Measuring Malware Evolution Using Support Vector Machines

Mayuri Wadkar
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Artificial Intelligence and Robotics Commons, and the Information Security Commons

Recommended Citation
DOI: https://doi.org/10.31979/etd.ed7k-dh6a
https://scholarworks.sjsu.edu/etd_projects/708

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.
Measuring Malware Evolution Using Support Vector Machines

A Project
Presented to
The Faculty of the Department of Computer Science
San José State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Mayuri Wadkar
May 2019
The Designated Project Committee Approves the Project Titled

Measuring Malware Evolution Using Support Vector Machines

by

Mayuri Wadkar

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Dr. Mark Stamp    Department of Computer Science
Prof. Fabio Di Troia    Department of Computer Science
Dr. Thomas Austin    Department of Computer Science
ABSTRACT

Measuring Malware Evolution Using Support Vector Machines

by Mayuri Wadkar

Malware is software that is designed to do harm to computer systems. Malware often evolves over a period of time as malware developers add new features and fix bugs. Thus, malware samples from the same family from different time periods can exhibit significantly different behavior. Differences between malware samples within a single family can originate from various code modifications designed to evade signature-based detection or changes that are made to alter the functionality of the malware itself. In this research, we apply feature ranking based on linear support vector machine (SVM) weights to identify, quantify, and track changes within malware families over time. We analyze numerous malware families over an extended period of time. Our goal is to detect and analyze evolutionary changes over a wide variety of malware families using quantifiable and automated machine learning techniques.
ACKNOWLEDGMENTS

It is my sincere pleasure to acknowledge the guidance of my graduate advisor Dr. Mark Stamp who provided knowledge, encouragement, feedback and valuable time. His insights and constructive criticism improved this report enormously.

I would also like to thank my committee members Prof. Fabio Di Troia and Dr. Thomas Austin for their valuable feedback and time.

Last but foremost, I would like to thank my dear husband for being patient and supporting me.
# TABLE OF CONTENTS

## CHAPTER

1. **Introduction** ................................................................. 1

2. **Related Work** ................................................................. 3

3. **Design and Implementation** .............................................. 5

   3.1 **Dataset** ................................................................. 5

      3.1.1 Malware Families .................................................. 6

      3.1.2 Portable Executable File Format ................................ 8

   3.2 **Feature Extraction** .................................................... 9

      3.2.1 Byte $N$-Grams .................................................... 9

      3.2.2 Features Extracted From PE Files .............................. 9

   3.3 **Support Vector Machine** ........................................... 10

      3.3.1 Feature ranking based on weights from linear SVM ....... 11

   3.4 **Decision Tree** ....................................................... 11

   3.5 **Chi-Squared Statistic** ............................................. 12

   3.6 **Experimental Approach** ........................................... 13

4. **Experiments and Results** ............................................... 14

   4.1 **System Configuration** .............................................. 14

   4.2 **Zbot Malware Family** ............................................... 14

      4.2.1 SVM based experiment using byte $n$-gram features ....... 14

      4.2.2 SVM experiment using PE file based features ............... 17

      4.2.3 Decision tree based experiment using PE file features ... 20
4.3 Chi-Squared Statistic . . . . . . . . . . . . . . . . . . . . . . . . . 21
4.4 Experiment With Two Similar Malware Families . . . . . . . . . 23
4.5 Experiment With Two Dissimilar Malware Families . . . . . . . 24

5 Conclusion and Future Work . . . . . . . . . . . . . . . . . . . . 26
5.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
5.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

LIST OF REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . 27

APPENDIX

Additional Results . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
A.1 Vobfus Malware Family . . . . . . . . . . . . . . . . . . . . . . 32
A.1.1 SVM based experiment using byte $n$-gram features . . . . . 32
A.1.2 SVM experiment using PE file based features . . . . . . . 33
A.2 Hotbar Family . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
A.3 Vinject Family . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
A.4 Winwebsec Family . . . . . . . . . . . . . . . . . . . . . . . . . 37
A.5 DelfInject Family . . . . . . . . . . . . . . . . . . . . . . . . . 39
A.6 Rbot Family . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
A.7 Obfuscator Family . . . . . . . . . . . . . . . . . . . . . . . . . 42
A.8 Bifrose Family . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
A.9 Zegost Family . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
A.10 Dorkbot Family . . . . . . . . . . . . . . . . . . . . . . . . . . 46
A.11 Hupigon Family . . . . . . . . . . . . . . . . . . . . . . . . . . 48
A.12 CeeInject Family . . . . . . . . . . . . . . . . . . . . . . . . . 49
**LIST OF TABLES**

1. Number of samples used in the experiments. . . . . . . . . . . . . 5
2. Features extracted from PE files . . . . . . . . . . . . . . . . . . . . . . . . 10
3. Indices of top 5 features with the largest variation in weight. . . . . 16
4. Top 3 PE based features with largest variation in weight. . . . . . . 19
5. Number of samples used in the experiment. . . . . . . . . . . . . . . . . . . 23
6. Number of samples used in the experiment. . . . . . . . . . . . . . . . . . . 24
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The PE File Format.</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Byte $N$-Gram Feature Extraction.</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Zbot: Raw weights of $n$-gram features from linear SVM.</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Zbot: Average weights of individual $n$-gram features from linear SVM.</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Zbot: Chi-square Statistic.</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>Zbot: Raw feature weights from linear SVM.</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>Zbot: Average weights of individual features from linear SVM.</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>Zbot: Chi-square Statistic.</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Zbot: Feature importances from Decision Tree Classifier.</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Zbot: Feature importances from Decision Tree Classifier.</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>Vbinject: Chi-squared statistics.</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>Hotbar: Chi-squared statistics.</td>
<td>22</td>
</tr>
<tr>
<td>13</td>
<td>CeeInject-Vbinject: Chi-squared statistics.</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>Zbot-Hotbar: Chi-squared statistics.</td>
<td>24</td>
</tr>
<tr>
<td>A.15</td>
<td>Vobfus: Raw weights of $n$-gram features from linear SVM.</td>
<td>32</td>
</tr>
<tr>
<td>A.16</td>
<td>Vobfus: Average weights of individual $n$-gram features from linear SVM.</td>
<td>33</td>
</tr>
<tr>
<td>A.17</td>
<td>Vobfus: Chi-square Statistic.</td>
<td>33</td>
</tr>
<tr>
<td>A.18</td>
<td>Vobfus: Raw feature weights from linear SVM.</td>
<td>34</td>
</tr>
<tr>
<td>A.19</td>
<td>Vobfus: Average weights of individual features from linear SVM.</td>
<td>34</td>
</tr>
<tr>
<td>A.20</td>
<td>Vobfus: Chi-square Statistic.</td>
<td>35</td>
</tr>
</tbody>
</table>
A.21  Hotbar: Raw feature weights from linear SVM.  . . . . . . . . . .  35
A.22  Hotbar: Average weights of individual features from linear SVM.  36
A.23  Vbinject: Raw feature weights from linear SVM.  . . . . . . . . .  36
A.24  Vbinject: Average weights of individual features from linear SVM.  37
A.25  Winwebsec: Raw feature weights from linear SVM.  . . . . . . . . .  37
A.26  Winwebsec: Average weights of individual features from linear SVM.  38
A.27  Winwebsec: Chi-squared statistics.  . . . . . . . . . . . . . . . .  38
A.28  DelfInject: Raw feature weights from linear SVM.  . . . . . . . . .  39
A.29  DelfInject: Average weights of individual features from linear SVM.  39
A.30  DelfInject: Chi-squared statistics.  . . . . . . . . . . . . . . . . . .  40
A.31  Rbot: Raw feature weights from linear SVM.  . . . . . . . . . . . .  40
A.32  Rbot: Average weights of individual features from linear SVM.  . .  41
A.33  Rbot: Chi-squared statistics.  . . . . . . . . . . . . . . . . . . . . .  41
A.34  Obfuscator: Raw feature weights from linear SVM.  . . . . . . . . .  42
A.35  Obfuscator: Average weights of individual features from linear SVM.  42
A.36  Obfuscator: Chi-squared statistics.  . . . . . . . . . . . . . . . . . .  43
A.37  Bifrose: Raw feature weights from linear SVM.  . . . . . . . . . . . .  43
A.38  Bifrose: Average weights of individual features from linear SVM.  . .  44
A.39  Bifrose: Chi-squared statistics.  . . . . . . . . . . . . . . . . . . . .  44
A.40  Zegost: Raw feature weights from linear SVM.  . . . . . . . . . . . .  45
A.41  Zegost: Average weights of individual features from linear SVM.  . .  45
A.42  Zegost: Chi-squared statistics.  . . . . . . . . . . . . . . . . . . . .  46
A.43  Dorkbot: Raw feature weights from linear SVM.  . . . . . . . . . . . .  46
| A.44 | Dorkbot: Average weights of individual features from linear SVM. | 47 |
| A.45 | Dorkbot: Chi-squared statistics. | 47 |
| A.46 | Hupigon: Raw feature weights from linear SVM. | 48 |
| A.47 | Hupigon: Average weights of individual features from linear SVM. | 48 |
| A.48 | Hupigon: Chi-squared statistics. | 49 |
| A.49 | CeeInject: Raw feature weights from linear SVM. | 49 |
| A.50 | CeeInject: Average weights of individual features from linear SVM. | 50 |
| A.51 | CeeInject: Chi-squared statistics. | 50 |
CHAPTER 1

Introduction

Malware is usually defined as 'malicious software' aimed to cause disruption or denial of activity, gathering private data without consent, allowing unapproved access to system resources, and other improper behavior [1]. Malware detection and prevention is a top priority for governments and businesses. With recent advances in technology (GPUs, Cloud, BYOD/IOT devices, etc.) malware evasion is of great concern too.

Building countermeasures for these threats is becoming very difficult due to increasing complexity of network systems and software. Creators of malware are aware of the details about networks and security mechanisms incorporated in end hosts. They are developing even more sophisticated and effective methods for subverting them. Hackers, enthusiasts, and organized criminals frequently introduce new features intended to enable their malware to evade detection. It is very likely that newer malware is written based of existing ones rather than starting from scratch [2]. Hence, malware can be seen as evolving over time. Usually in software development process, new software is written on top of the existing software. Thus, malware writers be inclined to reuse existing malware code and release new variants of the same malware [3].

Staying informed and up to date on which threats are out there and how they work, will help in developing countermeasures. Thus, understanding malware evolution and identifying the relationship between various malware samples, is of great significance. It will help us to understand the connections among existing malware families and might also be useful to detect unknown malware. Earlier works used reverse engineering techniques to extract knowledge about unknown malware. Using reverse engineering, malware is broken down to reveal its inner workings, architecture, code segments, and
design. In this research, we will use a class of machine learning techniques to measure malware evolution for a wide variety of malware families.

Support vector machines (SVMs) are a popular technique for supervised learning [4]. Feature rankings based on the weights of a linear SVM can be used to eliminate features that contribute little to the model or to gain insight into the training data [5]. In this research we conduct experiments using linear SVM and feature ranking technique to track and quantify changes within malware families over time.

Decision tree (DT) algorithm starts at the tree root and splits the data on the feature that maximizes the information gain. Using this information gain we can determine which features from the given set are most important [4]. In this research, we use feature importances from trained decision tree classifier to validate and support SVM based analysis.

We have considered static features such as byte n-grams for training linear SVM. In addition to byte n-grams, we have used features extracted from binary representation of portable executable (PE) files to train linear SVM and DT classifiers.

This report is organized as follows. In Chapter 2, we discuss related work. Chapter 3 describes machine learning techniques used — SVM and decision trees. These techniques have been discussed briefly along with dataset, features, design, and implementation. Chapter 4 describes experimental results and analysis. In Chapter 5, we conclude our work and also discuss possible future work related to our research.
CHAPTER 2
Related Work

There are many experimental studies of malcode detection published until now. S. Attaluri et al. [6] consider using profile hidden markov models for detecting metamorphic malware. D. Lin and M. Stamp [7] explore shortcomings of hidden markov model (HMM) based detection approach. Similar techniques have been successfully applied for masquerade detection [8]. Singular value decomposition has been exercised in order to detect metamorphic malware [9]. The emphasis of these studies was on a specific area such as detection of metamorphic malware or masquerade detection. Numerous experimental studies of malware classification have been presented until now. However, very few attempts have been made to understand the evolution of malware variants of a specific family.

Evolutionary characteristics of malware have been investigated in [10, 11]. A research that construes the phylogeny of malware was published by Ma et al. [12]. Their techniques discovered significant amount of code sharing between different malware families and accurately captured subtle changes among variants of the same family. Their approach and results are useful in understanding the evolution of remote code injection malware. According to A. Gupta et al. [13], an instance of malware may give rise to many others. Few malware families are active for many years, while others are active for just a few days. Their research revealed many families where specific traits are inherited after many months. They used graph pruning technique to demonstrate inheritance relationships between different variants of malware.

Our research is similar to these studies as in we also aim to establish evolutionary relationships among malware variants. However, in contrast with previous studies, our work focuses on malware variants spanning many years allowing us to offer a broad view into the evolution of malware features. We use automated machine
learning techniques in contrast to the reverse engineering technique used by previous researchers.

Many researchers study broad sets of features like third-party calls [14], API-calls, string-based features [15], permissions and network addresses to distinguish the nature of applications. Such features are extracted from the sets of benign as well as malware samples to build detection tools. The study presented in [16] uses a different method for feature selection, in which $n$-grams are ranked based on frequency and entropy. Their work shows that a class-wise feature selection can improve model efficiency. It involves extracting the top $k$ byte $n$-grams from samples and using those as the feature space. The research in [17] used a feature fusion technique to perform malware family classification. They used structural features such as $n$-grams, entropy, and image representation. Along with hex-dump based features, they also used features extracted from disassembled files such as opcodes, API calls, and sectional information of PE file. Their proposed method achieved a very high accuracy of approximately 0.998.

Byte $n$-grams are a popular feature type used in static analysis and have been used in many prior studies [18, 19, 20]. Dynamic analysis is not required in case of byte $n$-gram features. Without the knowledge of file format, these features enable analysts to learn great deal of information from header and other code sections of an executable file [21]. We also leverage the effectiveness of $n$-gram features in our project. This literature review helped us in choosing the appropriate features and machine learning techniques for our experiments.
CHAPTER 3
Design and Implementation

In this chapter, we first discuss about dataset and features used to conduct experiments. In later sections, we discuss about linear SVM, decision trees and the approach taken to analyze evolutionary changes in malware families.

3.1 Dataset

For this research, we used malware dataset comprised of 26,245 portable executable (PE) files belonging to 13 different malware families. We used malware samples from families listed in Table 1. Each family represents a class of malware which share some traits in how they carry out an attack, their purpose, kind of information they target, how good they are at hiding themselves, etc. We introduce these malware families in Section 3.1.1.

Table 1: Number of samples used in the experiments.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>Number of samples</th>
<th>Time Period (Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zbot</td>
<td>1624</td>
<td>2010 – 2012</td>
</tr>
<tr>
<td>Vbinject</td>
<td>3715</td>
<td>2009 – 2012</td>
</tr>
<tr>
<td>Winwebsec</td>
<td>1460</td>
<td>2011 – 2012</td>
</tr>
<tr>
<td>DelfInject</td>
<td>281</td>
<td>2010 – 2012</td>
</tr>
<tr>
<td>Rbot</td>
<td>782</td>
<td>2004 – 2007</td>
</tr>
<tr>
<td>Hotbar</td>
<td>2437</td>
<td>2011 – 2012</td>
</tr>
<tr>
<td>Obfuscator</td>
<td>2707</td>
<td>2004 – 2011</td>
</tr>
<tr>
<td>Bifrose</td>
<td>1048</td>
<td>2010 – 2012</td>
</tr>
<tr>
<td>Zegost</td>
<td>2452</td>
<td>2010 – 2012</td>
</tr>
<tr>
<td>Dorkbot</td>
<td>526</td>
<td>2010 – 2012</td>
</tr>
<tr>
<td>Hupigon</td>
<td>1442</td>
<td>2007 – 2011</td>
</tr>
<tr>
<td>Vobfus</td>
<td>6643</td>
<td>2010 – 2012</td>
</tr>
<tr>
<td>CeeInject</td>
<td>1128</td>
<td>2009 – 2012</td>
</tr>
</tbody>
</table>

In order to conduct experiments in an efficient manner, the dataset was organized in such a way that malware samples belonging to the same family were kept in the same directory. These were further split into sub-directories based on compilation
month and year of the PE files. Malware samples with altered compilation date or creation date were discarded during initial wrangling process of the dataset.

3.1.1 Malware Families

Zbot is a trojan horse that tries to steal confidential information from a compromised computer. It explicitly targets system data, online sensitive data, and banking information, yet can be modified through the toolbox to accumulate any kind of data. The trojan itself is basically disseminated through drive-by downloads and spam campaigns. It was first discovered in January 2010 [22].

VBinject comprises of worms and trojans that disguise other malware inside it. VBinject is a packaged malware, i.e. a malware that utilizes encryption and compression programming to obscure its contents. Hence, it is hard to recognize other malware that it is concealing. It was first seen in 2009 and again in 2010 [23].

Winwebsec trojan presents itself as an antivirus software. It shows misleading messages to the users stating that the device has been infected and persuades them to pay for the bogus product [24].

DelfInject worm enters a system as a file passed by other malware or as a file downloaded accidentally by clients when visiting malignant sites. DelfInject drops itself on to the system using an arbitrary document name, (for example, xpdsae.exe). It changes the registry entry, hence it runs at each system start. It then injects code into svchost.exe so it can create a connection with specific servers and download files [25].

Rbot is a backdoor trojan that enables attacker to control the computer using an IRC channel. It then spreads to other computers by scanning for network
shares exploiting vulnerabilities in the system. It allows attackers to launch
denial-of-service (DoS) attacks [26].

**Hotbar** is an adware program that installs itself on user’s computer or may be
downloaded by user from malicious website. It displays advertisements as user
browses the web [27].

**Obfuscator** hides its purpose through obfuscation. The underlying malware can
have any purpose [28].

**Bifrose** is a backdoor trojan that allows attacker to connect to a remote IP using
random port number. Some variants of Bifrose have capability to hide files and
processes from user. Attacker can view system information, retrieve passwords,
or execute files by gaining remote control of the system [29].

**Zegost** is a backdoor trojan that injects itself into svchost.exe allowing attacker to
execute files on the compromised system [30].

**Dorkbot** is a worm that steals user names and passwords by keeping tab on what
user does online. It blocks security update websites and can launch DoS attack.
It spreads through instant messaging applications, social networks, and flash
drives [31].

**Hupigon** is a main component of family of backdoor trojans. It opens a backdoor
server enabling other remote computers to control a compromised system [32].

**Vobfus** malware family downloads other malware onto user’s computer. It uses
Windows Autorun feature to spread to other devices like flash drive. It makes
long lasting changes to device configuration that cannot be easily restored just
by removing the threat [33].
CeeInject aims to conceal its motivation in order to prevent detection by security programming software. Many malware families use it as a shield to prevent detection. It can obfuscate a bitcoin mining client, which might be installed on a system to mine bitcoins without the user’s knowledge [34].

3.1.2 Portable Executable File Format

![PE File Format Diagram](image)

This file format is supported by Windows and is an extension of the Common Object File format. We can obtain useful information from executable files such as their compilation time, size of data, number of imports, characteristics, version information, etc. The structure of a typical PE file is shown in Figure 1. The PE design specification allowed us to create a parser to extract features from the executable file and forms the basis for our static analysis. Each component of a PE file has specific standard which can be found at [35]. Files that divert from the given standards might be malicious or infected.
3.2 Feature Extraction

For our initial SVM based static analysis, experiments conducted use byte $n$-gram features extracted from binary form of executable files. Later, the same experiments are repeated using a set of 55 features extracted from portable executable files.

3.2.1 Byte $N$-Grams

Byte $n$-grams are extracted by considering a file as a sequence of bytes. From this sequence, every possible unique combination of $n$ successive bytes is considered as a separate feature. First we read each malware file as sequence of bytes and extracted byte $n$-grams. Top $k$ byte $n$-grams that occurred most frequently in the samples formed our feature space. The $n$-gram frequencies for each malware file form our feature vectors. For every malware file, feature vector was normalized by dividing each cell by the total number of bytes read from that file. The process is summarized in Figure 2. For all our experiments, we considered byte bi-grams ($n=2$) as features.

![Figure 2: Byte N-Gram Feature Extraction.](image)

3.2.2 Features Extracted From PE Files

We extracted 55 novel features from PE files of malware samples which are listed in Table 2. This set of features was used to further enhance our SVM based analysis and results.
Table 2: Features extracted from PE files

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Features</th>
<th>Sr. No.</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Machine</td>
<td>29.</td>
<td>Size of Optional Header</td>
</tr>
<tr>
<td>2.</td>
<td>Characteristics</td>
<td>30.</td>
<td>Time Date Stamp</td>
</tr>
<tr>
<td>4.</td>
<td>Size of Code</td>
<td>32.</td>
<td>Size of Initialized Data</td>
</tr>
<tr>
<td>5.</td>
<td>Size of Uninitialized Data</td>
<td>33.</td>
<td>Address of Entry Point</td>
</tr>
<tr>
<td>6.</td>
<td>Base of Code</td>
<td>34.</td>
<td>Base of Data</td>
</tr>
<tr>
<td>7.</td>
<td>Image Base</td>
<td>35.</td>
<td>Section Alignment</td>
</tr>
<tr>
<td>12.</td>
<td>Size of Headers</td>
<td>40.</td>
<td>Checksum</td>
</tr>
<tr>
<td>13.</td>
<td>Subsystem</td>
<td>41.</td>
<td>Dll Characteristics</td>
</tr>
<tr>
<td>14.</td>
<td>Size of Stack Reserve</td>
<td>42.</td>
<td>Size of Stack Commit</td>
</tr>
<tr>
<td>15.</td>
<td>Size of Heap Reserve</td>
<td>43.</td>
<td>Size of Heap Commit</td>
</tr>
<tr>
<td>16.</td>
<td>Loader Flags</td>
<td>44.</td>
<td>Number of Rva and Sizes</td>
</tr>
<tr>
<td>17.</td>
<td>Number of Sections</td>
<td>45.</td>
<td>Mean Entropy of Sections</td>
</tr>
<tr>
<td>18.</td>
<td>Minimum Entropy of Sections</td>
<td>46.</td>
<td>Maximum Entropy of Sections</td>
</tr>
<tr>
<td>19.</td>
<td>Mean Raw Size of Sections</td>
<td>47.</td>
<td>Minimum Raw Size of Sections</td>
</tr>
<tr>
<td>20.</td>
<td>Maximum Raw Size of Sections</td>
<td>48.</td>
<td>Mean Virtual Size of Sections</td>
</tr>
<tr>
<td>21.</td>
<td>Minimum Virtual Size of Sections</td>
<td>49.</td>
<td>Maximum Virtual Size of Sections</td>
</tr>
<tr>
<td>22.</td>
<td>Number of DLL Imports</td>
<td>50.</td>
<td>Total Number of Imports</td>
</tr>
<tr>
<td>23.</td>
<td>Ordinal Number of Imports</td>
<td>51.</td>
<td>Number of Exports</td>
</tr>
<tr>
<td>24.</td>
<td>Number of Resources</td>
<td>52.</td>
<td>Mean Entropy of Resources</td>
</tr>
<tr>
<td>25.</td>
<td>Minimum Entropy of Resources</td>
<td>53.</td>
<td>Maximum Entropy of Resources</td>
</tr>
<tr>
<td>26.</td>
<td>Mean Size of Resources</td>
<td>54.</td>
<td>Minimum Size of Resources</td>
</tr>
<tr>
<td>27.</td>
<td>Maximum Size of Resources</td>
<td>55.</td>
<td>Load Configuration Size</td>
</tr>
<tr>
<td>28.</td>
<td>Version Information Size</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3 Support Vector Machine

Support vector machine (SVM) is one of the popular techniques for supervised learning and is used for data classification. It establishes a separating hyperplane by widening the separation between two classes of data [4]. Using SVM we can delineate input data to a higher dimensional feature space. SVM solves the following minimization problem given a set of instance-label pairs \((x_i, y_i), x_i \in \mathbb{R}^n, y_i \in \{1, -1\}, i = 1, \ldots, l:\)

\[
\min_{w,b} \frac{1}{2}w^Tw + C \sum_{i=1}^{l} \xi(w, b; x_i, y_i),
\]  

(1)
where $C \geq 0$ is a penalty parameter and $\xi(w, b; x_i, y_i)$ is a loss function. The kernel function for linear SVM is given as $K(x_i, x_j) = x_i^T x_j$. Our SVM experiments are based on a linear kernel function.

SVM trained model assigns weights to each feature used during the training phase. These weights can be used to decide the relevance of each feature [36]. Next subsection describes the feature ranking algorithm based on weights from linear SVM.

### 3.3.1 Feature ranking based on weights from linear SVM

Feature rankings based on the weights of a linear SVM can be used to eliminate features that contribute little to the model or to gain insight into the training data. These can be obtained by sorting the features according to the absolute values of weights in the model. Algorithm 1 briefly describes the steps involved in feature ranking technique.

**Algorithm 1 Feature Ranking**

**Data:** Training Set $x_i, y_i$ where $i = 0, 1, \ldots, n$.

**Result:** List of feature ranks in sorted order.

1: Train linear SVM on the given training set.
2: Rank features from 1 to $n$ based on absolute weights $|w_j|$ where $j = 0, 1, \ldots, n$ assigned by SVM.
3: Sort feature ranks in ascending order.
4: Return top $k$ ranked features.

The value of $|w_j|$ is directly proportional to the significance of $j$th feature. Thus, features are ranked using $|w_j|$ values [36]. However, this approach is limited to linear SVM.

### 3.4 Decision Tree

In decision trees, classification of samples start at root node and samples are sorted based on their feature values. Decision tree splits the dataset samples into smaller subsets in order to predict the target value [37]. Each node in a tree mimics a
feature and each edge represents a decision. The algorithm learns how to best split the dataset using a greedy strategy by making series of locally optimum choices about which feature to use for the split. There are numerous measures for evaluating the goodness of a feature such as entropy [38] and gini index [39]. These are derived based on degree of impurity of the child nodes in a given tree [40]. Typically used impurity measures in practice are mentioned below.

**Entropy** is calculated as,

\[
\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)
\]  

(2)

**Gini index** is calculated as,

\[
\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2
\]  

(3)

**Error** is computed as,

\[
\text{Classification error}(t) = 1 - \max_i [p(i|t)]
\]  

(4)

The splitting of dataset continues until a predefined stopping criterion is met or no further gain can be achieved. These measures enable us to identify the importance of features.

### 3.5 Chi-Squared Statistic

Chi-squared statistic is used to compare the observed frequency distribution with the expected frequency distribution [41]. This technique was proposed by Pearson in 1900 [42]. In mathematical terms, it is a normalized sum of square deviation between observed and expected frequency distributions. It is calculated as,

\[
\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}
\]  

(5)

where \( n \) denotes the number of features or observations, \( \chi^2 \) is the cumulative chi-square statistic, \( O_i \) is observed value of \( i^{th} \) instance, and \( E_i \) is expected value of \( i^{th} \) instance.
In our experiments, this statistic was used to calculate the power-divergence between SVM feature weights from a specific time period and the average feature weights.

### 3.6 Experimental Approach

This section describes an approach taken in order to analyze evolutionary changes that occurred in the malware family over period of time. We selected a malware family from the dataset and tagged each sample in this family according to the date on which it was compiled. Then we trained a linear SVM based on byte $n$-gram features over all samples from a one year time interval. Using this model, we then ranked all features based on the SVM weights. Next, we shifted ahead one month and again trained an SVM on all samples within a one year window, and again ranked the features. Continuing, we obtained a series of snapshots of ranked features based on overlapping sliding windows, each of one year duration and each offset by one month. We used this as a means to trace changes in the significance of various features and to quantify changes. This approach is useful in detecting slow evolutionary changes as well as sudden breaks in the evolution.

Next step was to be able to identify and investigate features of a malware family that undergo significant variation in terms of SVM weights when we shift the training window. Thus, we repeated experimental approach mentioned in previous paragraph using feature set described in Table 2. Choosing this feature set enabled us to analyze individual features and their importances.
CHAPTER 4
Experiments and Results

In this chapter, we present analysis and results for the experiments conducted as part of our research. Prior to that, we describe the configuration of system used to carry out those experiments.

4.1 System Configuration

Since our project involved working with malware, all experiments were conducted in a controlled virtual environment. The configuration for virtual machine is given below.

- Processor: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- CPU: 8 vCPUs
- RAM: 24 GB
- Operating System: Ubuntu 18.04.1 LTS Linux (64 bit).
- Plots and charts were generated using matplotlib library in Python.
- Python sklearn libraries were used for machine learning algorithms.
- PE file based features were extracted using pefile library in Python.
- N-Gram features extracted using N-Gram library in Apache Spark.

4.2 Zbot Malware Family

In this section, we discuss experiments conducted on 1624 malware samples from Zbot family. First we discuss results of SVM based experiments that use byte $n$-gram features. Next, we present results of SVM based experiments that use 55 features extracted from PE files. Lastly, we discuss results of decision tree related experiments.

4.2.1 SVM based experiment using byte $n$-gram features

Initially we trained linear SVM using byte $n$-gram ($n=2$) features of Zbot samples from one year time interval (Jan. 2011 – Jan. 2012). We recorded feature weights from this trained model. Next, we shifted ahead one month and again trained an
SVM on samples within one year time window (Feb. 2011 – Feb. 2012). In this way, we captured series on 12 snapshots, starting at Jan. 2011 and ending at Dec. 2012, based on overlapping sliding windows, each of one year duration and each offset by one month. Figure 3 shows a plot of raw feature weights from all overlapping time windows of size one year. Next, we computed average weights of each individual features across all time windows. The result can be seen in Figure 4.

Figure 3: Zbot: Raw weights of $n$-gram features from linear SVM.

Figure 4: Zbot: Average weights of individual $n$-gram features from linear SVM.
On comparing Figure 3 and Figure 4, it appears that in certain time windows for certain features the variation in weight is significant. Thus, in order to gain insight of features that have considerably large variation in weight from the average value, we fetched indices of top 5 features that vary the most in given time window. Refer to Table 3 for corresponding result of Zbot family. From this result, we see that weights for 593rd and 974th features vary the most in all time windows.

Table 3: Indices of top 5 features with the largest variation in weight.

<table>
<thead>
<tr>
<th>Time Window</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr. 2011 – Apr. 2012</td>
<td>593</td>
<td>974</td>
<td>183</td>
<td>3507</td>
<td>724</td>
</tr>
<tr>
<td>May 2011 – May 2012</td>
<td>593</td>
<td>974</td>
<td>3507</td>
<td>183</td>
<td>2080</td>
</tr>
<tr>
<td>Nov. 2011 – Nov. 2012</td>
<td>593</td>
<td>974</td>
<td>3507</td>
<td>183</td>
<td>724</td>
</tr>
</tbody>
</table>

Further, we computed chi-squared statistics for feature weights obtained from all time windows, which can be viewed in Figure 5. Using this statistic, we can measure power-divergence of observed feature weights from average (i.e expected) feature weights. It indicates time windows in which significant changes were incorporated in malware samples. Based on our analysis in Figure 5, it appears that new Zbot variants were released in February 2012, April 2012, May 2012, August 2012, and November 2012. Such an analysis might help to understand release cycle of a specific malware family. Magnitude can denote the amount of evolution that has occurred during specific time period.
4.2.2 SVM experiment using PE file based features

This experiment is based on Zbot samples compiled from year 2010 to year 2012. Initially, we trained SVM using 55 distinct features extracted from PE files of Zbot samples. Details about these features are given in Section 3.2.2. We captured linear SVM weights for multiple overlapping time windows. Along with raw feature weights, we also computed average weights of individual features. Corresponding results can be viewed in Figure 6 and Figure 7.

We observed that there exists a set of features for which the weight change is high in certain time windows. Next, we captured top 3 such features from each time window for which the variation is high. The result can be seen in Table 4. From this result, we see that TimeDateStamp feature is the most prevalent across all selected time windows. As a similar feature was used to tag samples during SVM training, it is proof of concept that the SVM model used this feature to maximize the margin between classes. Features such as AddressOfEntryPoint and SizeOfImage also played vital role in the separation of classes. However, we can gain useful information from features other
than prevalent features mentioned earlier. Because, these offer insight into attributes that might have been of interest to malware writers in certain time periods.

Figure 8 shows the chi-squared statistic for experiment based on features from PE
Table 4: Top 3 PE based features with largest variation in weight.

<table>
<thead>
<tr>
<th>Time Window</th>
<th>1st Feature</th>
<th>2nd Feature</th>
<th>3rd Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb. 2010 – Feb. 2011</td>
<td>TimeDateStamp</td>
<td>BaseOfCode</td>
<td>SizeOfUninitializedData</td>
</tr>
<tr>
<td>Mar. 2010 – Mar. 2011</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Apr. 2010 – Apr. 2011</td>
<td>TimeDateStamp</td>
<td>SizeOfImage</td>
<td>SectionsMeanVirtualsize</td>
</tr>
<tr>
<td>May 2010 – May 2011</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Aug. 2010 – Aug. 2011</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Sep. 2010 – Sep. 2011</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Nov. 2010 – Nov. 2011</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Feb. 2011 – Feb. 2012</td>
<td>TimeDateStamp</td>
<td>BaseOfData</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Mar. 2011 – Mar. 2012</td>
<td>TimeDateStamp</td>
<td>CheckSum</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Apr. 2011 – Apr. 2012</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>May 2011 – May 2012</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Aug. 2011 – Aug. 2012</td>
<td>TimeDateStamp</td>
<td>SectionMaxRawsize</td>
<td>SizeOfInitializedData</td>
</tr>
<tr>
<td>Sep. 2011 – Sep. 2012</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfImage</td>
</tr>
<tr>
<td>Nov. 2011 – Nov. 2012</td>
<td>TimeDateStamp</td>
<td>AddressOfEntryPoint</td>
<td>SizeOfUninitializedData</td>
</tr>
</tbody>
</table>

Figure 8: Zbot: Chi-square Statistic.
files. From Figure 8, it can be observed that feature weights in certain time windows diverged significantly from their average values. These time periods are April 2011, February 2012, May 2012, August 2012, and November 2012. Chi-squared statistics in Figure 5 and Figure 8 are identical. This indicates that our technique of detecting evolutionary patterns yields consistent results for both sets of features that are used in the analysis.

4.2.3 Decision tree based experiment using PE file features

Using the same 55 features as used in the SVM based analysis, we also trained decision tree classifier in order to obtain feature importances using entropy criterion. The purpose was to be able to relate these feature importances with corresponding feature weights obtained from linear SVM.

Figure 9 shows feature importances from decision tree classifier computed using entropy criterion for Zbot family. The model was trained on entire Zbot dataset from all time windows at once. From this figure, we can notice certain features that are paramount for Zbot malware family such as `SectionMaxRawSize`, `AddressOfEntryPoint`, etc.

![Zbot: Feature importances from Decision Tree Classifier](image)

Figure 9: Zbot: Feature importances from Decision Tree Classifier.
Similar to SVM based analysis, we iteratively trained decision tree classifier on overlapping time windows. In Figure 10, we visualize feature importances from all time windows.

Figure 10: Feature importances from Decision Tree Classifier.

As seen in Figure 10, features such as SectionMaxRawSize, AddressOfEntryPoint, SizeOfInitializedData, and SizeOfImage have high feature importances (gain). Thus, this result can be related to our SVM based analysis as these features also exhibit high feature weights as seen in Figure 4.

4.3 Chi-Squared Statistic

In this section, we present chi-squared statistics for Vbinject and Hotbar malware families. Based on this analysis, we see that every malware family has a different evolution pattern. For instance, feature weights from linear SVM trained on Hotbar malware samples compiled in August 2012 divert significantly from average feature weights. Thus, new functionality might have been introduced in Hotbar samples.
from that time period. Chi-squared statistic for Vbinject family can be seen in Figure 11. From this figure it is clear that Vbinject variants undergone changes rapidly. Refer to Appendix A for feature weight plots and chi-squared statistics of remaining malware families.

Figure 11: Vbinject: Chi-squared statistics.

Figure 12: Hotbar: Chi-squared statistics.
4.4 Experiment With Two Similar Malware Families

Until now we performed experiments on individual malware families. In this experiment we combined two similar family together in order to analyze SVM weights when sliding across the discontinuity. We chose CeeInject and VbInject malware families for this experiment as both the families are from the same category i.e. VirTool and have similar behavior in the wild. Table 5 shows number of samples used in this experiment.

Table 5: Number of samples used in the experiment.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>Number of Samples</th>
<th>Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>CeeInject</td>
<td>765</td>
<td>2011–2012</td>
</tr>
<tr>
<td>Vbinject</td>
<td>2137</td>
<td>2011–2012</td>
</tr>
</tbody>
</table>

From Figure 13, we can see a sudden spike when Vbinject samples were introduced in the training. However, it has flattened for Vbinject samples from June 2011 onwards.
Based on this result we can say that initial versions of Vbinject and CeeInject were different. Also, Vbinject samples compiled after June 2011 exhibit similar nature to that of CeeInject samples.

4.5 Experiment With Two Dissimilar Malware Families

For this experiment, we juxtaposed two dissimilar malware families – Zbot and Hotbar. Both families exhibit significantly different behavior. Zbot is a trojan horse, whereas Hotbar is an adware. Table 6 shows number of samples used for this experiment.

Table 6: Number of samples used in the experiment.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>Number of Samples</th>
<th>Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zbot</td>
<td>1472</td>
<td>2011–2012</td>
</tr>
<tr>
<td>Hotbar</td>
<td>2437</td>
<td>2011–2012</td>
</tr>
</tbody>
</table>

Figure 14: Zbot-Hotbar: Chi-squared statistics.

Figure 14 shows two spikes when Hotbar samples from June 2011 and Dec 2011
were introduced in the training. Apart from these two time periods, the deviation is almost negligible. Based on this we can say that these families share similar characteristics though belonging to different malware categories.
5.1 Conclusion

In this research, we used a novel technique of training SVM model on malware samples from overlapping time windows with some shift offset. This technique proved useful for analyzing malware evolution characterized by a limited complexity in feature design. Our goal was to track the changes in variants of a malware family using quantifiable machine learning techniques. To attain this goal, we used structural as well as content based features. Linear SVM and decision tree classifiers enabled us to analyze importance of features. Using chi-squared statistic, we measured and visualized slow as well as sudden evolutionary changes for 13 malware families.

5.2 Future Work

Our work primarily focused on static features such as byte n-grams and distinct features extracted from PE files. These were then analyzed using machine learning techniques like linear support vector machine and decision tree classifier. Other static features such as opcode n-grams can also be explored using our approach to track evolution. Such features will give a better picture of how malware code evolved over time.

Dynamic analysis helps in analyzing runtime behavior of malware. Dynamic features such as API calls, memory allocation patterns, files and network sockets operations, etc. can be considered for our analysis. These features, although complex to obtain, provide useful information about behavioral traits.

We can also use random forest, deep learning techniques, and genetic algorithm which are found to be effective on data sets we are handling. These techniques may shed light on newer ways in which malware can be detected and/or predict how they mutate.
LIST OF REFERENCES


27


28


[42] K. Pearson, “X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157--175, 1900. [Online]. Available: https://doi.org/10.1080/14786440009463897
APPENDIX

Additional Results

A.1 Vobfus Malware Family

In this section, we present experiments conducted on samples from Vobfus family compiled from year 2010 to year 2012.

A.1.1 SVM based experiment using byte $n$-gram features

Figure A.15 shows a plot of raw feature weights from all overlapping time windows of size one year. Next, we computed average weights of each individual features across all time windows. The result can be seen in Figure A.16. Lastly, we computed chi-squared statistic for feature weights obtained from all time windows. This can be viewed in Figure A.17. From Figure A.17, it appears that new Vobfus variants might have been released in April 2012 time period.

![Figure A.15: Vobfus: Raw weights of $n$-gram features from linear SVM.](image)

Figure A.15: Vobfus: Raw weights of $n$-gram features from linear SVM.
A.1.2 SVM experiment using PE file based features

For this part of experiment, linear SVM was trained using PE file based features. The result can be seen in Figure A.18, Figure A.19, and Figure A.20. From Figure A.20, it can be observed that feature weights from multiple time windows diverted significantly from their average values. From this figure it can be inferred
that variants of Vobfus undergone changes rapidly.

![Vobfus: Raw feature weights from linear SVM](image1)

**Figure A.18:** Vobfus: Raw feature weights from linear SVM.

![Vobfus: Average weights of individual features over time](image2)

**Figure A.19:** Vobfus: Average weights of individual features from linear SVM.

34
A.2 Hotbar Family

Figure A.20: Vobfus: Chi-square Statistic.

Figure A.21: Hotbar: Raw feature weights from linear SVM.
Figure A.22: Hotbar: Average weights of individual features from linear SVM.

A.3 Vbinject Family

Figure A.23: Vbinject: Raw feature weights from linear SVM.
Figure A.24: Vbinject: Average weights of individual features from linear SVM.

A.4 Winwebsec Family

Figure A.25: Winwebsec: Raw feature weights from linear SVM.
Figure A.26: Winwebsec: Average weights of individual features from linear SVM.

Figure A.27: Winwebsec: Chi-squared statistics.
A.5 DelfInject Family

Figure A.28: DelfInject: Raw feature weights from linear SVM.

Figure A.29: DelfInject: Average weights of individual features from linear SVM.
A.6 Rbot Family

![Figure A.31: Rbot: Raw feature weights from linear SVM.](image)

Figure A.31: Rbot: Raw feature weights from linear SVM.
Figure A.32: Rbot: Average weights of individual features from linear SVM.

Figure A.33: Rbot: Chi-squared statistics.
A.7 Obfuscator Family

Figure A.34: Obfuscator: Raw feature weights from linear SVM.

Figure A.35: Obfuscator: Average weights of individual features from linear SVM.
A.8 Bifrose Family

Figure A.36: Obfuscator: Chi-squared statistics.

Figure A.37: Bifrose: Raw feature weights from linear SVM.
Figure A.38: Bifrose: Average weights of individual features from linear SVM.

Figure A.39: Bifrose: Chi-squared statistics.
A.9 Zegost Family

Figure A.40: Zegost: Raw feature weights from linear SVM.

Figure A.41: Zegost: Average weights of individual features from linear SVM.
Figure A.42: Zegost: Chi-squared statistics.

A.10 Dorkbot Family

Figure A.43: Dorkbot: Raw feature weights from linear SVM.
Figure A.44: Dorkbot: Average weights of individual features from linear SVM.

Figure A.45: Dorkbot: Chi-squared statistics.
A.11 Hupigon Family

Figure A.46: Hupigon: Raw feature weights from linear SVM.

Figure A.47: Hupigon: Average weights of individual features from linear SVM.
Figure A.48: Hupigon: Chi-squared statistics.

A.12 CeeInject Family

Figure A.49: CeeInject: Raw feature weights from linear SVM.
Figure A.50: CeeInject: Average weights of individual features from linear SVM.

Figure A.51: CeeInject: Chi-squared statistics.