2016 International Conference on Collaboration Technologies and Systems

On the Anatomy of the Dynamic Behavior of Polymorphic Viruses

Armando Cabrera
College of Technology
Purdue University Northwest
Hammond, IN, USA
armandocjr@gmail.com

Ricardo Calix
College of Technology
Purdue University Northwest
Hammond, IN, USA
rcalix@pnw.edu

***Abstract*—The sophistication of novel strains of polymorphic viruses, such as Stuxnet, has increased over the last decade. Traditional tools such as anti-virus, firewalls, intrusion detection/prevention systems, etc. may be incapable of detecting such strains. As a result, new methods need to be introduced in order to detect this family of malware. Combining dynamic malware analysis techniques with machine learning tools can prove useful in the progression of developing an effective and efficient classifier. This paper explores the use of dynamic analysis of malware and machine learning to create a classifier for polymorphic virus detection.**

***Keywords*-Stuxnet; dynamic malware analysis; emulators; machine learning; malware detection.**

I. Introduction

As computer networks become more complex, computer network attacks also grow in sophistication. This increase in sophistication leads malware analysts to develop better tools and methods of mitigating and preventing attacks. Tools such as anti-virus scanners, firewalls, intrusion detection/prevention systems, etc. all aid in the defense of malware attacks. However, when polymorphic viruses such as Stuxnet, Duqu and Flame were released to sabotage intelligence and physical facilities [12], the dynamics of the defense against malware shifted. Traditional tools, noted previously, are not equipped to handle such complex attacks [7]. This is because complex malware strains utilize packing, encryption, and other methods to obfuscate their payloads, making detection and analysis more difficult. Because of this, malware analysts must refer to other methods in order to provide a means of defense against these new attacks.

Traditional approaches to malware analysis utilize static and dynamic techniques in order to extract signatures from an unknown strain of malware. Static techniques can range from basic techniques (using antivirus to scan a sample, using hashes to identify a malware, or skimming a malicious file's functions and attributes for potential hints of malicious activity) to more advanced techniques (reverse-engineering a malicious strain in a disassembler to observe what the strain actually does). Similarly, dynamic techniques can range from basic techniques (running the malware on an isolated but live machine to study its effects) to more advanced techniques (using a debugger to extract detailed information from a malicious executable). Although these techniques reveal a great level of detail, they can often times be slow in terms of the time needed to finally extract and generate a signature for the novel malware strain.

Prior research has suggested the use of machine learning techniques [5] in order to provide another means of analysis for these attacks. Machine learning techniques can be utilized to develop classifiers that can detect the presence of malware within a computer network. Many of the traditional techniques, mentioned previously, can benefit from the advantages of using machine learning techniques. However, in order to progress towards this automation, the feasibility and accuracy of such a process must first be analyzed and benchmarked.

The methodology executed in this research analyzed the feasibility of using a dataset generated by emulators, using polymorphic viruses and non-malware as their input, to classify an unknown polymorphic virus strain. More specifically, the research aimed to answer the following questions: (1) Which dynamic analysis techniques and features should be considered in order to best represent a malware's behavior? (2) What kind of accuracy can be achieved by a dataset based on polymorphic viruses and non-malware? (3) Can feature ranking improve the accuracy achieved by the same dataset?

II. Literature Review

The amount of malware generated daily has increased at a rate that does not make manual inspection and analysis feasible [7]. This is due to a shortage in the number of malware analysts available. Not only does such a shortage exist, but the entire process of malware analysis is tedious and slow [8]. Couple this with the fact that malware is growing in complexity, and a rather alarming scenario is occurring; one in which hackers and malicious users have the advantage. As a result, new tools and techniques must be derived to work in conjunction with already existing tools like anti-virus, firewalls, intrusion detection and intrusion detection systems to name a few.

978-1-5090-2300-4/16 $31.00 © 2016 IEEE
DOI 10.1109/CTS.2016.79

A. *Polymorphic Viruses*

Malware can be categorized into different types based on their characteristics and behavior. Some of the different types of malware include viruses, Trojans, worms, polymorphic viruses, etc. For the purposes of this research, a comparison of a simple virus compared to a polymorphic virus will be highlighted. A simple virus works by replicating itself across a machine or network and requires a user to execute and spread the virus itself [1]. On the other hand, polymorphic viruses can range in the sophistication of its spreading mechanism. Polymorphic viruses are often encrypted and also contain mutation engines that can obfuscate manipulate its code in order to hide if from malware detection tools [1]. This makes detecting polymorphic viruses more difficult to detect and create signatures for. Although polymorphic viruses are more challenging to create, the consequences of a polymorphic virus attack can be devastating. Recently, a virus called Stuxnet was identified as a polymorphic virus that caused a significant amount of damage to a system it was designed to sabotage.

B. *Stuxnet*

One of the first viruses coined a "cyber weapon" is known as Stuxnet. This virus was created in an effort to sabotage a nuclear facility located in Natanz, Iran [12]. The way Stuxnet successfully executed this mission was:

1. Stuxnet traversed through an air gapped network, undetected, using a zero day exploit found within a USB driver.
2. After making its way through the network, Stuxnet reported network information back to a Command & Control server (C&C server).
3. The information gathered was used to find and target a specific computer hosting a specific model of Siemens PLC's software.
4. Once Stuxnet located the target, the virus continued executing its mission by embedding itself into the PLC.
5. After the PLC was compromised, Stuxnet manipulated the PLC program to alter its operating parameters, thus destroying the centrifuges that powered the nuclear facility, rendering it useless.

Stuxnet was the first documented virus of its kind. It utilized an unprecedented four zero day exploits to accomplish its task. Stuxnet also utilized 15,000 lines of code [12]. Most advanced viruses relied on a maximum of one zero day exploit to execute their attacks. The advanced attack mechanisms and complex nature of Stuxnet made it extremely difficult to detect. Stuxnet avoided detection from some of the most sophisticated detection mechanisms (anti-virus, intrusion detection/prevention systems, and firewalls).

Its eventual discovery was an accident as it happened to generate unusual traffic patterns that first seemed like a glitch [12]. Had it not been for this coincidental discovery, Stuxnet could have roamed freely for much longer. Its discovery and dissection lead malware analyst on a new chase. This lead to the pursuit of new tools and techniques needed to detect even more complex viruses than Stuxnet.

C. *Static Analysis*

When a new strain of malware needs to be analyzed, one of the first steps that can be performed, is a static analysis [6]. Static analysis is a technique that requires a malware analyst to examine the code of a malware strain without actually running the malware. A binary file of the malware is usually needed in order to examine the strain. Although static analysis techniques can provide information, such as memory dumps, these techniques are usually slow and leave out much of the information produced by the malware [2]. This is because the malware is not running live. When malware is not run live, many of the functions and expected references and dependencies are left out, therefore leaving malware behavior out of the picture. These shortcomings however are covered by dynamic analysis.

D. *Dynamic Analysis*

Dynamic analysis involves running a sample of malware on a live machine. The effects of the malware can then be studied as the machine of interest will become infected. Once the machine is infected, the functions and behaviors of the malware strain can be studied in greater detail [7]. There are a variety of techniques that encompass dynamic analysis. One of the most common techniques is function call monitoring. A function usually consists of code that performs a specific task [3]. In order to intercept these functions, a technique called function hooking is implemented in order to study the effects of a specific function. This same technique is applied to malware strains as they contain functions that carry out the intended behaviors of the strain. The functions found in malware can manipulate registry entries, files and folders, elevate unauthorized users' privileges, etc. The results of function calls can be obtained from emulators.

Emulators allow live malware to be executed in order for their respective behaviors to be observed and documented [11]. Compared to static techniques, which do not allow live malware to be analyzed in real-time, dynamic tools such as emulators provide an efficient means of observing malware behavior. There are several emulators such as CWSandbox, Zero Wine, Anubis, and ThreatExpert which all generate reports on the behavior exhibited by the injected strain of malware. Some emulators have been geared toward automatic dynamic malware analysis, such as CWSandbox [11]. This allows malware experts to identify novel strains of malware. Many emulators work using API hooking and DLL injection techniques in order to identify and extract the behavior exhibited by malware. Although emulators provide a great deal of information regarding malware behavior, this alone does not provide the depth needed to detect more complex viruses like Stuxnet.

Stuxnet went undetected for a number of reasons. From the utilization and exploitation of zero day exploits to the advanced packing and compressing of its payload, Stuxnet utilized many advanced obfuscation and exploitation mechanisms to achieve its mission. Unfortunately, this meant that commercially available tools and techniques came up short when attempting to detect the complex nature of Stuxnet. Since the release of Stuxnet, more variants have been created, adopting the nature and structure of this advanced form of malware. This has lead experts to utilize machine learning techniques to aid in the prevention and mitigation of viruses.

E. Machine Learning

Machine learning involves techniques that provide a system with the ability to learn and predict patterns. These patterns can be derived from past behaviors or can be learned with a variety of algorithms. Algorithms can be a process or a set of rules that can be followed in order to perform a calculation or operation. Machine learning techniques have been applied to malware analysis and have proven themselves useful in discovery of new malware. In one case, a learning-based framework was derived to perform automatic analysis of malware behavior [6]. This approach first executed and ran malware binaries in a sandbox. Characteristics and behaviors such as systems calls were observed and stored in a report. The reports were then combined and inserted in a high-dimensional vector space. In this vector space, the similarity of behavior allowed for the design of powerful clustering and classification methods. Machine learning techniques lead to the creation of prototype vectors, providing an efficient and effective model for detection [6]. The techniques have been proven to identify some viruses.

Another study analyzed the use of machine learning in behavior based malware detection [5]. The study analyzed several algorithms (k-Nearest Neighbor, Naïve Bayes, Support Vector Machine, J48 decision tree, and Multilayer Perceptron) in order to determine the best accuracy possible. In the analysis, it was found that the best performance was achieved by the J48 decision tree algorithm with a true positive rate of 95.9%, a false positive rate of 2.4%, and an overall accuracy of 96.8% [5]. These results solidify confidence in the use of machine learning for malware detection. Unfortunately, other polymorphic viruses have utilized more advanced techniques such as packers which are used in order to obfuscate themselves from these types of techniques. Packing is a process by which a piece of software is run through what is called a runtime packer. These runtime packers compress the software making the original code unreadable. In the case of malware, the content of the malware will not be revealed unless decompressed into memory.

A system named Malwise was derived in order to detect packed and polymorphic malware [2]. The system's approach used entropy analysis and flow graphs to observe and document the behavior of packed malware. The result of this approach was an algorithm that automatically unpacked malware and provided detection of several synthetic malware samples. The system effectively identified variants of malware and showed that there is a high probability that new malware is a variant of existing malware [2].

III. Methodology

This study will focus on using machine learning techniques to extract and generate a classifier for polymorphic virus detection.

A. Data Collection

Before any features could be extracted from the emulators, a database containing various strains of malware, including Stuxnet was created. For the benign software, executable files of well-known software were downloaded and placed into their own database. A combination of malware and non-malware were included in order to balance the dataset described later in the methodology. This balance would reduce the amount of bias generated by skewed data. Features were extracted from Stuxnet and other samples of software that were run through several emulators (Zero Wine, Anubis, and Threat Expert). Features are defined as data generated by the samples as they execute their respective payloads. This data could include function calls, registry edits, DLL injections, file manipulations etc. For each emulator a sample was run through, a CSV file was created containing the features that the respective emulator detected.

The CSV files containing the features were then sanitized by stripping away any XML headers and other artifacts that were generated by the emulators. Once the data was sanitized, a CSV file containing the raw features was left. This process was repeated for each sample that was run on the emulators. Once all of the samples were run through the emulators, the normalized CSV files were then examined by a script that identified each unique feature and then added it to a single CSV file. This was done so that any duplicate features found within the emulator outputs were removed. After this process was completed, a CSV file containing the feature vector combining all of the unique features was generated. Figure 1 below details the pipeline of the process.

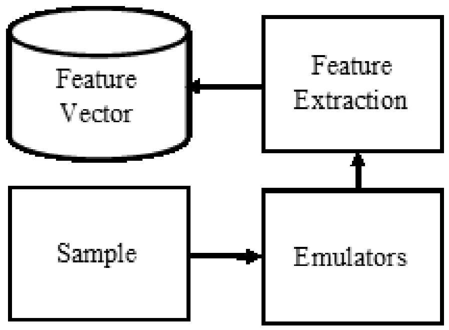

Figure 1. Process pipeline

After the complete feature vector was compiled, each sample (malware and non-malware) was compared to the feature vector so that a binary matrix of each feature matched can be recorded. First, the feature vector and sample vector were loaded. With both files loaded, the feature vector and sample vector were compared for any similarities. If a feature in the sample matched a feature in the feature vector a '1' was added to column signifying that the feature was in fact present in the feature vector. If the feature was not present a '0' was placed in the column instead. This was repeated until a binary matrix of all the features and their class were generated. Table I below details an example of a binary matrix used by this methodology.

TABLE I. BINARY MATRIX EXAMPLE

1	2	3	…	n	class
0	1	1	1	0	**malware**
1	1	0	0	1	**non-malware**

The first row of this binary matrix contains a number that is used to cross reference the feature related to that column. This was done because it was easier to cross reference the large feature vector with a file that contains the list of the names of the features then to locate the name of the feature in a long sequence. After all of the samples were analyzed, a 100 by 52,236 matrix was left to be analyzed. 100 represents the number of samples run (50 malware and 50 non-malware) and the 52,236 represents a cumulative total of the unique activities generated by the samples. The end of each row in this binary matrix contained the class associated with the sample's classification (malware or non-malware). The complete binary matrix would be used for further analysis.

B. *Feature Selection*

The last step in this instance of research was to experiment with feature ranking in order to determine whether or not using ranked features would improve the accuracy of the classifiers. To accomplish this, the binary matrix generated by the previous step was uploaded into Weka. Weka contains a collection of machine learning algorithms and tools that can be used for data pre-processing, classification, clustering and other data mining tasks [10].

With the binary matrix loaded, the select attributes tool in Weka was used. Here, several attribute evaluators could have been selected from. For this research, the *InfoGainAttributeEval* was used in conjunction with its ranker search method. The *InfoGainAttributeEval*, in Weka, evaluates and measures how each feature contributes in decreasing the overall entropy of the input dataset. Using this method would allow a good attribute (feature) to be selected over another. A good attribute is one that contains the most information, thus reducing the most entropy. This was used in order to rank each of the features from the highest impact to the lowest impact. Impact is defined as the amount of influence a particular feature might have with respect to the entire feature vector. For example if a certain feature was found more commonly amongst non-malware, it would rank higher because it was found more often. The features with the highest impact were examined in order to identify whether this smaller collection of features could represent/replace the larger feature vector that contains 52,236 features. These features were grouped into four main categories.

The four categories were: System Calls, Registry Edits, File Modifications, and DLLs. The categories reflect the action a feature was associated with and are the main output of the emulators used in this study. For example, if a sample created a new file as part of its program, the action taken by the program was classified a being related to file modification. These categories were later used to evaluate the selected attributes.

IV. ANALYSIS AND RESULTS

A. *Dataset*

The dataset used in this research consisted of 50 malware samples and 50 non-malware samples. The 50 malware samples all contained malware strains that exhibited characteristics of polymorphic behavior. The 50 non-malware samples included executables of programs that had a reputation of being "safe" to run. The non-malware samples included executables ranging from basic programs (such as file compression software) to more intrusive programs to more intrusive programs (such as anti-virus) The 100 samples were individually run through several emulators and generated a total of 52,236 unique features. An equal amount of samples for both classes (malware and non-malware) was selected in order to avoid skewing the dataset resulting in unbalanced data.

B. *Classification Accuracy (before feature selection)*

Before applying feature selection to the dataset, all 52,236 features were used in evaluating the accuracy of five classifiers found within Weka. The five classifiers used to evaluate the accuracy provided by the dataset are: Naïve Bayes, k nearest neighbor (kNN), J48 decision trees, sequential minimal optimization (SMO), and the RBF classifier. Each classifier yielded a different accuracy highlighted in table II below.

TABLE II. CLASSIFICATION ACCURACY (BEFORE FEATURE SELECTION)

Classifier	ACC (%)	F-measure (%)
Naïve Bayes	61.0	58.4
RBF	72.0	70.3
SMO	73.0	73.0
kNN	81.0	82.0
J48	66.0	65.0

C. *Classification Accuracy (after feature selection)*

Classification was also performed after feature selection was applied to the dataset. Using feature selection, a large dataset still remained. Therefore, several cutoffs were made when selecting the top features used to represent the entire dataset. The cutoffs used were 1000, 2000, 3000, 4000, and 5000 features. These cutoffs were selected because they represented the features with the highest impact compared to the rest of the dataset.

TABLE III. CLASSIFICATION ACCURACY (AFTER FEATURE SELECTION)

Classifier	ACC (%)	F-measure (%)
Naïve Bayes	74.0	74.0
RBF	74.0	74.0
SMO	74.0	73.0
kNN	80.0	80.0
J48	70	70.0

Once the cutoffs were detected, the same classifiers were applied to the modified datasets. The results of the classifier accuracy are presented in Table III below. Although the accuracy and F-measures of some algorithms increased with the application of feature selection, kNN still yielded the best results overall. It was also observed that after selecting a cutoff greater than 4000 of the top features, both accuracy and its respective F-measure decreased. Figure 2 below shows the average accuracy at the different cutoffs selected.

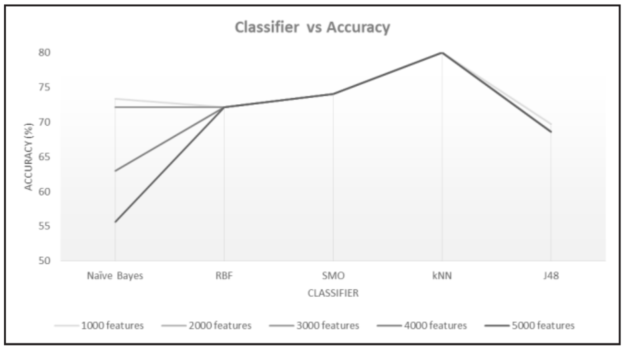

Figure 2. Classifier vs Accuracy

Likewise, the F-measure closely followed the same pattern as shown in figure 3 below.

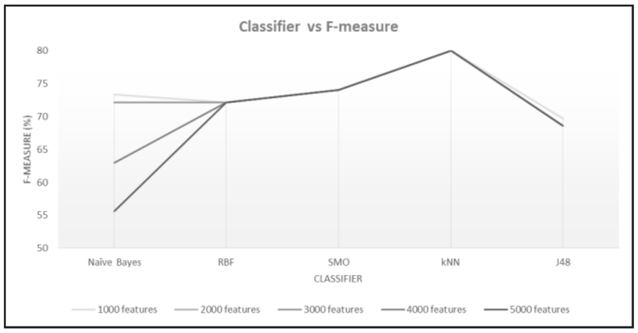

Figure 3. Classifier vs F-Measure

With each selected cutoff, the set of features remaining from the feature selection process was evaluated in order to discuss the category makeup. This was done in order to evaluate the contribution that each feature category had on the overall classification task. As discussed earlier, each feature was grouped into one of four categories depending on the nature of the feature's behavior. This process was performed for the 1000 top features as well as the 2000 top features.

For a complete list of the feature ranking results and a list of the individual feature details the reader is referred to [13].

D. Feature Composition

Analyzing the top 1000 features selected for feature selection, it was observed that 77.4% of the features belonged to the System Call category, 9.2% belonged to the File Modification category, 7.8% to the DLLs category, and 5.3% to the Registry Edits category (figure 4 below).

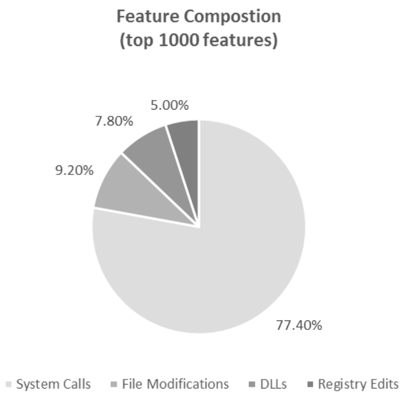

Figure 4. Feature Composition (top 1000 features)

Similarly, for the top 2000 selected features, 88.7% were System Calls, 4.6% were DLLs, 3.9% were File modifications, and 2.65% were Registry Edits (figure 5 below). From this data it can be suggested that the System Call category makes up for most of the features in the feature selection process.

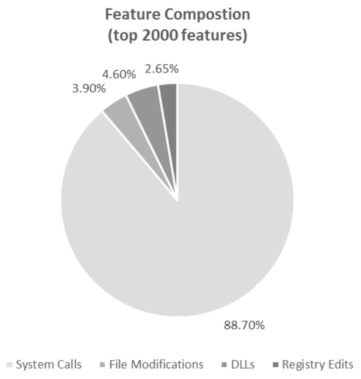

Figure 5. Feature Composition (top 2000 features)

V. CONCLUSION

This paper explored the possibility of utilizing dynamic malware analysis and machine learning in order to create a dataset that would allow for classifiers to determine if a sample binary is either malware or non-malware. Several classifiers were applied to this dataset in order to determine its accuracy. The classifiers applied were Naïve Bayes, k nearest neighbor (kNN), J48 decision trees, sequential minimal optimization (SMO), and the RBF classifier. The classification was also divided into two separate types of analysis. The first type utilized all 52,236 features generated from the dataset. In this approach, kNN yielded the best average accuracy and F-measure with 81% correctly classified instances. The second approach utilized Weka's feature ranking ability in order to identify those features that could best represent the entire feature vector. Because a significant amount of features were still left after ranking (>32,000), several cutoffs were selected in order to best represent the dataset. The cutoffs utilized the top 1000, 2000, 3000, 4000, and 5000 samples in order to observe the accuracy across each of these cutoffs. The classifier that yielded the best accuracy for this approach was kNN again with an accuracy of 80% correctly classified instances. It was also determined that selecting more than the top 4000 samples generated by feature selection decreased the accuracy yielded by the classifiers.

Future work will include using more binaries of both malware and non-malware to create an even larger dataset. Although kNN generated positive results in detecting polymorphic viruses and non-malware, the classifiers could benefit from a larger dataset which in turn could yield higher accuracies.

Finally, the features listed in the feature selection process were evaluated and categorized based on the nature of the features' behavior. It was noted that most of the features (>70%) belonged to the System Call category. This could suggest that malware detection classifiers may benefit from a more detailed analysis of the system calls generated by executing programs. If the feature selection process relies heavily on this category, perhaps more emphasis should be placed on these system calls in order to generate the best results in terms of classification accuracy.

REFERENCES

[1] C. Nachenberg, "Understanding and managing polymorphic viruses," The Symantec Enterprise Papers, vol. 30, p. 16, 1996.

[2] S. Cesare, Y. Xiang and W. Zhou, "Malwise--An Effective and Efficient Classification Ssytem for Packed and Polymorphic Malware," IEEE Transactions on Computers, pp. 1193-1205, 2013.

[3] M. Egele, T. Scholte, E. Kirda and C. Kruegel, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools," ACM Computing Surveys, pp. 1-49, 2012.

[4] N. Falliere, L. Murchu O and E. Chien, "Symantec," February 2011. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[5] I. Firdausi, C. Lim and A. Erwin, "Analysis of machine learning techniques used in behavior-based malware detection," in Advances in Computing, Control and Telecommunication Technologies (ACT), 2010.

[6] K. Rieck, P. Trinius, C. Willems and T. Holz, "Automatic Analysis of Malware Behavior using Machine Learning," Journal of Computer Security, pp. 1-30, 2011.

[7] M. Sikorski and A. Honig, Practical Malware Analysis, San Francisco: No Starch Press, 2012.

[8] P. Szor, Virus Reseach and Defense, Maryland: Addison-Wesley Professional, 2005.

[9] P. Trinius, C. Willems, T. Holz and K. Rieck, "Malheur," 17 December 2009. [Online]. Available: https://ub-madoc.bib.uni-mannheim.de/2579/1/mist.pdf.

[10] "Weka," 2016. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/.

[11] C. Willems, T. Holz and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," IEEE Security and Privacy, pp. 32-39, 2007.

[12] K. Zetter, Countdown to Zero Day, New York: Crown Publishers, 2014

[13] Link: http://rcalix.com/research/Top2000Features.csv