Spring 5-20-2020

Detection and Analysis of Malware Evolution

Sunhera Barunkumar Paul
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

Paul, Sunhera Barunkumar, "Detection and Analysis of Malware Evolution" (2020). *Master's Projects*. 922.
DOI: <https://doi.org/10.31979/etd.sntt-9mg7>
https://scholarworks.sjsu.edu/etd_projects/922

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.

Detection and Analysis of Malware Evolution

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

Sunhera Barunkumar Paul

May 2020

© 2020

Sunhera Barunkumar Paul

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Detection and Analysis of Malware Evolution

by

Sunhera Barunkumar Paul

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2020

Dr. Mark Stamp Department of Computer Science

Dr. Thomas Austin Department of Computer Science

Fabio Di Troia Department of Computer Science

ABSTRACT

Detection and Analysis of Malware Evolution

by Sunhera Barunkumar Paul

Malware is a malicious software that causes disruption, allows access to unapproved resources, or performs other unauthorized activity. Developing effective malware detection techniques is a critical aspect of information security. One difficulty that arises is that malware often evolves over time, due to changing goals of malware developers, or to counter advances in detection. This evolution can occur through various modifications in malware code. To maintain effective malware detection, it is necessary to detect and analyze malware evolution so that appropriate countermeasures can be taken. We perform a variety of experiments to detect points in time where a malware family has likely evolved. We then conduct further experiments to confirm that such evolution has actually occurred. We validate our approach by considering a number of malware families, each of which includes a significant number of samples collected over an extended period of time. All of our experiments are based on machine learning models, and hence our techniques require minimal human intervention and can easily be automated.

ACKNOWLEDGMENTS

I would like to thank, my project adviser, Dr. Mark Stamp, for his guidance and patience throughout the course of this project, without his persistent help, the goal of this project would not have been realized. I also would like to express my gratitude to my committee members, Dr. Thomas Austin and Mr. Fabio Di Troia, for their assistance and help with the project. Lastly, I want to thank my family and my friends as well, for their continuous efforts to motivate, support and encourage me.

TABLE OF CONTENTS

CHAPTER

1	Introduction	1
2	Background	3
3	Implementation	6
3.1	Dataset	6
3.2	Feature Extraction	9
3.3	Background of Classification Techniques	10
3.3.1	Support Vector Machines (SVM)	10
3.3.2	χ^2 Statistic	12
3.3.3	Word2Vec	12
3.3.4	Hidden Markov Model (HMM)	17
3.3.5	Experimental Approach	20
4	Results and Analysis	22
4.1	Hidden Markov Model (HMM)	22
4.2	Opcodes — SVM	24
4.3	Word2Vec — SVM	29
4.4	HMM — SVM	32
5	Conclusion And Future Work	37
5.1	Future Work	39
	LIST OF REFERENCES	40

APPENDIX

Appendix	43
--------------------	----

LIST OF TABLES

1	Number of samples used in experiments	6
2	Sliding time window example	12
3	HMM notation	18

LIST OF FIGURES

1	Word2Vec model architecture	13
2	Hidden layer of Word2Vec model	14
3	Types of Word2Vec models	15
4	Using softmax classifier in Word2Vec	16
5	Using negative sampling in Word2Vec	17
6	Hidden Markov Model	18
7	Hidden Markov model trained before and after the spike	23
8	χ^2 similarity graph for Zegost family using opcodes	25
9	χ^2 statistics on using opcodes as input	26
10	χ^2 statistics of Zegost family on using n -gram	27
11	χ^2 statistics of different families on using n -gram	28
12	χ^2 statistics of Zegost family on using Word2Vec-SVM	29
13	χ^2 statistics of other families on using Word2Vec-SVM	31
14	χ^2 statistics of Zegost family on using HMM-SVM	33
15	χ^2 statistics of other families on using HMM-SVM	34
A.16	HMM verification technique on Figure 12 showing evolution on Zegost using Opcode-Word2Vec-SVM	43
A.17	HMM verification technique on Figure 13(a) showing evolution on Bho using Opcode-Word2Vec-SVM	43
A.18	HMM verification technique on Figure 13(h) showing evolution on Vobfus using Opcode-Word2Vec-SVM	44
A.19	HMM verification technique on Figure 13(d) showing that the family has no evolution on Ceeinject using Opcode-Word2Vec-SVM	44

A.20	HMM verification technique on Figure 15(c) showing evolution on Bho using Opcode-HMM2Vec-SVM	44
A.21	HMM verification technique on Figure 15(d) showing evolution on Bifrose using Opcode-HMM2Vec-SVM	45
A.22	HMM verification technique on Figure 15(e) showing evolution on Ceeinject using Opcode-HMM2Vec-SVM	45
A.23	HMM verification technique on Figure 15(h) showing evolution on Hupigon using Opcode-HMM2Vec-SVM	45
A.24	HMM verification technique on Figure 15(f) showing that the family has no evolution on DelfInject using Opcode-HMM2Vec-SVM	46
A.25	HMM verification technique for second spike on Figure 15(g) showing that the family has no evolution on Dorkbot using Opcode-HMM2Vec-SVM	46

CHAPTER 1

Introduction

Malware is malicious software that causes disruption in normal activity, allows access to unapproved resources, gathers private data of users, and similar improper behavior. Developing measures to detect malware is a critical aspect of information security.

Malware often evolves due to changing goals of malware developers, advances in detection, and so on [1]. This evolution can occur through various modifications in the code. It is useful to detect and analyze malware evolution so that appropriate measures can be taken to improve the effectiveness of detection techniques [2].

In most previous works on malware evolution, reverse engineering methods have been employed, since a large number of malware samples need to be analyzed, this is a highly labor-intensive process. Whereas other researches have analyzed properties of malware evolution through a graph pruning technique [3]. Recent work has been conducted on malware evolution, which is primarily based on PE file features, support vector machines (SVM), and χ^2 similarity [4]. This latter research was successful and has the advantage of being easily automated, with no reverse engineering required. Our proposed research can be viewed as an extension of and improvement on the groundbreaking work in [4].

We have performed several experiments in this research to detect points in time where a malware family has likely evolved, and further experiments to confirm that such evolution has actually occurred have also been performed. These experiments have been conducted using a significant number of malware families, each of which including a large number of samples over an extended period of time. And, all these experiments are based on machine learning models.

For a particular given malware family, we have first separated all the samples based on windows of time and the opcode sequence from every sample has been extracted. We have then used these opcodes as the input features to detect malware evolution. We have implemented different ways in which we can modify the feature vectors on which a linear SVM is trained and then using the average feature weights obtained from the linear SVM model, we plot a χ^2 similarity graph and obtain our observations about whether a family has evolved or not from it. To confirm malware evolution, we have also trained Hidden Markov Models (HMMs) on either side of the most significant spikes in the χ^2 similarity graph, and then have used both of these models to score samples on either side of the spike. If a clear distinction between scores across such a spike is observed, it shows that the samples differ substantially on either side.

Conclusively, the main objective of this project is to implement different machine learning models and use them to detect malware evolution in a malware family. 15 different malware families have been experimented with, in this project utilizing the above mentioned techniques, and interested results for the same have also been obtained. Furthermore, Chapter 2 involves a discussion about the previous work that has been done in this area. Chapter 3 provides an overview about the dataset that is used and the different machine learning models and techniques that have been used in the implementation of this project. Chapter 4 discusses the results that have been obtained by implementing the various mentioned techniques. The conclusion of the project work and the future scope of this project has been described in Chapter 5.

CHAPTER 2

Background

Comparing with the work done in the fields of malware classification and detection using various machine learning techniques, it can be evidently seen that very little work has been done in the field of malware evolution. A vast amount of phenomenal research has been seen in the field of malware classification and its detection. Although the research is vast and extremely informative, it is equally important that malware evolution is also studied and researched, with respect to its detection techniques using machine learning. It might be very easily possible that the intent of the malware family might change, and with its continuous evolution, it is also possible that the technique required for detection will require evolving as well. Hence to protect and prevent unnecessary malware attacks, focus should be equally given to malware evolution.

A robust research work on malware is observable in [3]. This research is built on extensive and diverse malware data-set spanning from nearly two decades, demonstrating that inheritance properties of malware also form to be a major factor in the consideration. A graph pruning technique has been practiced to deduce this data-set. This research likewise also has claimed and revealed that many specific traits of families have been “inherited” ,from other families of the data-set. However, it is also not notably attested that all of the mentioned “inherited” traits are indeed inherited, as opposed to the matter that these traits may have risen from alternative sources or independently developed. Furthermore, the graph-based analysis in [3] requires “extensive manual investigation” ,in contrast to the technique which is considered in this paper.

As opposed to considering changes in Android non-malware, or goodware, the authors in [5] have extracted a range of features, determining the trends based on standard metrics of software quality by considering changes in the quality of Android

malware. This work fairly surprisingly also shows that the trends in malware and good-ware are noticeably similar, implying that malware writers have been striving to improve their software necessarily to the same level of efficiency as the good-ware writers on the other hand.

The paper [6] displays the plausibility to detect new variants, which is also an essential evolution detection problem. The authors have rather considered the problem of detecting malware variants, which might otherwise defeat the machine learning based detectors. This approach mainly relies on an extensive and vast feature set, employing semi-supervised learning, while the approach in this paper has a greater advantage of being an unsupervised technique along with general code modifications being easily detectable.

The work in [7] nominally focuses on malware taxonomy, however, displaying a great deal of detail in their analysis. This analysis provides significant insight into malware evolution, including “genealogical trajectories”, with this work being outlined on features that have been extracted from malware encyclopedia entries (developed by antivirus software vendors, such as Trend Micro). Based on support vector machines (SVM), using language processing concepts, a sophisticated machine learning approach has been applied to these features that have been extracted from the malware entries.

Machine learning models are mainly trained on features. Thus, extracting appropriate features from the dataset to obtain the results is very important. Malware researchers have broadly classified the features as static and dynamic. The ones that can be obtained without executing the code are known as static features and the ones collected by executing the code are known as dynamic features. Thus, static features are more efficient and easy to collect whereas dynamic features are more robust with techniques [8].

The work in [4] is focused on using static features of the malware samples namely, the PE file features. The research uses the PE file features of the malware samples in a family and uses them as feature vectors to train a linear SVM. The SVM models are trained over various different windows of time periods to analyze in which time period a change has occurred. Then a χ^2 distribution graph is plotted using the average feature weights obtained from the models. This graph shows a significant change over the different time windows which can be termed as a change in the malware family, that is, an evolution. This approach was successful and was easily automated with no reverse engineering required.

The research in this project is related to the work seen in [4]. The experiments in this project can be viewed as the extension of the groundbreaking work done. Using the above mentioned research, we want to experiment using opcodes as the input features to train the linear SVM model. We also use different machine learning models on the opcodes to help obtain feature vectors and then train SVM models on it. Since, there was no technique used to confirm the evolutionary changes in a malware family in [4], this project also deals with the verification of the results obtained from the χ^2 distribution graph.

CHAPTER 3

Implementation

In this chapter, we give a broad summary of the malware families and the dataset used in the research. We will also be discussing the different features and machine learning models that were used to conduct our experiments. All these taken together forms the basis of our evolutionary analysis.

3.1 Dataset

Not explicitly defined, although, a malware family normally represents a collection of malware samples that have major traits in common. Certainly, over time, every family will have a code base that would definitely have many shared similarities in their characteristics. This research makes use of a malware dataset consisting of 26,245 Windows portable executable (PE) files, belonging to 15 distinct malware families. Table 1 exhibits the distinct malware families and the number of samples per family that have been used in the research.

Table 1: Number of samples used in experiments

Family	Samples	Years
Adload	791	2009–2011
Bho	1116	2007–2011
Bifrose	577	2009–2011
CeeInject	742	2009–2012
DelfInject	401	2009–2012
Dorkbot	222	2005–2012
Hupigon	449	2009–2011
Ircbot	59	2009–2012
Obfuscator	670	2004–2017
Rbot	127	2001–2012
Vbinject	2331	2009–2018
Vobfus	700	2009–2011
Winwebsec	1511	2008–2012
Zbot	835	2009–2012
Zegost	506	2008–2011

Of the data-set used in this research, two malware families (Winwebsec and Zbot) are acquired from the Malicia dataset [9], while the remaining thirteen families are extracted from a larger data-set collected using VirusShare [10]. The latter data is greater than half a terabyte in size and contains about 500,000 malware executables. Upon request, these data-sets are made available, by the above mentioned authors.

The prime motivation for choosing the mentioned specific malware families for this research, is that these families have sufficient amount of samples required for training, also satisfactory to the time window that is in consideration. We need sufficient number of malware samples in each family to perform our experiments.

To enhance the efficiency of experimentation in this work, we have organized all the malware samples in each family, as per its creation date. During this initial wrangling process, samples having altered creation date or those that have undergone altered compilation, have been discarded [4].

Further on, we briefly discuss each of the malware families in our dataset. Note that these families represent a wide variety of types of malware, including Trojan, worm, adware, backdoor, and so on.

Bifrose is another backdoor trojan that allows an attacker to connect to a remote IP using a random port number. Some variants of Bifrose have the capability to hide files and processes from the user. An attacker can view system information, retrieve passwords, or execute files by gaining remote control of the system [11].

CeeInject aims to conceal itself from detection, and malware families use it as a shield to prevent detection. For example, CeeInject can obfuscate a bitcoin mining client, which might be installed on a system to mine bitcoins without the user's knowledge [12].

DelfInject is a worm that enters a system from a file passed by other malware, or as a file downloaded accidentally by clients when visiting malignant sites. DelfInject drops itself onto the system using an arbitrary document name (e.g., xpdsae.exe) and alters the relevant registry entry so that it runs at each system start. The malware then injects code into svchost.exe so that it can create a connection with specific servers and download files [13].

Dorkbot is a worm that steals user names and passwords by tracking a user's online activities. It blocks security update websites and can launch DoS attack. Dorkbot is spread via instant messaging applications, social networks, and flash drives [14].

Hotbar is an adware program that installs itself on a user's computer, or it may be downloaded by a user from a malicious website. Hotbar displays advertisements as the user browses the web [15].

Hupigon is a family of backdoor trojans. This malware opens a backdoor server enabling other remote computers to control a compromised system [16].

Obfuscator hides its purpose through obfuscation. The underlying malware can have virtually any payload [17].

Rbot is a backdoor trojan that enables an attacker to control an infected computer using an IRC channel. It then spreads to other computers by scanning for network shares and exploiting vulnerabilities in the system. Rbot includes many advanced features, and it has been used by attackers to launch denial-of-service (DoS) attacks [18].

VBinject primarily serves to disguise other malware s that can be hidden inside of it. VBinject is a packaged malware, i.e. a malware that utilizes techniques of encryption and compression to obscure its contents. Thus making it difficult to recognize any other malware that it is concealing. VBinject was first seen in 2009 and appeared again in 2010 [19].

Vobfus is a malware family that downloads other malware onto a user’s computer. It uses the Windows autorun feature to spread to other devices such as flash drives. Vobfus makes long lasting changes to the device configuration that cannot be restored simply by removing the malware from the system [20].

Winwebsec is a trojan that presents itself as an antivirus software. It shows misleading messages to the users stating that the device has been infected and attempts to persuade the user to pay to remove non-existent malware [21].

Zbot is a Trojan horse that attempts at stealing confidential information from a compromised computer. It explicitly targets system data, online sensitive data, and banking information. It can also be easily modified to accumulate other kinds of data. The Trojan itself is generally disseminated through drive-by downloads and spam campaigns. Zbot was originally discovered in January 2010 [22].

Zegost is a backdoor trojan as well, that injects itself into svchost.exe, thus allowing an attacker to execute files on the compromised system [23].

3.2 Feature Extraction

Opcodes are known to be machine level language instructions which specify a particular operation that has to be performed [24]. For this project, we have malware portable executable files as the dataset, so we extract opcodes from these executable

files to use as input features to the machine learning models. We have also segregated the malware samples in each family according to its creation year. Thus, as features we are using the opcodes from the malware samples in each family grouped according to its timestamp.

3.3 Background of Classification Techniques

In this section we will be discussing the techniques that we have used in this project.

3.3.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) is amongst the popular class of supervised machine learning algorithms, attempting to finding a separating hyper-plane, between two labeled classes of data [25]. By utilizing the so-called “kernel trick”, an SVM easily maps input data to a higher dimensional space, where the additional space has the ability to afford a greater opportunity, to find a separating hyperplane. For every individual feature used in training, a linear SVM has assigned a well-defined weight to it improving the efficiency of ranking of the features as these weights primarily define the relative importance that SVM has placed on each feature.

The aforementioned feature ranking process, is based on the weights obtained from linear SVMs, which makes it a very crucial aspect of the SVMs resulting in making them very useful and as the main machine learning technique to tackle the malware evolution problem that we are considering in this project.

The described feature rankings, that are based on the weights of a linear SVM, can be utilized in eliminating features that contribute very little to the model but are beneficial in obtaining an insight into the training data. The ranking is produced by directly sorting the features according to the absolute values of the corresponding weights in a linear SVM model.

The following steps are followed to train a linear SVM:

1. Given training data (X_i, z_i) for $i = 1, 2, \dots, n$ where each X_i is a real-valued feature vector of length m , and each $z_i \in \{-1, +1\}$ is the corresponding class label.
2. Train a linear SVM on the given training set to obtain the set of model weights w_j , for $j = 1, 2, \dots, m$.
3. Rank the features in decreasing order of significance, based on the absolute value of the corresponding weights. That is, determine a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_m)$ for which the weights of the linear SVM satisfy $|w_{\pi(j)}| \leq |w_{\pi(j+1)}|$, for $j = 1, 2, \dots, m - 1$.
4. Return the m ranked feature weights, $w_{\pi(1)}, w_{\pi(2)}, \dots, w_{\pi(m)}$.

Here, the value of $|w_j|$ is directly proportional to the significance of j^{th} feature in the trained linear SVM. Since the use of a nonlinear kernel function does not enable us to directly determine the relative importance of individual features, it is important to note that this feature ranking technique is restricted to a linear SVM.

The two classes in the SVM are defined as follows: All the samples within the most recent one-year time window comprise class are classified as class “+1”, while all samples from the current month are defined as class “−1” [4]. For example, in Table 2 we give three consecutive time windows, along with the time frames corresponding to the two classes in each case.

Table 2: Sliding time window example

Time Window	Class +1	Class -1
Jan. 2011–Jan. 2012	Jan. 2011–Dec. 2011	Jan. 2012
Feb. 2011–Feb. 2012	Feb. 2011–Jan. 2012	Feb. 2012
Mar. 2011–Mar. 2012	Mar. 2011–Feb. 2012	Mar. 2012

3.3.2 χ^2 Statistic

Defining in mathematical terms, the χ^2 statistic is a normalized sum of square deviation between the observed and expected frequency distributions. This statistic is calculated as

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

where n denotes the number of features or observations, χ^2 is the cumulative statistic, O_i is the observed value of the i^{th} instance, and E_i is expected value of i^{th} instance.

For experimentation purposes in this project, this statistic is considered over a specific time window and is used in quantifying the differences obtained from SVM feature weights, as compared to using the average feature weights over extended periods of time. In this context, the extended periods of time is termed as a year and the specific time window as a month. Thus, to explain in simplified terms, we consider computing the difference in the χ^2 statistic between models for a month and then use the average model over the most recent year. This χ^2 statistic is the plot through overlapping time windows, and any points in the graph where there is a substantial change (i.e., a “spike”), the graph indicates a point where the code base changes significantly from the baseline. These are points of interest, since they indicate the times at which the code has been significantly modified [4].

3.3.3 Word2Vec

This section in the paper discusses about the Word2Vec language model technique. This technique here has been used to obtain word embedding for the extracted sequence

of opcodes from the malware sample files. The word embeddings are essentially the vector representations of a word and the Word2Vec model technique makes the use of these word embeddings to find the similarity between words, as the main objective of using this technique is to find words with similar context, i.e close spatial positions [26].

Word2Vec is a shallow, two-layer neural networks as in Figure 1 which is trained to recreate the linguistic contexts of words. It creates a vector space, ranging to a dimension that is as large as several hundred dimensions, by taking in a large corpus of words [27]. Every word from this corpus is assigned to a corresponding vector in the space, with words sharing common contexts in the corpus placed in close proximity of others vectors in the space and are accordingly positioned in the vector space. Word2Vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text [27].

Figure 1: Word2Vec model architecture

Besides having a single hidden layer, like all neural networks, it has weights, and during training, its goal is to adjust those weights to reduce a loss function. However, here we are not using Word2Vec for the task it was trained on previously, instead, we are just taking its hidden weights, and using them as our word embeddings as depicted in Figure 2.

Figure 2: Hidden layer of Word2Vec model

In neural networks, a word cannot be fed as a string. Thus, here we feed the words as one-hot vectors, which essentially means that the vector is of the same length as the vocabulary, filled with zeros, except at the index that represents the word we want to represent, which is assigned “1”.

The hidden layer of Word2Vec is a standard, fully-connected (Dense) layer, with its weights being the word embeddings [27]. The output layer of the Word2Vec produces the probabilities for the target words from the vocabulary, and the rows of the hidden layer weight matrix, are the word vectors (word embeddings) we need. This results in the hidden layer operating as a lookup table and the output of the hidden layer is just the “word vector” for the input word (Figure 2). This Word2Vec model can be implemented using two distinct approaches, the continuous bag-of-words model and the skip-gram model as in Figure 3. Algorithmically, these models are similar.

Figure 3: Types of Word2Vec models

A *continuous bag-of-words* (CBOW) model predicts target words (e.g. ‘mat’) from the surrounding context words (‘the cat sits on the’). Statistically, it has the effect that CBOW smooths over a lot of the distributional information (by treating an entire context as one observation). This turns out to be a useful thing for smaller data-sets, for most parts in consideration [27].

In contrast to CBOW, a *skip-gram* model predicts surrounding context words from the target words (practically the inverse of CBOW). Looking at the statistics, skip-gram treats each context-target pair as a new observation, and this tends to perform better when we are dealing with larger data-sets.

A Word2vec model can also be trained with hierarchical softmax and/or negative sampling, but usually, only the negative sampling is used. A *hierarchical softmax* method uses a Huffman tree to reduce calculation, to approximate the conditional log-likelihood a model seeks to maximize. Hierarchical softmax works better for infrequent words [27]. Naturally, as training epochs increase, hierarchical softmax stops being useful.

Figure 4: Using softmax classifier in Word2Vec

Alternative to sampling the whole vocabulary, *negative sampling* reduces computation by sampling just n negative instances along with the target word. Technically, negative sampling ignores most of the ‘0’s in the one-hot label word vector [27]. More concretely, the negative sampling samples the negative instances(words), along with the target word and minimizes the log-likelihood of the sampled negative instances, while also maximizing the log-likelihood of the target word. These negative samples are chosen using a unigram distribution. Normally, the probability of selecting a word as a negative sample is related to its frequency, with more frequent words being more likely to be selected as negative samples.

In our experiments, we are using this technique by considering the malware sample as the document and the extracted opcodes from it as the words. Thus we obtain the word embeddings for the 20 most frequent opcodes in the sample and then use the vectors of each of these opcodes to train the linear SVM model. We also experiment with different number of vector sizes for the embeddings to observe if a particular vector size gives us better results.

Figure 5: Using negative sampling in Word2Vec

3.3.4 Hidden Markov Model (HMM)

This section discusses the Hidden Markov Model (HMM). We have used an HMM model to obtain feature vectors for the extracted opcode sequence from the malware samples. These are then used to train a linear SVM to obtain average feature weights. Before we use the HMM model for our implementation let us discuss about what the technique is.

The Markov Model is a statistical model comprising of states and known probabilities of the state transitions [28]. The hidden Markov model (HMM) has states that are not directly observable in contradiction to the states of the Markov Model that are directly visible to the observer. HMM in Figure 6 is a machine learning technique and it acts as a state machine.

For examining a set of observation symbols, every state is associated with a probability distribution with the probabilities for this transition between the states being fixed. To represent a data, it is easy to train an HMM using the observation sequences [29]. This observation sequence can be matched against a trained HMM to determine the probability of seeing such a sequence. The observation sequence is

similar to a training sequence if the probability sequence is high. As mentioned in [30], the following table shows the notations used in the hidden Markov models:

Table 3: HMM notation

Notation	Explanation
T	Length of the observation sequence
N	Number of states in the model
M	Number of observation symbols
Q	Distinct states of the Markov process, $q_0, q_1, \dots, q_{N-1}$
V	Possible observations, assumed to be $0, 1, \dots, M-1$
A	State transition probabilities
B	Observation probability matrix
π	Initial state distribution
$\mathcal{O}$	Observation sequence, $\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{T-1}$

The observations are always denoted by $\{0, 1, \dots, M-1\}$, since this simplifies the notation with no loss of generality. That is, $\mathcal{O}_i$ for $i = 0, 1, \dots, T-1$. A generic hidden Markov model is illustrated in Figure 6, where the X_i represent the hidden state sequence and all other notations are as defined above. The Markov process—which is hidden behind the dashed line—is determined by the current state and the A matrix. We are only able to observe the $\mathcal{O}_i$, which is related to the (hidden) states of the Markov process by the matrix B . A hidden Markov model is defined by the matrices A, B and π . An HMM is denoted as $\lambda = (A, B, \pi)$.

Figure 6: Hidden Markov Model

The following three problems can be solved efficiently using the HMM algorithms:

Problem 1 : Given a model $\lambda = (A, B, \pi)$ and an observation sequence $\mathcal{O}$, we need to find $P(\mathcal{O}|\lambda)$. That is, an observation sequence that can be scored to see how well it fits a given model.

Problem 2 : Given a model $\lambda = (A, B, \pi)$ and an observation sequence $\mathcal{O}$, we can determine an optimal state sequence for the Markov model. That is, the most likely hidden state sequence can be uncovered.

Problem 3 : Given $\mathcal{O}, N, M$, we can find a model λ that maximizes probability of $\mathcal{O}$. This is the training of a model in order to best fit an observation sequence.

In this project, we have utilized the algorithms for Problems 1 and 3 above. An HMM is trained to represent a set of data, that is in the form of observation sequences. We trained this model using variety of compilers and malware generators. This is equivalent to using the algorithm for Problem 3. The resulting HMM is then used to score the malware samples against the model.

In this project, we are training HMM models on the top 20 most frequent opcodes and an ‘other’ category [28]. These opcode sequences are extracted from the malware samples. For every malware sample we obtain an HMM model from which we use the B matrix as the input features to train linear SVM models. The B matrix gives us information about the observation transition probabilities and we use the B matrix vector as input to the SVM model which generates feature weights for all the observations that is the opcodes. This approach might further give us better results and prove an evolutionary change in the malware family.

3.3.5 Experimental Approach

This section outlines the approach we follow to automatically determine points in time where significant evolutionary changes occur in malware families. First, we select a malware family from our dataset and tag each sample in the family according to the date on which it was compiled. Once this is done, we then extract the opcode sequences from each of the malware samples. Then we train a linear SVM based on the opcodes discussed above. We also use word embeddings (vectors) of varying vector sizes obtained from Word2Vec model and also the B matrix from HMM as input feature vectors to the linear SVM model.

As discussed above, when training, one class consists of all samples belonging to the specific family within a one-year time window, while the other class consists of the samples from the subsequent one-month time window. Such a model contrasts the family characteristics over a one month period to the characteristics of the previous one-year time interval. From each such model, we rank all features based on the linear SVM weights. Then we shift our time window one month ahead, and again train an SVM and rank the features as just described [4].

In this way, we obtain a series of snapshots of ranked features based on overlapping sliding windows, each based on a one year duration and each offset by one month. We use this as a basis for tracking changes in the significance of various features and we quantify those changes using the χ^2 statistics, as discussed above. To confirm the changes observed in the statistics obtained from the above approach, we also train HMMs on either side of the significant spike in the distribution. If the samples score very differently on each of the models, this confirms our analysis of evolutionary changes in the family.

In the next chapter, we first verify if the different approaches discussed in this chapter enable us to detect changes within a given timeline. We then compute and graph the χ^2 statistics for all families in our dataset to determine likely evolutionary change points.

CHAPTER 4

Results and Analysis

In this chapter, we discuss the results and analysis of the experiments performed by training various pipelines of machine learning techniques on the Malware data-set. This chapter has four sections, each discussing the results of experiments that have been conducted using different techniques.

4.1 Hidden Markov Model (HMM)

We have already seen that by using PE file features of the malware samples we can detect evolutionary changes in a malware family. These features are used to train linear SVM models over sliding window periods of time and a χ^2 distribution graph is then plotted using the feature weights obtained from them. The graph tells us about the changes in the malware family according to the feature weights obtained. The changes or spikes as observed in the graph were considered as the points of evolution in the malware family. But the research didn't mention about using any other types of methods to confirm the evolutionary points in the family. The confirmation of the evolution taking place in a particular malware family is done using Hidden Markov Models in this section.

This section particularly deals with training HMM models using the top 20 opcodes and an 'other' category extracted from the malware samples. We will be discussing the use of HMM model in confirming the evolutionary changes and also use this method in asserting the results obtained in the experiments performed later in this project.

Once distinct spikes have been obtained from the χ^2 similarity graph from the linear SVM models over different time periods, we train an HMM model on both sides of the most significant spikes using the extracted opcode sequences from the malware samples. Both the models are then used for scoring malware samples of the same

family. We then use these scores to plot a line graph denoting the two different scores of the same sample on the two models. This distinction in the scores is due to the change in the models. If the two models obtained had been the same, the samples would have given same scores but the difference in the scores of the same samples on both the models conclusively tells us, that the models are different. Hence, the malware samples should also be termed as different that is evolved. In this way, we can verify the evolutionary changes obtained in a family using various methods.

Figure 7: Hidden Markov model trained before and after the spike

The Figure 7 depicts two line graphs of the scores obtained from scoring the malware samples before and after the spike on the χ^2 distribution graph on two HMM models. First, we train an HMM model using samples from before the spike and then train another one using the samples from after the spike. These models are then used for scoring samples before and after the spike.

Figure 7 (a) shows the graph for the malware samples before the spike. The blue line indicates the scores of the HMM model trained before the spike and the green line indicates the scores of the model trained after the spike. All the scores are obtained on scoring both the models using samples from before the spike. In the similar way, Figure 7 (b) shows the scores of the samples from after the spike. The violet line shows the scores of the model which is trained on samples before the spike and the

orange line shows the scores of the model trained with samples after the spike. In all cases, the scores are log likelihood per opcode (LLPO) that is, the scores have been normalized using the length of the opcode sequence.

It can be observed from both the graphs that for every sample that has been scored, the scores are distinct from each other. This clearly implies that the two models are fairly different from each other because if the models had been the same, the scores would have been the same or with very less difference. This distinction in the scores leads us to believe that the malware family has evolved because without an evolution the models should be similar. Thus, it can be concluded that this technique give us an insight in confirming the evolutionary points obtained from the other machine learning techniques. This approach is also used for the further experiments. Whenever any experiment is said to find an evolutionary point in the family, the claim has been confirmed using this approach.

4.2 Opcodes — SVM

In this section, we discuss the experiments on the 15 malware families. Each family has numerous samples which are segregated according to their timestamp. From each of these malware samples, we then extract opcodes for our experiments. These opcodes are then used as inputs to the linear SVM model. The SVM model is trained for sliding windows of time periods and then, a χ^2 distribution graph is plotted using the feature weights obtained from each of the models trained.

In the Figure 8, we discuss experiments on the Zegost family for 506 malware samples. We perform our SVM feature analysis on the Zegost family based on the most frequent 20 opcodes and the ‘other’ category. We have also experimented with higher number of opcodes, but it showed same results and thus we choose to use 21 opcodes. Specifically, these experiments utilize Zegost samples from 2008 through 2011. As

above, we determine linear SVM weights for multiple overlapping time windows.

Figure 8 gives the χ^2 statistic timeline for the Zegost family for our SVM feature weight experiments. It can be observed from the graph that the weights of the opcodes keep fluctuating and as a result we can see that there are multiple spikes in the graph. On analyzing these spikes, that is training HMM models on either side of them to check evolution, we couldn't find any convincing evidence which could determine that these points were the points of evolution. Thus, we can say that the changes observed in the graph by using opcodes to train the linear SVM models does not give us valuable information to detect evolutionary changes in the Zegost family.

Figure 8: χ^2 similarity graph for Zegost family using opcodes

We also obtained the resultant χ^2 distribution graphs for the remaining 14 malware families as well. For Adload and Bho family in Figure 9, from the graph we can observe that the graph does not show any significant spikes except the one at the extreme end which is in the last time period. Thus, from the graph it should mean that there has been no evolution in the family except in the last time period. But on confirming this observation using the HMM models, we could find that there have been evolutions in the family but the graph obtained by the current approach that is,

by using opcodes to train the linear SVM models does not show it and thus give us false results. Thus we can say that this method is not beneficial to detect evolutionary points.

Using this experiment for Bifrose, Ceeinject, Hupigon and Rbot from Figure 9, we can clearly observe that there a lot of fluctuations in the graph, but these fluctuations do not impart any meaning in detecting evolutionary points in the families. They imply that there are a lot of changes in the opcode sequences of the family.

Figure 9: χ^2 statistics on using opcodes as input

There can be a lot of variations in the opcode sequences extracted from the malware samples which depend either on which developer has written a particular malware sample or maybe on the purpose of the sequence written. Thus, opcodes sequences even though for a particular malware family but might contain a lot of variations in the sequences in which they are used and thats why, when we use the raw opcode sequences in training linear SVM models, over different time periods, we obtain variations in the feature weights as the opcode sequences keep changing from sample to sample in the same family. Thus, we should be refining the input features that we are training the SVM model on. Training the SVM model on stabilized input features giving information about the opcodes from the entire sample might be more beneficial in obtaining results.

Thus, we tried using the byte n -gram approach with the opcodes and use the n -grams as an input to train the SVM model. A n -gram language model is a contiguous sequence of n items from a given sample. Here our given sample is the malware sample so we will be considering n -gram of opcodes, that is the sequences of opcodes in which they are occurring in the malware sample in a particular family. We experimented over different values of n for the n -gram approach where $n = 2, 3, 5, 10$.

(a) Zegost 2-gram

(b) Zegost 5-gram

Figure 10: χ^2 statistics of Zegost family on using n -gram

Figure 10 shows the χ^2 similarity graph for the various n -gram approaches on the Zegost family. From the results, there is not significant change in the average weights when we try different n -grams of opcodes. It does not give us any understanding of any evolutionary changes in the malware family. There are a lot of fluctuations in the graph but they do not impart any meaningful information regarding the evolution in the family. Thus, using n -grams of opcodes with SVM does not give us any distinct results about the evolution of a malware family, as we are dealing with the opcode sequences and not an input feature which tells us about the whole malware sample.

Figure 11: χ^2 statistics of different families on using n -gram

In Figure 11 we implemented the n -gram approach with the other families too, to check if this approach provides same observations for all the families. From the results obtained, we can say that this technique does not give us any insight about the evolution in a malware family. We need to use the opcode sequences obtained from the malware samples and obtain input features from it which would contain

information about the whole malware sample. In the next sections, we use Word2Vec model embedding vectors and the B matrix of an HMM as the input feature vectors to train the linear SVM models. These input features would contain the information regarding the whole sample rather than a single instruction sequence.

4.3 Word2Vec — SVM

In this section we discuss the experiments on 15 malware families by using Word2Vec models. In this approach, we train Word2Vec models on the extracted opcodes for each malware sample in the family and then use the Word2Vec embeddings of each of the 20 most frequent opcodes and the ‘other’ category as feature vectors to train a linear SVM model over each time window. We then compute the χ^2 similarity with the feature weights of all the models, and we plot these values on a χ^2 similarity graph. Spikes in this graph indicate points in time where the weights of the corresponding SVM’s differ significantly, which are points where it is likely that the malware has evolved to a significant degree.

(a) Zegost Vector Size 2

(b) Zegost Vector Size 3

Figure 12: χ^2 statistics of Zegost family on using Word2Vec-SVM

Figure 12 gives the χ^2 statistic timeline for the Zegost family based on our Word2Vec-SVM feature weight experiments. From Figure 12, it can be observed that feature weights in certain time windows diverge significantly from their average values.

Specifically, these time periods are November 2010 and May 2011, as can be seen from the graph.

We also use different vector sizes of the Word2Vec embeddings. But as we see in Figure 12 (a) and 12 (b), both the vector sizes show us that the significant spike in the distribution graph is in the same time period. Thus we can say that a change has taken place in that particular time period, in this case, November 2010 - May 2011, as compared to the other time periods that we are experimenting with.

On analyzing these results using the HMM verification technique, we can confirm that the time period obtained from the graph is the time where the malware family has evolved. The different vector sizes are not of much importance here, as we are obtaining similar results for both. Thus, we can say that the Word2Vec approach gives us the time periods in which malware evolution has taken place. In both the figures 12 (a) and 12 (b), we do observe some other small spikes in the graph but that is mainly due to noise in the feature vectors on which the linear SVM model is trained. These spikes do not impart any meaningful information to our observations. But this is in the case of the Zegost family. We should consider the same approach on other families as well.

Figure 13 shows the χ^2 statistics of the Word2Vec feature vectors and the linear SVM approach in the other malware families. Four families out of these results namely, Bho, Bifrose, Adload and Vobfus perform very well with this approach and give us significant spikes in the χ^2 distribution graph.

In Figure 13 (a) it can be observed that a significant spike is obtained from the χ^2 distribution graph. On applying the HMM verification approach, we can positively say that this spike is a point of evolution in the malware family. We train Word2Vec models on the extracted opcode sequences from the samples and then use the opcode embedding vectors to train the linear SVM model. In training the Word2Vec model,

(a) Bho vector size 2

(b) Bifrose vector size 2

(c) Adload vector size 2

(d) Ceeinject vector size 2

(e) Rbot vector size 2

(f) Hupigon vector size 2

(g) Vobfus vector size 3

Figure 13: χ^2 statistics of other families on using Word2Vec-SVM

we go through the most frequent 20 opcodes and an ‘other’ category in every malware sample. Thus, the embedded vectors obtained from the Word2Vec model to use as the input features are dependent on the entire malware sample and considers the complete malware sample. This is the reason why we obtain convincing results of malware evolution from this approach as compared to the earlier approach.

The other malware families Bho, Bifrose and Vobfus also show significant spikes in the χ^2 distribution graph. These results have been confirmed by the HMM verification method too. Thus, we can positively say that the Word2Vec method is useful in detecting evolution. But for some of the other families, the Word2Vec approach does not seem to be performing well. In Figure 13, we can see that for Ceeinject, Dorkbot, Hupigon and Rbot this technique seems to not give any conclusive results. The graph shows a lot of fluctuations in it and a result cannot be obtained from them. This might be because of the changes in opcode sequences in each family.

Every malware sample might have a variety of different opcode sequences and thus, on calculating the average feature weights from the linear SVM models on these samples, they show a lot of fluctuations in the graph because the sequences of opcodes in the sample keep changing as well. Every family has its own unique opcode sequence and thus, some approaches might work well with some families but not all of them.

4.4 HMM — SVM

In this section we discuss the experiments on 15 malware families by using HMM models. In this approach, after extracting the opcode sequences from each malware sample, we train an HMM model on them using 20 most frequent opcodes and the 'other' category that occurs in the sample. We then use the B matrix obtained from HMM, that is, the resulting observation probability matrix as a feature vector to train linear SVM models over each time window. We then compute the χ^2 similarity of the SVM weights of nearby (in time) models, and we plot these values on a χ^2 similarity graph. Spikes in this graph indicate points in time where the weights of the corresponding SVM's differ significantly, which are points where it is likely that the malware has evolved to a significant degree.

(a) Zegost without restarts

(b) Zegost with 10 restarts

Figure 14: χ^2 statistics of Zegost family on using HMM-SVM

Figure 14 gives us the χ^2 statistic timeline for the Zegost family (506 samples) for our HMM-SVM feature weight experiments. From this figure, we do not observe a significant change in the average weights which would indicate that evolutionary change has taken place. There are fluctuations in the graph telling us that there are changes in the instructions of the malware samples and thus a feature vector from an HMM model cannot be used to display evolutionary changes in Zegost family.

But we need to make sure if the HMM model is being converged in order to get the most accurate vectors from the model and thus, we have also implemented random restarts in obtaining a converged HMM model. But the results obtained from the various number of random restarts were the same as that of the results obtained from the model with no random restarts implying that the HMM model is getting converged in this case. We also experimented using two different values of the number of states for the HMM model, $N = 2, 3$. But the results obtained from these experiments were also not helpful in observing a distinct spike in the χ^2 distribution graph for Zegost family. But the results on just one family might not be enough to conclude upon an approach. Thus we implement this approach on the other families as well.

Figure 15 shows the χ^2 statistics of the HMM B matrix as feature vectors and the

(a) Adload

(b) Zbot

(c) Bho

(d) Bifrose

(e) Ceeinject

(f) Delfinject

(g) Dorkbot

(h) Hupigon

Figure 15: χ^2 statistics of other families on using HMM-SVM

linear SVM approach in the other families as well. Overall, the HMM-SVM approach seems to be working better than the Word2Vec approach as we could detect malware evolution in more families through this approach. From Figure 15 we can make the following observations about each figure:

Adload — An evolutionary event takes place in the time window Sep 2010–May 2011.

Zbot — No significant spike is observed in the graph.

Bho — A malware evolution takes place in the time window Nov 2008–May 2011

Bifrose — A malware evolution takes place in the time window March 2010–June 2011. The other spike in the period June 2009–Jan 2011 is a part of noise in the data. This was confirmed by training HMMs on both sides.

Ceeinject — A malware evolution takes place in the time window June 2009–April 2011.

Delfinject — No significant spike is observed in the graph.

Dorkbot — No significant spike is observed in the graph.

Hupigon — A significant spike in the time period June 2010–Jan2011 is observed in the graph.

From Figure 15 we can conclude that we can observe significant spikes in almost all the families using the HMM-SVM approach. For the families Adload, Bho, Bifrose, Ceeinject and Hupigon we can observe significant spikes in the χ^2 distribution graph. When confirmed using the HMM verification technique, these spikes are detected to be the points of evolution in the malware families. Other than the significant spikes there are also a few number of smaller spikes in the graph, they are mainly due to noise that is there due to the variations in the B matrix obtained from the trained HMM models on the opcode sequences. Thus, we can assert that the HMM-SVM approach is useful in observing evolutionary changes in a malware family.

While comparing the families in which we could detect evolutionary changes using Word2Vec approach and the HMM approach, we can clearly observe that the

evolutionary points obtained by both the approaches are the same and they also match with the evolutionary points obtained in the research earlier.

While some families did not perform well with the HMM approach as well, but we can conclusively say that it is because of the variations in the B matrix obtained from the HMM models trained on every malware sample. But overall, this technique is seen to have performed better than all the approaches that we have seen in this project.

CHAPTER 5

Conclusion And Future Work

In the previous research, a significant spike in the chi square similarity graph obtained by using PE file features with a linear SVM model proved that there was a change in the malware family which was termed as malware evolution. In this project, we have confirmed the change in the malware family as an evolution by training HMM on either side of the significant spike in the graph and then scored samples on both the models. The scores obtained from these models were quite distant from each other, hence, proving the point that the two models are different, and a malware evolution has taken place.

Since PE file features of the malware samples were used to determine malware evolution in the earlier work, we used opcodes as the input feature in all our experiments. We extracted the opcode sequences from each of the malware samples from each of the families and used them as our input feature. All through our experiments we have used the top 20 most frequent opcodes and an ‘other’ category for our analysis.

In the first set of experiments, we used the opcodes and trained a linear SVM model over different windows of time periods in a single malware family. The resultant average feature weights obtained from the model were then used to plot a χ^2 distribution graph. We observed the graphs of 15 different malware families but couldn’t find any significant change in the graph which could imply an evolution in the family. The graphs had a lot of fluctuations in them which meant that the weights associated with the opcodes were changing a lot and thus, a proper result cannot not be obtained from them.

In the next set of experiments, we used a byte n -gram approach with the opcodes as the input to the linear SVM model. The χ^2 distribution graph for this too had a lot of fluctuations in them and thus, no observation about an evolution could be made

from them other than the fact that this approach cannot be used to detect malware evolution in a family and also that using the raw opcode sequences with the linear SVM model does not give us any meaningful explanation about the malware family used.

We then trained a Word2Vec model using the extracted opcodes from the malware files. The word embeddings of the opcodes were then used to train a linear SVM model. We used different number of vector sizes of the word embeddings to feed to the SVM model to see which vector size worked better but there wasn't a particular number that worked for all so it meant that it was a little random. The same type of experimentation was done by training an HMM using the opcode sequences and then using the B matrix obtained from the HMM as the input feature to the linear SVM model. For training the HMM and also to ensure the convergence of the model, we tried different number of random restarts.

On plotting the χ^2 distribution graphs using both the approaches mentioned above, for some of the families we observed significant spikes in the distribution graph for both the methods. Although some noise could be seen in these graphs along with the distinct spike in it, but overall, we could see a change in the malware family. On training an HMM on either side of the spikes and scoring some samples on it, we could successfully state that a malware evolution has taken place in the families. Thus, using the Word2Vec model and the HMM model on the opcodes before feeding it into the linear SVM model proved to be a successful approach in detecting evolution. While this was true for some families, there were also some families which didn't show any significant spikes in their graphs and were filled with a lot of fluctuations. These graphs implied that these families had a lot of changes in their code and these approaches cannot detect the evolution in them if present.

5.1 Future Work

Our research mainly focused on using opcodes extracted from the malware samples. Various other features can also be used to detect changes in the malware family. Combinations of features can also be explored as input features. They might yield additional observations about malware evolution over different windows of time periods.

Malware obfuscation techniques like dead code insertion affect the static features of a malware file but it would not have much effect on evolution tracking that is based on dynamic features. Thus, using dynamic features might help in gathering more insights about malware evolution in a family.

Another area to consider would be using deep learning techniques to analyze malware samples in a family to detect changes in them. Even though such techniques are difficult to implement and are complex, but it would be very interesting to observe the results obtained from using deep learning methods.

LIST OF REFERENCES

- [1] M. Barat, D.-B. Prelipcean, and D. Gavriluț, “A study on common malware families evolution in 2012,” *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 4, pp. 171–178, 2013.
- [2] J. Bai, S. Mu, and G. Zou, “The application of machine learning to study malware evolution,” *Applied Mechanics and Materials*, vol. 530-531, no. Advances in Measurements and Information Technologies, pp. 875–878, 2014. [Online]. Available: <http://search.proquest.com/docview/1762084930/>
- [3] A. Gupta, P. Kuppili, A. Akella, and P. Barford, “An empirical study of malware evolution,” in *2009 First International Communication Systems and Networks and Workshops*. IEEE, 2009, pp. 1–10.
- [4] M. Wadkar, F. D. Troia, and M. Stamp, “Measuring malware evolution using support vector machines,” 2019.
- [5] F. Mercaldo, A. Di Sorbo, C. A. Visaggio, A. Cimitile, and F. Martinelli, “An exploratory study on the evolution of android malware quality,” *Journal of Software: Evolution and Process*, vol. 30, no. 11, pp. n/a–n/a, 2018.
- [6] J. Ouellette, A. Pfeffer, and A. Lakhotia, “Countering malware evolution using cloud-based learning,” *Proceedings of the 8th international conference on malicious and unwanted software*, p. 85–94, 2013.
- [7] Z. Chen, M. Roussopoulos, Z. Liang, Y. Zhang, Z. Chen, and A. Delis, “Malware characteristics and threats on the internet ecosystem,” *The Journal of Systems & Software*, vol. 85, no. 7, pp. 1650–1672, 2012.
- [8] A. Damodaran, F. Troia, C. Visaggio, T. Austin, and M. Stamp, “A comparison of static, dynamic, and hybrid analysis for malware detection,” *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017.
- [9] A. Nappa, M. Rafique, and J. Caballero, “The malicia dataset: identification and analysis of drive-by download operations,” *International Journal of Information Security*, vol. 14, no. 1, pp. 15–33, 2015.
- [10] S. Kim, “PE header analysis for malware detection,” Master’s thesis, San Jose State University, Department of Computer Science, 2018.
- [11] “Win32 bifrose detected with windows defender antivirus,” <https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/bifrose>, 2012.

- [12] “Virtool: Win32 ceeinject detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject>, 2007.
- [13] “Virtool: Win32 delfinject detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/DelfInject&ThreatID=-2147369465>, 2007.
- [14] “Worm: Win32 dorkbot detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm%3AWin32/Dorkbot>, 2011.
- [15] “Adware: Win32 hotbar detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware%3AWin32%2FHotbar>, 2006.
- [16] “Backdoor: Win32 hupigon detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor%3AWin32%2FHupigon>, 2006.
- [17] “Win32 obfuscator detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FObfuscator>, 2011.
- [18] “Win32 rbot detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FRbot>, 2005.
- [19] “Virtool: Win32 vbinject detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/VBInject&ThreatID=-2147367171>, 2010.
- [20] “Win32 vobfus detected with windows defender antivirus,” <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fvobfus>, 2010.
- [21] “Win32 winwebsec detected with windows defender antivirus,” <https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fWinwebsec>, 2010.
- [22] “Win32 zbot detected with windows defender antivirus,” <http://www.symantec.com/securityresponse/writeup.jsp?docid=2010-011016-3514-99>, 2011.
- [23] “Win32 zegost detected with windows defender antivirus,” <https://www.symantec.com/security-center/writeup/2011-060215-2826-99>, 2011.

- [24] S. Rezaei, F. Rezaei, A. Afraz, and M. R. Shamani, “Malware detection using opcodes statistical features,” in *2016 8th International Symposium on Telecommunications (IST)*. IEEE, 2016, pp. 151–155.
- [25] M. Stamp, *Introduction to machine learning with applications in information security*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2018.
- [26] I. Popov, “Malware detection using machine learning based on word2vec embeddings of machine code instructions,” in *2017 Siberian Symposium on Data Science and Engineering (SSDSE)*. IEEE, 2017, pp. 1–4.
- [27] J. Gilyadev, “Word2vec explained,” <https://israelg99.github.io/2017-03-23-Word2Vec-Explained/>, 2017.
- [28] C. Annachhatre, T. Austin, and M. Stamp, “Hidden markov models for malware classification,” *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 59–73, 2015.
- [29] S. Kazi and M. Stamp, “Hidden markov models for software piracy detection,” *Information Security Journal: A Global Perspective*, vol. 22, no. 3, pp. 140–149, 2013. [Online]. Available: <http://www.tandfonline.com/doi/abs/10.1080/19393555.2013.787474>
- [30] M. Stamp, “A revealing introduction to hidden Markov models,” <https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf>, 2018.

APPENDIX

Appendix

Figure A.16: HMM verification technique on Figure 12 showing evolution on Zegost using Opcode-Word2Vec-SVM

Figure A.17: HMM verification technique on Figure 13(a) showing evolution on Bho using Opcode-Word2Vec-SVM

(a) Before spike

(b) After spike

Figure A.18: HMM verification technique on Figure 13(h) showing evolution on Vobfus using Opcode-Word2Vec-SVM

(a) Before spike

(b) After spike

Figure A.19: HMM verification technique on Figure 13(d) showing that the family has no evolution on Ceeinject using Opcode-Word2Vec-SVM

(a) Before spike

(b) After spike

Figure A.20: HMM verification technique on Figure 15(c) showing evolution on Bho using Opcode-HMM2Vec-SVM

(a) Before spike

(b) After spike

Figure A.21: HMM verification technique on Figure 15(d) showing evolution on Bifrose using Opcode-HMM2Vec-SVM

(a) Before spike

(b) After spike

Figure A.22: HMM verification technique on Figure 15(e) showing evolution on Ceeinject using Opcode-HMM2Vec-SVM

(a) Before spike

(b) After spike

Figure A.23: HMM verification technique on Figure 15(h) showing evolution on Hupigon using Opcode-HMM2Vec-SVM

(a) Before 2010

(b) After 2010

Figure A.24: HMM verification technique on Figure 15(f) showing that the family has no evolution on DelfInject using Opcode-HMM2Vec-SVM

(a) Before spike

(b) After spike

Figure A.25: HMM verification technique for second spike on Figure 15(g) showing that the family has no evolution on Dorkbot using Opcode-HMM2Vec-SVM