointers. In a PE file.

Some malware writers compress their codes to make them undetectable by anti malware systems or difficult to disassemble. There are multiple compression tools to pack PEs or writers may pack their files by their own homemade compressors. So we used number of most popular Compressor/Decompressor tools such as UPX, ASPack, PECompact, etc. to make the PEs recognizable by the PE analyzer and near 6,000 malicious or benign PEs were unpacked.

PE analyzer extracts all API function names concatenated the library name for each PE implicitly linked to. Figure 2 shows a sample PE analyzer output for file Blaster.exe.

Finally, a total of 44,605 distinct API calls were extracted from selected files where malicious executables called a sum of 2,813,564 API functions and benign executables called 438,819 API procedures. After this, dataset is available for feature generation and feature selection which are described in following sections 5 and 6.

4. FEATURE ANALYSIS

In this section we have described two important parts of our system in detail. The first one , Feature Generation, creates the features and the second component, Feature Selection, selects between the features produced by the first component.

4.1 Feature Generation

Given a set of API call sequences that are labeled either positive (malware) or negative (benign). We have selected two types of features. One type is semantically relevant to the possible behavior of an executable and the other is based on discriminative behavior.

A simple way to use each API call as a feature. Every executable can be represented as a binary vector of API calls, namely A, where $A_i = 1$ if and only if the PE file has imported the i_th API call and $A_i = 0$ if corresponding PE file has not used the API call. As will be shown in experiments section, these features cannot be used to classify the data efficiently since single API calls as features are not discriminative. in contrast combination of API calls as features can predict malicious behavior more profoundly.

Considering similarities between PE files, frequent API call sets can be also considered as a feature set. On our data set a frequent itemset can be defined as follows:

Definition 1. *(Frequent API call set) Given an API set dataset ADB, support(APIset) is the percentage of API sets in ADB, where APIset is a subset. An API sequence is frequent if its support is no less than a given support min_sup.*

Representation of each PE file shares the same framework used in API-based approach for feature generation. Where every PE file is represented as a vector of frequent API call sequences. The vector is 0, 1 valued. Naming this vector F, $F_i = 1$ if and only if the PE file has imported the whole i_th frequent set of API calls and Fi=0 if and only if corresponding executable doesn't import all the API procedures included in the i_th frequent API call set. For mining frequent patterns from transactions of PE files which were introduced in section 4, we used fp-growth algorithm introduced by Han et al. [8].

Having 29000 features and a large number of instances, it's still time consuming to train a classifier with a moderate hardware. As an alternative, closed frequent patterns can be mined as features.

Definition 2. *(Closed Frequent API call Set) A frequent API call set is closed if there exists no superset that has the same support.*

To mine closed frequent API call sets we used Clospan algorithm by Han et al. [26], the algorithm is already implemented by professor Han's group and made available in a partially open source data mining package named Illimine for academic uses on the web [6].

Setting minimum support to 10%, we get a batch of 2994 features that makes it more feasible to run different algorithms on the data.

To overcome the sparseness problem we introduce new features that have an accumulative nature which makes it possible to combine multiple features in a smaller set of features without losing behavioral characteristics of Those API calls. Based on our domain knowledge, we decided to choose features which are more relevant to the application domain. To satisfy this objective we chose two new feature spaces.

First, considering every DLL as a feature seems being useful because we can characterize kind of behavior that an API call might have by looking at the library it is imported from. For example Netutils.DLL is a library which is used for a set of utility API calls contributing to network operations under Microsoftő Windows Seven environment. When using these features every PE file is transformed to a vector of 890 elements. Each element of this vector corresponds to a DLL library (e.g. Kernel32.dll). Value of each element is equal

Algorithm 1 Fisher Score Based Feature Selection.

```
INPUT:
  $API = Initial\ set\ of\ API\ calls$
  $PEDB = PE\ transaction\ database$
  $\delta = Coverage\ threshold$
OUTPUT:
  $API_s = Set\ of\ selected\ API\ calls$

  Sort API in descending order of Fisher score
  for All transactions $\tau$ in PEDB do
    if $count(intersection(API_s,\ \tau)) \geq \delta$ then
      skip this iteration
    end if
    for All API call $\alpha$ in API do
      if $\alpha \epsilon \tau$ and $count(API_s,\ \tau) < \delta$ then
        $API_s = API_s \cup \alpha$
        $API = API - \alpha$
      end if
    end for
  end for
```

to number of API calls the PE file has imported from the corresponding DLL.

Second set of domain relevant features, are API call categories. To do this we used a categorization made by Microsoftő on MSDN website [14]. This categorization makes 95 categories of most used API calls in the API call database. Setting these categories as features we have a new representation of PE files as 95-element vectors. In this vector any element corresponds to a specific category and its value is equal to the number of API calls that the PE file has used from the corresponding category.

Considering the novelty of categorizing features in this field, we have observed a high boost in classification results which are discussed in Section 5.2.

4.2 Feature Selection

The set of categories created upon the set of benign and malicious PE files are considered as the initial set of features. To improve the classification accuracy we decided to select the top discriminative API calls and add them to the current feature set.Assuming the initial set of API calls is *APIf* $=\{\ APIf_1,\ \ APIf_2,\ \ ...,\ \ APIf_n\ \}$, where each API call $APIf_i$ represents a feature. Having a PE file P and a feature set APIf, x is the feature vector representation of P. Then,

$$x_i = \{ \begin{array}{ll} 1 & \text{if } P \text{ imports } APIf_i \\ 0 & \text{otherwise} \end{array} \quad (1)$$

For discriminative power evaluation, Fisher score [5] is popularly used and has led to good results in recent work [15]. The score is defined as:

$$Fr = \frac{\sum_{i=1}^{c} n_i * (\mu_i - \mu)^2}{\sum_{i=1}^{c} n_i \sigma_i^2} \quad (2)$$

where n_i is the number of data samples in class i, μ_i is the average feature value in class i, σ_i is the standard deviation of the feature values in class i, and μ is the average feature value in the whole dataset.

As could be understood easily from equation 2, Fisher score is maximized when σ_i is minimized and sum of differences between class averages μ_i gets maximized. It shows that features with more similar intra-class values and different inter-class values have a bigger Fisher score. So if for a given API call as a feature its Fisher score gets larger when count of it being seen in malicious and benign files has a large difference and API call is said to be discriminative.

Based on the Fisher score we have used an algorithm introduced in a recent work [15] with a little change to choose between features. A pseudo code of this algorithm is shown in algorithm 1.

By applying this algorithm on API call categories as features and whole data, top four categories by Fisher's score are:

1. File Management

2. Process and Thread

3. Console

4. Registry

To interpret these features, when going back to malware definitions and their known major behaviors we can conclude that file management is the most important activity in viruses, because this kind of malware is known to copy itself multiple times and mutate itself every now and then if it's a polymorphic/metamorphic malware. Another difference when studying malware and benign applications is the type of process management done by malicious code that is intended to hide from, or spoof antivirus tools and process monitors. since getting hidden is more complicated than running normally, this approach needs more calls to process management procedures.

The third discriminative category is Console, which shows the difference in user interaction. Benign applications are most purposed to interact with users when malware specially Worms and Viruses, except a small number of Trojan and Backdoor tools, are aimed to do their objective in background. Next category is the Registry, malicious executables have heavily used Windows Registry to change system menus, user privileges, startup programs, and file extension handling attributes to ease malware distribution or to destruct the target computer.

Considering knowledge contained by these features, and originality of this approach we expected a high classification performance that will be shown in section 5.2.

5. EXPERIMENTS

In these experiments we aimed to compare the classification results on feature sets discussed in section 4. To do so three data sets are created with the whole malicious and benign executables which totally makes up 34,820 records in each data set. Data sets only differ in their feature sets introduced in section 4.2, and have exactly the same number and distribution of records.

First data set "Close", is created using the closed frequent patterns defined in definition 2 as the features. The second data set "DLL", is based on DLL file names as the features, and the third one "Cat+" is created upon API call categories in addition to the set of most discriminative API calls which are selected using the method introduced in section 4.2 as features. In our experiments malicious PE files are labeled as positive and benign PE's are considered as negative samples.

Table 1: Classification results on all datasets.

Data	Classifier	Precision	Recall	AUC	FA_{rate}
Close	RF 10	95.10	95.40	0.94	4.15
Close	NB	90.50	27.80	0.78	2.02
Close	J48	95.10	95.30	0.87	4.47
DLL	RF 10	96.20	96.40	0.95	3.30
DLL	NB	94.00	93.50	0.63	6.60
DLL	J48	94.00	93.60	0.62	6.58
Cat+	NB	96.70	87.90	0.61	3.29
Cat+	J48	98.40	99.30	0.93	1.62
Cat+	RF 10	98.40	99.70	0.97	1.64
Cat+	RF 100	98.50	99.70	0.98	1.51

In this part of paper we first talk about imbalanced data issue in section 5.1, and latter in section 5.2 we will compare the results. All of our experiments were carried out on a 2.16GHz Intel Core 2 Duo PC with 2GB physical memory, using WEKA [25] under Microsoft Windows XP environment.

5.1 Evaluation of Imbalanced Data

As mentioned before in section 1, the data is highly imbalanced. So using traditional accuracy results is not suitable to evaluate the classification. Some techniques have been introduced before [2] to make the evaluation more reliable when dealing with imbalanced data sets. We chose precision and detection rate (recall) in addition to AUC and accuracy to evaluate our experiments.

Detection rate is defined as the portion of the total malicious PE files that are classified as malware. Precision refers to the probability that a PE is classified as a malicious PE correctly. Though precision and detection rate are usually contrary our results show that we've achieved good detection rate while holding precision up to 98.5%.

AUC stands for Area Under ROC (Receiver operating characteristic) Curve resulted from a classification, and is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

Classifiers are evaluated using 10-fold cross validation which allows us to use the whole data as test and train data to make the test results more reliable.

5.2 Results

Multiple classifiers were trained and tested on the data. Due to imbalanced nature of our data, we foresaw that Random Forest classifier outperforms other classifiers because of its characteristics including Bagging, ensembling and attribute raising that are widely used to overcome imbalanced data set problems and for classifier accuracy improvement. We analyzed the data sets with different classifiers. We tried to determine which data set holds more knowledge about the behavior of PE files. At the same time we have tried to determine the best classification method for our data.

Table 5.1 compares classification results related to different classifiers on all three data sets. As can be intuitively understood, Naive Bayes method gives the lowest detection rate where the highest detection rate in each case belongs to the Random Forest.

Random Forest has good performance on data with skewed class distribution. Also it is shown that Random Forest has the best performance in classifying all created data sets in respect to AUC, please refer to table 5.1.

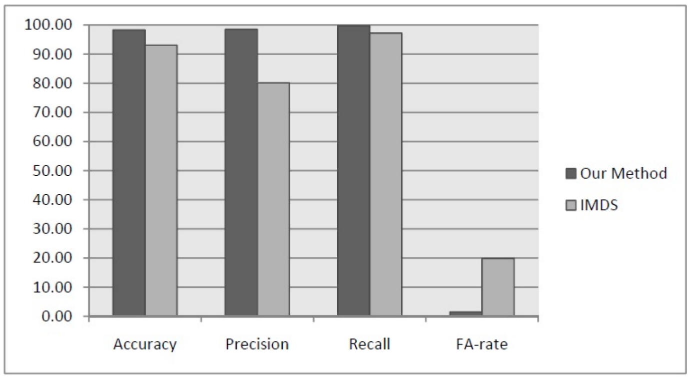


Figure 3: Our method's results versus IMDS'.

Table 2: Our method's Improvements to IMDS.

Measure	IMDS	Our method	Improvement
Accuracy	93.07%	98.31%	5.24%
Precision	80.13%	98.5%	18.37%
Recall (DR)	97.19%	99.7%	2.51%
FA_{rate}	19.86%	1.51%	18.36%

To choose a methodology among these feature generation/selection and classification methods, our approach is lowest false alarm rate (FA_{rate}) and highest detection rate.

The method Random Forest on Cat+ data set, with 100 trees, by having a detection rate of 99.7% and FA_{rate} equal to 1.51%, is recognized as our best method. In addition maximum AUC is achieved by this method.

Intelligent Malware Detection System [27] (IMDS), is the most recent approach that we know. The methodology behind IMDS is to use frequent pattern mining for feature generation and uses Max-Relevance [17] to select best features. After that the IMDS system classifies PE files using a Classification Based Association classifier. While Figure 3 compares our system versus IMDS, Table 5.2 shows improvements we have achieved The most emphasizing improvements are in FA_{rate} and precision, although it seems that the improvement in detection rate is only 2.51%, but the improvement is made in a distance of less than 1.7% to the full detection solution. So the improve in detection rate is the most important achievement in our approach. We have achieved detection rate of 99.7% while keeping accuracy as high as 98.3%. Our method improved state of the art technology in several aspects: accuracy by 5.24%, detection rate by 2.51% and false alarm rate was decreased from 19.86% to 1.51%.

6. CONCLUSIONS AND FUTURE WORK

In this work, we have developed a framework based on mining API calls of executables for detecting malware and malicious codes. This framework includes PE analyzer, feature generator, feature selector and classifier. After exporting Windows API calls from PEs by PE analyzer, various features are generated and selected form API call sets. Then by applying classification methods we can classify PEs as benign or malicious.

Since no data has been published from previous studies, we had to collect malicious executables from various sources such as [10]. Additionally we have made our data set publicly available including the binary sources and the list of extracted API calls for each malware and benign executable.

We examined various classification methods, as the results are showing studied framework outperforms all previous works on malware detection studies using data mining techniques.

To evade being by this framework one can encrypt the whole executable file or use irrelevant API calls in unreachable code. The first evasion technique does not come in handy because the executable which is encrypted as a whole cannot be executed and is harmless by itself. But to overcome the second technique which is easy to perform (yet hardly useful) for malware writers, we are planning to develop a framework for runtime API call detection. For this purpose we need to run executables in a virtual environment and capture all executable's calls and then we can mine execution patterns to extract useful information about the behavior of an executable and address the unreachable code and fake API call problem.

7. ACKNOWLEDGMENTS

This research is financially supported by Iran Telecommunication Research Center (ITRC) under supervision of APA Malware Research and Educational Center at Shiraz University.

8. REFERENCES

[1] Malicious software (malware): A security threat to the internet economy. Technical report, Organization For Economic Co-operation And Development, 2007.

[2] G. Batista, R. Prati, and M. Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 2004.

[3] L. Breiman and A. Culter. Random forests. *Machine Learning*, 2001.

[4] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium*, 2003.

[5] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Wiley Interscience, 2nd edition, 2000.

[6] J. Han. Illimine project. University of Illinois at Urbana-Champaign Database and Information Systems Laboratory, 2005.

[7] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* DataManagement Systems. Morgan Kaufmann, 2nd edition, 2006.

[8] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation. *Springer*, 2004.

[9] S. Hashemi, Y. Yang, D. Zabihzadeh, and M. Kangavari. Detecting intrusion transactions in databases using data item dependencies and anomaly analysis. *Expert Systems*, 25(5), November 2008.

[10] V. Heavens. http://vx.netlux.org.

[11] J. Kephart and W. Arnold. Automatic extraction of computer virus signatures. In *Proceedings of 4th Virus Bulletin International Conference*, pages 178–184, 1994.

[12] J. Kolter and M. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of KDD'04*, 2004.

[13] T. Lee and J. Mody. Behavioral classification. In *Proceedings of 2006 EICAR Conference*, 2006.

[14] M. D. Library. Functions by category (windows). http://msdn.microsoft.com/en-us/library/aa383686(VS.85).aspx.

[15] D. Lo, H. Cheng, J. Han, S. Khoo, and C. Sun. Classification of software behaviors for failure detection: A discriminative pattern mining approach. In *Conference on Knowledge discovery and data mining.* ACM SIG-KDD, 2009.

[16] R. Loand, K. Levitt, and R. Olsson. A malicious code filter. In *Comput. Secur.*, pages 541–566, 1995.

[17] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27, 2005.

[18] M. Pietrek. *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format.* Microsoft, http://msdn.microsoft.com/en-us/magazine/ms809762.aspx, February 1994.

[19] M. Pietrek. *An In-Depth Look into the Win32 Portable Executable File Format.* Microsoft, http://msdn.microsoft.com/en-us/magazine/cc301805.aspx, February 2002.

[20] J. Rabek, R. Khazan, S. Lewandowski, and R. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 76–82, 2003.

[21] M. Schultz, E. Eskin, and E. Zadok. Data mining methods for detection of new malicious executables. In *Security and Privacy Proceedings IEEE Symposium*, pages 38–49, May 2001.

[22] A. Sung, J. Xu, P. Chavez, and S. Mukkamala. Static analyzer of vicious executables (save). In *Proceedings of the 20th Annual Computer Security Applications Conference*, 2004.

[23] Symantec. Symantec internet security threat report: Trends for july-december 2007 (executive summary). Technical report, Symantec Corp, April 2008.

[24] J. Wang, P. Deng, Y. Fan, L. Jaw, and Y. Liu. Virus detection using data mining techniques. In *Proceedings of IEEE International Conference on Data Mining*, 2003.

[25] H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2nd edition, 2005.

[26] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. *Proceedings of 3rd SIAM International Conference on Data Mining (SDM'03)*, May 2003.

[27] Y. Ye, D. Wang, T. Li, and D. Ye. An intelligent pe-malware detection system based on association mining. In *Journal in Computer Virology*, 2008.