Spring 2022

Concept Drift and Malware Detection

Xiaoli Tong

Concept Drift and Malware Detection

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

Xiaoli Tong

May 2022

The Designated Project Committee Approves the Project Titled

Concept Drift and Malware Detection

by

Xiaoli Tong

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2022

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

ABSTRACT

Concept Drift and Malware Detection

by Xiaoli Tong

In software development, new software is often based on a previous version with some improvements or new features. A similar software development practice holds true for malware writers, that is, hackers tend to add features to existing malware and release revised versions, which can be viewed as belonging to existing malware families. Therefore, a malware family typically evolves over time. In this paper, we build on recent research that has demonstrated that malware evolution can be detected using machine learning techniques. Specifically, we account for concept drift in the context of malware evolution, in the sense that we retrain our models whenever substantial evolution is detected. By accounting for concept drift, we obtain improved results as compared to models that do not consider concept drift.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest appreciation to Dr. Mark Stamp as my advisor for his continuous support and guidance through the research. He is always there and ready to help.

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

I would also like to thank Samanvitha Basole for her time and advice. She sacrifices her lunch time to have a meeting with me.

I would especially thank my parents for supporting me and educating me. Thanks my husband, my two kids for their love and their understanding. Without their support, I could not imagine how I can finish all my studies in the graduate school.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware, short for "malicious software", is software that is designed to cause harm, gain unauthorized access, or perform other malicious activities [1]. Malware can be classified into different types, such as virus, worm, trojan, trapdoor, rabbit, etc. With the popularity of the Internet in recent years, millions of people have been the victims of malware attacks. The number of malware variants have grown rapidly, in parallel with the emergence of new technologies. According to [2], the number of malware is more than 1 billion in 2022, and 560,000 new malware are detected every day. Consequently, malware is one of the biggest security concerns for individuals and businesses.

There is an urgent need to detect malware. Classically, there are three general approaches to detecting malware [3], namely, signature detection, change detection, and anomaly detection. Signature detection is used to detect patterns in a specific malware. Signature detection is an effective technique to detect a specific, known malware. Moreover, signature detection poses a minimal burden for users, because the signature files can be kept up to date automatically. Change detection consists of find any modifications to files [3]. False negatives can be avoided when change detection is used, and unknown malware can also be found. However, the false positive rate can be high. Anomaly detection is based on abnormal or virus-like behavior of software. While anomaly detection can be used to discover new malware variants, it also tends to suffer from high false positive rates.

Malware writers develop different strategies in response to advances in detection methods. The resulting arms race between malware writers and malware detection makes malware detection a challenging and constantly evolving task.

In software development, new software is often based on a previous version with

some improvements or new features. A similar software development practice holds true for malware writers. That is, hackers tend to add features to existing malware code and release revised versions of malware, which can be viewed as belonging to an existing malware families [1]. Therefore, a malware family can be viewed as evolving over time [4].

Most previous studies on malware evolution focus on reverse engineering [5], which is a time and labor-intensive process. In paper [6], graph pruning methods were applied to the malware evolution detection problem. The series of papers [5, 7, 8] apply machine learning techniques to detect malware evolution, and the research in this paper can be viewed as an extension of this previous work. Whereas previous work was focused entirely on detecting when a malware family has evolved, we will take this process one step further. Specifically, when we detect evolution we will update our machine learning models. We will then compare the effectiveness of these updated models to the case where the models are static. Updating models in this manner can be viewed as a means of dealing with concept drift (also known as data drift) that occurs within a malware family [9, 10].

We propose a method to detect malware evolution in three steps. The contribution of our research is that our models can detect drift at specific time periods, instead updating models using sliding windows. Our approach can be used to automatically update models if the threshold of level of detection is reached.

The rest of the paper is structured as follows. In Chapter 2, relevant background topics and related work on malware evolution are discussed. In Chapter 3, we present the data set used in our experiments, as well as a brief review of the machine learning algorithms we employed; concept drift is also introduced in this section. Our experiments and results are covered in Chapter 4, while conclusion and potential future work are discussed in Chapter 5.

CHAPTER 2

Related Work

In this chapter, some previous research on malware evolution detection is covered. Understanding the evolution of malware not only help to know how malware evolves overtime, but also it may be possible to guide the future research on malware families by making prediction on the direction where the malware is heading. Past research on malware evolution can be classified into two categories, implementing machine learning methods and others that not utilizing machine learning methods. We will give a brief review on these researches in this chapter.

The paper [11] is based on malware samples spanned almost twenty years. The goal of the paper is to understand how malware has evolved over time. It also focus on the relation between the different malware families. Graph pruning is introduced in this paper. The author claims that some malware inherit features from other malware families over a specific time period. However, comprehensive examination is needed to identify the related families manually. With a large number of malware families existing and new malware emerging everyday, it would be very time consuming to do the investigation.

The authors in [12] analyze the changes of malware based on the Adroid malware samples. A large number of features are extracted from these samples. The trends are detected by employing standard software quality metrics. The authors use the trends in malware compared those trends from goodware, and conclude that the trends in malware follow the similar path as those of goodware. The paper [8] argues that there may be a large number of overlap between malware and goodware because Andriod malware contains significant number of goodware.

Machine learning algorithms are used in [5, 7, 8] to detect the evolution of malware. All these approaches can be implemented automatically without a large amount of

time to reverse engineering the code. In [5], 54 static features are extracted from Windows portable executable (PE) files of malware samples. Linear SVMs are fitted on one year time windows. χ^2 statistics is utilized to quantify the difference between the coefficients of two different linear SVM models. χ^2 similarity plot are drawn based on all the results from χ^2 statistics. The spikes in the χ^2 indicate changes in the code for a specific malware family. In paper [7], opcode sequence is extracted from malware samples. Linear SVM is trained over on the data with one year time span each time. χ^2 is used to compare the feature coefficients from the linear SVM models. χ^2 similarity plots are obtained based on the results from all the linear SVM models. If there is a spike in the χ^2 graph, it may indicate that there is some changes at this point for this malware family. Then HMM is trained on both sides of the spikes. If the score from HMM models are different on both sides of the spike, more evidence is shown that there is some changes at this specific time period. Paper [8] is also based on the opcode sequence extracted from malware samples. HMM models are trained on 31 different observation symbols and two states. Among the 31 observations, the most top thirty frequent opecodes are selected and the rest are classified into one observation symbols. The samples from two consecutive months are used to train and test HMM separately. Then the same process continues until it reaches the end of the entire malware family. If the two scores from consecutive months are different, it may indicate there is some changes between these two months. HMM2Vec and Word2Vec are implemented in [8] to detect the changes in a malware family.

Studies have been done on concept drift in malware detection [13, 14]. The research presented in [13] declares that it is crucial to take the concept drift into account for malware detection. Two measures are proposed to track the drift in malware families, relative temporal similarity and metafeatures. Byte 2-grams and Mnemonic 2-grams are chosen as static features of malware samples. The cosine

similarity measures the commonality between the features of two samples. The decrease of cosine similarity based on timestamp indicates a characterized drift among the malware samples. Tracking metafeatures is another approach to measure the concept drift. However, it is difficult to determine the exact evolution point based on these two methods.

The study in [14] retrains models periodically so that the updated models could capture the drift and adapt changes over time. However, the training process is very expensive and time consuming, especially with a large volume and a long time span data set.

Features are the vital components in machine learning. Generally, there are two types of malware features used in malware analysis, namely, static features and dynamic features. Static features are collected from malware files without executing the code, such as bytes,opcode N-Grams. Dynamic features are extracted while executing malware. In this paper, static features are used. So, we only list some researches based on static features below. The work in [15] explores different feature types for malware families. It concludes that the features selected from PE headers has more discriminative power than other feature types in malware classification. Moreover, the cost of extracting PE features are lower compared with other feature extracting methods. Studies [16] are based on byte n-gram features. The K n-grams with large information gain are selected as features among all of the n-grams. Research in [17] detects malware using opcode n-gram features. The authors have done experiments to determine the size of n-gram, which ranges from 1 to 6. In our research, static features extracted from PE files are selected based on the broad range of characteristics of the executable files. Section 3.2 covers the details on the features selected in this research.

CHAPTER 3

Implementation

In this chapter, we will give a brief introduction of our dataset and the malware families in our research. Parts of the features of malware are introduced in this chapter. Related machines learning algorithms, such as Support Vector Machine, Cosine Similarity, and concept drift, are covered at the end of this chapter.

3.1 Dataset

The dataset used in this research contains of Windows executable files. There are 1419 malware families in total, and each of the family can spread across different time windows. We select 15 malware families due to the availability of the sample size. Table 1 shows the number of samples of each family in our dataset. These malware families belong to a variety of malware categories, such as worm, virus, trojan, etc. A brief discussion of each of families is as follows.

VBInject is a malware targeting at Windows System. Encryption and compression methods are utilized to prevent from detection. Another malware is concealed inside it, so detection will become hard. Generally, the symptom is mild if a system is infected by this malware. An alert may be sent out from Windows security software [18].

Winwebsec reports malicious software by pretending scanning the computer. It persuades users to pay for some illegal software by claiming infections on the system [19].

Renos makes the system download unwanted software automatically, and convince users to purchase service in order to remove the malicious software. It may cause unstable of the system [20].

OnLineGames a is a huge malware family. The goal is to steal the private information from oneline game players [21].

BHO, short for Browser Helper Objects, is the add-ons for Microsoft browser. Most of the time, users install the add-ons without clear notification from distributors. The toolbars contain different links to sites with competing resources [22].

Startpage changes the start page of internet explorer [23].

Adload has the ability to open a backdoor of a system and install unwanted software on the system. It belongs to Trojan family [24].

VB is a malware family targeting windows system. It takes control of infected computers and causes the computer functions slow [25].

FakeRean is a very severe malware family. The malware pretend to scan the infected computer. It will trick the user to pay fee by declaring malware on the infected computer. [26].

Lolyda.BF sends users information of the infected computer to its remote server. Once the computer is infected, it can also monitor the activity of the computer [27].

Vobfus is a kind of worm, which can be downloaded by other malware. Removable devices are also resources to help it spread. The malware must be run on computers by users before it it is activated [28].

CeeInject is obfuscated, so it can hide its purpose to avoid detection. This malware can have any purpose to hide in the computer. One example is obfuscating in a Bitcoin mining client. The infected computer mines bitcoin without user's knowledge [29].

Zbot, a specific Trojan virus, steals sensitive financial information from infected computers. Several attacks can launch, such as Man-in-the-browser attacks, keystroke logging from grabbing, CryptoLocker ransomware attacks, and so on [30].

Allaple.A is a specific worm virus. It is a network worm that can spread to the computers within local network. It may conduct denial-of-service attack on the target websites [31].

Table 1: Malware family list

No	Family	Sample Size
1	VBInject	1302
2	Winwebsec	1078
3	Renos	788
4	OnLineGames	712
5	BHO	666
6	Startpage	653
7	Adload	590
8	VB	516
9	FakeRean	494
10	Vobfus	538
11	CeeInject	498
12	Lolyda.BF	505
13	Zbot	486
14	Allaple.A	470
15	Agent	476

Agent belongs to Trojan virus family. It can change the configuration setting of Windows Explorers [32].

3.2 Features

In this paper, the dataset is in the format of portable executable(PE) file. The PE format is used for windows operating systems, which contains a variety of information. The purpose of introduction of PE file is to have the same file format for all version of Windows, such as Windows NT, Windows95, etc. [33] Windows operating system utilizes information in the PE file to manage the executable code. PE file contains headers and sections. Figure 1 shows the basic PE file structure. Each file contains DOS MZ header, DOS stub, PE header, Section table, and a number of distinct sections [34]. Header and sections are two parts in the PE file. Information about PE files are located in the header part, while details about executable are located in the section part. The header can be broken into small parts, which are DOS header,

PE header, optional header, and section table. PE header contains information about the PE structure. The optional header consists of variety of information, such as the checksum of the file, address of entry point, data directories, etc. The description of these parts are as follows.

- PE header.The information about the PE structure are located here.
- The optional header. It consists of variety of information, such as the checksum of the file, address of entry point, data directories, etc.
- Section table. Section table summarizes information of all sections in the PE file. It provides essential information about the PE files, such as the location of the section, and the name of section, section flags, etc.

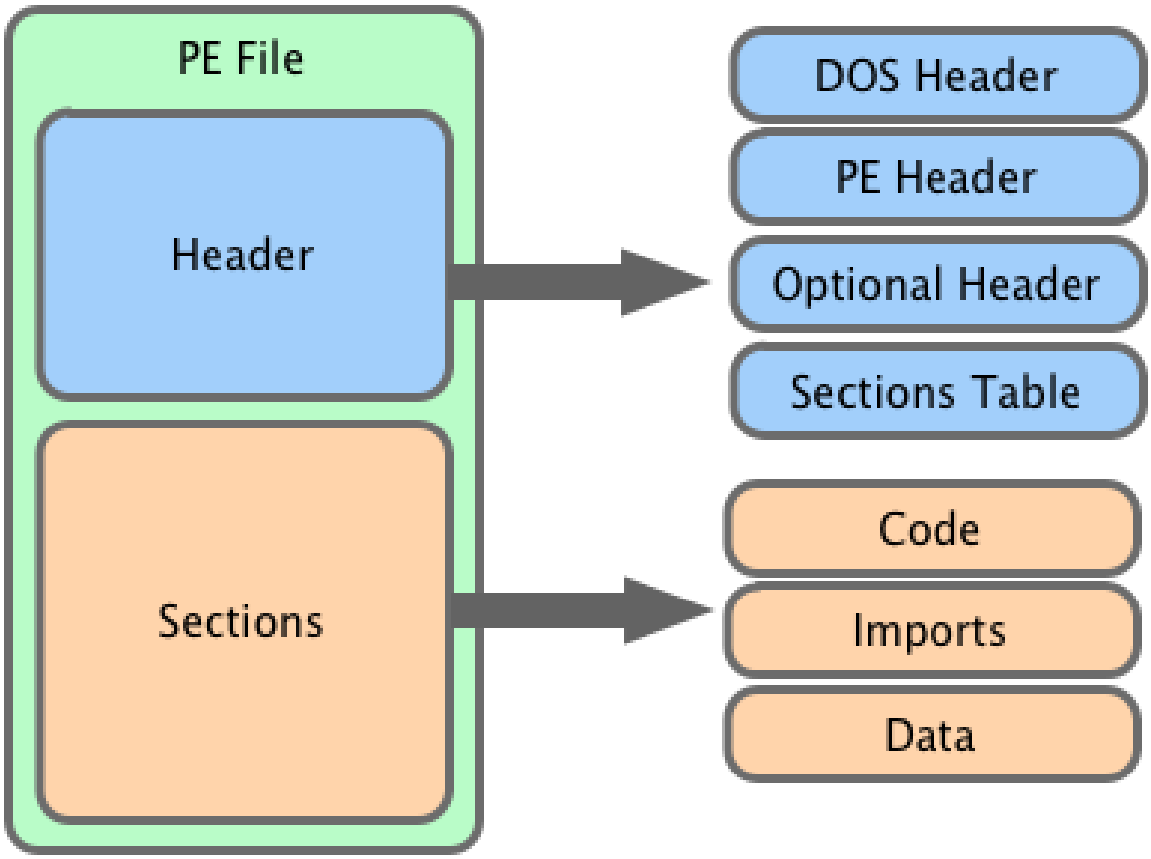

Figure 1: PE file structure.

Table 2: Features selected from PE file

No	Features	No	Features
1	Machine	2	SizeOfOptionalHeader
3	Characteristics	4	TimeDateStamp
5	MajorLinkerVersion	6	MinorLinkerVersion
7	SizeOfCode	8	SizeOfInitializedData
9	SizeOfUninitializedData	10	AddressOfEntryPoint
11	BaseOfCode	12	BaseOfData
13	ImageBase	14	SectionAlignment
15	FileAlignment	16	MajorOperatingSystemVersion
17	MinorOperatingSystemVersion	18	MajorImageVersion
19	MinorImageVersion	20	MajorSubsystemVersion
21	MinorSubsystemVersion	22	SizeOfImage
23	SizeOfHeaders	24	CheckSum
25	Subsystem	26	DllCharacteristics
27	SizeOfStackReserve	28	SizeOfStackCommit
29	SizeOfHeapReserve	30	SizeOfHeapCommit
31	LoaderFlags	32	NumberOfRvaAndSizes
33	SectionsNb	34	SectionsMeanEntropy
35	SectionsMinEntropy	36	SectionsMaxEntropy
37	SectionsMeanRawsize	38	SectionsMinRawsize
39	SectionMaxRawsize	40	SectionsMeanVirtualsize
41	SectionsMinVirtualsize	42	SectionMaxVirtualsize
43	ImportsNbDLL	44	ImportsNb
45	ImportsNbOrdinal	46	ExportNb
47	ResourcesNb	48	ResourcesMeanEntropy
49	ResourcesMinEntropy	50	ResourcesMaxEntropy
51	ResourcesMeanSize	52	ResourcesMinSize
53	ResourcesMaxSize	54	LoadConfigurationSize
55	VersionInformationSize		

In our research, we extract features from each of PE file. These features are selected based on the research from [5], where 55 features are selected. The detailed information of the features is shown in the Table 2.

3.3 Support Vector Machine

Support vector machine, abbreviated as SVM, is a very popular supervised machine learning algorithm that can be used to solve classification and regression problems. However, it is used more widely in classification problem than in regression problem. The purpose of applying a SVM algorithm is to find a hyperplane, which can distinctly separate the data points in multidimensional space [35]. The hyperplane is chosen that it can maximize the margin of a dataset. The margin here refers to the minimum distance between the hyperplane and any data point of the dataset. Another important feature in SVM is "Kernel trick" [35], which allows users to separate data points by projecting them into higher dimensional space. The trick may refer to no remarkable penalty is paid by working in a higher dimensional space. Three kernel functions has been widely used, which are linear kernel, polynomial kernel, and Gaussian kernel.

Figure 2 is an example of linear SVM classification. Two classes are denoted by red circles and green squares. The optional hyperplane is shown with a solid line. The support vectors are the red circle and green square on the dash lines. The distance between the two dash lines are margin. Our goal is maximize the margin to get our optional hyperplane.

In our research, 55 features are extracted from a large number of Windows portable executable (PE) files. Linear SVM models are trained on one year size of windows of malware samples. For samples within each window, the two classes of SVM are obtained in a following way. One class, defined as "+1", is the samples at the current month, while the other class, defined as "-1", is the samples within a specific time window. A series of feature weights from linear SVM models are obtained. We use these feature weight to calculate cosine similarity shown in Chapter 4.

According to paper [36], scaling is very important to train a SVM. There are two

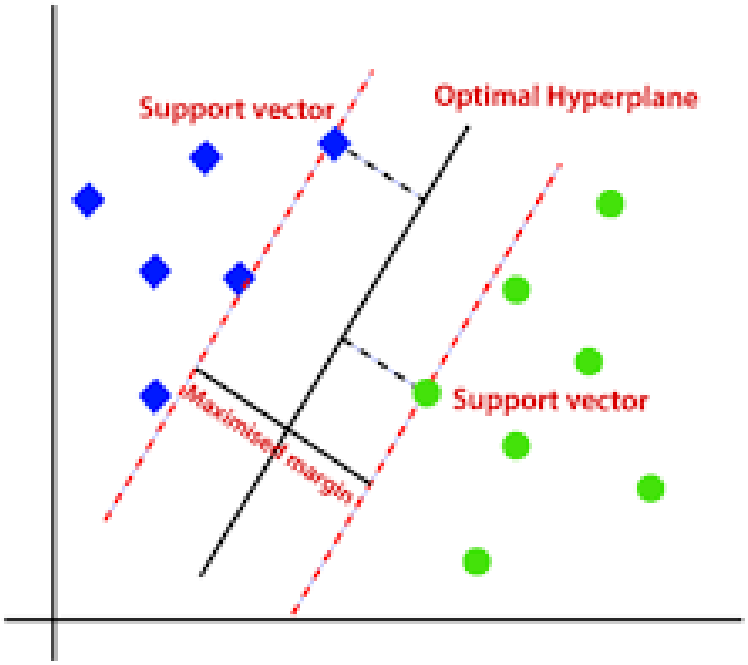

Figure 2: Linear SVM.

advantages for normalizing the data. First, it can avoid the situation that attributes in large scales dominate those in relatively smaller scales. Second, it can increase the efficiency during the calculation. In this research, it takes hours to train a linear SVM with 55 features of 300 data samples before normalizing the data. After normalization, it takes less than 1 minutes to train a linear SVM on the same data set.

3.4 Cosine Similarity

Cosine similarity is a similarity measurement to find the cosine of two non-zero vectors. The cosine similarity is calculated as

$$S(X,Y) = cos(\theta) = \frac{\sum_{i=0}^{n-1} X_i Y_i}{\sqrt{\sum_{i=0}^{n-1} X_i^2}\sqrt{\sum_{i=0}^{n-1} Y_i^2}} \tag{1}$$

where S(X,Y) stands for the cosine similarity between two non-zero vectors, X and Y. The value of cosine similarity is between -1 and 1. If two vector have a value of cosine similarity is 1, it means that two vectors have the same orientation. If the

value of cosine similarity is 0, it means two vectors are perpendicular to each other. If the value of cosine similarity is -1, it means two vector are in the opposite direction. Figure 3 illustrates the cosine similarity between two vectors. The two vectors are in the same direction and the angle between these two vectors are around 30 degrees. Therefore, we can say the two vectors has around 86% similarity based on the cosine angles.

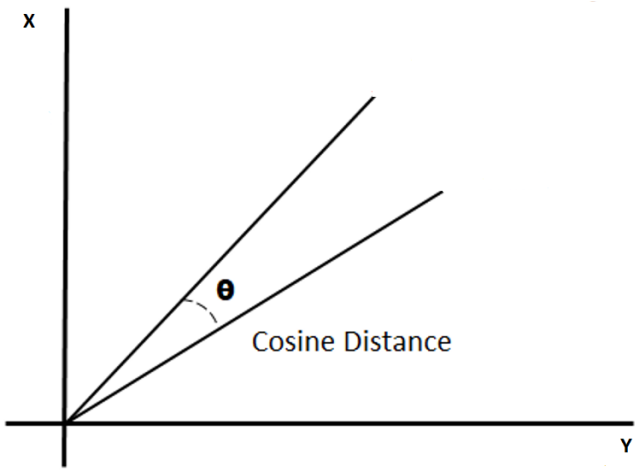

Figure 3: Cosine Similarity.

In the research, we use cosine similarity plot to detect if there is any significant change in a specific malware family. The cosine similarity is calculated from feature coefficients of linear SVM models. For example, two linear SVM models are fitted on two data samples. The cosine similarity of these two data samples are calculated based on the coefficients of these two linear SVM models. Cosine similarity plot is drawn based on all the results from a series of cosine similarity values. If there is a spike in the plot, it may indicate a significant change at this point for a malware family.

3.5 Concept Drift

In machine learning, concept drift is a vital component to be considered to ensure the accuracy of a machine learning algorithm in the long run. Concept drift denotes that the relationship between the independent variables and predictors changes over time under in unforeseen circumstances [37, 38]. An example of concept drift is online shopping data. Online companies collect customer data, such as their preferences, their browsing history, to help the companies better understand about customers' behavior and to make prediction for the future sales. However, over time, customer behavior and preferences may change. The old models built based on the unchanged data could not reflect the changes. Therefore, the accuracy of the models decreases. The model should be able to adapt to these concept changes if better prediction would be achieved [39].

The definition of concept drift can be illustrated using probability. Suppose $X = \{x_1, x_2, x_3....x_n\}$ is feature vectors, y is the label. Given time t_0, t_1, there exists X, so that,

$$P_{t_0}(X, y) \neq P_{t_1}(X, y) \tag{2}$$

Where P is the joint probability of feature vectors X and labels y.

Equation 2 [9, 40] is the same as

$$P_{t_0}(X) * P_{t_0}(y/X) \neq P_{t_1}(X) * P_{t_1}(y/X) \tag{3}$$

There are two types of concept drift, real concept drift and virtual concept drift [9]. The changes in $P(y/X)$ can be viewed as real concept drift. In real concept drift, it does not matter that $P(X)$ changes or not. Virtual concept is the changes in $P(X)$ while $P(y/X)$ stay the same. Figure 4 shows a binary classification, and green and black dots represent one class, respectively. For real concept drift, the boundary of two classes changes, while the boundary stays the same for virtual drift.

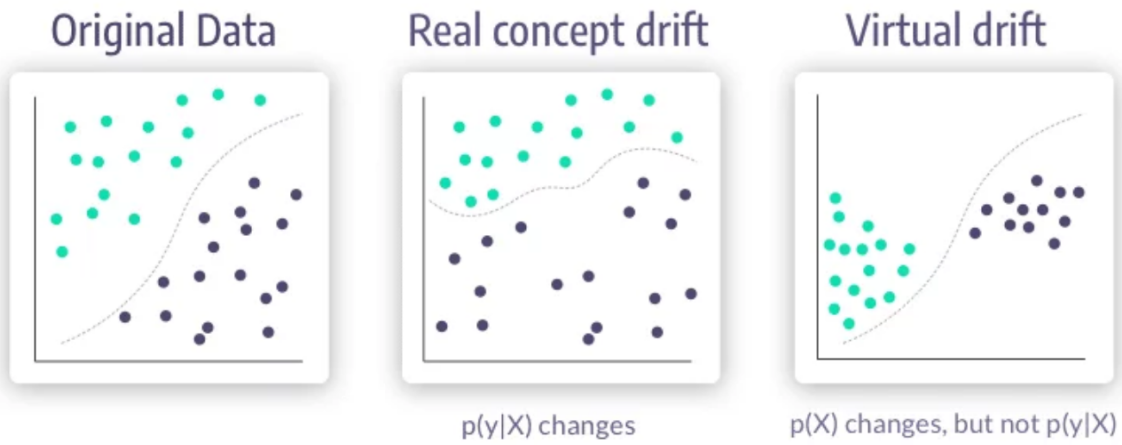

Figure 4: Real Concept Drift vs Virtual Concept Drift.

Understanding concept drift mostly refers to answering three questions, "when", "How", and "where" on the drift information [10]. To be specific, we need to understand when the concept drift happened, how to measure the level of concept drift, and locate the region where it happened. All of the these questions can be answered from the drift detection algorithms. It is crucial to determine when the drift occurs. PCA drift detection [41] is implemented. The drift alarm will be triggered when large p-value of Wilcoxon test is obtained. Paper [10] summarizes a series of algorithms, such as DDM [42],STEMPD [39], to detect the drifts. The level of drift can be quantified by measuring similarity of the two concepts, current one and previous one.

In this research, we will detect when the evolution happens using cosine similarity for a specific malware family. The cosine similarity values will be tested to identify the level of drift.

CHAPTER 4

Experiments and Results

In this chapter, we discuss the experiments and results on the malware evolution. We will break into two sections in this chapter. In Section 4.1, we perform concept drift detection on Juxtaposed two families. While in Section 4.2, concept drift detection is conducted on three individual malware family separately. In total, there are four families used in the experiments, VBInject, Winwebsec, CeeInject and Renos. Table 3 shows the sample size and the spread period for the four malware families mentioned above. We delete some of the samples by removing duplicate and unreasonable samples for all these four families. For example, some sample of Winwebsec family are from February, 2038. We remove all these data before applying our machine learning algorithms.

4.1 Juxtaposed Malware Families

We juxtapose two malware families, VBInject family and Winwebsec family. There are two reasons that these two families are chosen. First, the sample size of these two families are the largest ones in our dataset. Second, the timestamp of the samples from these two families overlaps between 2010 and 2012. By pretending these samples are from one family, we use one year as sliding window to fit a linear SVM. To be specific, starting from the first family, we choose samples at the current month as one class, marked as "+1", the samples one year before this month are chosen as

Table 3: Sample information

Family	Size	Time period
VBInject	1302	September 2010 - December 2012
Winwebsec	1078	May 2011 - Feburary 2012
CeeInject	498	April 2009 - January 2012
Renos	788	June 1992 - September 2009

the other class, marked as "-1". The window keeps sliding down until it reach the end of this family. Then we gradually add one month of the second family until it reach the end of the dataset. Cosine similarity is calculated based on the coefficients from the linear SVM models.

Figure 5 shows the plot of cosine similarity for the families, VBInject and Winwebsec. The cosine similarity is approaching -0.3 when we start adding a new family. Although it shows one month before adding the new family, it is still reasonable. When we calculate the cosine similarity, we compare the coefficients of the current month with the ones of previous month. Therefore, we move one month ahead when computing the cosine similarity. It is another big spike, which shows above 0.6 of the value of cosine similarity. This is the place where the samples size of VBInject family almost 0. It makes sense because the dataset are doninmant by Winwebsec family at the point. These points indicate big changes occurs. We could use these points to update our models in the next section. It also gives us confidence that the pike on the cosine similarity could be a big changes with one specific malware family.

Machine learning algorithm, SVM, is selected to investigate and verify our observations from the similarity plot. The timestamp is ignored to get a better understanding about our models. We concatenate two samples from VBInject and Winwebsec, and label the samples manually as one class,"-1". Then samples from CeeInject is added, and tag the samples as another class, "+1". The big spike we are interesting in is the join of the two families. We compared fixed models versus retrained models. For fixed model, an initial point is chosen. SVM is trained using the samples before the initial point and tested using those after initial point. For retrained models, second point is selected around the big spike. Two models are trained. For the first model, training data are the same as the fixed model mentioned above. Test data are the samples between the initial point and second point. For

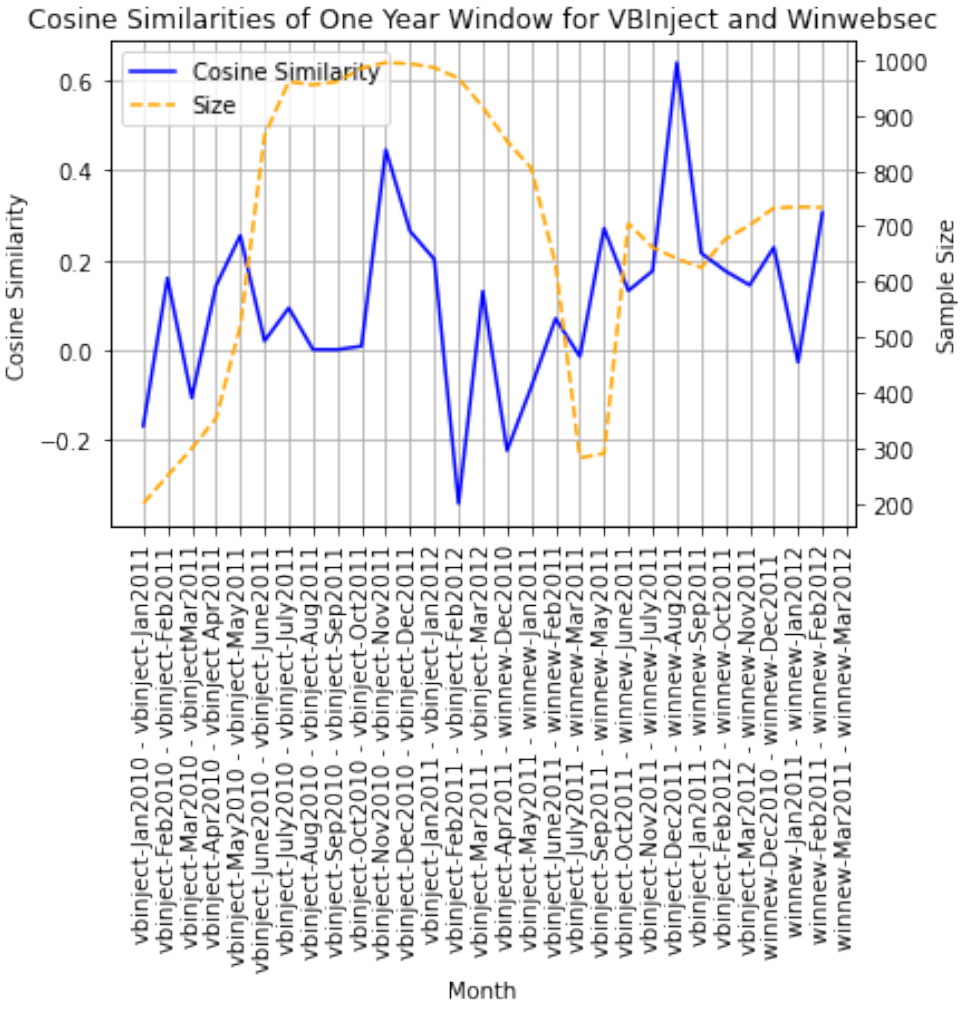

Figure 5: Cosine similarity for VBInject and Winwebsec

the second model, the training data are the same as the test data in the first model. The rest samples after the second point are used as test data. The comparison of the accuracy from three models is displayed in Figure 6. By retraining models, the accuracy increases from 59% to 82%. The experiment proves our idea that the big similarity change could be a indicator of malware evolution at a specific period. We will apply the idea in each individual malware family next.

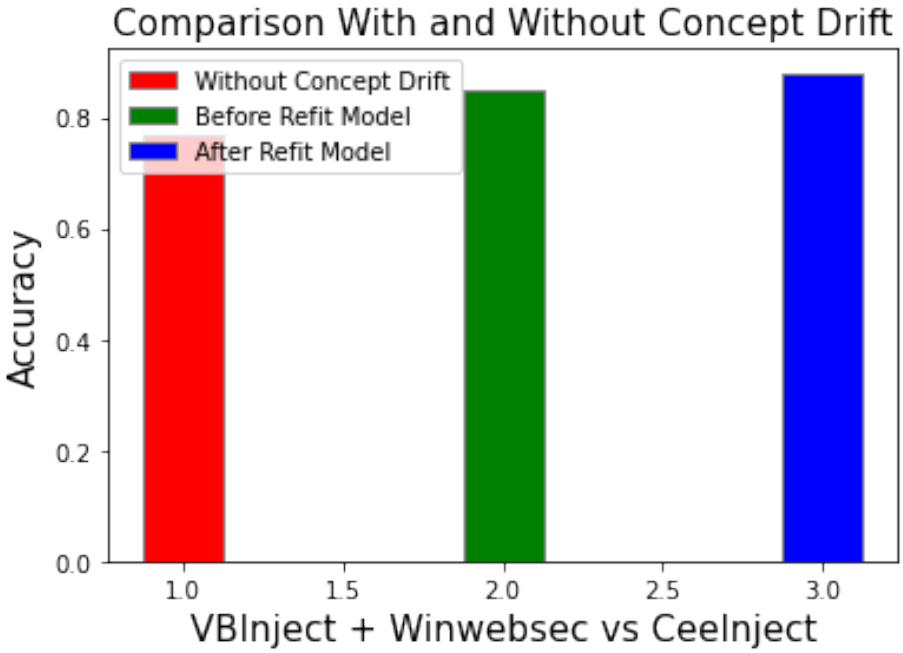

Figure 6: Model Updating for Juxtaposed Families.

4.2 One Family Drift Detection

According to [43], drift detection is the approaches to identify changes time points or intervals and to compute the changes. Researchers[10] has summaries four stages to detect the concept drift, which are retrieving data, modeling data, test statistics calculation and hypothesis testing. In our research, three steps have been done to detect the drift for each of selected malware family. The three steps are data analysis, drift measurement, and drift investigation. First step, we analyze the samples from all families, such as size, features. We choose the families that have relatively large size to perform the detection. For each selected family, further analysis and feature selection have been done. We observe that the samples are not evenly distributed among the months. The second step is model fitting. Linear svm is implemented because the weights of coefficients can be used to determine the importance of the feature. The third step is to quantify the level of the drift using cosine similarity. The

larger value of cosine similarity, the smaller difference between the feature weights from two linear SVM. We explore machine learning algorithm, such as SVM, to verify our observations in the last step.

4.2.1 Winwebsec Experiments

In this section, we discuss the experiments and results performed on the Winwebsec malware. There are 1078 Winwebsec samples spanned from 2011 to 2012. We follow three steps to detect the drift and verify the drift.

The first step is data analysis. Figure 7 shows the distribution of Winwebsec samples based on their time stamp. This gives us the general view on the samples and provide insights for analysis in the next step. The samples are not uniformly distributed within the timeline.

Drift measurement is the second step. Here, we conduct analysis on the 55 features selected, which are listed in Table 2. As discussed in Section 3.3, starting from the first month of the samples, one year window is chosen as sliding window. Samples within the window are treated as one class. The samples in the month after the one year window are treated as the other class. Then we implement linear SVM and obtain the feature coefficients from the model. The same procedure repeats with the one year window sliding forward until it reaches the end of the samples. The cosine similarities are computed on the basis of the feature coefficients acquired from each model. Figure 8 gives us the overall cosine similarity among feature coefficients from all linear SVM models. We observe that there is a big spike located in August 2011. The value of cosine similarity jump from approximate 0 in June, 2011 to around 0.7 in August 2011. Then it quickly decreases to around 0 level. That may indicate some changes occurred at this particular period. Further investigate needs to be done to confirm our observations. We also notice a small bump at the time of January

2012. The value of cosine similarity shifts from 0.1 to 0.3.

The last step is drift investigation. The purpose of this step is to verify if there is drift by updating models on the points. SVM models is chosen by comparing the accuracy of the models. In the experiment, we perform binary classification on two malware population. Samples from VBInject are chose as one class, while samples from Winwebsec are the other class. If the data from stationary malware population, the accuracy of models with updating and without updating for binary classification would have no big difference. In other words, if the evolution causes change in malware population, retraining models would increase the accuracy of binary classification. First, we train the models on the initial starting point and use the model to test on the samples after this point. This model is called the fix model. Then we retrain the model at the big spike, shown in Figure 8. The spike is treated as one big evolution. That is, the samples from June 2011 to September 2011 are considered as one big evolution. Then we tested the model using data after the training set. This model is called model with drift. Figure 9 is the accuracy comparison from two models, fix model and one with drift. The accuracy increase from 51% to 78%. There is a considerable big increase with respect to model accuracy after retraining model. For the second experiment, we retrain model twice on two parts of spike. That is, one model is retrained with data from June 2011 to August 2011, and test the model on samples before second spike. Then retrain model with data from previous training set, and test with the rest of sample. The reason why training twice is because there are two big jumps with respect to cosine similarity on the spike. One is increase of cosine similarity about 0.6, while the other is decrease of cosine similarity about 0.5. That may indicate two big changes in malware population. The accuracy of three different models are shown in Figure 10. Compared to the first experiment, retraining on two parts of spike increases the accuracy. Furthermore, it may indicate concept drift with

cosine similarity shifting about 0.5. The third experiment is on the small bump. The cosine similarity changes about 0.2. The results is shown in Figure 11. Retraining model does not improve the model accuracy. In this case, we conclude that it is not a good indicator of concept drift with values of cosine similarity shifting around 0.2.

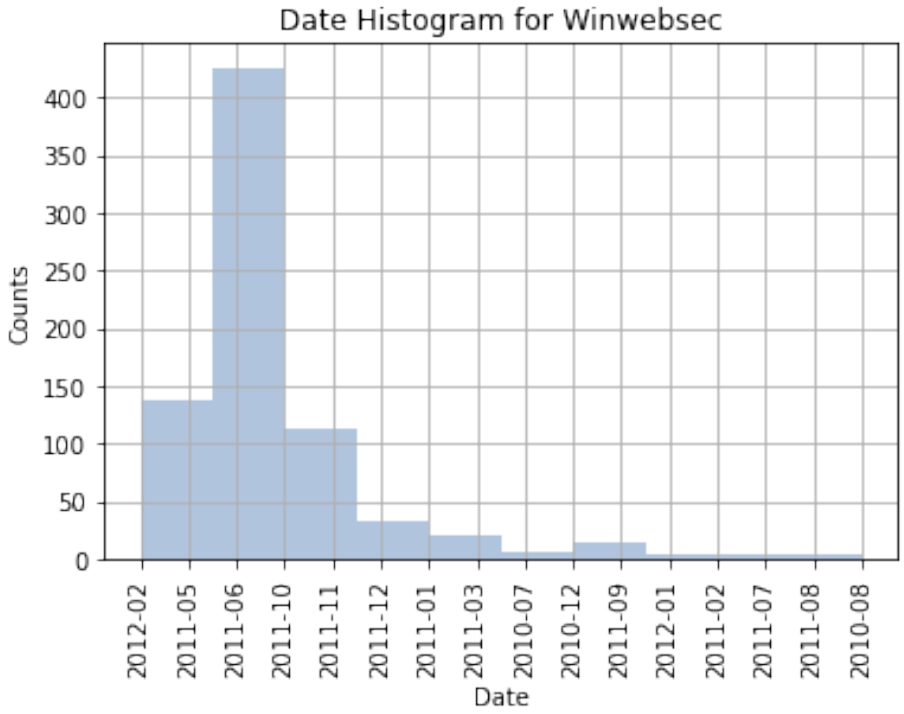

Figure 7: Date Histogram for Winwebsec family.

Figure 9 shows that there is a significant improvement after retraining models at the evolution point, which is the big spike in Figure 8. However, we only use part of samples from Winwebsec family for the retraining model. It would be fair to compare the models by taking the size into account. Figure 12 shows that the accuracy of retraining model increases to 99% if we compare the fixed model with the same size.

4.2.2 VBInject Experiments

In Section 4.2.1, we have performed three steps to detect and verify concept drift. In this section, the similar steps have been done as in Section 4.2.1. We perform

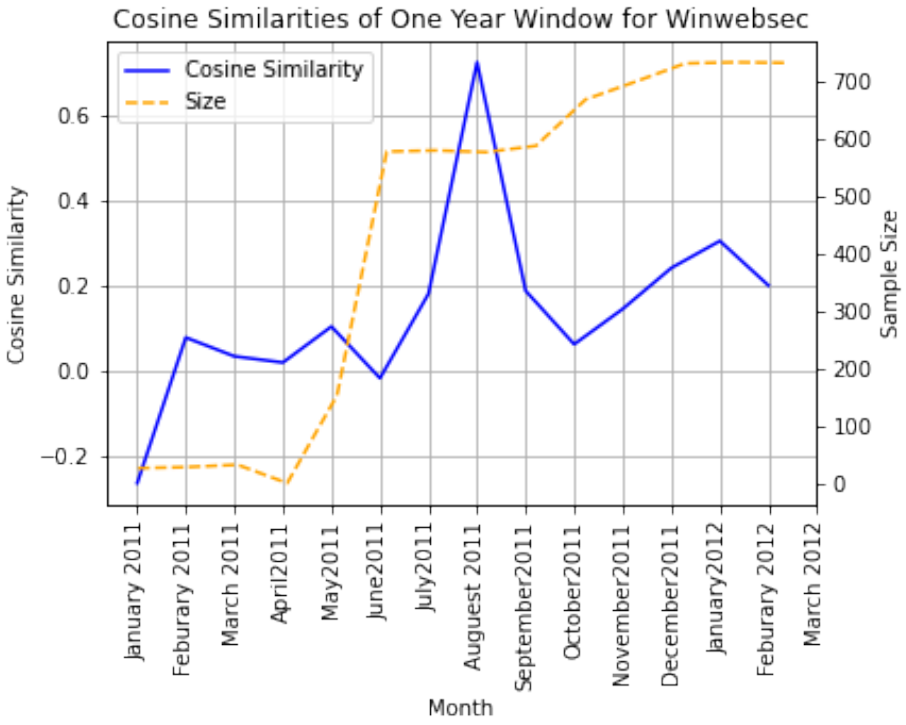

Figure 8: Cosine Similarity for Winwebsec Family.

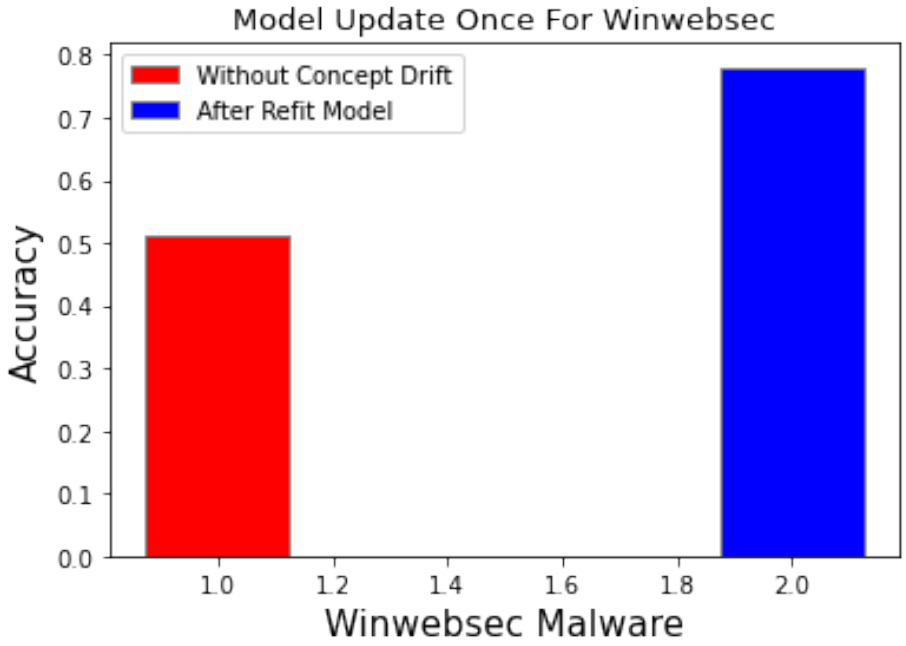

Figure 9: Model Updating Once For Winwebsec.

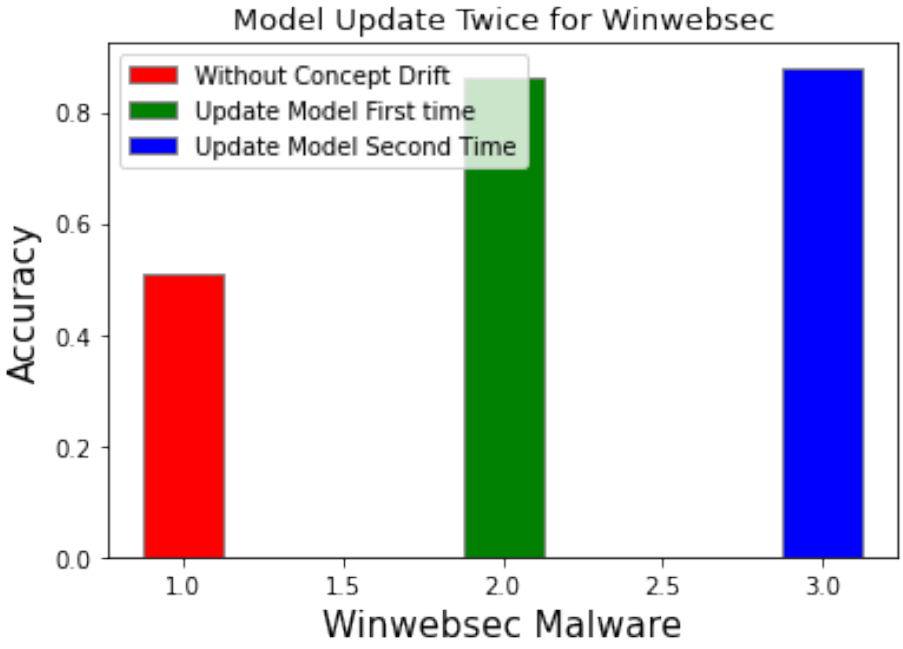

Figure 10: Model Updating Twice For Winwebsec.

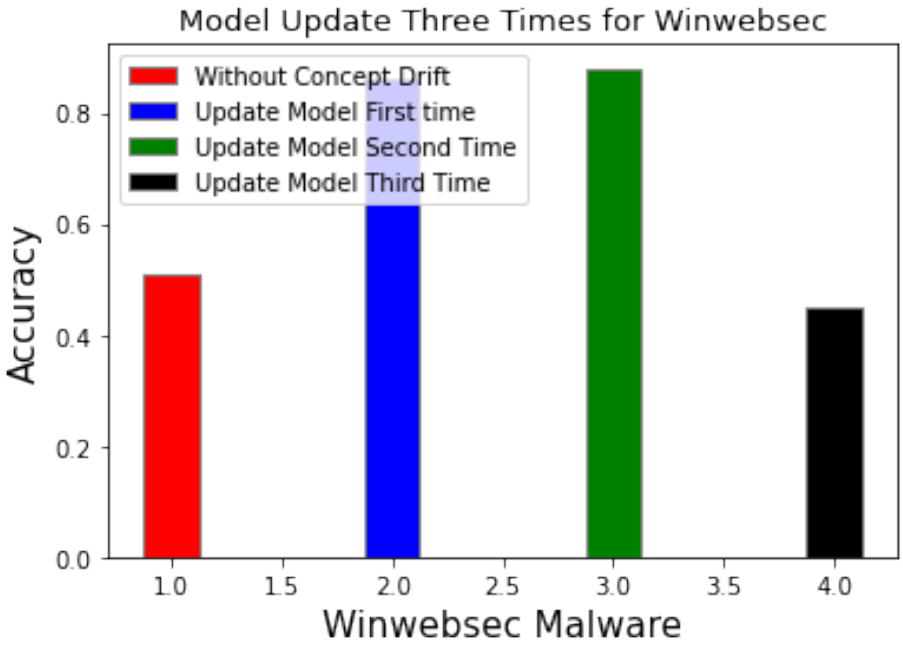

Figure 11: Model Updating Three Times For Winwebsec.

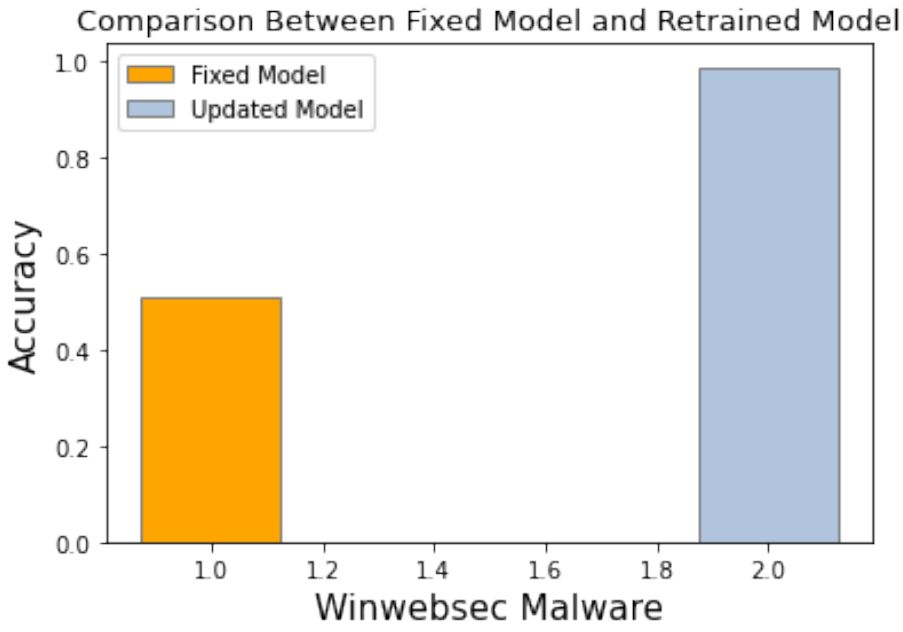

Figure 12: Comparison Between Fixed Model and Retraining Model for Winwebsec.

preliminary analysis on VBInject malware samples, and remove the inaccurate samples. Figure 13 shows the distribution of VBInject samples. The malware samples are not uniformly distributed. The size of samples in June 2011 and February 2011 are extremely larger compare with those in January 2010 and March 2012. When training the samples, extra attention is needed on these samples. Cosine similarity for VBInject family is displayed in Figure 14. There are three obvious spikes that need to be investigate, one is around May 2011, the second one is around November 2011, and the third one is around November 2011. Figure 15 shows the comparison of model accuracy before and after refitting model using data on the big spike. To be specific, we retrain SVM model using data from October 2011 to December 2011, and test the model using data after December 2011. The model accuracy increases from 51% to 94% with retraining model. We also update model twice before and after big spike, the accuracy increase from 90% to 98% shown in Figure 16. However, it may be not worth updating model twice in that case. There are two reasons. First, it would be

time consuming to retrain models in such a short period with a large volume of data. Second, accuracy is not improved dramatically. Figure in Appendix C are the results from retraining models on small spike, the data we used are from April 2011 to June 2011. In this case, it depends on the threshold defined by the experiment. If the value of 0.3 cosine similarity is chosen to retrain model, we do not need to retrain model here.

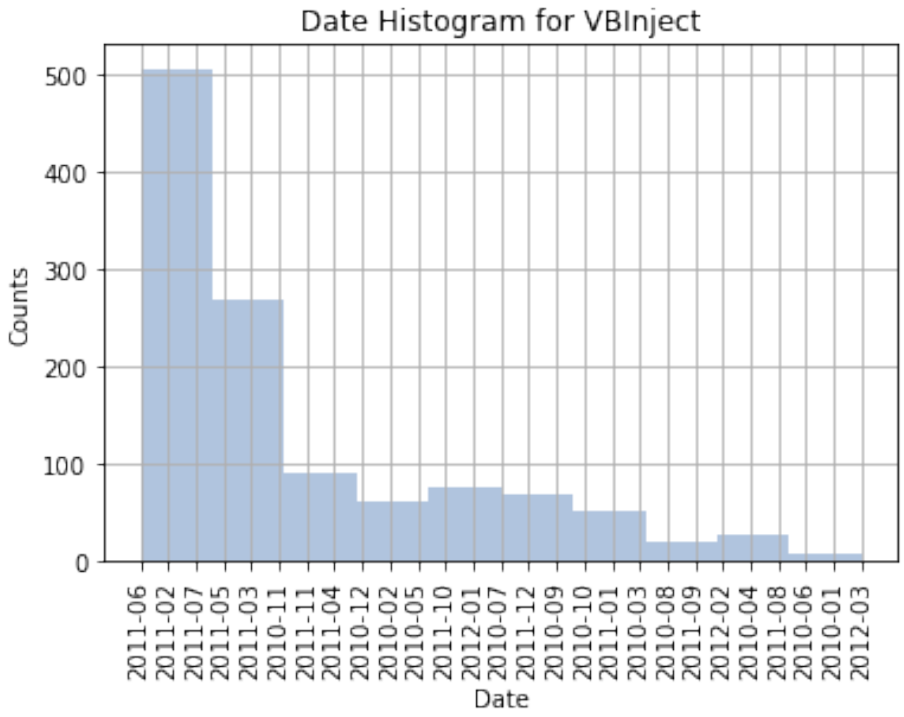

Figure 13: Date Histogram for VBInject family.

4.2.3 CeeInject Experiments

In this section, similar experiments have been done as in Section 4.2.1 and Section 4.2.2. Histogram in Figure 17 plot shows the distribution of CeeInject samples are extremely uneven. The sample size in most of months are below 10. There are one relatively big jump shown in Figure 18. Cosine similarity jumps from 0 to 0.3. There is no big difference in accuracy by retraining model compared with the fixed model

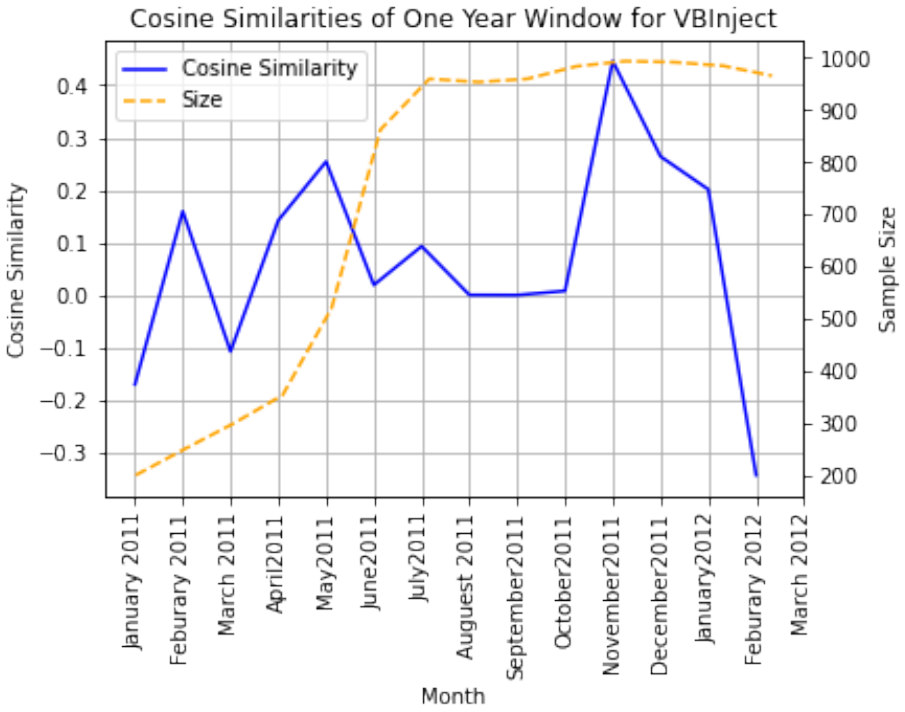

Figure 14: Cosine Similarity for VBInject Family.

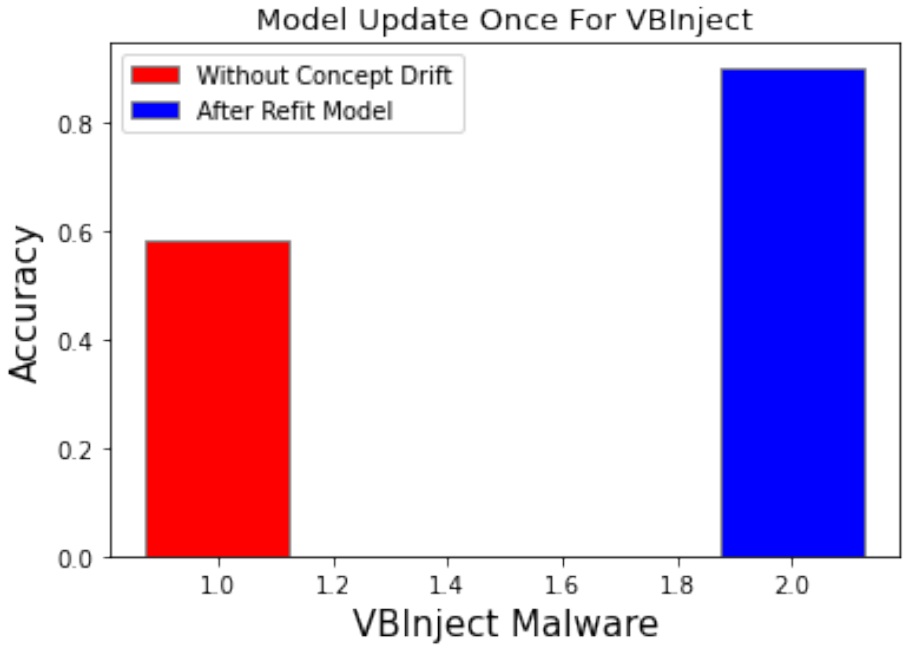

Figure 15: Model Updating Once For VBInject.

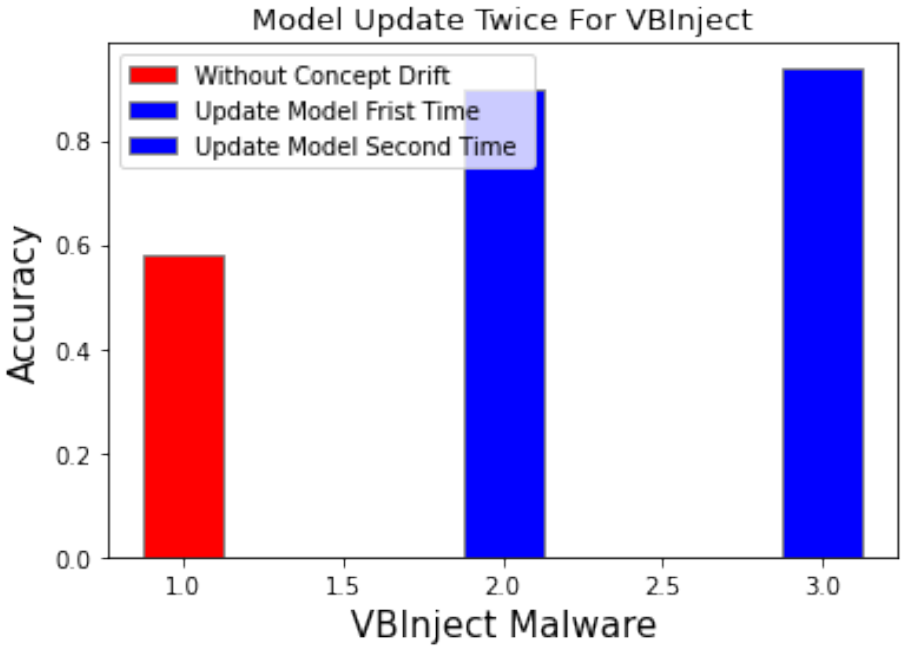

Figure 16: Model Updating Once On Small Spike For VBInject.

on the spike. The result is illustrated in Figure 19. We also test on retraining model with samples from May 2011 to July 2011. The reason why this period is chosen is because there is a relatively big change in cosine similarity, which is around 0.3. The accuracy increases about 0.7%, as shown in Figure 20. As discussed in Section 4.2.2, if the threshold for updating model set as 0.3, it may indicate drift occurs at this point. Further research can be done on how to define the threshold with more data involved.

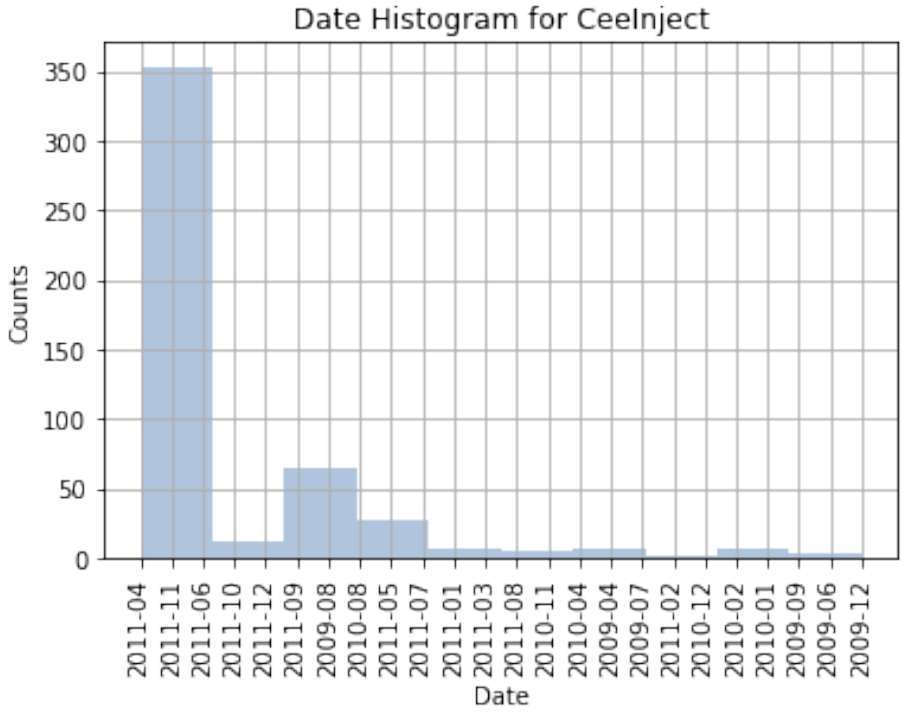

Figure 17: Date Histogram for CeeInject family.

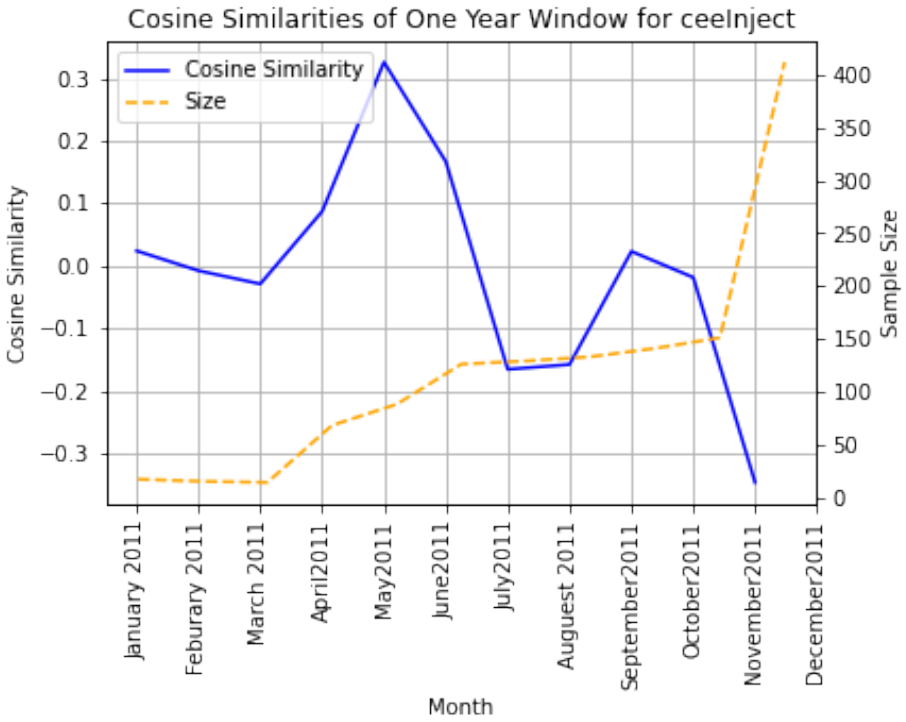

Figure 18: Cosine Similarity for CeeInject Family.

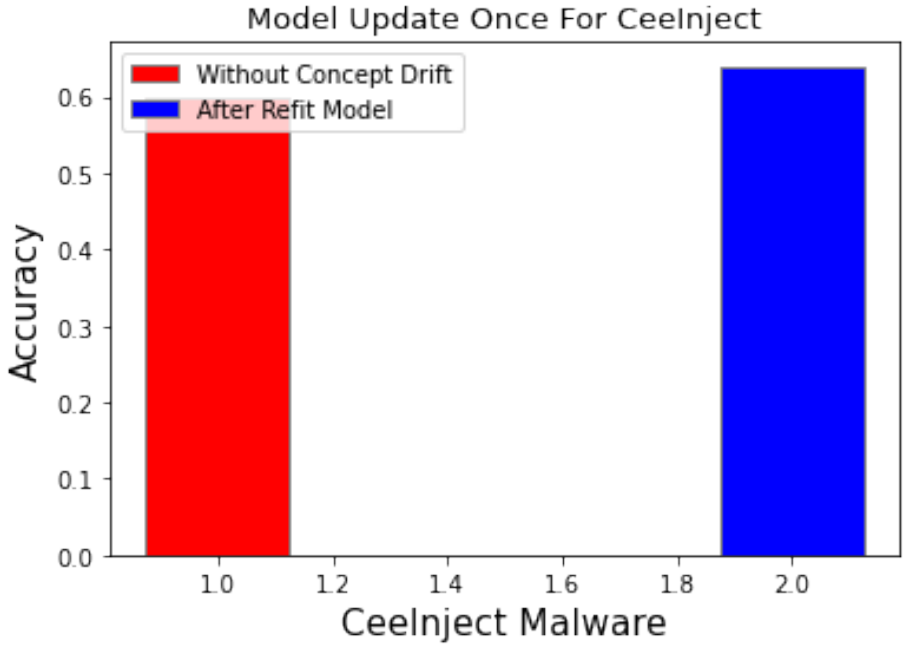

Figure 19: Model Updating Once For CeeInject.

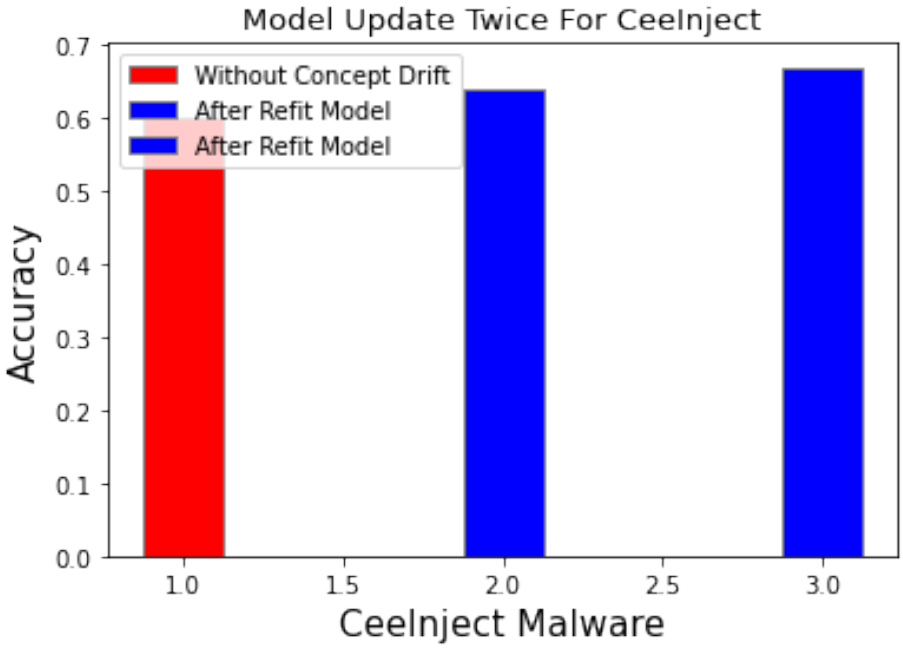

Figure 20: Model Updating Twice For CeeInject.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

In this study, we have used cosine similarity and machine learning algorithm on static features extracted from PE files to detect concept drift. Three steps are implemented to complete our goals. First step is data preliminary analysis. Histogram is used to generate a general view of the distribution of malware samples. The second step is measure the level of concept drift. Linear SVM and cosine similarity are implemented. We generate graphs of cosine similarity for each family selected. The third step is to verify the drift. We select SVM and experiment on different kernels to investigate and verify the drift. We start our experiment by juxtaposing two malware families. It confirms our idea that there is drift on the boundary of two families. Retraining models increase the model accuracy. Then we apply to 3 individual malware family, and for all of these family, we found that drift has at different levels. It worth investigating if there is a change of cosine similarity with value over 0.3. It would be a indicator of drift among malware population.

5.2 Future Work

Our experiments focus on 55 features selected from PE files. We have done experiments to rank the important features based on coefficient weights of linear SVM. Parts of the results are shown in Appendix B. Tracking individual feature or features with higher weights would be likely produce additional insights into the evolution of malware family.

Our study of malware detection is based on static feature extract from PE files. Other static features, such as opcode n-grams, byte n-grams would be other optional approaches to be considered. In addition, dynamic features may also be helpful to detect the concept drift.

For the measurement of drift, cosine similarity is selected to measure the level of concept drift in this study. Other measurement of dissimilarity or distance measurement can also be explored. Chi-square, Jaccard Distance, or Euclidean distance can also be explored. This may provide interesting understanding of concept drift.

We explore SVM machine learning algorithm to investigate the drift detected in our study. Further research can be done using many other machine learning algorithms. For example, random forest, logistic regression, k-Nearest neighbors can also applied to malware evolution problem. Feature selection methods can also be explored in the future research. Ideally, framework can be built to capture the evolution and retrain the model automatically, so better prediction for future malware evolution can be reached.

LIST OF REFERENCES

[1] J. Aycock, *Computer Viruses and Malware*, 01 2006, vol. 22.

[2] ''A not-so-common cold: Malware statistics in 2022,'' https://dataprot.net/statistics/malware-statistics/, 03 2022, (Accessed on 04/10/2022).

[3] M. Stamp, *Information Security: Principles and Practice*, 04 2011.

[4] A. Walenstein, M. Venable, M. Hayes, C. Thompson, and A. Lakhotia, ''Exploiting similarity between variants to defeat malware "vilo" method for comparing and searching binary programs,'' in *Proceedings of BlackHat DC*, 2007.

[5] M. Wadkar, F. Di Troia, and M. Stamp, ''Detecting malware evolution using support vector machines,'' *Expert Systems with Applications*, vol. 143, p. 113022, 10 2019.

[6] 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*, 2009, pp. 1--10.

[7] S. Paul and M. Stamp, ''Word embedding techniques for malware evolution detection,'' 03 2021.

[8] L. Tupadha and M. Stamp, ''Machine learning for malware evolution detection,'' 07 2021.

[9] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, ''A survey on concept drift adaptation,'' *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1--37, 2014.

[10] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, ''Learning under concept drift: A review,'' *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, pp. 1--1, 10 2018.

[11] A. Gupta, P. Kuppili, A. Akella, and P. Barford, ''An empirical study of malware evolution,'' 02 2009, pp. 1 -- 10.

[12] 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, p. e1978, 08 2018.

[13] A. Singh, A. Walenstein, and A. Lakhotia, ''Tracking concept drift in malware families,'' in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, 2012, pp. 81--92.

[14] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. Tygar, ''Approaches to adversarial drift,'' in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, 2013, pp. 99--110.

[15] G. Yan, N. Brown, and D. Kong, ''Exploring discriminatory features for automated malware classification,'' in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 41--61.

[16] Z. Fuyong and Z. Tiezhu, ''Malware detection and classification based on n-grams attribute similarity,'' in *2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, vol. 1. IEEE, 2017, pp. 793--796.

[17] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, ''Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey,'' *information security technical report*, vol. 14, no. 1, pp. 16--29, 2009.

[18] ''Virtool:win32/vbinject,'' https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/VBInject%26ThreatID=-2147367171, 11 2010, (Accessed on 04/02/2022).

[19] ''Win32/winwebsec,'' https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Winwebsec&threatId=, 08 2010, (Accessed on 04/02/2022).

[20] ''Win32/renos,'' https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/VBInject%26ThreatID=-2147367171, 04 2007, (Accessed on 04/02/2022).

[21] ''Trojan-psw:w32/onlinegames,'' https://www.f-secure.com/v-descs/trojan-psw_w32_onlinegames.shtml, (Accessed on 04/10/2022).

[22] ''Browser hijack objects (bhos),'' https://blog.malwarebytes.com/threats/browser-hijack-objects-bhos/, 06 2016, (Accessed on 12/10/2021).

[23] ''Trojan.startpage,'' https://blog.malwarebytes.com/detections/trojan-startpage/, (Accessed on 12/10/2021).

[24] ''Trojan.adload,'' https://blog.malwarebytes.com/detections/trojan-adload/, (Accessed on 12/10/2021).

[25] W. Woodham, ''Trojan:win32/vb (vb trojan) — virus removal guide,'' https://howtofix.guide/trojan-win32-vb/, (Accessed on 12/10/2021).

[26] ‘‘Win32/fakerean,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FFakeRean, 02 2011, (Accessed on 04/03/2022).

[27] ‘‘Pws:win32/lolyda.bf,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FLolyda.BF, 02 2011, (Accessed on 04/03/2022).

[28] ‘‘Win32/vobfus,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fvobfus, 03 2010, (Accessed on 12/10/2021).

[29] ‘‘Virtool:win32/ceeinject,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject, 11 2007, (Accessed on 12/10/2021).

[30] ‘‘Zeus trojan (zbot),’’ https://www.hypr.com/zeus-trojan-zbot/, (Accessed on 12/10/2021).

[31] ‘‘Worm:win32/allaple.a,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=worm:win32/allaple.a&ThreatID=2147574777, 06 2007, (Accessed on 12/10/2021).

[32] ‘‘Trojan:w32/agent,’’ https://www.f-secure.com/v-descs/agent.shtml, (Accessed on 12/10/2021).

[33] ‘‘An in-depth look into the win32 portable executable file format,’’ https://docs.microsoft.com/en-us/archive/msdn-magazine/2002/february/inside-windows-win32-portable-executable-file-format-in-detail, 10 2019, (Accessed on 12/10/2021).

[34] ‘‘Pe format,’’ https://docs.microsoft.com/en-us/windows/win32/debug/pe-format, 11 2021, (Accessed on 12/10/2021).

[35] M. Stamp, *Information Security: Principles and Practice*, 2010.

[36] C.-w. Hsu, C.-c. Chang, and C.-J. Lin, ‘‘A practical guide to support vector classification chih-wei hsu, chih-chung chang, and chih-jen lin,’’ 11 2003.

[37] G. Widmer and M. Kubat, ‘‘Learning in the presence of concept drift and hidden contexts,’’ *Machine learning*, vol. 23, no. 1, pp. 69--101, 1996.

[38] N. Lu, G. Zhang, and J. Lu, ‘‘Concept drift detection via competence models,’’ *Artificial Intelligence*, vol. 209, pp. 11--28, 2014.

[39] R. Klinkenberg and T. Joachims, ‘‘Detecting concept drift with support vector machines.’’ in *ICML*, 2000, pp. 487--494.

[40] V. Losing, B. Hammer, and H. Wersing, ''Knn classifier with self adjusting memory for heterogeneous concept drift,'' in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291--300.

[41] J. Shao, Z. Ahmadi, and S. Kramer, ''Prototype-based learning on concept-drifting data streams,'' in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 412--421.

[42] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, ''Learning with drift detection,'' in *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286--295.

[43] M. Basseville, I. V. Nikiforov, *et al.*, *Detection of abrupt changes: theory and application*. prentice Hall Englewood Cliffs, 1993, vol. 104.

APPENDIX A

Juxtaposed Malware Family

We also juxtapose CeeInject and VBInject families to test our idea in Section 4.1. The results is shown in Figure A.21.

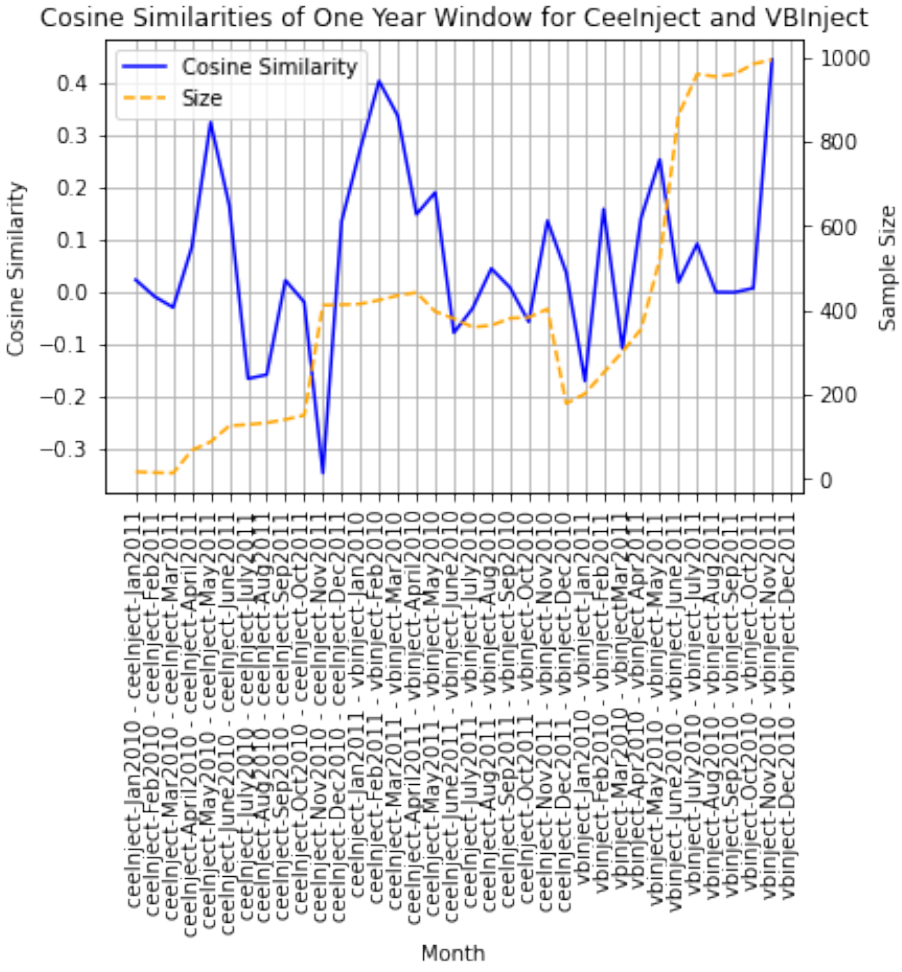

Figure A.21: Cosine Similarity for CeeInject and VBInject.

APPENDIX B

Important Features

Parts of feature importance based on the linear SVM coefficients is listed as Figure B.22, Figure B.23, Figure B.24. The samples are from VBInject malware used in this research.

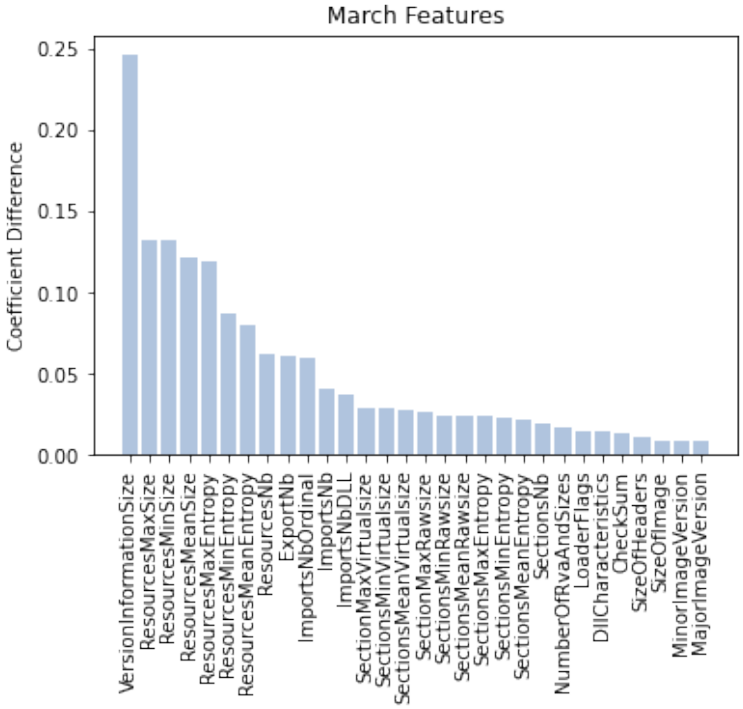

Figure B.22: important features on March

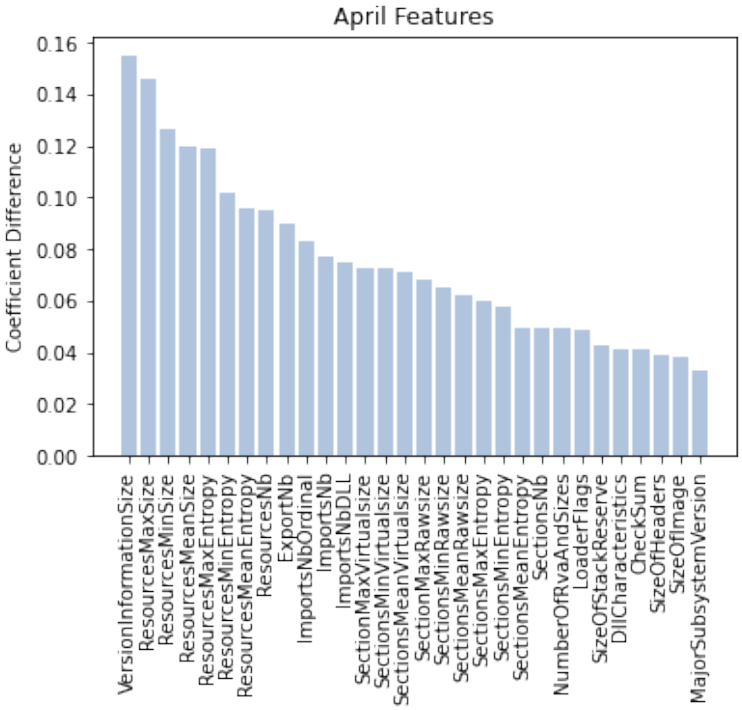

Figure B.23: important features on April

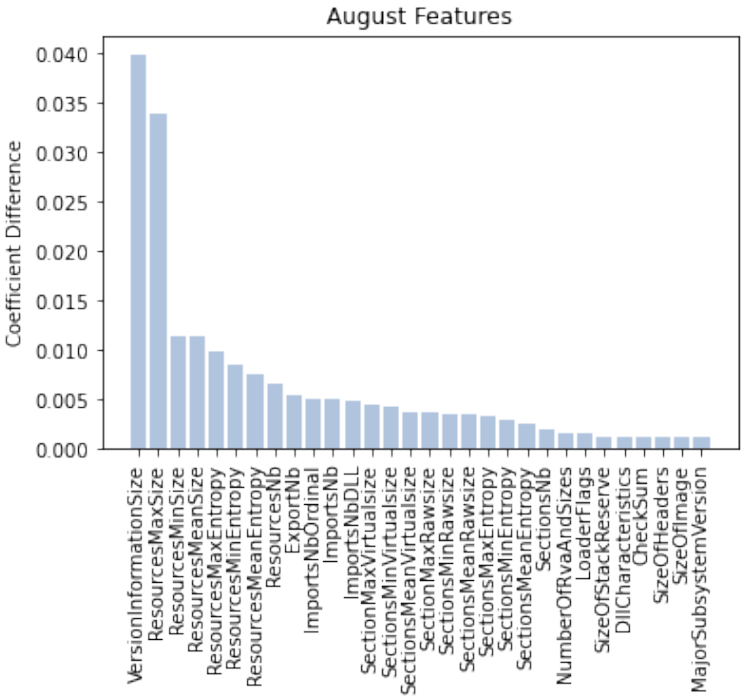

Figure B.24: important features on August

APPENDIX C

Results from Retraining Models

C.0.1 Juxtaposed Families Experiments

Experiments have also been done on three families, VBInJect, Winwebsec, and Renos to test our idea in Section 4.1. Results is shown in Figure C.25.

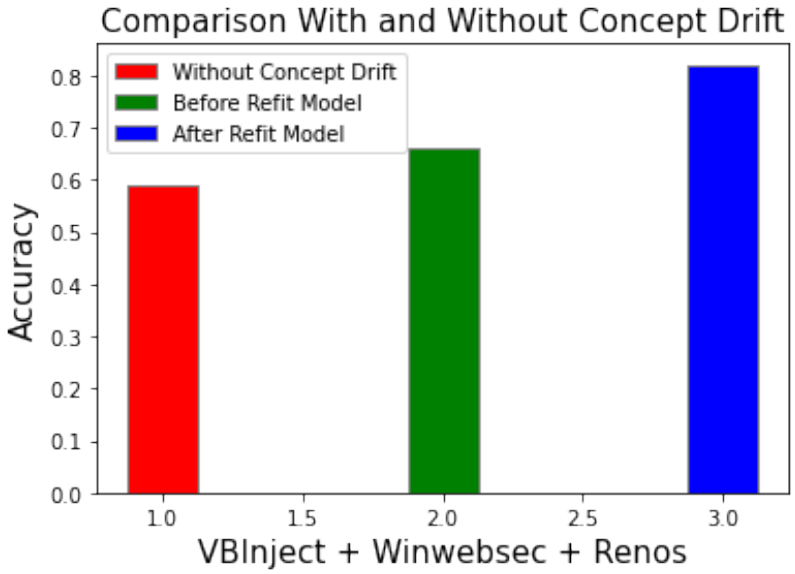

Figure C.25: Model Updating Once For Juxtaposed Families.

C.0.2 VBInject Experiments

Part of the results is displayed in Figure C.28 and Figure C.27.

C.0.3 CeeInject Experiments

Figure C.28 are the results using samples from April 2011 to July 2011.

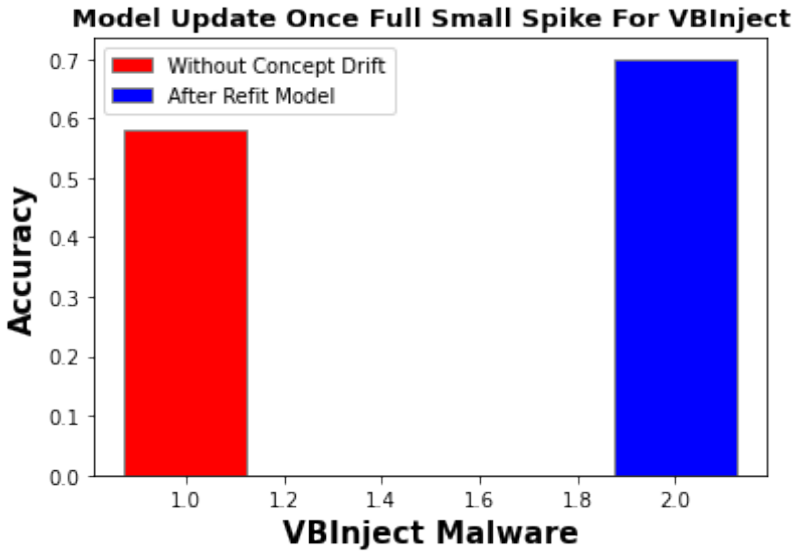

Figure C.26: Model Updating Twice For VBInject.

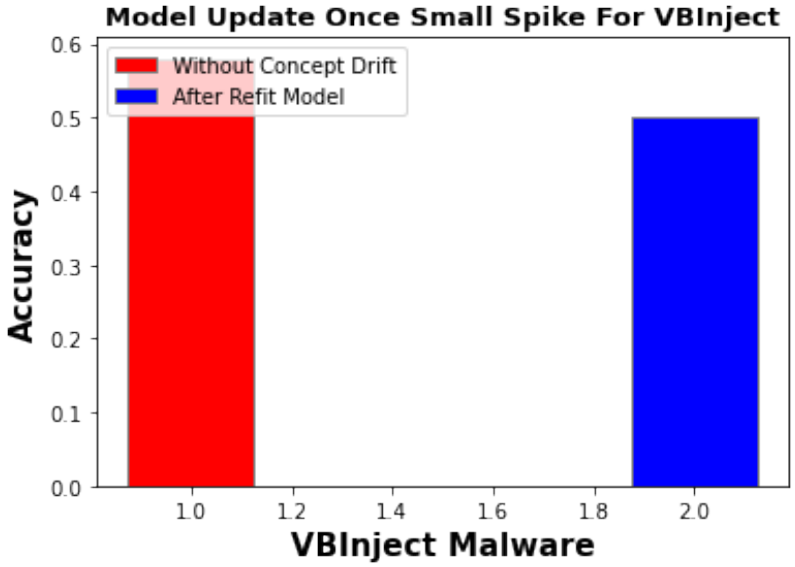

Figure C.27: Model Updating Once On Small Spike For VBInject.

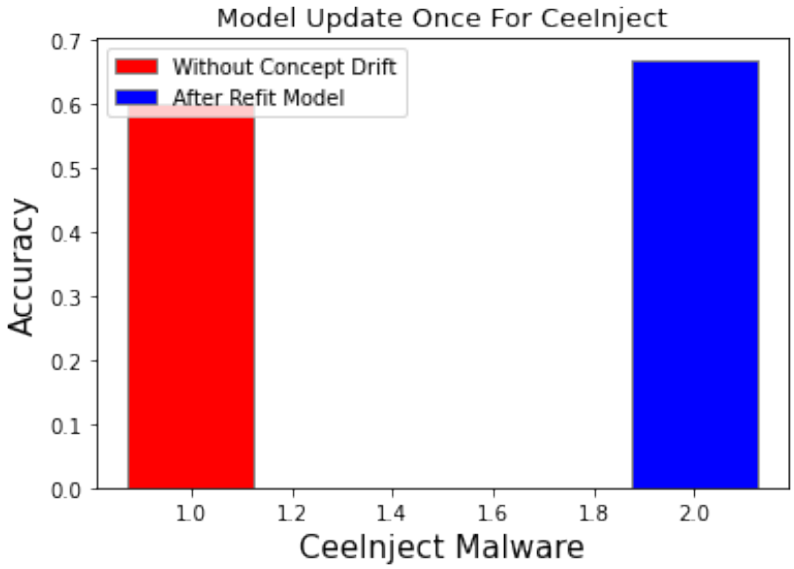

Figure C.28: Model Updating Once For CeeInject.