Spring 2021

Machine Learning to Detect Malware Evolution

Lolitha Sresta Tupadha
San Jose State University

Machine Learning to Detect 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

Lolitha Sresta Tupadha

May 2021

The Designated Project Committee Approves the Project Titled

Machine Learning to Detect Malware Evolution

by

Lolitha Sresta Tupadha

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2021

Dr. Mark Stamp	Department of Computer Science
Dr. Mike Wu	Department of Computer Science
Fabio Di Troia	Department of Computer Science

ABSTRACT

Machine Learning to Detect Malware Evolution

by Lolitha Sresta Tupadha

Malware evolves over time and anti-virus must adapt to such evolution. Hence, it is critical to detect those points in time where malware has evolved so that appropriate countermeasures can be undertaken. In this research, we perform a variety of experiments to determine when malware evolution is likely to have occurred. All of the evolution detection techniques that we consider are based on machine learning and can be fully automated—in particular, no reverse engineering or other labor-intensive manual analysis is required. Specifically, we consider analysis based on hidden Markov models and various word embedding techniques, among other machine learning based approaches.

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. Mike Wu and Mr. Fabio Di Troia, for their assistance and help with the project. Finally, I want to thank my family for their continuous efforts to motivate, support and encourage me.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is software that is designed to cause harm by, for example, gaining access to private computer systems, stealing sensitive information, infecting files on a system, or spreading its infection. This is a software whose intent is malicious, or whose effect is malicious [3]. Since the creation of the ARPANET in 1969, we have witnessed exponential growth in the number of users of the internet. The widespread use of computer systems along with continuous internet connectivity of the "always on" paradigm makes modern computer systems prime targets for malware attacks.

Malware comes in many forms, including viruses, worms, backdoors, trojans, adware, spyware etc. Malware is a continuously evolving threat to information security. In the field of malware detection, a signature typically consists of a string of bits that is present in a malware executable. Today, signature-based detection is the most popular method of malware detection used by anti-virus (AV) software [3]. But malware has become increasingly difficult to detect with standard approaches, such as signature scanning [4]. Virus writers have developed advanced metamorphic generators and obfuscation techniques that enable malware to easily evade signature detection. In [5], the authors prove that carefully constructed metamorphic malware can successfully evade signature detection.

Koobface is a recent example of an advanced malware. This malware was designed to target the users of social media. This infection is spread in the form of spam that is sent through social networking websites. Once a system is infected, it gathers user's sensitive information like their credentials. On an infected system, it blocks the user from accessing any anti-virus or security websites [6]

Malware writers modify their code to deal with advances in detection, as well as to add new features to the existing malware [7]. Hence malware can be perceived as

evolving overtime. To date, most research into malware evolution has relied on reverse engineering techniques [8], which can be extremely labor intensive. Here, our goal is to detect malware evolution automatically, using machine learning techniques. We want to find points in time where it is likely that significant evolution has occurred within a given malware family. It is important to detect such evolution, as these points are precisely where modifications to existing detection strategies are urgently needed.

We consider several machine learning techniques to identify potential malware evolution. Our experiments are conducted using a significant number of malware families, each of which contains a large number of samples collected over an extended period of time. We extract the opcode sequences from each malware sample and these sequences are used as features for our experiments. We then group the available samples based on time periods and we train machine learning models on each of these time windows. We compare the models to determine likely evolutionary points—substantial differences in models across a time boundary indicate significant changes in the samples on either side of the boundary. Specifically, we consider experiments using hidden Markov models (HMM) as well as word embedding techniques, including Word2Vec and HMM2Vec. We also consider logistic regression.

The remainder of this paper is organized as follows. Chapter 2 provides background on malware and malware evolution. In Chapter 3, we provide an overview of relevant previous work in the area of malware evolution. Chapter 4 contains an introduction to the dataset used in this project and we introduce the machine learning techniques that we have employed. Chapter 5 contains our the experimental results, while Chapter 6 gives out conclusion and a discussion of future work.

CHAPTER 2

Malware and its Evolution

As discussed in Chapter 1, malware is a software whose intent is malicious, or whose effect is malicious [3]. The motivation behind writing a malware could range from throwing a harmless prank to performing some serious technical and financial loss to an organization. The very early malware, known as Morris worm was written by a graduate student as a prank [9]. According to [10] there are more than one billion malware programs out there with 560,000 new malware pieces discovered every day. In 2020, the number of detected malware variants rose by 62% with trojans accounting for 58% of the entire malware. Figure 1 shows a graph showcasing the growth in the malware attacks. The x-axis shows the year and y-axis represents the number of attacks causing financial losses of more than $1 Million dollars for that year. We can see that malware attacks are increasing despite many advanced malware detection techniques.

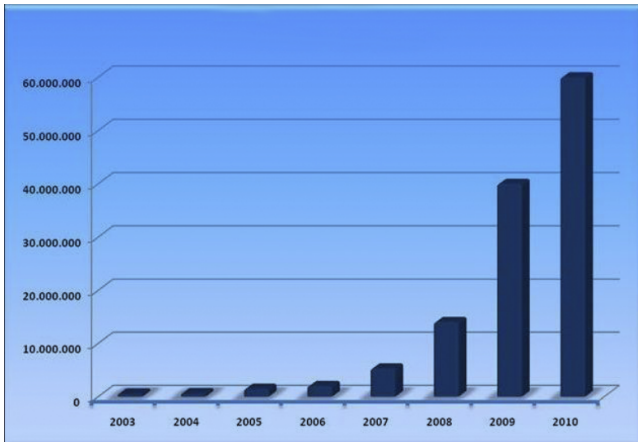


Figure 1: Malware Growth [1]

This chapter gives a brief introduction to different kinds of malware. Malware can be subdivided into many sub–categories that include worms, virus, trojan, trapdoor etc. In this section we will look in to the details of different kinds of malware.

2.1 Worms

A computer worm is a kind of virus but its only difference from virus is that it spreads by itself without need of external tools [11]. That means worms replicate themselves using the network rapidly and can cause the entire network to collapse. Some of the infamous worms in history are Code Red, Blaster, Stuxnet and Santy.

2.1.1 Internet Worms

Internet worms target some widely used or famous websites with security vulnerabilities. They initially infect the site and then replicate themselves on the computers being used by spreading through the internet or the Local Area Network (LAN)

2.1.2 Email Worms

Email worms are kind of worms that are sent via compromised email attachments. These worms are sent as either attachments or some external link through email. Once the recipient of email, either downloads the attachment or clicks on the link, the worm infects the system and the same infected file will automatically be sent to addresses from their contact list [9].

2.1.3 Instant–Messaging Worms

These are the same as Email worms but will be spread through popular messaging apps like messenger, whatsapp etc. It tricks the user with short messages like "You have got to see this!" or "So funny" and tempts him into clicking any external link. Once the link is clicked, it infects the system and will be forwarded to all the contacts of the user.

2.1.4 File–sharing Worms

Although file–sharing is being restricted by organizations and many users, it is still prevailing. These kinds of worms transmit between the system as some shared files and when the user installs it or downloads it the worm starts to infect the system.

2.2 Virus

This is the most popular form of malware that is used interchangeably with malware most of the time. A computer virus is similar to a worm but it needs outside assistance to transmit from one system to another. They can live anywhere in the system ranging from the boot sector, applications, macros, library routines, compilers and even in the virus detection softwares. They can enter the system through a variety of sources like the internet, file–transfer and external storage like floppy disks etc. They use techniques like encryption, polymorphism and metamorphism in order to evade detection.

2.2.1 Encrypted Virus

Most of the antivirus softwares tries to detect the malware by searching for an already known signature. A simple way to evade signature detection is to encrypt the signature with different keys only known to this particular malware. This is not a foolproof technique to evade signature detection as it is still possible to search for the encrypted key.

2.2.2 Polymorphic Virus

Polymorphic viruses are complicated viruses that can change their structures and create a new version of themselves while retaining basic jobs they have to do. Since the structure of the file is changed it can easily evade signature detection. They rely on mutation engines to alter their decryption routines every time they infect the machine. These mutation engines use billions of decryption routines making it impossible for any conventional systems to flag them as malicious [12].

2.2.3 Metamorphic Virus

From a bird view both metamorphic and polymorphic viruses change their structures to create newer versions of themselves. Polymorphic virus achieve this by

encrypting itself with variable encryption keys so that each copy or each signature appears different whereas metamorphic virus rewrites its code to make it appear different each time [9].

2.3 Trojan

A trojan horse or trojan is a malicious software that appears to be an innocent file but has some sort of malicious payload. The difference between trojan and virus is that the virus is self–replicating whereas trojan is not. Some examples of trojans are Crack2000, BetBus [12]

2.4 Trapdoor

Trapdoor/Backdoor allows unauthorized access to the system [9]. This sits in the system and compromises security checks of the system allowing malicious users to enter into the system.

2.5 History of Malware

To this point, we have discussed different kinds of malware. In this section we consider famous malware attacks. These attacks give us some awareness about the intensity of the malware and the necessity to analyze and detect them effectively.

2.5.1 Creeper

Creeper was a worm that was discovered in the early 1970's. This worm transmitted through modems and showed the message to the user that read "I'm a Creeper. Catch me if you can". Following this attack, another malware called rabbit was discovered in 1974. This malware replicated itself so fast that the system crashed. The author in [9] mentions that several malware attacks that happened in the 1970's were unintentional and occurred as part of some software bugs. These attacks were not so harmful and thereby did not awake the people regarding the necessity for cybersecurity.

2.5.2 Brain Virus

Brain virus is declared as the first official virus. It occurred in 1986. Even though it was not very harmful, it became a prototype for later viruses. Brain virus used to mutate itself so that it was very difficult to remove it from the system. It was placed in the boot sector of the system and shielded the access to the disk in order to evade its detection. Computer security professionals consider this virus as a warning showcasing the potential of malware. But at that time it was ignored since it did not cause any potential damage to the system.

2.5.3 Morris Worm

The first major malware attack that shook the computer society was when Morris worm hit the internet in 1988. It is surprising to know that Morris worm was created by a single graduate student who claimed that this worm is a result of testing gone bad. The Morris worm spread over the internet attacking the systems. This is a very smart and sophisticated piece of software that remains undetected after infecting the system.

The attack of Morris worm was declared as the internet crisis. Computer Emergency Response Team (CERT) was created after this attack [9]. This evidently brought down the entire internet and made the software community realize the potential and damage possible through the malware attacks and emphasized on necessity for malware analysis.

2.5.4 Code Red

The next major malware attack was in 2001 when Code Red infected more than 300,000 machines in just 14 hours. This worm targeted machines worldwide whereas 43% of infected machines were located in the United States, 11% in Korea and rest in China and Taiwan. At the peak, Code Red infected 2000 machines every minute.

Code Red gained access to the system by exploiting buffer overflow in Microsoft IIS server. Code Red performed Distributed Denial of Service (DDoS) attack on www.whitehouse.gov. after gaining access to the system. The speed with which it attacked many systems was very new to the internet community [9].

2.5.5 SQL Slammer

The internet community in 2003 witnessed the fastest malware attack when SQL Slammer infected more than 75,000 machines in less than 10 minutes. SQL slammer doubles its spread every 8 seconds during its peak [9]. Once a machine is affected, it searches for new susceptible hosts by randomly generating IP addresses. SQL Slammer was just 376 bytes of code which resided in Microsoft SQL software and tried to connect to every server over UDP port 1434. Microsoft released a patch to this bug which acted as a cure to this worm.

2.5.6 WannaCry

This is an advanced malware that attacked over 200,000 computers around 150 countries in May 2017. This virus targeted organizations using the Windows operating system. It encrypts all the data in the system and demands the user to pay ransom money to gain access back to the system. Once the payment is done, it reveals the key to decrypt the data [13].

Looking at a few famous malware attacks in the past we can analyze malware evolution. There is definitely a shift in malware trends and also motivation of the attackers. Most attacks in the recent times are targeting a specific user or an organization by reverse engineering the software. Also, malware is available as a service, meaning the expensive tools required to develop malware are available for a cheaper price.

Since signature detection is the popular method of malware detection, attackers

invest plenty of time in coming up with solutions to evade signature detection. One of the first moves made by hackers in order to evade signature detection is to encrypt the file. If the malware uses a different or changing key every time to encrypt itself, it will never have a single signature. But the information security community found a fix for this problem. The encrypted malware is supposed to be decrypted and it should contain a few lines of code in order to perform decryption. In most cases decryption code is just a few lines making it more difficult to obtain a signature. But signature scanning can still be applied to detect this kind of malware.

As a next step malware authors came up with polymorphic and metamorphic viruses in order to evade signature detection. Polymorphic virus can be detected using emulation whereas we still do not have a concrete solution to detect metamorphic virus. Metamorphic virus changes the entire code structure of itself before infecting the system making it impossible to be detected by signature detection.

We have seen malware has gone through a lot of evolution ever since its first discovery in the 1970's as a harmless worm to flash worm which can take down entire internet in less than a minute. Consequently we need to come up with new techniques that can detect any advanced malware, and machine learning is a promising line of research [4].

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

techniques required for detection will require evolution. Hence to protect and prevent unnecessary malware attacks, focus should be equally given to malware evolution.

CHAPTER 3

Related Work

We can find a lot of research work in the area of machine learning for malware detection but comparatively very few articles can be found in the area of malware evolution analysis. Malware evolutionary analysis based on code injection is considered in [14]. This works deals with shellcode extracted from malware samples. The researchers used clustering techniques in order to analyze shellcode and determine relationships between the samples. This work has been successful in determining the similarities between samples showing that a significant amount of code has been shared between samples. A drawback with this paper is, the authors only considered analysis with shellcode. Not much of feature engineering has been employed. Though this work provided results on samples that are similar, it did not provide considerable results on analyzing the evolution of malware.

Malware evolution research has been performed by the researchers in [15]. The good thing about this experiment is that it considers a huge dataset that spans over two decades. The authors use techniques based on graph pruning. This paper claims that specific properties of various families are inherited from other families over a period of time. However, it is not clear if those specific properties are inherited from other families or developed independently. Because of this problem, this work requires a lot of manual investigation. But in our research we are going to use machine learning techniques that minimize the manual intervention.

The research present in [16] is focused on detecting malware variants, which can be perceived as the malware evolution problem. The authors here used semi-supervised learning techniques on the malware samples that are proven to evade machine learning based detection. In contrast, in this paper we use unsupervised learning techniques and our work shows significant evolutionary points in time.

The authors of [17] extract variety of features from the Android malware samples and then determine the trends using standard software quality metrics. These results are then compared to the trends generated by the Android goodware. This work shows that the trends in the Android malware and goodware are similar and changes in malware has followed same path as goodware. This work has not provided many insights regarding the evolution problem we are interested in.

Our work in this paper is a continuation of the work presented in [18], where static PE file features of malware samples are used as the basis for malware evolution detection. They employed support vector machine (SVM) technique to train the samples of a particular family over a period of time. Linear SVM was considered here. Then the weights of the resulting models are compared using a χ^2 technique. Huge difference in model weights is given by a spike in the graph which refers to the evolutionary point in time.

The work presented in [19] is focussed on malware taxonomy. This work gives reasonable insights into malware evolution in terms of genealogical trajectories. This research is based on features extracted from malware encyclopedia entries (as developed by antivirus software vendors, such as TrendMicro). The authors use SVMs and language processing techniques on the extracted features to generate the desired results.

Malware researchers have always considered wide variety of features for malware analysis, that can be broadly classified into two types of features, static and dynamic. Static features are those that can be collected without executing the code where as dynamic features requires emulation. In general, static features are easy to collect, while dynamic features are very robust as given by the authors of [20]

The authors of [21] use multiple static features to perform malware classification among the families. The static features that are considered here are n-grams, entropy

and image representations. Apart from these, hex-dump based features are also used along with the features extracted from disassembled files, such as opcodes, API calls and sectional information from portable executable (PE) files. This works presents us with good insights on wide variety of static features that can be considered.

There is also similar work given in [22] which is derivative of work presented in [18]. In this paper the authors use opcode sequences from malware samples to analyze the malware evolution problem. Similar to the work given in [18], data is divided into time–windows and support vector machine (SVM) technique is used to observe evolutionary points in the malware samples. They later use HMMs to confirm the evolution points in the malware. Our work present in this paper is an extension to the above work. We perform experiments with statistical techniques like HMM, word embedding techniques like Word2Vec, HMM2Vec to analyze malware evolution problems. We determine that we can observe significant evolution in the malware family using our experiments.

CHAPTER 4

Dataset and Classification Techniques

In this chapter, we are going to discuss about the malware families and the dataset used in the research. Also we will also be discussing the feature extraction process and the machine learning techniques that were used to conduct our experiments. All these put together contributes to the basis of our evolutionary analysis.

4.1 Dataset

We acquired a dataset which contains Windows portable executable files belonging to 15 different malware families. We obtained two families (winwebsec and zbot) from the Malicia dataset [23] while the rest of the families are extracted from a larger dataset using VirusShare [24]. Each malware family contains a good number of samples spread over a time period. A malware family is supposed to have very similar characteristics probably because of a shared code base. Number of samples present in each family is given in Table 1.

The main motivation behind selecting the given set of malware families is that we have a large number of samples in each of these families spread over an extended period of time. This would help us in better understanding the evolution of malware over a considerable period of time. We have organized the malware samples in each family based on their creation date. Few samples with unclear creation date or altered compilation date are discarded in the initial feature analysis phase. We have considered samples that have a creation date in the time window mentioned in Table 1.

4.2 Malware Families

In this section, we discuss details of the malware families used in our experiments. We made sure that the malware families we use encompass wide variety of types, including virus, trojan, backdoor, worms etc. We also have malware families that uses encryption and obfuscation techniques to evade signature detection.

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
Total	11037	

Bifrose is a backdoor trojan. Trojan means it appears as an innocent looking file and tricks the user into installing it. And since it is also a backdoor, once it is installed, it gives malicious users unauthorized access into the system. A hacker can then enter the system and perform malicious operations [25].

CeeInject performs any malicious operations on the user system while remaining undetected. CeeInject uses obfuscation techniques to evade signature detection [26].

DelfInject is a worm that resides on some popular websites and is downloaded into the user's machine on visiting the website. This malware is downloaded as some executable file and is executed whenever the system is restarted [27].

Dorkbot is a worm that notices the user's activities and steals the credentials of

the user. It performs denial of service (DoS) attack and is spread via messaging applications [28].

Hotbar is an adware virus that resides on websites and is downloaded onto the user system when a user visits this malicious site. This is more annoying than harmful. It displays advertisements whenever the user browses the internet [29].

Hupigon is also a backdoor trojan similar to bifrose. That means it enters the system as a naive file and then compromises the security of the system [30].

Obfuscator evades signature detection using obfuscation techniques. Once it stays undetected, it can perform any activity it wants to [31].

Rbot is also a backdoor trojan that allows attackers into the system through the IRC channel. This is one of very advanced malware and is used mostly to launch denial of service (DoS) attacks [32].

VbInject uses encryption techniques to evade the signature detection. Its primary purpose is to disguise other malware that can be hidden inside of it [22]. Its payload can vary from harmless to very intense [33].

Vobfus is a trapdoor that lets other malware into the system. It exploits the vulnerabilities of the Windows operating system's autorun feature to spread in the network. This malware makes some serious changes to the system configuration that cannot be easily recovered [34].

Winwebsec is a trojan that tricks users into paying money by portraying itself as an anti–virus software. For the user, it gives deceptive messages saying that the device has been infected and motivates him to pay to remove the non–existing software in the system [35].

Zbot is also a trojan that was first discovered in 2010. It steals valuable information from the affected system and uses it for its own benefit. It can target information like system data, online sensitive data, and banking information. It can be easily modified to acquire other kinds of data. The Trojan is generally spread through the spam. Zbot was originally discovered in January 2010 [36].

Zegost is another backdoor trojan that gives access to malicious users into the compromised system [37].

4.3 Feature Extraction

We obtained the PE files of the families discussed above. Opcodes are known to be machine level language instructions which specify a particular operation that has to be performed [38]. For our experiments, we have extracted mnemonic opcodes from these PE files. Since opcodes encapsulate the overall structure of the program we can use opcode sequences in our experiments. We have also segregated the samples from each family according to its creation date. These samples will be input to our machine learning techniques.

4.4 Classification Techniques

In this section we will be discussing the techniques that we have used in this project. We used different kinds of machine learning techniques varying from statistical models, word embedding techniques to neural networks. This section presents a brief introduction to all the techniques we used in this research.

4.4.1 Hidden Markov Model

This section here presents the hidden Markov model (HMM). We conducted several experiments using HMM technique. Before discussing the details of our experiments, let us take a deep dive into the HMM technique.

HMMs are based on discrete probability. HMM includes a Markov process which

is a statistical model comprising states and known probabilities of state transitions [2]. The states and state transitions of HMM are not directly observable by the user where as the states of the Markov model are directly visible to the user. HMMs are built based on observation symbols where every unique observation symbol in the observation sequence is treated as a state. HMM as a state machine is shown in Figure 2.

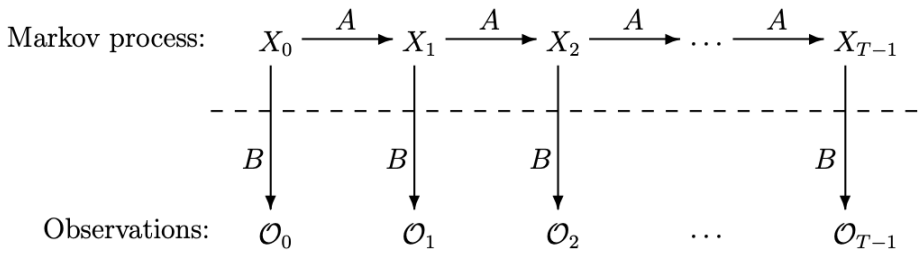


Figure 2: Hidden Markov Model [2]

Figure 2 gives generic view of HMM. The observations are always denoted by $\{0, 1, \ldots, M-1\}$. Here, the states X_i are determined by row stochastic matrix A of dimensions $N \times N$. The states determined by X are original states but are not directly observable. On the contrary observations $\mathcal{O}$ can be observed. Hidden states and observations are related. The probabilities with which they are related is given by row stochastic matrix B. Here N is the number of hidden states, M is distinct observation symbols and T is length of observation sequence. Initial state distribution is given by row stochastic matrix π. Now, we can define the HMM model as $\lambda = (A, B, \pi)$. Table 2 summarizes the HMM notations

There are three fundamental problems that can be solved using HMMs. Here, we give introduction to three problems and we determine solutions to solve these problems.

Problem 1: Given a model $\lambda = (A, B, \pi)$ and an observation sequence $\mathcal{O}$, determine

Table 2: 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, $q_0, q_1, \ldots, q_{N-1}$
V	Possible observations, assumed to be $0, 1, \ldots, M-1$
A	State transition probabilities
B	Observation probability matrix
π	Initial state distribution
$\mathcal{O}$	Observation sequence, $\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1}$

$P(\mathcal{O} \mid \lambda)$. That is, we want to compute a score for the observed sequence $\mathcal{O}$ with respect to the given model λ

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

Problem 3: Given an observation sequence $\mathcal{O}$ and the parameter N, determine a model $\lambda = (A, B, \pi)$ that maximizes probability of $\mathcal{O}$. This is training a model to best fit the observed data.

In our project, we have extensively used solutions for problem 1 and problem 3. We train an HMM to best fit the given observation which in our case is a set of opcodes. This gives us a model as determined in problem 3. The resulting HMM models are used to score malware samples of the same family against the model. This can be achieved by using solution for problem 1. More details about the implementation will be given in chapter 5. But the three problems of the HMM are solved by the following three algorithms.

- Forward algorithm

- Backward algorithm
- Baum–Welch re-estimation algorithm

The forward algorithm is used for calculating the probability of an observation sequence, given a model. This is achieved by calculating $\alpha_t(i)$. $\alpha_t(i)$ is the probability of being in state q_i at time t given an observation sequence $\mathcal{O}$. For $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, define

$$\alpha_t(i) = P\left(\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_t, x_t = q_i \mid \lambda\right)$$

We have seen that the probability of partial observation sequence up to time t is given by $\alpha_t(i)$. Using the basic ideology mentioned above, forward algorithm computes $P(\mathcal{O} \mid \lambda)$. For $i = 0, 1, \ldots, N-1$, compute $\alpha_0(i)$ as

$$\alpha_0(i) = \pi_i b_i\left(\mathcal{O}_0\right)$$

For $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, compute $\alpha_t(i)$ as

$$\alpha_t(i) = \left(\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\right) b_i\left(\mathcal{O}_t\right)$$

From the above formula, we can derive $P(\mathcal{O} \mid \lambda)$ as

$$P(\mathcal{O} \mid \lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i)$$

The solution to HMM problem 2 is given by backward algorithm. This algorithm is contrast to forward algorithm. It starts from end and works its way to the start. For $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, we define

$$\beta_t(i) = P\left(\mathcal{O}_{t+1}, \mathcal{O}_{t+2}, \ldots, \mathcal{O}_{T-1} \mid X_t = q_i, \lambda\right)$$

Now, we compute $\beta_t(i)$ recursively as

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j\left(\mathcal{O}_{t+1}\right) \beta_{t+1}(j)$$

Once we solved forward and backward algorithms, we use these parameters to solve problem 3. We define $\gamma_t(i)$ as the most likely state sequence for given time t. For $t = 0, 1, \ldots, T-2$ and $i = 0, 1, \ldots, N-1$, define

$$\gamma_t(i) = P\left(X_t = q_i \mid \mathcal{O}, \lambda\right)$$

Since $\gamma_t(i)$ measures the relevant probability up to time t and $\beta_t(i)$ measures the relevant probability after time t, we can write $\gamma_t(i)$ as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O} \mid \lambda)}$$

We now have all the required parameters, to present the Baum–Welch re-estimation algorithm. This algorithm is presented in [2].

1. Initialize $\lambda = (A, B, \pi)$ with some random values.

2. Compute $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$ and $\gamma_t(i, j)$. $\gamma_t(i, j)$ can be given as

   $$\gamma_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j\left(\mathcal{O}_{t+1}\right) \beta_{t+1}(j)}{P(\mathcal{O} \mid \lambda)}$$

   Relationship between $\gamma_t(i, j)$ and $\gamma_t(i)$ is given as

   $$\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i, j)$$

3. Re-estimate the model parameters as

   $$\pi_i = \gamma_0(i)$$

   For $i = 0, 1, \ldots, N-2$ and $j = 0, 1, \ldots, N-1$ compute

   $$a_{ij} = \sum_{t=0}^{T-2} \gamma_t(i, j) \Big/ \sum_{t=0}^{T-2} \gamma_t(i)$$

   For $j = 0, 1, \ldots, N-1$ and $k = 0, 1, \ldots, M-1$ compute

   $$b_j(k) = \sum_{t \in \{0,1,\ldots,T-1\}} \gamma_t(j) \Big/ \sum_{t=0}^{T-1} \gamma_t(j)$$

4. If $P(\mathcal{O} \mid \lambda)$ increases, goto step 2.

4.4.2 Word2Vec

Word2Vec is a word embedding technique that is used more significantly in language processing. This model uses a shallow feed forward neural network that learns word association from a larger corpus. The input to word2vec model is a text corpus whereas output is a set of vectors each vector representing words in the corpus. This idea behind word2vec is that words also occur as part of sequential data where there exists transitional probabilities between words. Word2Vec can group similar word vectors in vector space, making words of similar meaning appear closer together. The vector representations of the words are used to analyze word associations with another word or use as feature vectors to other techniques [39]. We initially convert our words to one–hot vector representation and that is given as input to our word2vec network. Word2vec contains only one hidden layer, which is a fully–connected dense layer. Figure 3 gives the architecture of word2vec model.

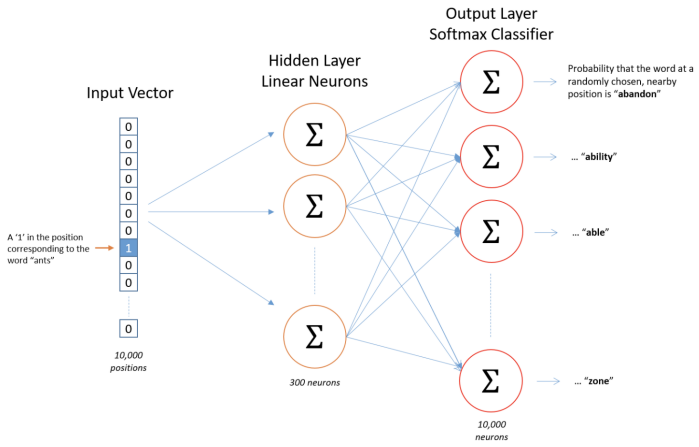


Figure 3: Word2Vec model architecture

The word2vec model can be implemented using two distinct approaches called continuous bag–of–words (CBOW) model and the skip–gram model. The architecture of these models is shown in the below Figure 4. The way the continuous bag–of–words model works is that it tries to predict the target word by reading the surrounding

words or the context words. The CBOW model tries to take as input “i want some orange” and adjusts the network to output “juice”.

Skip–gram model tries to predict the context words given the target word. We need to define whether to use skip–gram or the CBOW model based on the context of the problem. Basically skip–gram works well with smaller dataset and also excels at representing rare words. On the other hand CBOW model works well with large datasets and is very good at representing frequent words. For our problem we obtain better results with the CBOW model as we have a large dataset and also we are only using the most frequent opcodes from the dataset.

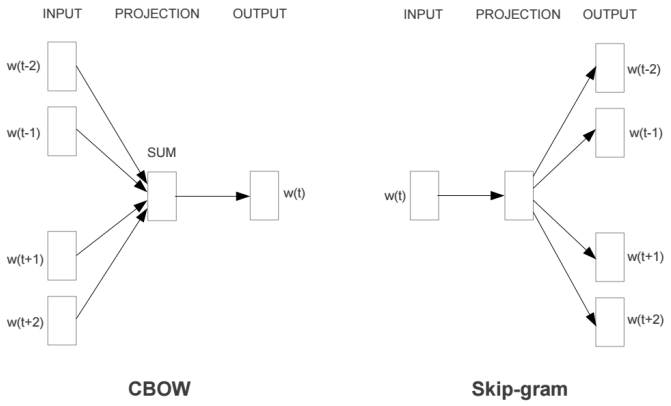


Figure 4: Types of Word2Vec model

4.4.3 Logistic Regression

Unlike the other techniques we have seen till now, logistic regression is used widely for classification problems. Logistic regression uses the sigmoid function (also knows as the logistic function) and hence the name. Sigmoid function is given as

$$S(x) = \frac{1}{1 + e^{-x}}$$

Logistic regression is modification on linear regression for better performance. It models the probability that observation takes one of the two binary values. Linear regression

makes unbounded predictions whereas logistic regression converts the probability in to clear 0 or 1 by using the sigmoid function. Plot of sigmoid function is given in the Figure 4.

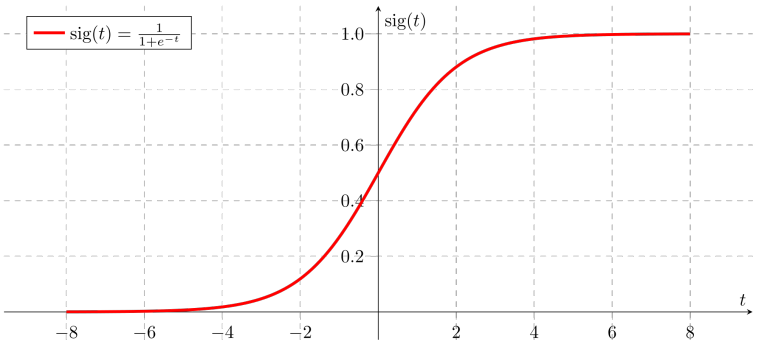


Figure 5: Graph of sigmoid function

We can see from the above graph that our logistic regression model output the values only between 0 and 1.

CHAPTER 5

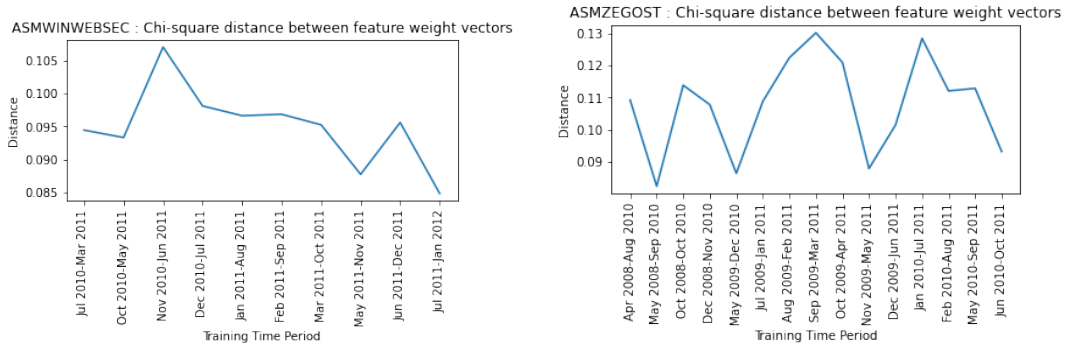
Experiments, Results and Analysis

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

5.1 Logistic Regression

This experiment is inspired from using SVM technique for malware analysis. The authors in [22] use linear SVM to analyze malware evolution problem. Similar to SVM, logistic regression is used extensively for classification problems. In this experiment, we train logistic regression model using time-windows approach. We divide our data into overlapping time windows of one year time period with a slide length of one month. All the samples from most recent one year time window are considered as +1 class whereas samples from the current month are considered as -1 class. We train our logistic regression model in this approach. Once we have a model we fetch the weights from the model and we calculate the Euclidian distances between vectors generated from all the models. We plot those distances on a graph to analyze the evolution. Figure 6 shows results of logistic regression on three families.

Results given by this experiment in Figure 6 shows that we do not find a significant spike in the families and the graph keeps fluctuating. Even though our logistic regression model achieved high accuracy in classification of samples, the weights of the hidden layer did not determine much about changes in the malware samples. This could be because with more number of weights we tend to get more noise and can make little sense out of the data. So, we could not deduce any valuable information from the results given by logistic regression. Hence we experimented with other techniques.



(a) Winwebsec (b) Zegost

Figure 6: Results from malware families on using Logistic Regression

5.2 Hidden Markov Model

We conducted two experiments using HMM technique. Both the approaches give us some insightful results into malware evolution problem. In both the approaches we extensively use solutions for HMM problem 1 and HMM problem 3. That is both the approaches have two phases, training phase and scoring phase. That is we train models to fit a given observation sequence and then we score samples on given models. Now, let us discuss these approaches in some detail.

5.2.1 HMM Approach 1

We know that HMM models are very good at deducing the structure of the data. We start our experiments by processing each malware family to find out the top thirty most frequent opcodes.

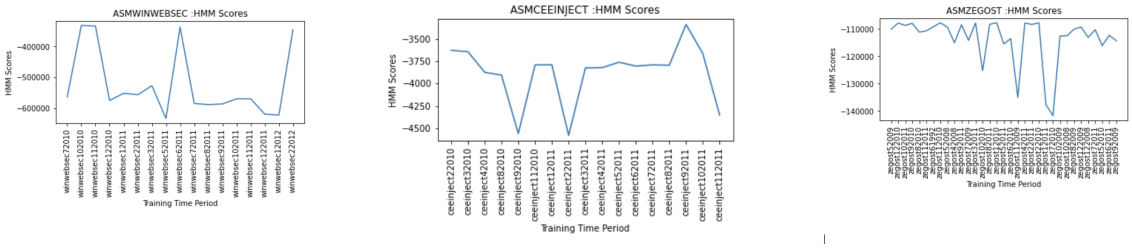
For each unique time period in our dataset, we trained an HMM model. Suppose we have malware samples that are created in May 2012, June 2012 and July 2012, we will train three different HMM models for these three time frames. We generate an observation sequence by treating opcodes as numbers. We consider the top thirty opcodes and every other opcode comes under “other” category. Now, will solve the HMM Problem 3, given in Chapter 4. In other terms, we are generating a model that

best fits the given data.

In our experiments, we have considered data from oldest time period as our test data. We have not trained any model for this time period, this is solely used for testing purpose. Suppose, we have data from May 2011, May 2012 and May 2013 we considered malware samples from May 2011 only for test purpose. We extract the observation sequence from test directories by only considering the top thirty opcodes (since the opcodes are analyzed for the entire family) and every other opcode comes under "other" category. We generate a single observation sequence by appending opcodes from all the samples. Now that we have some test data and HMM models from the same family, we execute solution from HMM Problem 1. That is we score the test data from the same family on each of the models. The idea behind this is, since each model is trained on the data from that particular time period, if the scores given by two models on an observation sequence is similar, we assume that the models are similar there by not much of change occurred between these models. If scores given by two models are drastically different, that means a significant change has occurred in that time period.

The length of observation sequence in training an HMM model ranges from Length of observation sequence ranges from 619226–1000000. We determined the N parameter to be two. The number of distinct observation symbols or M is thirty. We will discuss the results of these approaches on the malware families given below. Figure 7 shows results from this approach.

We have experimented with more number of opcodes as well but our results were similar. We can observe a significant spike in the graphs which confirms that evolution of malware has taken place during that time period. For HMM, it is important to make sure that our model converges. So, we experimented with random restarts as well. But our results were similar which tells us that the model is converging. We can



(a) Winwebsec (b) Ceeinject (c) Zegost

Figure 7: Results from malware families on using HMM approach 1

thus confirm that this approach can be used to understand significant evolution in the malware families. We will analyze more about the results in the later parts of this chapter.

5.2.2 HMM Approach 2

Now, we will consider another experiment we performed using HMM. We start the experiment by processing the top thirty opcodes in each malware family just like the previous experiment. Our data is segregated according to its creation date grouped by each month. So we have malware samples for each month. We split the data from each time period in to training and testing samples. We used 75% of samples for training and 25% of samples for testing. Now we train an HMM model for data from each month. We determined the number of states to be two. The length of observation sequence and the distinct opcodes were similar to the previous experiment.

Suppose we have data from May 2015, June 2015, July 2015 etc. We use training samples to generate an HMM model of each of these time periods. So we will have three HMM models. Now let us consider two HMM models of adjacent time periods t_1 and t_2 as λ_{t1} and λ_{t2}. We score test samples from t_1 with both the models λ_{t1} and λ_{t2}, giving us two vectors of same length. If we have, m test samples our vector will be of length m. Since each sample might be of different length, we normalize

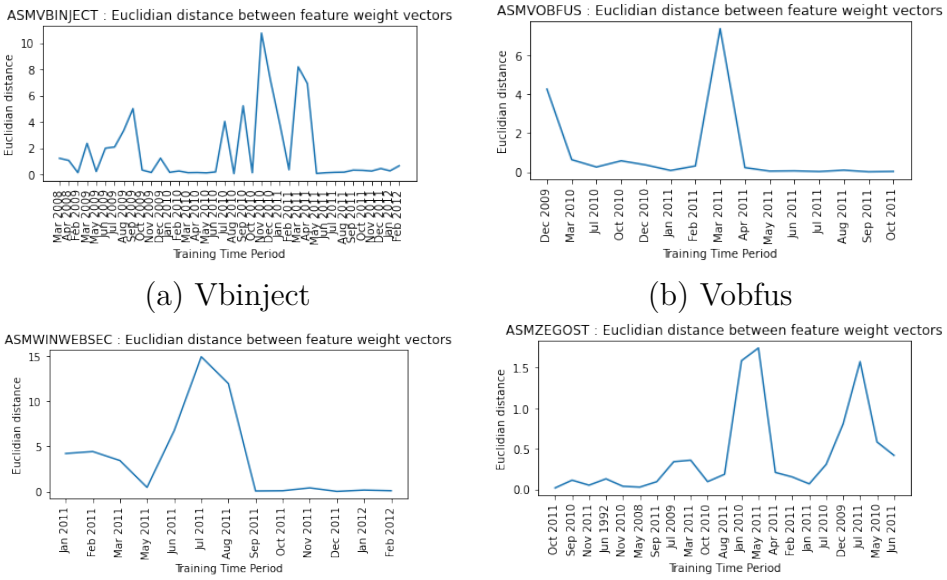
each score by dividing it with the length of the observation sequence generating log likelihood per opcode (LLPO). We generate LLPO, in order to directly compare the scores irrespective of the length of observation sequences.

Once we generate two vectors by scoring the samples with the models λ_{t1} and λ_{t2}, we compute Euclidean distance between the vectors. Let us say this is $distance_1$. We do the same for the test samples for time period t_2. We score the samples from time t_2 with the models λ_{t1} and λ_{t2}, generate vectors and compute Euclidean distance between the vectors as $distance_2$. Now we average the two distances and store it as the final distance. We do the same process for all the samples from all the time periods and compute the distances.

Once we have distances, we plot the graph to look for evolution. Small distance suggests minimal change, whereas larger distances indicate potential evolution points. The idea behind this approach is if the distance between two feature vectors generated by scoring same samples on two different models is high, it implies that the samples are scored differently indicating how different the models are. Hence larger distances indicate potential evolution points in the graph.

Figure 8 gives results from malware families on using the above mentioned approach. This experiment has been very successful in determining the exact points in the time where evolution occurred.

Results generated by both the HMM approaches give us some significant results in analyzing the malware evolution. In approach 1, we followed a simple approach by generating the models and compared the scores to look for the evolution. In approach 2, we took some strict measures to eradicate considering minor changes in the models. This is done by calculating score vector distances from samples of two time periods and averaging the distances to generate final distance. Hence both the approaches give successful results in analyzing the malware evolution.



(a) Vbinject (b) Vobfus

(c) Winwebsec (d) Zegost

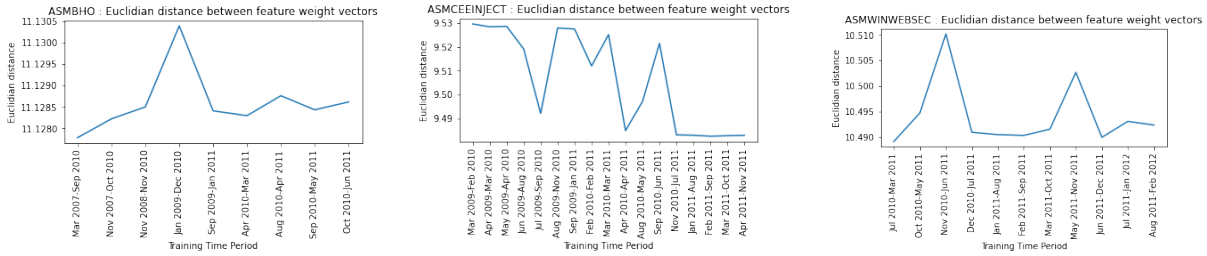
Figure 8: Results from malware families using HMM approach 2

5.3 HMM2Vec

In this section we will discuss our HMM2Vec technique and its results on the dataset. We start our experiment by processing the entire malware family to identify the most frequent thirty opcodes. Going forward we only consider these opcodes and every other opcode comes under one category.

We divide the dataset into an overlapping time–window of one year with a slide length of one month. We train HMM models on each window with two states. Length of observation sequence ranges from 619226–1000000. The observation probability matrix or the B matrix of the resulting model will contain two rows each of length thirty (since we have thirty distinct opcodes). We use vectors from this matrix as our feature vectors. We consider the two rows from the model as two vectors. We convert it into one vector by appending the feature vector of the first state followed by the second state. We obtain the second vector by appending the feature vector of the second state followed by the first state. In HMMs there is a chance that the states

of the observation matrix might be interchanged. In order to eliminate this problem, we make two vectors from each model. We then compare the distance between the vectors. We compute Euclidean distance in this case and the minimum distance between two vectors is taken as the final distance between the two models. We then plot the distances on the graph to analyze the similarities between the data.



(a) Bho (b) Ceeinject (c) Winwebsec

Figure 9: Results from malware families using HMM2Vec

Analyzing the results from Figure 9 tells us that HMM2Vec is successful in identifying significant evolution in most families. We observe significant spikes or evolutionary points for families like Ceeinject, Winwebsec, Bho, Dorkbot and Delfinject. Thus this experiment confirms to us that HMM2Vec is efficient in analyzing potential evolutionary points in the malware family.

5.4 Word2Vec

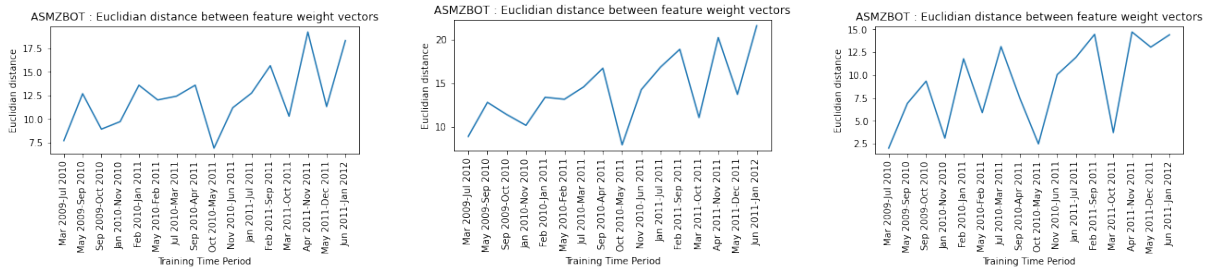
We have seen in the previous research given in [22] that the authors used vectors generated from Word2Vec technique as input to other machine learning techniques like support vector machines (SVM). In this experiment we use vectors generated from Word2Vec technique for direct comparison. Word2Vec is extensively used to generate word embeddings. These word embeddings are word vectors in space, where vectors of similar words are closer in vector space.

We employ Word2Vec experiment in the sliding window approach. We divide the dataset into an overlapping time window of one year with slide length of one month.

Here we use CBOW instead of the skip–gram model. We chose CBOW over skip–gram because CBOW performs well with large datasets and it is efficient in representing frequently occurring words.

In this approach, we have considered top thirty most frequent opcodes for a family and every other opcode is treated under one category. Since we cannot feed opcodes directly to the Word2Vec model as input, we generate one–hot vectors and these are given as input to the model. This vector will have thirty components (since distinct opcodes we consider are thirty) with 1 in the position corresponding to particular opcode and 0s in all other positions.

Window size implies the number of context words. We have experimented with different window sizes but window size of 5 works well in our case. We also experimented with different vector sizes. Vector size represents number of hidden layers in our Word2Vec model. There was not much of difference with different vector sizes. Hence we conducted all the experiments with vector of size 2. Figure 10 gives experiments on Zbot family with different vector sizes.



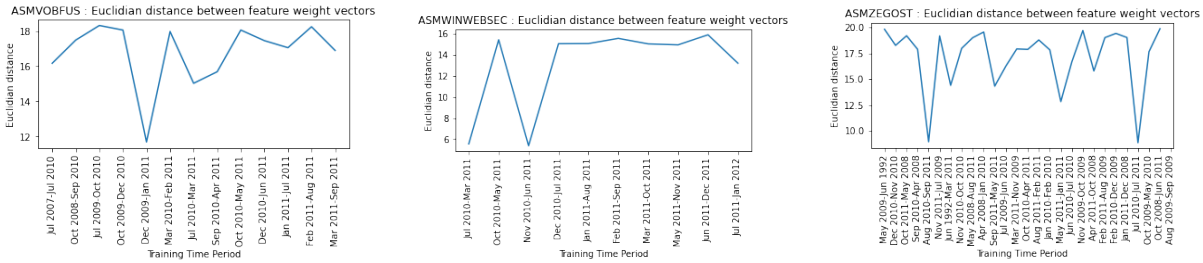
(a) Zbot vector size 2 (b) Zbot vector size 3 (c) Zbot vector size 5

Figure 10: Results from zbot family with different vector sizes

Once our word embeddings are generated from our Word2Vec model, we append word embeddings of each opcode together to generate a feature vector. Once we have vectors generated, we compute an average weight vector and distance between these vectors is calculated as our scores. If the distance between two vectors is close the

models are similar else the models are very different from each other showing potential evolution.

We plot the scores on a graph to analyze the evolution in the malware family. Results from Word2Vec technique are given in Figure 11



(a) Vobfus (b) Winwebsec (c) Zegost

Figure 11: Results from malware families on using Word2Vec

Results from Word2Vec approach shows potential evolutionary points in almost all the malware families. We can see significant evolution in the families Vobfus, Winwebsec and Zegost. Word2Vec showed significant evolutionary points in other families as well. Conclusively, Word2Vec proved to be a strong technique to detect malware evolution.

5.5 Analysis of Results

In this section we will analyze the results given by different techniques, compare and contrast it with the results given by the previous work. First we will discuss the results given by each technique, compare the similarities of results given by each technique then discuss the similarities with previous work.

We want to identify if the evolution point is determined by the number of samples used in the training process. Figure 12 shows the evolution points given by Word2Vec experiment and also the number of samples used in each time window. The results clearly show that our techniques are able to detect the evolution points despite the number of samples used.

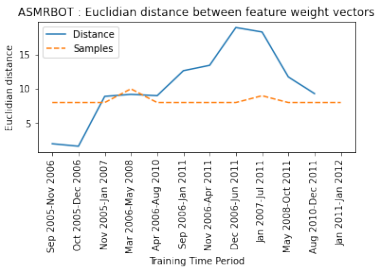


Figure 12: Results from Rbot family given by Word2Vec technique

We started our experiments using HMM technique. In this case we scored samples on HMM models and compared the generated score vectors. Results given by HMM clearly shows us that the scores are distinct from each other. This clearly implies that two models are significantly different from each other because if the models had been same, the scores would have been the same or with very less difference. This distinction in the scores proves that the malware family has evolved over time because without an evolution the models should be similar generating similar scores. Hence, it is evident that HMMs are successful in detecting malware evolution.

In our next experiment using HMM2Vec technique, we obtain successful results by employing the B matrix of HMM model as feature vector for comparison. If the distance between vectors generated by B matrix of two models is less, this implies that two models are similar. In contrast if the distance between vectors is large, this implies that the models are significantly different from each other showing an evolution occurred in the malware family because without an evolution, models on two samples should be similar by generating similar vectors. Hence the results given by this approach shows actual evolution not any random fluctuation.

Our approach used in Word2Vec is also similar to the HMM2Vec approach. This experiment proves the strength of word embedding models. Results given by this technique shows successful evolution in almost all the malware families. Since

Word2Vec calculates word transition probabilities, if two vectors generated by two models are closer to each other, this shows that models are similar. If the vectors are distant from each other, this shows that models are drastically different proving that malware evolution occurred in the family.

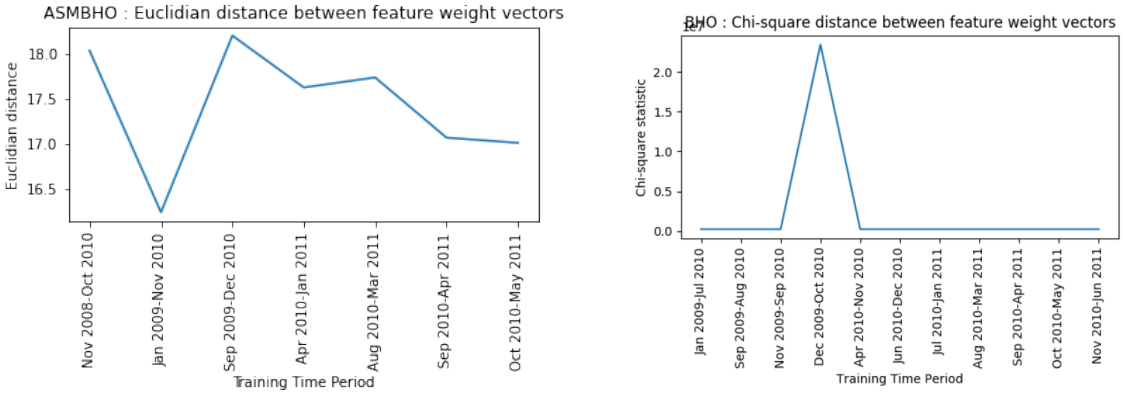
We achieved significant results using all our approaches. HMM2Vec and Word2Vec models gives us the time windows where evolution has taken place. Analyzing the results given by Word2Vec and HMM2Vec techniques on Winwebsec family shows us that malware evolution took place during the time period November 2010–June 2011. Now experimenting with HMM approach on Winwebsec family shows us the exact point of evolution as June 2011. We are able to identify the potential time windows of evolution as well as exact point in time where evolution occurred for all the families using our experiments.

We can compare some of our work to the work given in [22] as same dataset is used by the authors in this paper. The authors in [22] used SVM technique to detect the malware evolution. They used vectors generated by Word2Vec approach as input to SVM model. Once a χ^2 graph is generated, they confirm the evolution by training HMM on either sides of the spike. Comparing our results with the results given in [22], we are successful in detecting malware evolution for the families for which the previous research could not detect. And for several other families our results match the results given in [22]. Previous researchers could not detect malware evolution for families like Delfinject, Dorkbot and Zbot whereas we could detect malware evolution for all the families using our approach. Also, the previous work could only show that evolution occurred in a particular time window of one year whereas we could determine the exact time when malware evolution occurred. Hence the research given in this paper gives better insights in to malware detection problem. In the below section, we will discuss the results on each of the malware families and compare the results given by

our experiments on each family to the results given by the previous research.

Adload: For this family, both HMM2Vec and Word2Vec did not give any significant spike in the graph showing a strong evolution. But the graph keeps changing showing that malware samples differed over a period of time. On the contrary results given by HMM approach shows significant spikes for this family.

Bho: The results generated by Word2Vec on this family shows that malware evolution occurred during September 2009–December 2010. Whereas our experiment conducted by HMM approach tells us that malware evolution happened during October 2010. We compare this with results given by authors in [22]. Figure 13 shows comparison of results between Word2Vec approach given in this paper and SVM approach given in [22]. Both the graphs, shows the evolution at the same time window. Overall graph might not look the same because, we are comparing Euclidian distances between the vectors where as research given in [22] compares χ^2 distances between the SVM models.



(a) Word2Vec results (b) SVM results in [22]

Figure 13: Comparison of results from bho family

Bifrose: The results generated by HMM2Vec did not give much information regarding the malware evolution. We could only understand that malware samples had

changes over time but the change was not so strong that we found a spike in the graph. Where as graph generated by the Word2Vec approach gives us a better understanding of changes in the malware family. We could see that a significant evolution occurred during November 2009–March 2011. Results given by the HMM approach narrows down the evolution point to March 2011. A similar graph is given by authors in [22] showing evolution during November 2010–May 2011.

Ceeinject: For Ceeinject malware family, we get good results from all the experiments we performed. Results given by HMM2Vec and Word2Vec shows significant evolution during August 2010–July 2011 and we identify the exact month of evolution as November 2010 using HMM approach. Results given by [22] shows similar evolution during September 2010–May 2011.

Delfinject: We obtained good results for this family using the HMM approach. This shows a significant evolution occurred during January 2011. We did not observe significant spikes on the graph from Word2Vec or HMM2vec technique. In retrospect work given in [22] could not find any significant results on this family where as we could analyze the evolution using HMM technique.

Dorkbot: We achieved good results on this family from all our experiments. A significant spike is observed in the graphs plotted by all the three techniques revealing that malware evolution happened during 2011. We tried to compare the results with previous work given in [22]. Techniques mentioned in [22] could not find proper evolution in this family.

Hupigon: The results received from Word2Vec technique shows that significant malware evolution in this family happened during July 2010–April 2011. Results

by HMM approach points out that evolution occurred during February 2011. Results given by the SVM approach in [22] identifies that malware evolution happened during June 2010–January 2011.

Ircbot: The results generated by the Word2Vec technique identifies that malware evolution occurred during 2011. There is no significant spike observed but the graph shows that the malware evolved over a period of time.

Obfuscator: We could not derive significant information from this family. Graph plotted on this family using has many spikes which did not give us much information regarding the malware evolution.

Rbot: Graph generated by the Word2Vec technique shows significant evolution in the malware family. We did not observe significant results on this family from any previous researches.

Vbinject: We could not observe a significant spike in this malware family through our experiments. Results given by the experiments on this family shows so many fluctuations in the graph showing no significant evolution.

Vobfus: The results generated by our experiments shows that a significant evolution in this family takes place during December 2009–January 2011. The results given in [22] shows a significant evolution during November 2010–May 2011.

Winwebsec: We observed significant evolution in this malware family using our Word2Vec experiment. We observed a significant spike in the family showing that malware evolved during December 2010–July 2011. On the other hand previous research given in [22] could not find any significant evolution for this family.

Zbot: Experiments conducted on this family shows significant changes in the family. We can see a spike showing significant evolution in this family happened between April 2011–November 2011.

Zegost: Experimental results given by Word2Vec technique shows significant spikes in the graph. This graph shows significant evolution taking place between August 2010–September 2011 and another significant evolution during July 2010–July 2011.

Results given by all the experiments shows significant information. Our experiments shows significant evolution in almost all the malware families. By comparing the results given by HMM, HMM2Vec and Word2Vec techniques we can see that there are clear similarities in the results given by these three approaches on many families. Since we observed similar evolution points given by all the experiments, there is a confirmation that malware family evolved during that particular time period. Also, not just finding out a time–window in which malware evolution happened, we are able to find out the exact point in time where malware family significantly changed. Even though HMM approach give us exact points in time, where evolution occurs, there are few tradeoffs we need to consider. HMMs are computationally intensive. Execution of Word2Vec experiments on winwebsec family takes around two minutes of time whereas HMM execution takes around forty minutes. Hence, given our requirements and the time constraints, we can choose any of the experiments to detect malware evolution. Evolutionary points generated for few families by previous research given in [22] matches with our experiments. On the other hand, our techniques gave successful and better results on many families which earlier research could not detect.

We have conducted experiments with several techniques. We used neural networks, statistical techniques and word embedding techniques. Although results generated

by logistic regression did not give great insights into malware evolution analysis, we received valuable results from all the other techniques. We have seen results generated by our experiments on all the families and in most cases we obtained significant results showing point in time, when malware evolution occurred. HMM, HMM2Vec and Word2Vec techniques proved to be very efficient in detecting the malware evolution and they provided valuable results.

CHAPTER 6

Conclusion And Future Work

In previous research, authors used secondary tests in order to confirm the evolutionary changes in the malware families. Any significant changes in the malware family was given by the χ^2 graph which was called as malware evolution. In this research, we have confirmed malware evolution by training models for overlapping time window and we scored each model with the samples from the same family or calculated euclidian distances from each other. The scores obtained were quite distant from each other showing us that potential change has occurred in these families. Thus this technique is successful in identifying malware evolution.

In all our experiments we used opcodes extracted from the PE files. We processed each family to identify the top thirty most frequent opcodes and we used these in all our experiments, with all other opcodes grouped into one "other" category.

In the first set of our experiments, we segregated our data into overlapping time-windows of one year with a slide length of one month. We then trained an HMM on each time window. Once our models are ready, we scored each model with the samples from the same families. If the scores generated by the models are similar no significant change has occurred but if the scores differ significantly that shows that potential change in the malware family has occurred. This technique is successful in identifying the potential evolution points in the families.

We then use word embedding techniques Word2Vec and HMM2Vec to identify the malware evolution. We train the models for an overlapping time-windows and the vectors generate are used for comparison. We compute the distance between the vectors and we plot the distance to analyze the evolution in the malware family. We achieved successful results from both the techniques and results from both the techniques confirm the similar evolutionary points for each families.

We carried out next set of experiments with logistic regression. We trained a logistic regression model on over-lapping time windows. Data from previous one year time period is considered in to +1 class whereas samples from current month are considered in to -1 class. Once the models are generated we compare the feature weights of the models. We compare the distance from each other to analyze evolution. This experiment did not give any useful results in understanding malware evolution.

In this research, we we derived successful results from Word2Vec, HMM2Vec and HMM techniques. This clearly shows us that word embedding techniques are very powerful in detecting malware evolution. We were also able to determine significant results for many families for which previous researches did not produce any valid information. Thus our research proves the strength of word embedding techniques in malware evolution problem.

6.1 Future Work

In this paper, we conducted our experiments using mnemonic opcodes from the malware samples. In the future, we can consider experiments with other features from the malware samples. This might give us an understanding of different features of malware samples.

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] A. Gostev, O. Zaitsev, S. Golovanov, and V. Kamluk, "Kaspersky security bulletin. malware evolution 2010," *Kaspersky Lab (April 2009)*, vol. 5, 2011.

[2] M. Stamp, "A revealing introduction to hidden markov models," https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf, 2018.

[3] J. Aycock, *Computer viruses and malware*. Springer Science & Business Media, 2006, vol. 22.

[4] M. Stamp, *Introduction to machine learning with applications in information security*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2018.

[5] J.-M. Borello and L. Mé, "Code obfuscation techniques for metamorphic viruses," *Journal in Computer Virology*, vol. 4, no. 3, pp. 211–220, 2008.

[6] N. VILLENEUVE, R. Deibert, and R. Rohozinski, "Koobface," 2010.

[7] M. Barat, D.-B. Prelipcean, and D. Gavriluţ, "A study on common malware families evolution in 2012," *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 4, pp. 171–178, 2013.

[8] 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.

[9] M. Stamp, *Information security: principles and practice*. John Wiley & Sons, 2011.

[10] 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.

[11] Symantec, "What is difference between viruses, worms and trojans," http://www.symantec.com/business/support/index?page=content&id=TECH98539, 2009.

[12] 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.

[13] S. Mohurle and M. Patil, "A brief study of wannacry threat: Ransomware attack 2017," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, pp. 1938–1940, 2017.

[14] J. Ma, J. Dunagan, H. J. Wang, S. Savage, and G. M. Voelker, "Finding diversity in remote code injection exploits," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006, pp. 53–64.

[15] 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.

[16] J. Ouellette, A. Pfeffer, and A. Lakhotia, "Countering malware evolution using cloud-based learning," in *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)*. IEEE, 2013, pp. 85–94.

[17] 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, p. e1978, 2018.

[18] M. Wadkar, F. Di Troia, and M. Stamp, "Detecting malware evolution using support vector machines," *Expert Systems with Applications*, vol. 143, 2020.

[19] 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.

[20] 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.

[21] S. Rezaei, A. Afraz, F. Rezaei, and M. R. Shamani, "Malware detection using opcodes statistical features," in *2016 8th international symposium on telecommunications (IST)*. IEEE, 2016, pp. 151–155.

[22] S. Paul and M. Stamp, "Word embedding techniques for malware evolution detection," in *Malware Analysis Using Artificial Intelligence and Deep Learning*. Springer, 2021, pp. 321–343.

[23] A. Nappa, M. Z. 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.

[24] S. Kim, "PE header analysis for malware detection," Master's thesis, San Jose State University, Department of Computer Science, 2018.

[25] "Win32 bifrose detected with windows defender antivirus," https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/bifrose, 2012.

[26] "Virtool: Win32 ceeinject detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject, 2007.

[27] "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.

[28] "Worm: Win32 dorkbot detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm%3AWin32/Dorkbot, 2011.

[29] "Adware: Win32 hotbar detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware%3AWin32%2FHotbar, 2006.

[30] "Backdoor: Win32 hupigon detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor%3AWin32%2FHupigon, 2006.

[31] "Win32 obfuscator detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FObfuscator, 2011.

[32] "Win32 rbot detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FRbot, 2005.

[33] "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.

[34] "Win32 vobfus detected with windows defender antivirus," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fvobfus, 2010.

[35] "Win32 winwebsec detected with windows defender antivirus," https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fWinwebsec, 2010.

[36] "Win32 zbot detected with windows defender antivirus," http://www.symantec.com/securityresponse/writeup.jsp?docid=2010-011016-3514-99, 2011.

[37] "Win32 zegost detected with windows defender antivirus," https://www.symantec.com/security-center/writeup/2011-060215-2826-99, 2011.

[38] 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.

[39] J. Gilyadev, "Word2vec explained," https://israelg99.github.io/2017-03-23-Word2Vec-Explained/, 2017.

APPENDIX

Appendix

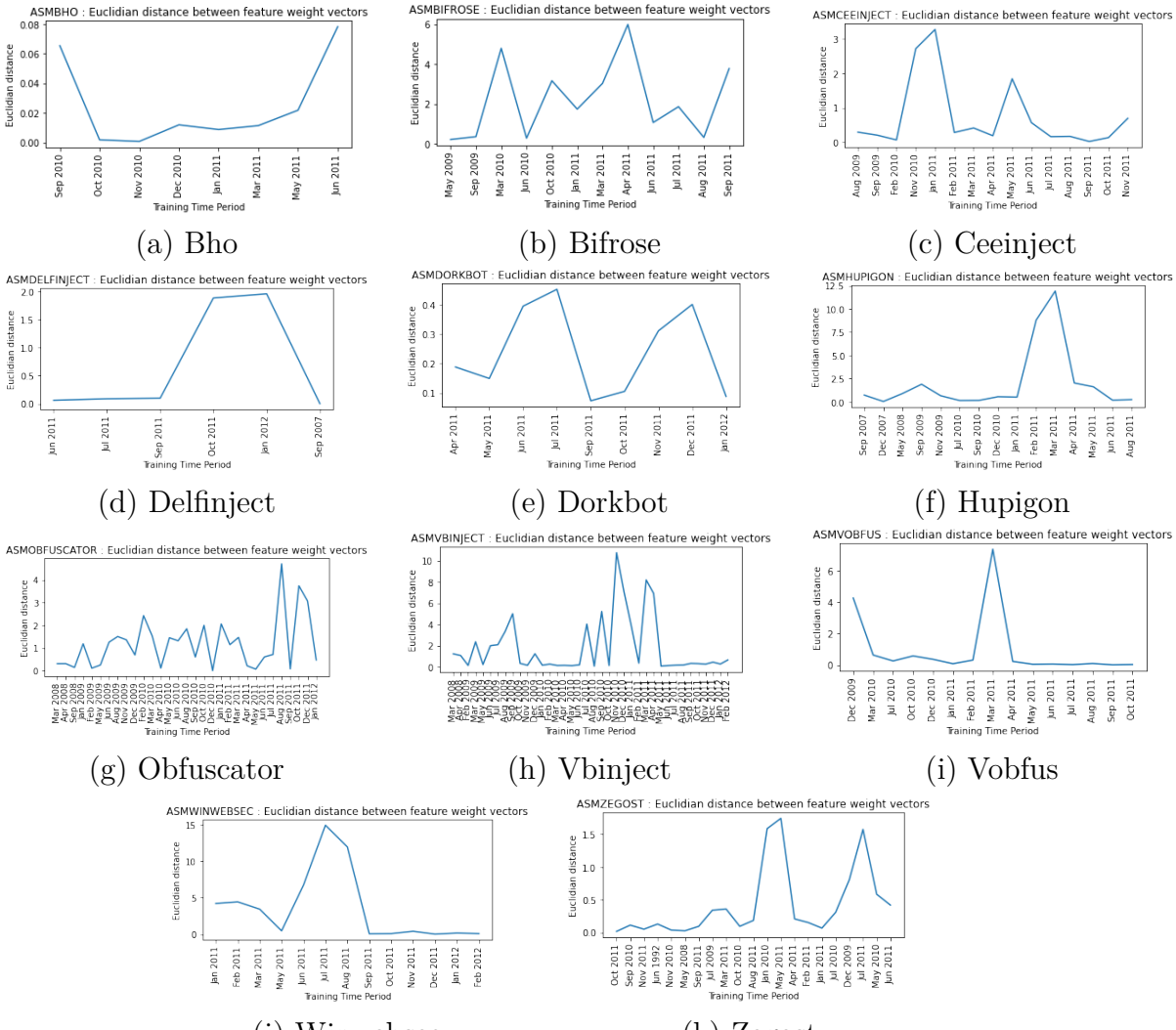
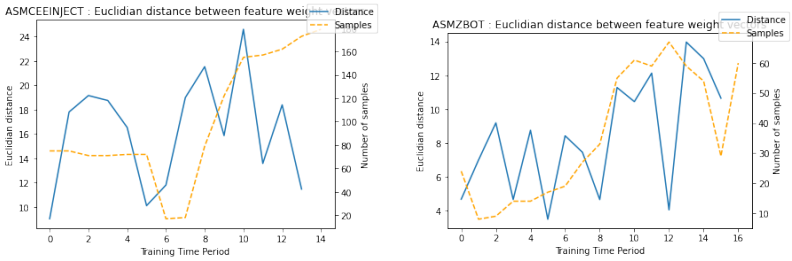
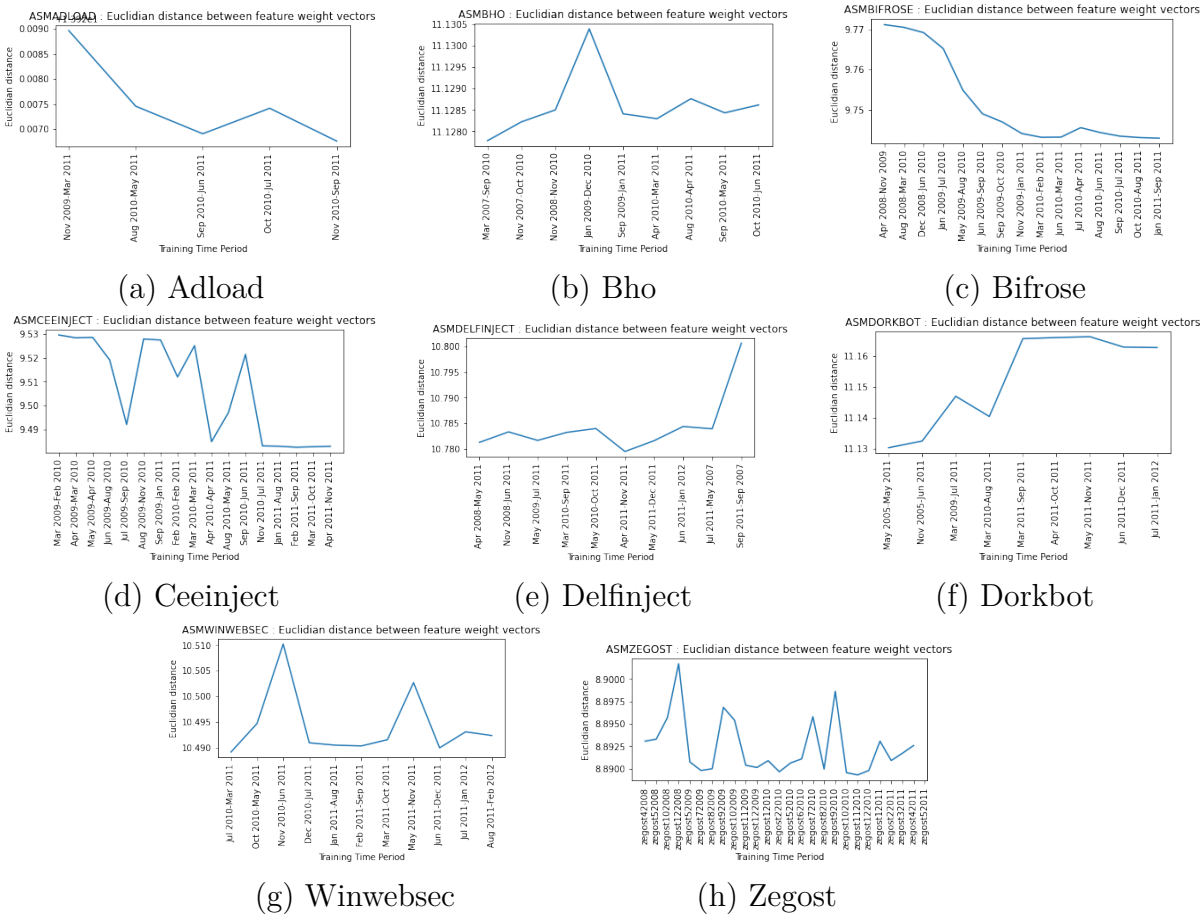


Figure A.14: Results from malware families on using HMM Approach 2



(a) Ceeinject (b) Zbot

Figure A.15: Results from malware families on using Word2Vec showing number of samples

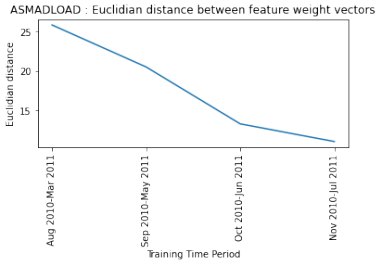


(a) Adload (b) Bho (c) Bifrose

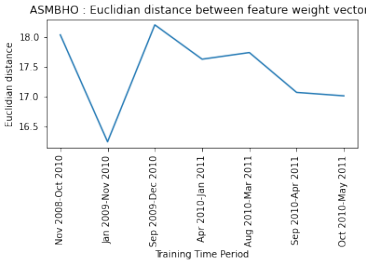
(d) Ceeinject (e) Delfinject (f) Dorkbot

(g) Winwebsec (h) Zegost

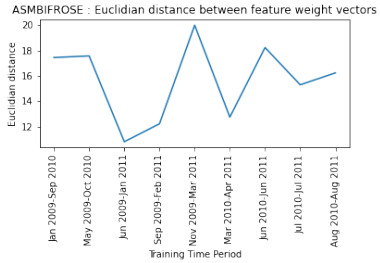
Figure A.16: Results from malware families on using HMM2Vec



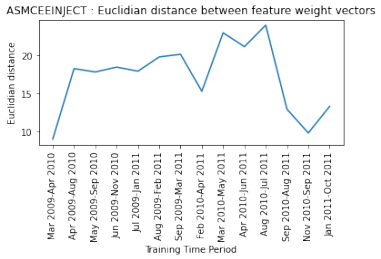
(a) Adload



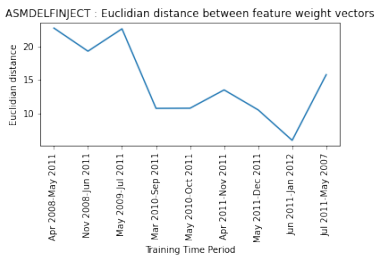
(b) Bho



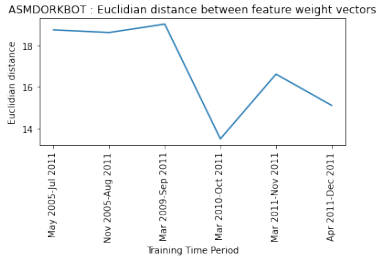
(c) Biforse



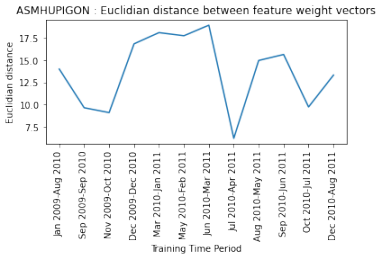
(d) Ceeinject



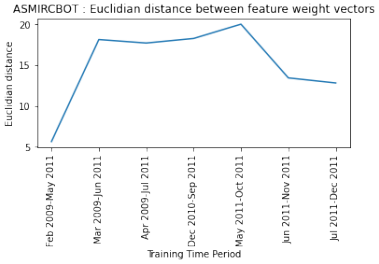
(e) Delfinject



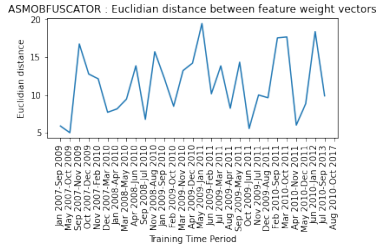
(f) Dorkbot



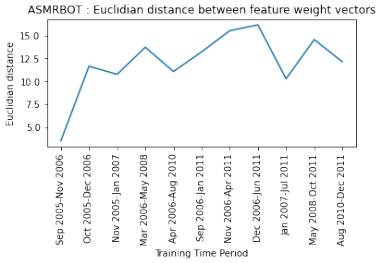
(g) Hupigon



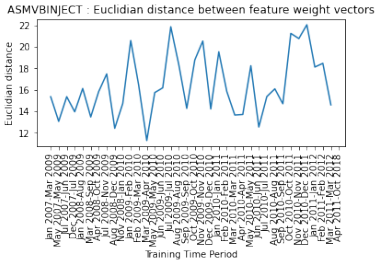
(h) Ircbot



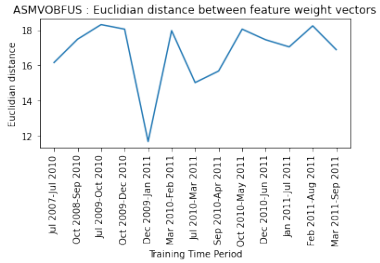
(i) Obfuscator



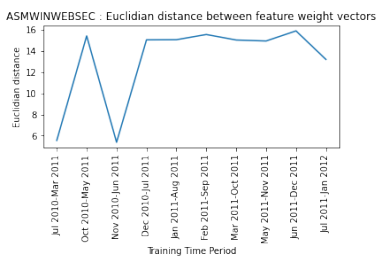
(j) Rbot



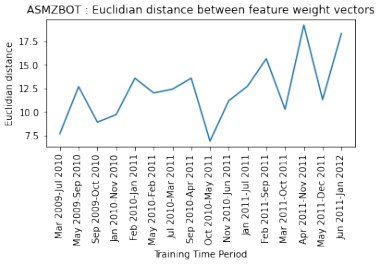
(k) Vbinject



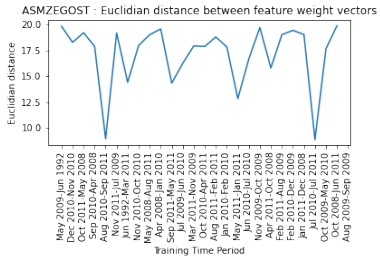
(l) Vobfus



(m) Winwebsec



(n) Zbot



(o) Zegost

Figure A.17: Results from malware families on using Word2Vec