Windows API based Malware Detection and Framework Analysis

Veeramani R, Nitin Rai

Abstract**—** Detection of zero day malware has been the great challenge for researchers from long time. Traditional signature based anti-malware scanners detect malware based on their unique signatures. The major drawback of such traditional signatures based scanners is that it has no protection against zero-day or unseen malware. Further usage of packers and obfuscation techniques empowered the malware writers to recreate malware variants quickly with slight or no change in malcode. These new variants are undetectable by traditional signature based scanner until their signatures are not present in database. Therefore researchers are working towards finding patterns or features which have unchangeable characteristics of malware even though the malware mutates or obfuscates itself. To address the limitation of traditional signature based scanner, we propose the malware detection method based on extracting relevant application programming interface (API) calls from sub categories of malware. These malware are categorized based on their infection mechanism and actions performed. And because of their fundamental difference in infection mechanism, they do not share similar type of API calls in all malware categories. In this paper, we elucidate an automated framework for analyzing and classifying executables based on their relevant API calls. We explain all the software components used to make the framework fully automatic for extracting API calls. We further explain the Document Class wise Frequency feature selection measure (DCFS) to get the relevant API calls from the extracted API calls to increase the detection rate. We conclude the paper with our experimental result and discussion.

Index Terms- Malware; Packers; Disassembly; Windows API; Document Classwise frequency

1. Introduction

Any program having the malicious intent can be classified as malware or computer infection program. It is a collective term for any malicious software which performs hidden unintended actions affects the integrity of the system and cause damage, loss without the knowledge of user. The definition of the malware given by McGraw and Morrisett [1] as "any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of system". Filiol [2] defines malware as "a simple or self-replicating program, which discreetly install itself in a data processing system, without user's knowledge or consent, with a view to either endangering data confidentiality or data integrity and system availability or making sure that user to be framed for computer crime". New malware variants are discovered at an alarmingly high rate, some malware families featuring tens of thousands of currently known variants. The need for security is in fact a response to the increasing number of attacks led against information systems. Previous works and literature [3] [4] has shown that one single technique alone cannot detect all types of malware. The two most popular techniques for the malware detection are commonly known as Signature-based and Anomaly-based. Signature based techniques is well known for its high detection rate for known malware whose signatures are present in the database but has low detection or provide no protection from zero day malware. Using the obfuscation technique malware writers are creating new malware without changing the essence of malware. These new malware are the variant of known malware and easily bypass the detection. Anomaly based detection techniques work on the concept of normal and anomalous behavior of the program to decide the maliciousness. The key advantage of anomaly based detection technique is its ability to detect zero-day attack. The success of these techniques depends on what features should be learnt in training phase to discriminate malware and benign accurately. In this work, we followed the static analysis approach to analyze the PE executable based on their API calls. We used static analysis tool IDA Pro to disassemble the binary file to analyze and extract the Windows API.

This paper is organized as follows: the next section briefly describes various types of malware. Section III describes the contribution of the research work. Section IV highlights the background of the work, and Section V elaborates the implementation of automated framework used for the extraction of API calls. The analysis and experimental results are shown in Section VI followed by Section VII mentioning the limitation and future work and finally conclusion in Section VIII.

2. Malware Types

2.1 Virus

A computer virus is a program, a block of executable code, which attaches itself to another program in order to reproduce itself without the knowledge of User. It needs a host program to cause harm. There are different types of computer viruses for example boot sector viruses, parasitic viruses, polymorphic viruses and metamorphic viruses.

2.2 Worm

A computer worm does not require any host program and replicates itself by executing its own code independent of any other program. In general, virus attempt to spread through programs/files on single computer while worm spread throughout a network aiming to infect other connected computers. Well-known examples of worms are Code Red, Slammer, Mydoom, Netsky. They usually exploit security flaws in communicate applications or in network protocols.

2.2 Trojan Horse

A Trojan horse is a software program that appears to be very useful. It performs the desirable function for the users as stated but secretly performs some unauthorized actions like stealing information or harm the integrity of the system. Once installed in the victim's computer, it secretly enables the attacker to access all or part of victim's computer resources. Such Trojan horses can be classified as spyware as well. The most popular Trojan horse programs are Back Orifice, Netbus and Subseven.

2.3 Logic Bombs

A logic bomb is a non self-reproducing malware, which install itself into the system and waits for some trigger incident or external event, such as the arrival of a specific date or time, or the creation or deletion of a specific data item such as a file or a database entry, before performing a damaging or an offensive function.

3 Contribution of the Paper

The main contributions of this paper are as follows:

- Building of malware dataset to perform experiments in the absence of publicly available malware dataset.
- Analysis of the fully automated framework to extract the API calls.
- Malware category-wise relevant API feature selection using Document class-wise frequency and classification.

We build the dataset by downloading the variety of malware executables from VX Heavens source [14]. Even though the malware executables were easily downloadable, these cannot be analyzed straight forward because they were packed and obfuscated. Identification and unpacking of malware is the pre-requisite requirement for the research work. Without proper unpacking and de-obfuscation, the statistical analysis cannot be performed accurately. To deal with packing-unpacking and de-obfuscation of these malware, we implemented an automatic system for identification of packer and unpacking the malware and keep the log record for analysis. After unpacking the malware executables, we used automatic framework to extract the API calls invoked by the executables. Further we performed the DCFS based feature selection measure to get the relevant API calls for each malware category separately to increase the detection and classification accuracy.

4 Background

Static analysis and Dynamic analysis are two primary approaches dominated in this area. Dynamic analyses refer to techniques to profile the actions of the malware binary at runtime [7]. Static analyses refer to techniques to disassemble and analyse the logical structure, flow, and data content stored within the binary itself. While both analysis techniques yield important (and sometimes complementary) insight into the capabilities and purpose of a malware binary, these techniques also have their unique advantages and disadvantages.

Dynamic analysis provides only a partial "effects-oriented" profile of the full potential of a given malware binary. Dynamic analysis cannot reveal the effects of programming logic that fails to execute during the runtime analysis. For example, the malware binary may include unsatisfied trigger conditions (e.g., logic revealed only when certain environmental or temporal conditions are satisfied), or suicide logic that can be triggered when process tracing is detected or when other self-protection conditions are met.

Static program analysis offers the potential for a more comprehensive assessment of the entire code and data of the program. For example, by analysing the sequence of invoked system calls and APIs, performing control flow analysis, and tracking data segment references, it is possible to infer logical code bombs, temporal triggers, and other malicious system interactions. Features such as the presence of network communication logic, registry and OS manipulations, and object creations (e.g., files, processes, inter-process communication) can be detected, whether or not these capabilities are exercised at runtime.

User-level malware programs require the invocation of system calls to interact with the OS in order to perform malicious actions. Therefore, analysing and extracting malicious behaviours from these programs requires the identification of system calls invoked within the code. Although system calls in operating systems are predefined mechanisms for trapping to the kernel and asking for services, application programs may interact with other standard helper modules provided by the OS that eventually trap into the kernel. As an example, in Windows, the Win32 API is a collection of services provided by helper DLLs that reside in user space, while the native APIs are services provided by the kernel. In such a design, the user-level API allows a higher-level understanding of behaviour because most of the semantic information is lost at the native level. Therefore, an in-depth binary static analysis requires the identification of all Windows API calls, and call sequences, made within the program.

Literature has shown that API call can be explored to model the program behavior. Analysis of API calls has been explored for generation of birthmark on portable execution (PE) in [5]. The study on the related area [11] [12] clearly indicates the scope of the API calls and their call sequence in order to detect the program behavior. Other than API, assembly features and hex-bytes features have been successfully tried out on N-gram based statistical analysis for the malware detection but they are not feasible and practical because of very large numbers of features generated.

Understanding the issue of API resolution requires understanding how Windows executables refer to APIs from the various Windows libraries. The PE executable file format includes an import section that determines what DLLs are imported by the executable. The information contained in the The unpacked malware were arranged in the respective categories and fed to the disassembler for API call extraction. For each category the extracted API's were further refined using DCFS measure. Fig. 1 shows the system architecture of an automated process. The following are the steps followed by the automated system. import table is used by the loader at runtime to identify the addresses of the referred APIs so that whenever an API is called, a jump to the API code is executed

5 Implementation of Framework

In this section we elaborate the complete framework for API extraction. Most of the malware in the dataset were compressed, packed and obfuscated. The freely available unpackers like UPX, ASPack, FSG and UPack are used in the automated system to unpack the executables before disassembly and analysis.

- Unpack the malware.
- Extraction of API Calls using IDA Pro and export into Mysql database using ida2sql python plugin.
- Selection of relevant API Calls.

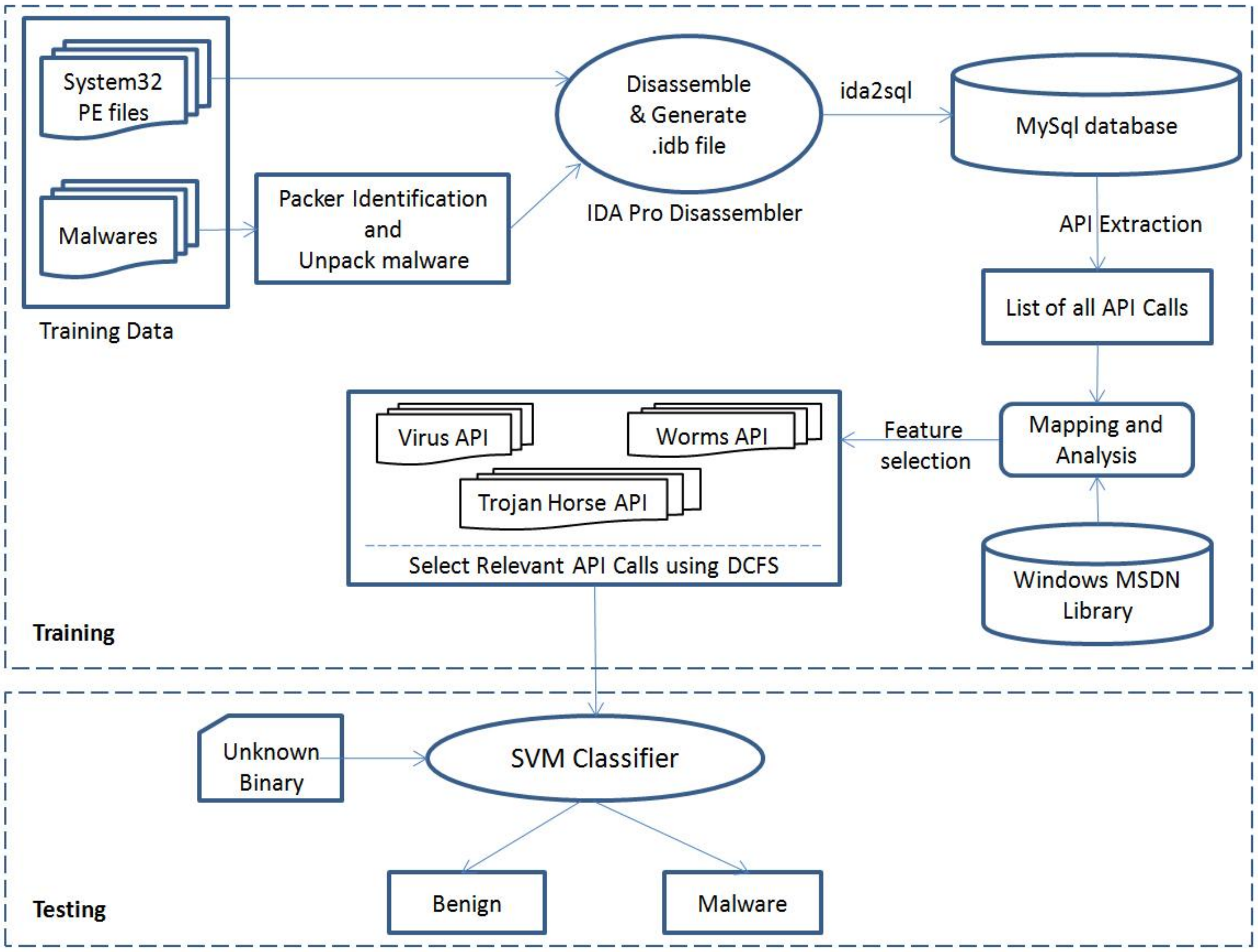

Figure 1. Framework for API Call Extraction and Feature Selection for Malware Detection

Step 1: Unpack the malware

Packers have become the favorite tool for malware writers to create more variants of the existing malware with slight change in malware code. This generates totally new malware having the same malware logic with different signature to avoid detection. Identification of right packer is core of this step. Partial unpacked malware will not provide proper information about the executable. We have implemented java based program to automate the identification of the packer used for malware and then unpack it for further analysis. This is achieved by using the packer identification tool.

Step 2: Extraction of API Calls

The framework uses the most reliable disassembly tool for static analysis, namely, Interactive Disassembler Pro (IDA Pro) [8] since it can disassemble all types of non-executable and executable files (such as ELF, EXE, PE, etc.). It automatically recognizes API calls for various compilers and provide the hooks to call custom defined plugins resulting in incredibly powerful implementation with flexible levels of analysis and control. IDA Pro loads the selected file into memory to analyse the relevant portion of the program. IDA Pro generates the IDA database files into a single IDB file (.idb) after disassembling for analysing the information extracted from the binary.

IDA Pro provides access to its internal resources via an API that allows users to write plugins to be executed by IDA Pro. We have used *idapython* [9] plugin which facilitates us to run the disassembly module automatically for generating the consolidated .idb database. The *ida2sql* plugins used to export .idb database into MySQL database for better binary analysis.

ida2sql plugin generates 16 tables (Address comments, Address reference, Basic blocks, callgraph, control_flow_graph, data, expression_substitution, expression _tree, functions, instructions, metainformation, modules, operand_expressions, operand_strings, operand_tuple, sections) [10] for each and every binary executable. Each of them contains different information about the binary content. For example *Function table* contains all the recognizable API system calls and non-recognizable function names and the length (start and the end location of each function). *Instructions table* contains all the operation code (OP) and their addresses and block addresses. We extracted the list of API calls using Function table. Reference from the Microsoft Developer Network (MSDN) [15] is used for matching and in identifying the windows API's. Java based program was implemented to compare and match the API from MSDN and the API calls generated in the database for the malware sample set. To list all the API calls that are associated with malcode are collected using machine opcodes such as Jump and Call operations as well as the function type.

Step 3: Selection of relevant API Calls

We have used the concepts of relevant API calls and *Class-wise document* frequency for selecting the relevant API calls. The aim is to identify a set of API calls that are common to the set of malware and similarly another set of API calls that are common to the benign executables. Let D be the training set containing a set of malware programs V and a set of benign programs B, i.e., $D = B \cup V$. Pujari et al. [13] proposed a feature extraction method based on n-grams and the feature selection measure is a variant of document frequency. For n-gram Ng, the document frequency δ (Ng, P) of Ng with respect to a program P is 1 if Ng is present in P and 0, otherwise. The *Class-wise document* frequency of Ng with respect to a class C is

$$\sum_{P \in C} \delta(Ng, P).$$

In other words, the Class-wise document frequency is the number of executable programs in C that contain Ng. Fig. 2 is a flow chart describing the selection of relevant API calls using DCFS.

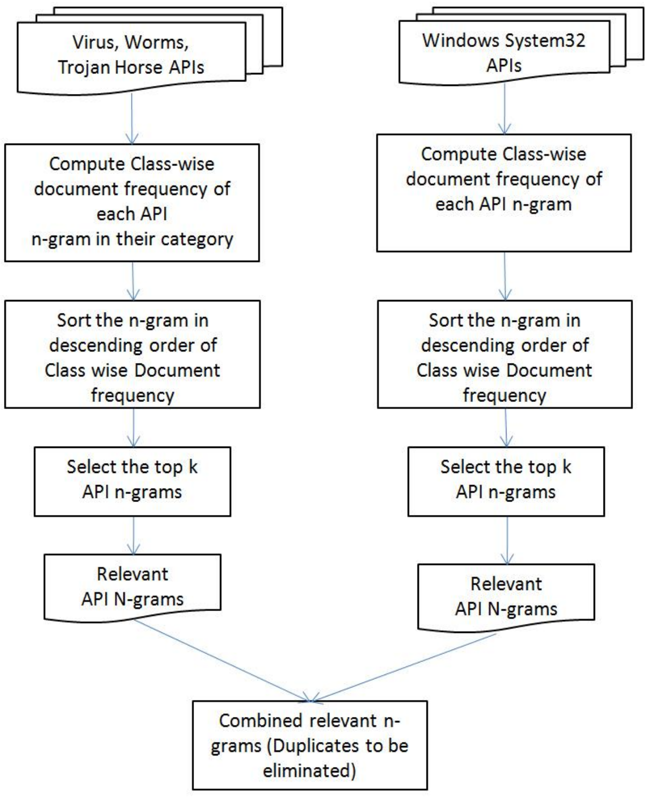
Virus, Worms, Trojan Horse APIs
Windows System32 APIs
Compute Class-wise document frequency of each API n-gram in their category
Compute Class-wise document frequency of each API n-gram
Sort the n-gram in descending order of Class wise Document frequency
Sort the n-gram in descending order of Class wise Document frequency
Select the top k API n-grams
Select the top k API n-grams
Relevant API N-grams
Relevant API N-grams
Combined relevant n-grams (Duplicates to be eliminated)

Figure 2. Flow Chart for selection of Relevant API calls

6 Experimental Results

Malware dataset was built using the executables downloaded from the VX-Heavens [14] website. The malware dataset was prepared keeping variety of malware. We collected 210 malware executables from [14] and 300 benign executables from system32 folder in Windows XP system. All the executables were in Windows PE format. A statistical analysis of the Windows API calling sequence reflects the behaviour of a particular piece of code. In this research work, the relevant API calls were extracted for each category of malware to capture their behaviour. The extracted API's were further refined using document class-wise frequency measure to extract relevant API calls. These relevant API calls were provided to classifier as training to prepare the model to classify the given program as malicious or benign. The experiments were performed on various values of n-gram on SVM classifier. Experiments results are shown in Table 1.

Table 1 Experimental Results For Various Size of N-Grams

Size of n-gram	*Accuracy*
1	97.23 %
2	94.47%
3	93.96 %
4	91.70%

7 Limitations and Future Work

In this work, we focused on the Windows API calls and hence it will be limited to the detection of Windows PE malware. Identification of packer is very important step in this framework. The executables packed with unknown packer cannot be analysed until the correct packer is identified. Hybrid features can be tried to increase the detection accuracy and reduce the false positive.

8 Conclusion

A statistical analysis of Windows API calls in malware reflects the behavior of a piece of code. In this research project, the relevant APIs were extracted from each malware category and further refined using DCFS measure to classify the executable as malicious or benign. The entire static detection process was fully automated for classification system. The experimental results for different sizes of n-grams are promising as a benchmark for improvement of the framework with different set of features thereby increasing the accuracy.

Acknowledgment

This research was conducted at the Centre for Artificial Intelligence & Robotics (CAIR), Bangalore. The authors would like to thank Director of CAIR and Division Head of ISD, CAIR for their support throughout this project. Special thanks to Dr. R. Anitha and Ms. Shina Sheen of PSG College of Technology for their helpful comments and suggestions.

References

[1] Gary McGraw, Greg Morrisett, “Attacking Malicious Code: A report to the Infosec Research Council”, IEEE Software, Sep/Oct 2000.
[2] Eric Filiol, “Computer viruses: from theory to applications”, First edition, IRIS International Series, Springer Verlag France, ISBN 2-287-23939-1, June 2005.
[3] Vinod, P., Jaipur, R., Laxmi, V. and Gaur, M., “Survey on Malware Detection Methods”, Hack. 2009, 74.
[4] Mihai Christodorescu and Somesh Jha, “Testing Malware Detectors”, in Proceedings of ISSTA’04, July 11 - 14, 2004, pages 33-44, Boston, MA USA, ACM Press.
[5] Sharif, M., Yegneswaran, V., Saidi, H., Porras, P. & Lee, W., “Eureka: A framework for enabling static malware analysis”, Computer Security - ESORICS, Lecture Notes in Computer Science (LNCS), Springer, 2008, 5283/2008, 481-500.
[6] Wang, C.; Pang, J.; Zhao, R. & Liu, X., "Using API Sequence and Bayes Algorithm to Detect Suspicious Behavior", 2009 International Conference on Communication Software and Networks, 2009, 544-548.
[7] Willems, C.: CWSandbox: Automatic Behaviour Analysis of Malware (2006), http://www.cwsandbox.org/
[8] Hex-Rays SA, IDA Pro, http://www.hex-rays.com/idapro/, 2008.
[9] Idapython, http:// code.google.com/p/idapython/, 2009.
[10] Zynamics BinNavi, http://www.zynamics.com/binnavi.
[11] Sami, A., Rahimi, H., Yadegari, B., & Hashemi, S., “Malware Detection Based on Mining API Calls”, ACM Symposium on Applied Computing, April 2010.
[12] Alazab, M., Venkatraman, S. & Watters, P., “Malware Detection Based on Structural and Behavioural Features of API Calls”, 1st International Cyber Resilience Conference, Edith Cowan University, Perth Western Australia, 23rd August 2010.
[13] D. Krishna Sandeep Reddy, Arun K. Pujari, “ N-gram analysis for computer virus detection”, Journal in Computer Virology, 231-239, Volume 2, Number 1, August 2006.
[14] VX Heavens, http://vx.netlux.org
[15] Windows API Functions, MSDN, http://msdn.microsoft.com/enus/ library/aa383749%28VS.85%29.aspx., January 2010.
[16] H. Witten and E. Frank., “Data Mining: Practical machine learning tools and techniques”, Morgan Kaufmann, 2nd edition, 2005.
[17] Fanglu Guo, Peter Ferrie, Tzi-cker Chiueh, “A Study of the Packer Problem and Its Solutions”, LNCS, Springer-Verlag Berlin Heidelberg, 2008.
[18] Wei Yan, Zheng Zhang, Nirwan Ansari, “Revealing Packed Malware”, IEEE Computer Society, 2007.

[19] M. Zubair Sha, S. Momina Tabish1, Fauzan Mirza, Muddassar Farooq, "PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime", In Proceedings of the 2009 Recent Advances in Intrusion Detection (RAID) Symposium-Springer.

[20] Osterman Research White Paper, "The Global Malware Problem: Complacency Can be Costly", http://www.ostermanresearch.com/downloads.htm#Security, June 2011.

[21] Abou-Assaleh,T.,Cercone,N.,Keselj,V., Sweidan,R, "Detection of new malicious code using n-grams signatures", In: PST, pp. 193–196, 2004.

[22] O.Kostakis, J.Kinable, H.Mahmoudi, Kimmo Mustonen, "Improved Call Graph Comparison Using Simulated Annealing", SAC 11, ACM, Taiwan, March 2011.

[23] H. Flake, "Structural comparison of executable objects", in Proceedings of the IEEE Conference on Detection of Intrusions, Malware and Vulnerability Assessment (DIMVA), 2004, pp.161 – 173.