Malware Detection with Computational Intelligence

Brittany Byrd, Rakesh Malik, Venkata Kandalam and Qingzhong Liu
Department of Computer Science
Sam Houston State University, Huntsville, TX 77341, USA
Emails: {bas050, rxm058, vnk001, liu}@shsu.edu

Abstract - This work seeks to detect malware with learning classifiers. In this study, three learning classifiers: fisher linear classifier, automatic neural network classifier, and support vector machine are utilized for the analysis and detection of malicious code within a given data set. To achieve this goal we collected and investigated the sequential component of API calls native to the Windows 32-bit host operating system using API Monitor v2 and general sandboxing techniques. The findings indicate that implementation of a support vector machine algorithm (sustaining a 94.9% detection rate) to conclude the malicious nature of an executable is ideal in behavior-based malware detection.

Keywords: malware detection, API system calls, learning classifiers

1 Introduction

The anonymity of the digital world provides an invisibility cloak for cybercriminals to lurk behind undetected by Internet users worldwide. In the past decade, the incentive for malware development and distribution by malware writers has shifted from mere bragging rights or proof of concept, to economical profit (e.g., trade of bank account information, theft of credit card numbers, the collection and sale of sensitive user information, etc.). Motivated by monetary gain, cybercriminals are increasingly encouraged to employ more advanced and persistent modes of evasion to undermine malware detection strategies and tools.

The prevalence of novel malware and its variants have insinuated the cyber-world. Created at an alarming rate of millions per day, the exploitation of automation and obfuscation techniques enable cybercriminals to generate new malware more rapidly than analysts can develop detection signatures [1]. Malware authors seek to exploit the vulnerabilities of the cyber-world via copious avenues (e.g., malware can be propagated through web, e-mail, and human-transported media) through the implementation of varying tactics (e.g., executable binary file, script, and active content). Researchers unanimously agree that new detection methods are crucial in the battle against malware identification, detection and eradication – preferably fast and accurate methods that also use automation techniques and output a low false positive rate.

Created from merging the terms 'malicious' and 'software', malware is generally defined as software that performs varying nefarious actions on host systems (e.g., computer and/or mobile devices) without the authorized consent of the host's owner [2]. Implementation of the malicious code continuously poses an imminent threat to our society's capabilities to safeguard both the security and privacy of cyber users.

The purpose of this work seeks to determine via verifiable methods the detection capabilities of three learning classifiers: 1. fisher linear classifier, 2. automatic neural network classifier, and 3. support vector machine – to detect malware in executable files via segment analysis using inputted Microsoft Windows (32-bit operating system) API system calls derived from both the malicious and benign executables. Testing implemented in this work, aims to demonstrate the detection ability of the constructed testing algorithm, based on the analysis of the collected data set outputted by the executable monitoring software, API Monitor v2 created by Rohitab.com [3].

To achieve this task, we collected hundreds of malicious and benign program samples from VX Heaven (a malware collection library website) and the local Windows 32-bit host directory. Analysis of the samples were conducted using the API Monitor v2 tool to extract an output of API system calls, as input into the malware detection algorithm, as a distributed framework for automatic detection.

We explored the malware detection capabilities of varying learning classifiers, as proof of concepts for determining whether a program is malicious or benign through the analysis of API system calls, specific to the Windows 32-bit operating systems.

The remainder of this paper is organized as follows: Section 2 presents prior and related work. Section 3 discusses implementation of the methodology, including an explanation of the data set, the intricate testing procedures, and the results. Lastly, Section 4 we briefly conclude and present future implications.

2 Related Works

This section discusses the contributions of prior and related work, regarding the developmental effort towards malware-detection methodologies that demonstrate advancement against sophisticated malware evasion strategies.
Current research explicably demonstrates that signature-based detection techniques are an ineffective detection tools against advanced stealth and obfuscation techniques. Signature-based detection use a pattern matching schema (signatures or scan strings) to identify malware [8 – how antivirus]. Prominent among commercial malware detectors (e.g., virus scanners), this exhaustive human-error prone reactive approach only proves effective against known malware – and is dependent on providing detection results after infection, due to its emphasis on syntactic( specific characteristics of individual malware instances) as a complete representation of all malware development and complexities. Hence, the practical shift towards behavior-based detection techniques that offer analysts the opportunity to create detection methods that instead target the semantics (general behavior specifications exhibited by an entire family of malicious code) to detect malware [6, 8, 9].

There is a large body of work in the area of behavior-based malware detection. We focus on the areas most related to our work.

Zhou & Inge [8] presented a malware detection technique that categorizes a program as malware, according a learning engine created by the authors which uses an adaptive data compression model. Our work seeks to analyze malicious code, in contrast to benign executables (native to the Windows 32-bit host operating system). We focused on how each program subset interacted with the Windows API system calls, specifically which DLLs were called and in what order. Similarly, Christodorescu et al. [9] proposed a malware detection algorithm that undermines the weaknesses associated with the use of signature-based detection by incorporating instruction semantics to detect malicious program traits. Their contribution demonstrated that the algorithm can detect all variants of certain malware, has no false positives, and is resilient to obfuscation. Fredrikson et al. [1] also implemented a novel algorithm called HOLMES that demonstrated a detection accuracy of 86% for new unknown malware, with 0 false positives. Their work further elaborated on the effectiveness of behavior-based detection methods overall, in contrast to commercial signature-based anti-virus products.

In contrast, Kolbitsch et al. [10] only focused on the dissection of one malware sample to create a novel malware detection approach. They analyzed the intimate structure and characteristics of the malicious program – later using the analysis results to contrast and test against the runtime behavior of unknown programs to demonstrate the efficiency of their malware detection approach to detect running malicious code on a host system with small overhead. Our data set consisted of the collection of hundreds of benign and malicious executables, wherein our data detection learning strategies were tested via static analysis (e.g., without preprocessing, such as unpacking or disassembling) according to their ability to accurately determine whether a program was malicious based on data analyzed from our sample collection.

3 Detection Method and Experiments

To demonstrate the efficiency of an automated malware detection algorithm, we collected and analyzed the Windows API system calls of a 32-bit host operating system called by both malicious and benign programs.

3.1 Data Set

We collected real malware executable files, specifically known to target the Windows 32-bit operating system, from VX Heaven. VX Heaven is a library collection website that provides a vast selection of known malware source codes that are widely used as an experimental data set [8]. We downloaded and analyzed 200 malware scripts of multiple categories (e.g., backdoor, worm, Trojan, etc.) for this study. These files converted into executables within the Windows platform, form the malware data set.

160 benign executable programs (native to the Windows 32-bit local host operating systems) were retrieved via the directory path C:\Windows\System32. These collected samples were used to perform contrast analysis, in comparison to the malicious files to create detection dependency graphs as input for our detection algorithm. Following, the collection of the sample data set – the execution process of each program was monitored and collected via the API Monitor tool. Our malware samples are derived from varying malicious codes, collectively grouped under the common term malware. Hence, the malware used in our data set can be categorized into six distinct subsets, according to their actions [4, 5, 6, 7]; virus, worms, Trojan, rootkit, distributed denial of service, and backdoor.

3.2 API System Call Extraction

We utilized the API Monitor v2 monitoring tool [3], to construct API system call detection tables as input for the learning classifiers. API system calls used by the both the malicious and benign executables, were individually inputted into the detection tables and numbered in order of call sequence, according to each executable collected.

Collection of the API system calls (specifically the malicious samples) were executed under delicate conditions via dynamic analysis, to appropriately analyze each malicious sample. To achieve this task, the malware must be executed in a realistic environment, while ensuring that any potential for leaks into the live network are removed. For this, we used Oracle VirtualBox virtualization tool as a sandbox to download and run the malware for monitoring and API system call capture. After collecting the API system calls, we constructed detection tables based on the extracted segmented features. Figure 1 shows the output via API Monitor v2 of the API system calls monitored and detached by the malware sample, named Backdoor.Win32.711.). Figure 2 displays a detection table of the API system calls outputted by API Monitor v2, in correspondence with the specified monitored malware samples.
Applications that originated from the same development platform tend to share the same libraries and resources. As a result, malware and benign programs generated using the same development platform (e.g., third-party software companies) can resemble each other through the observation of their execution process regarding the use of API system calls [11].
3.3 Learning Classifiers

To determine whether a program is classified as a malicious executable, the extracted API system calls were inputted into learning classifiers, as segmented classification features.

Classifiers enable statistical classification (the use of an object's characteristics to identify which class the object belongs to) by evoking a classification decision premised on the object's characteristics combination value.

We used three learning classifiers: fisher linear classifier [12]; automatic neural network classifier [13, 15], and support vector machine [14], to perform our malware detection approach by testing and contrasting the detection rate capabilities of each.

3.4 Experimental Results and Analysis

The results indicate that by extracting meaningful features, such as the API system calls from within each different subset (malicious and benign), malware detection is possible via learning classifiers.

In each experiment, 50% benign and 50% malicious of the executables were randomly selected from the data set were used to create the training data set. The other 50% benign and 50% malicious from the data set were used for the testing (prediction) data set – with a mean accuracy of over 100 experiments. The results are 83.2% by using fisher linear classifier, 85.7% by using automatic neural network classifier and 94.9% by using support vector machine with a linear kernel.

Implementation of the three learning classifiers to our data set indicates that each strategy is capable of distinguishing between malicious and benign applications via the extraction and input of API system calls as determinants. However, it is also evident that use of the support vector machine is the more efficient approach.

The results further support the notion that measurable and identifiable differences exist between malicious and benign executables. Hence, deeper investigation into these instances can provide analysts in the development of better detection approaches in the future.

4 Conclusion

Our study shows that it is effective for detecting malicious code by analyzing the Windows 32-bit API system calls with the aid of learning machine techniques.

Our future implications aspire to generate work that will further explore malware detection efforts, from the perspective of algorithm modification (specifically the HOLMES algorithm). This current paper will serve as a platform to our future endeavor that seeks to incorporate the concept of infection markers (detectors used by malware to avoid infecting the same host system more than one time) into undermining the effects of malicious code on Windows Registry keys.

Additionally, other areas of malware analysis using raw features extracted from Windows executables (such as, PE header information, security labels, and instruction code sequences, assembly language mnemonics) are practical options for data set derivation for behavior-based malware detection efforts).

5 References


