Tracking Concept Drift in Malware Families

Anshuman Singh
Center for Advanced Computer Studies
University of Louisiana at Lafayette
asingh@louisiana.edu

Andrew Walenstein
School of Computing and Informatics
University of Louisiana at Lafayette
walenste@ieee.org

Arun Lakhotia
Center for Advanced Computer Studies
University of Louisiana at Lafayette
arun@louisiana.edu

ABSTRACT

The previous efforts in the use of machine learning for malware detection have assumed that malware population is stationary i.e. probability distribution of the observed characteristics (features) of malware populations don't change over time. In this paper, we investigate this assumption for malware families as populations. Malware, by design, constantly evolves so as to defeat detection. Evolution in malware may lead to a nonstationary malware population. The problem of nonstationary populations has been called concept drift in machine learning. Tracking concept drift is critical to the successful application of ML based methods for malware detection. If the evolution causes the malware population to drift rapidly then frequent retraining of classifiers may be required to prevent degradation in performance. On the other hand, if the drift is found to be negligible, then ML based methods are robust for such populations for long periods of time.

We propose two measures for tracking concept drift in malware families when feature sets are very large–relative temporal similarity and metafeatures. We illustrate the use of the proposed measures with a study on 3500+ samples from three families of x86 malware, spanning over 5 years. The results of the study show negligible drift in mnemonic 2-grams extracted from unpacked versions of the samples. The measures can likewise be applied to track drift in any number of malware families. Tracking drift in this manner also provides a novel method for feature type selection, i.e., use the feature type that drifts the least.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Invasive software (e.g., viruses, worms, Trojan horses)

Keywords

Concept Drift, Malware, Temporal Similarity, Metafeatures

AISec'12, October 19, 2012, Raleigh, North Carolina, USA.

1. INTRODUCTION

Recently machine learning (ML) based methods have have been proposed for developing malware detectors [35, 1, 24, 29, 31]. ML-based methods promise to improve upon the state-of-the-art by being less laborious. Frequently AV analysts manually analyze new malware to determine their unique characteristics and then handcraft methods for detection. By contrast, ML-based detectors are trained using a collection of malware samples and the system automatically learns the characteristics that distinguish each sample (or a class).

Though ML-based methods are promising, their success is predicated on an important assumption that may not hold when applied to malware. They assume that the training data is sampled from a stationary population. In other words, they assume that the data source (rather, the probability distribution of their observed characteristics) does not change over time. Malware does not fit this profile. The entire population of malware is constantly evolving either in response to external pressures—new technologies and new detectors—or due to internal pressures—the need for new capabilities. The evolution of malware in response to new detectors has a direct impact on many observed characteristics. This raises questions about whether a collection of malware identified today may be a representative of malware that may be generated tomorrow. The previous efforts in the use of ML in malware detection have assumed that malware population is stationary. Since malware is known to evolve in order to evade detection, this assumption needs investigation.

The term "concept drift" has been used in ML literature to describe a non-stationary population [39, 38]. Concept drifts are classified as either "real" or "virtual". In real concept drift, there is a change in statistical distribution of data as well as change in concept. In contrast, a virtual concept drift, as is the case for spam and malware, the distribution of data may vary, but not the concept. Many methods have been proposed in ML literature to detect and track drift as well as adapt the classifier to drift [38].

In this paper, we propose two measures to track drift in static features of malware. Static features were chosen, instead of dynamic features, because the predominant methods currently used by malware authors to prevent detection of their products is to scramble their static properties, which in turn is because historically malware detectors have significantly relied on static features. Hence, in the current generation of malware we expect drift, if it exists, to be more pronounced in static features. Drift can be tracked

or monitored by tracking each feature individually and then observing an overall trend. Although, static features like n-grams are easy and efficient to extract, they are usually very large in number, even after feature selection, thus making it infeasible to track each one of them individually. We solve this problem by using two types of feature summary measures—relative temporal similarity and metafeatures.

We then apply the proposed methods to track concept drift in real world malware populations. The findings of this type of study are critical to the successful application of ML-based methods for malware detection. If the evolution causes the malware population to drift rapidly then frequent retraining of classifiers may be required to prevent degradation in performance. On the other hand, if the drift is found to be negligible, then ML-based methods may be robust for such populations so long as training data poisoning attacks like boiling frog attacks [34] do not happen. Tracking drift in malware can also give clues to the kind of evolution in malware that causes the observed drift.

Studying drift in malware poses a challenge – what constitutes a malware population? There is no predefined notion of what constitutes a malware. While there are classes of malware, such as, virus, worms, Trojans, backdoors, etc. none of these classes define a population in the sense that a random sample of it can be used for statistical analysis. We resolve this issue by considering a malware "family" as a population. The notion of malware family is used by the anti-malware companies to define a collection of samples that share some characteristics, such as, sharing certain code base. Though the term "malware family" is quite fuzzy, in that two malware samples may not always be placed in the same family by two different AV analyst, yet such grouping is used in the industry for sharing data and information about malware. GData Software recently reported that the number of malware families have remained relatively constant over the years where as the number of variants within a family has been growing rapidly [14]. This report suggests that it may be prudent to study concept drift within malware families.

Besides studying drift in individual malware families, we also narrow the population further to unpacked malware. This constraint was guided by prudence. Most packed malware encrypt their payload and randomly vary the key. Since encryption results in statistically garbled data it would be counterproductive to look for pattern or drift in it. However, once the code is decrypted it would be reasonable to expect some pattern that is retained across variants.

The results of our study show negligible drift in mnemonic 2-grams extracted from 3500+ unpacked malware samples belonging to three real world malware families and spanning over 5 years. There were three pieces of evidence indicating negligible drift in these families. First, we found that the similarity (cosine similarity) of later variants in a family with respect to a randomly picked early variant did not change significantly (Figure 2). Second, we found that some metafeatures for instruction mnemonic characterization were stable in samples over time (Figure 3). Third, results from retraining experiments show that a retrained classifer had almost the same accuracy as the original classifier (Table 6). We would like to point here that the negligible drift observed in our study is not indicative of negligible drift in most real world malware in general. Our results may have been influenced by the limited dataset and the choice of features. However, the study does give a trend in the chosen malware families.

On the surface the results may appear counter-intuitive since malware has been believed to be evolving rapidly to evade various detection methods. However, it appears from the results of our study that the families we investigated use two independent evolution mechanisms–evolution of the protection layer, a la due to the packer and evolution of the behavior, a la the actual malicious logic. If one can remove or peek through the protection layer, one can access malware code whose statistical properties do not vary signficantly over time. Our study also indicates that ML classifiers trained on static features extracted out of samples from such families need not be trained frequently without any drop in performance.

Though detection of novel classes in malware data streams has been studied before [28], to the best of our knowledge this is the first work on tracking of concept drift in malware. The main contributions of this paper are:

- It proposes a similarity based method for tracking concept drift in malware.
- It proposes to track metafeatures to solve the problem of tracking large number of static features in malware.
- It presents results of tracking concept drift in three real world malware families.
- It studies the relationship between concept drift in malware and malware evolution.

The rest of the paper is organized as follows. Section 2 gives some backgound on concept drift and an overview of related works. Section 3 discusses different types of evolution in malware and their possible impact on concept drift. Section 4 describes our proposed methods for tracking drift in malware. Section 5 presents our empirical study of concept drift in real world malware families. Section 6 discusses some limitations of our proposed method and our empirical study and also gives some directions for future work. Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

ML-based works on malware detection have not considered the applicability of some of the assumptions of the ML method to malware. In the following we describe the context in which one of these assumptions appear and then explain the notion of concept drift and related works.

Supervised machine learning methods have their origin in statistical classification [11]. *Statistical classification* is a special type of statistical estimation problem where the conditional probability distribution function $p(C|X)$ is estimated for all classes C given the data attributes are expressed as a vector X. *Statistical estimation* itself is concerned with generalizing about the population from a sample [19]. For example, if a population is of size N, then a sample $S = (s_1, s_2, \ldots, s_n)$, where $n << N$, can be used to estimate, within an error bound, the mean and variance of a Gaussian population. This is an example of parametric estimation where the type of the distribution function $f(X; \bar{\lambda})$ (e.g. Gaussian) is assumed and the problem is reduced to finding the parameters $\bar{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ (e.g.

mean,variance) of the distribution. In contrast, nonparametric estimation makes no assumption about the type of probability distribution function.

A basic assumption in statistical estimation is that the data is sampled from a stationary population—a population that does not change over time. In supervised classification, if the population of one or more classes changes over time, estimation should be done again using a new sample (training data) to update the conditional distribution $p(C|X)$ with respect to the changed populations of classes. In machine learning literature, the stationarity assumption of statistics is stated as part of the IID assumption. IID means that data is independent and identically distributed. The second part of the IID assumption–identically distributed data– means that all the data is coming from the same population.

The problem of nonstationary populations has been called *concept drift* in machine learning literature. Concept drift is defined as any change in the conditional probablity distribution $p(C|X)$ over time [21, 41]. From Bayes theorem,

$$p(C|X) = \frac{p(X|C)p(C)}{p(X)}$$

Hence, concept drift can result from change in $p(X|C)$ or $p(C)$. Sometimes, change in $p(X|C)$ may not affect class membership due, for example, to symmetric drift in opposite directions. Hence, change in $p(X|C)$ is also called virtual drift. Since, change in class priors $p(C)$ alone may change $p(C|X)$, any observable change in $p(C|X)$ is called real drift.

Drift can be gradual or sudden. Different methods have been proposed in literature to handle both kinds of drift [38]. In general, there are two types of approaches to handle concept drift [13]:

- **Direct adaptive**. The classifier adapts at regular intervals of time without verifying the occurrence of concept drift. These methods include fixed size time windows on training data and instance weighting [38, 22].
- **Detect and adapt**. The concept drift is detected by monitoring or tracking certain indicators of drift. If the drift is detected, then the classifier adapts according the extent and direction of the drift. There are three types of indicators for monitoring or tracking concept drift [23]:
  - *Properties or features of data*: Some properties or individual features of data are tracked over time to detect drift. The scale and direction of change can be determined easily based on the values of features [21, 9].
  - *Classifier parameters*: Some classifiers like Support Vector Machines have parameters that vary with drifting concepts. These parameters can be monitored over time to detect concept drift [9].
  - *Classifier performance measures*: Various classifier performance measures like accuracy, precision, recall etc. can be monitored w.r.t training windows to detect drift. A significant decrease in the performance is an indicator of concept drift.

Early work on concept drift was theoretical in nature and based on computational learning theory [25, 18]. Widmer [39] described concept drift in a non-theoretical setting and proposed window-based adaptive methods for retraining to handle concept drift. The impact of concept drift on the performance of classifiers was studied by Kelly et al. [21]. Kelly et al. tracked individual features over time to empirically detect concept drift. More rigorous approaches for detecting concept drift using test statistics exist in statistics literature [12, 16, 9].

There has been some work on handling concept drift in spam. Delany et al. [8, 7, 6] describe a lazy-learning based approach to handle concept drift in spam. Fdez-Riverola et al. [10] improve Delany et al.'s work by introducing a term relevance score to track concept drift. These works use direct adaptive approaches to handle drift. An empirical or statistical study of properties of spam and spam features directly showing the occurrence of concept drift is absent in these works.

Studies on classification of executables using machine learning [35, 1, 24, 29, 31] have used static datasets. Works on malware clustering have also used static datasets, and have not taken into account the temporal changes in characteristics of malware [2, 3]. Malware phylogeny generation [20] has also been atemporal in nature. Masud et al. [28] proposed a new ensemble based method to detect malware in evolving data streams. Evolving data streams are data streams in which new concepts may appear over time. Concept evolution in data streams may occur in addition to and independent of concept drift.

To the best of our knowledge, there has not been any work on monitoring concept drift in malware domain. We believe montoring or tracking concept drift can give significant insights into evolution of malware. These insights can be helpful in determining the right type of features for detecting drifting malware.

3. MALWARE EVOLUTION AND IMPACT ON CONCEPT DRIFT

Malware, like any other software, evolves due to changes in the environment in which it survives and operates. Many of these evolutionary changes occur to avoid detection by AV scanners [32]. The technology used by AV scanners usually exerts pressure at certain points in malware to evolve and evade detection. Evolution could also be due to changes in the environment in which malware is created. Broadly speaking malware evolution, as impacting the byte sequences in a binary, may be classified into three types: natural, environmental, and polymorphic. We describe each one of these types and their possible impact on concept drift.

3.1 Natural evolution

Natural evolution in malware occurs due to changes resulting from adding features, making bug fixes, porting to a new environment, as such. The changes in the binary are a reflection of changes in the code made by a programmer. Variations introduced in a binary due to natural evolution are expected to be introduced relatively infrequently. That is, successive versions of binaries are expected to show high similarity, except when the code base undergoes significant refactoring. Similarly, the similarity of an early version of a binary with that of future version is expected to degrade slowly. Hayes et al. [17] argued that malware can be expected to to evolve by gradual accumulation of capabilities (the so-called "creeping feature" problem in software [26])

and through a model of "punctuated equilibrium" in which small changes are usually made between versions but occasionally abrupt changes are made. These "punctuations" are typically explained as being due to accumulated "maintainance debt" that lead to mass refactoring.

3.2 Environmental evolution

Environmental evolution happens due to changes in the software development environment, such as, evolution in the compiler, or using different compiler switches, or using a different compiler itself. Changes in the libraries that are linked to the malware can also lead to significant change in the overall code base of malware. These changes are not a result of changes to the malicious software base itself. The drift due to environmental evolution is expected to be infrequent and separated in time since changes to the underlying software development environment are relatively slow. Compiler updates are often separated by months, if not years. Decision of changing compilers are much more infrequent. Rosenblum et al. [33] give a method of identifying compiler provenance given the binary. Their work shows there is enough information in a binary to determine the compiler used to generate the binary. Hence, changes in version of compilers and libraries can introduce noticeable changes at binary level.

3.3 Polymorphic evolution

Polymorphic evolution occurs due to changes imposed by semantics-preserving program transformations in the form of automated obfuscations [5]. Typically, these transformations are done by packers and protectors. The intent of polymorphic evolution is to create artificial diversity so as to evade detection [5, 15]. Polymorphic evolution is the only real means of generating a large number of variants of a program with high diversity. The high diversity is due to the use of encryption with different keys or compression with different parameters. Since packing converts most of the code into encrypted or compressed data, the drift in data itself is random and useless to track. Hence, any useful drift tracking should be on unpacked malware. The evolution in unpacked malware can be attributed to factors decsribed in Sections 3.1 and 3.2.

4. MEASURES FOR TRACKING CONCEPT DRIFT IN MALWARE

We now describe the measures we propose for tracking concept drift in malware. These measures were developed to address the specific challenges posed by malware. Static features are one of the most commonly used features in ML-based malware detection and clustering. They can be efficiently extracted and can be parameterized by the type of static object and the size of the object. For example, n-grams can be extracted for different values of n and for bytes, instruction mnemonics,or CFG nodes as grams. A major challenge in tracking concept drift using static features in malware is that the number of n-grams is usually very large. For a typical malware executable the number of n-grams can quickly run into millions. It is not feasible to track such a large number of features individually. The multivariate tests for drift detection [9] do not scale up to tens of thousands of features. Hence, the methods required for tracking concept drift in malware should use measures that summarize the information in features that can be tracked over time. The pattern of change in the measure should give the indication of the kind of drift. We propose two such measures for detecting concept drift in malware. In addition, we also describe a modification of the use of retraining performance as an indicator of concept drift.

4.1 Relative temporal similarity

We propose tracking the similarity score between executables to get a sense of the direction and pattern of concept drift. Some methods have been proposed in software engineering literature to measure similarity of versions of large software systems. For instance, Yamamoto et al. [40] gave a method for measuring source code based similarity between versions of large open source systems like BSD Unix. Their method uses line matchings in source code for computing similarity. The method we propose, in contrast, uses similarity between executables. Similarity score between two executables represented as feature vectors, captures at a macro level the feature based commonalities between samples. Two commonly used similarity scores are cosine similarity and Jaccard index [4]. The similarity score summarizes the commonality between numerous features from the two samples into a single scalar. The similarity score can be computed for time ordered pairs of samples obtained by combining a base sample with every temporally successive sample. More precisely, let $(P_1, P_2, \ldots, P_n)$ be the temporal order on executables based on their PE header timestamp and $(\phi_1, \phi_2, \ldots, \phi_n)$ be the corresponding temporal order on the feature representation of executables. Let $sim : X^n \times X^n \rightarrow Z$ be some similarity function and $\sigma_1 = sim(\phi_1, \phi_2)$, $\sigma_2 = sim(\phi_1, \phi_3)$, $\sigma_3 = sim(\phi_1, \phi_4)$,..., $\sigma_{n-1} = sim(\phi_1, \phi_n)$ be the similarity scores for temporally ordered pairs of executables, then the $(1, i)$ temporal similarity string $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$ can be used to infer the direction of concept drift.

Some of the drift patterns resulting from different properties of the $(1, i)$ temporal similarity string $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$ are:

- *Monotonically decreasing*: This occurs when $\sigma_1 < \sigma_2 < \ldots < \sigma_{n-1}$
- *Recurring*: This occurs when the values in the temporal similarity string oscillate between certain lower and upper bounds.
- *Stable*: The variation in the values of similarity is within a small interval.
- *Split stable*: The similarity values lie at two or more different levels and the values in each level are stable.

Each of the properties of the temporal similarity string described above characterizes a type of drift.

We evaluated the $(1, i)$ relative similarity measure by tracking drift in naturally evolving malware. Natural evolution should result in a monotonically decreasing pattern. We tracked the $(1, i)$ relative temporal similarity in Agobot samples. Hayes et al. [17] simulated natural evolution in samples of malware family Agobot by turning on and off the features in the Agobot kit. The features used by Hayes et al. in generating these samples are shown in Figure 1. Each new sample was generated by adding a new feature to the

set of already existing features. We used the samples generated by Hayes et al. The similarity plot is shown in Figure 1. The similarity decreases monotonically as expected.

ID	Value	Description
1	BOT COMPNICK	Use computer name as nick name
2	BOT SECLOGIN	Enable login by channel messages
3	BOT RANDNICK	Assign random nickname to bot
4	BOT MELSERVER	Melt original server file
5	BOT TOPICCMD	Execute topic commands
6	DO SPEEDTEST	Do speedtest on startup
7	DO AVKILL	Enable AV software disabling
8	DO STEALTH	Enable stealth mode
9	AS ENABLED	Enable Autostart
10	AS SERVICE	Install as service
11	IDENTD ENABLED	Enable the server
12	CDKEY WINDOWS	Get Windows CD keys
13	SPAM AOL ENABLED	Enable AOL spamming
14	SNIFFER ENABLED	Enable sniffer
15	INST POLYMORPH	Polymorph on installation

Table 1: Agobot features

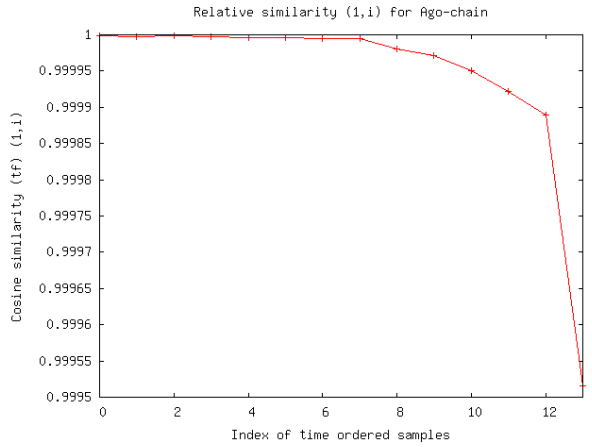


Figure 1: Similarity in Agobot samples

4.2 Metafeatures

Certain properties of features can be considered as features of features. We call such features *metafeatures*. Metafeatures summarize information from a large number of features. For example, metafeatures of n-grams can be the number of unique n-grams in an executable, the n-gram with highest frequency in an executable, etc. Tracking or monitoring metafeatures is easier than tracking large number of individual features. Another advantage of tracking metafeatures is that they can be more resilient to noise in data.

More precisely, computing metafeatures involves computing an integer valued function on a subset of features obtained by a projection function applied to the set of all features. Let $F = \{f_1, f_2, \ldots, f_n\}$ be the set of all features in a set of executables where features are integer valued i.e. $f_i \in Z$. Then, the projected subset is given by $F_\Pi = \{f_i : \forall i \in ind(\Pi(F))\}$ where Π is the projection function and *ind* returns the set of indices of elements in the set F. The metafeatures are then obtained by an integer valued function $F_M : Z^m \rightarrow Z$ that takes the set of projected features (of size m) as input.

4.3 Retraining performance

Retraining with time windows over training data and then comparing performance has been a commonly used indicator for detecting concept drift. However, this method is not applicable as it is to malware. Many malware samples have the exact same timestamp. This could be due to use of the same payload executable for packing that generated *almost* identical unpacked samples with the same timestamp. Or it could be due to mechanically generated executables using malware generation kits. If a significant number of samples with the same timestamp fall in two training windows, then performance comparison results may not be indictive of drift even if it occurs in the most recent training data window.

We propose to use variable length training data window such that all malware samples with the exact same timestamp occur in one window. The window size is increased or decreased to fit or remove all such samples, respectively. We beleive performance comparison using the training data windows adjusted for samples with the exact same timestamp would be more robust against mechanically generated malware in training data.

5. EMPIRICAL STUDY OF DRIFT IN MALWARE FAMILIES

We conducted three studies to learn about concept drift in malware families. These studies track drift in malware using the methods described in Section 4. The first study, described in Section 5.3, tracks drift using relative similarity of malware over time. The second study, described in section 5.4, uses two metafeatures for the same purpose. The third study, described in Section 5.5, examines the impact of retraining with new malware samples on the performance of classifiers.

5.1 Dataset

The malware dataset for our experiments were provided by an antivirus company. The dataset comprised of unpacked samples from three malware families-Agent, Hupigon and Pcclient. Agent is a family of trojans that have multiple components, Hupigon is a family of backdoor trojans and Pcclient is a family of backdoor trojans with a keylogger and a rootkit [30]. The samples in each family were time ordered based on their PE header timestamp. This timestamp gives the time and date when the binary was created. The timestamp characteristics of samples in each family are given in Tables 2 and 3. Table 2 gives yearly frequency of PE header timestamps of samples in each family. The samples are spread over 7 years in each family with the Agent family mostly from 2008. Table 3 shows that there were many samples with the same timestamp, probably because they were generated at once by a malware generation kit.

The ground truth of each malware sample was verified by majority voting on 3 different antivirus scanners. Though, this method of filtering of samples on which most AV scanners disagree has been criticized by Li et al. [27], there were very few samples in our dataset on which the majority of AV scanners did not agree.

5.2 Features

We used static features of executables to study concept drift in malware. Static features are obtained by considering the executable or some abstraction of the executable

(e.g. disassembled executable) as text. N-grams are one of the commonly used static features in malware detection [35, 1, 24] since they can be efficiently extracted. N-grams are obtained by sliding a window of size N along the executable. We considered two kinds of n-grams in our experiments:

- **Byte 2-grams**: Byte 2-grams are extracted by sliding a 2 byte window along the executable. The feature vector of each sample contained the number of occurences of each byte 2-gram. Although, 4-grams have been shown to be the best performing features with bytes, the difference in performance between 4-grams and 2-grams is not much [24]. We chose 2-grams to keep the number of features manageable without sacrificing much performance. There were a total of 65,536 byte 2-grams and all were used in the experiment.

- **Mnemonic 2-grams**: Each malware executable can be disassembled to obtain some more information than just bytes. Each instruction in the disassembled malware contains one or more of the instruction mnemonic, label, registers or data. The mnemonic can be extracted from each instruction and the malware executable can be represented as a sequence of mnemonics. We extracted mnemonic 2-grams by sliding a two mnemonic window on the mnemonic representation of malware. We chose 2-grams over other values of n in n-grams since they have been shown to be the best performing features for mnemonics (more specifically, opcodes) [31]. The feature vector of each malware sample was obtained by calculating the *TF-IDF* for each mnemonic 2-gram. *TF-IDF* is a commonly used feature weight in text categorization [36]. *TF-IDF* of feature i is defined as $TF_i * IDF_i$ where TF_i is the term frequency and IDF_i is the inverse document frequency. Term frequency is defined as the number of occurrences of feature i in malware sample S normalized by the total number of features in S i.e. $TF_i = |f_i|/|S|$. Inverse document frequency weighs down more commonly occurring features. It is defined as $IDF_i = \log_{10}(|S|/DF_i)$ where DF_i is document frequency or the number of malware samples containing the feature i.

Year	Malware Family		
	Agent	Hupigon	Pcclient
2003	3	6	23
2004	6	10	16
2005	3	13	197
2006	4	120	27
2007	98	351	113
2008	1,656	170	400
2009	132	44	781

Table 2: Yearwise number of samples from different malware families based on PE header timestamp

5.3 Relative temporal similarity

The purpose of this study was to track concept drift in real world malware families using relative similarity described in Section 4.1. We used cosine similarity as a measure of similarity between samples. Cosine similarity [37] between two samples represented as feature vectors $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ is given by

$$cos(A, B) = \frac{A \cdot B}{|A||B|}$$

where $A \cdot B = a_1b_1 + a_2b_2 + \ldots + a_nb_n$, $|A| = (a_1^2 + a_2^2 + \ldots + a_n^2)^{1/2}$, and $|B|$ is defined similarly.

The $(1, i)$ temporal similarity string was computed for two types of static features and the patterns in the string were studied using scatter-plots. We used byte 2-grams and mnemonic 2-grams as features in this experiment.

5.3.1 Byte 2-grams

We computed relative similarity with byte level features using byte 2-grams. Two treatments (not to be confused with folds) were created to introduce randomly selected base malware sample. Each treatment used a different base sample against which all other samples were compared. The number of samples from each malware family in each treatment are given in Table 4.

Family	Byte 2-grams		Mnemonic 2-grams	
	Tr-1	Tr-2	Tr-1	Tr-2
Agent	1,886	1,881	1,875	1,873
Hupigon	679	676	566	524
Pcclient	1,524	1,514	1,425	1,373

Table 4: Number of samples in two treatments for each family

Our expectation was a decrease in similarity of samples over time with the amount of decrease varying with family. However, the results of this study (Figures 2(a),(c),(e)) do not support a monotonically decreasing drift model. The figure shows plots for the first treatment, the results of the other treatment were almost identical. The y-axis in the plots is the cosine similarity $(1, i)$ and x-axis is the index i of temporally ordered sample pairs. The plots in Figure 2(c),(e) indicate that at the byte level there is hardly any decrease over time in similarity between samples in Hupigon and Pcclient families implying negligible drift in Hupigon and Pcclient families. However, in Agent (Figure 2(a)) there are four bands of similiarity at different levels. The small band on the top left(cosine similarity ≈ 1) is at the same level as the one on top right indicating high similarity between these samples. The are two distinct bands at cosine similarity ≈ 0.73 and 0.98, respectively. A closer look at the bands revealed that a large number of samples in each of these bands had identical timestamps. The first 795 samples in the band at level 0.73 had the timestamp 12/25/2008 05:34:49 followed by 248 samples with timestamp 12/21/2008 04:57:01. The band at level 0.98 was amost entirely composed of 505 samples with timestamp 12/30/2008 20:59:39.

The results can be explained considering that all the samples were unpacked. Most malware found in the wild is usually packed. Packing a malware involves converting malicious code to data using encryption or compression and attaching an unpacker that converts the data back to code upon execution. Large number of variants can be generated by using different encryption key or compression parameter and there are also polymorphic packers that can generate a

Agent		Hupigon		Pcclient	
#	Timestamp	#	Timestamp	#	Timestamp
795	12/25/2008 05:34:49	12	03/22/2008 02:40:31	352	02/02/2009 17:47:58
505	12/30/2008 20:59:39	8	08/04/2004 01:01:37	228	02/04/2009 15:52:12
248	12/21/2008 04:57:01			127	06/13/2008 23:52:17
11	8/7/2007 10:37:35			116	01/12/2009 17:48:29
7	7/27/2009 01:27:29			108	12/03/2008 16:59:35
7	3/31/2007 03:17:56			38	02/05/2009 17:52:58
				26	08/16/2008 21:47:34
				23	06/13/2008 23:52:19
				20	11/25/2007 06:33:11
				20	02/02/2009 17:48:00

Table 3: Number of samples with the same PE header timestamp

different unpacker for every variant. The process of unpacking a packed executable using a generic unpacker gives an executable that is similar but not identical to the original executable. The generic or custom unpackers used by AV companies may introduce some artifacts that may not be in the original executable. Hence, unpacking the variants produced by packing the same executable will give executables with different hash signature. The results in Figures 2(a),(c),(e) indicate that the bands consist of samples that were originally one executable but unpacking produced very different executables that are very similar at byte level. This implies that a single executable was packed to generate variants in Hupigon and Pcclient ((Figure 2 (c),(e)) explaining a single high similarity band. However, three different executables were used in Agent (Figure 2 (a)) that show up as bands at three different levels.

5.3.2 Mnemonic 2-grams

The use of instruction mnemonics as features for computing relative similarity can give some insight into how malware code drifts. We studied drift in mnemonic 2-grams in the three malware families. As for byte 2-grams, two treatments were created to account for the variability in results introduced by the choice of the base sample. The number of samples in each treatment are shown in Table 4. The counts are smaller than those for byte 2-grams since some of the malware executables could not be successfully disassembled. Since the number of all possible mnemonic 2-grams is very large, we used feature selection to prune the feature set. Document frequency with a threshold of 10 was used as the selection criteria i.e. those mnemonic 2-grams were selected that occurred in more than 10 samples in the family. The number of features before and after selection are given in Table 5.

Family	Total features	Selected features
Agent	93,285	33,720
Hupigon	188,469	88,966
Pcclient	61,176	19,182

Table 5: Number of mnemonic 2-grams before and after feature selection

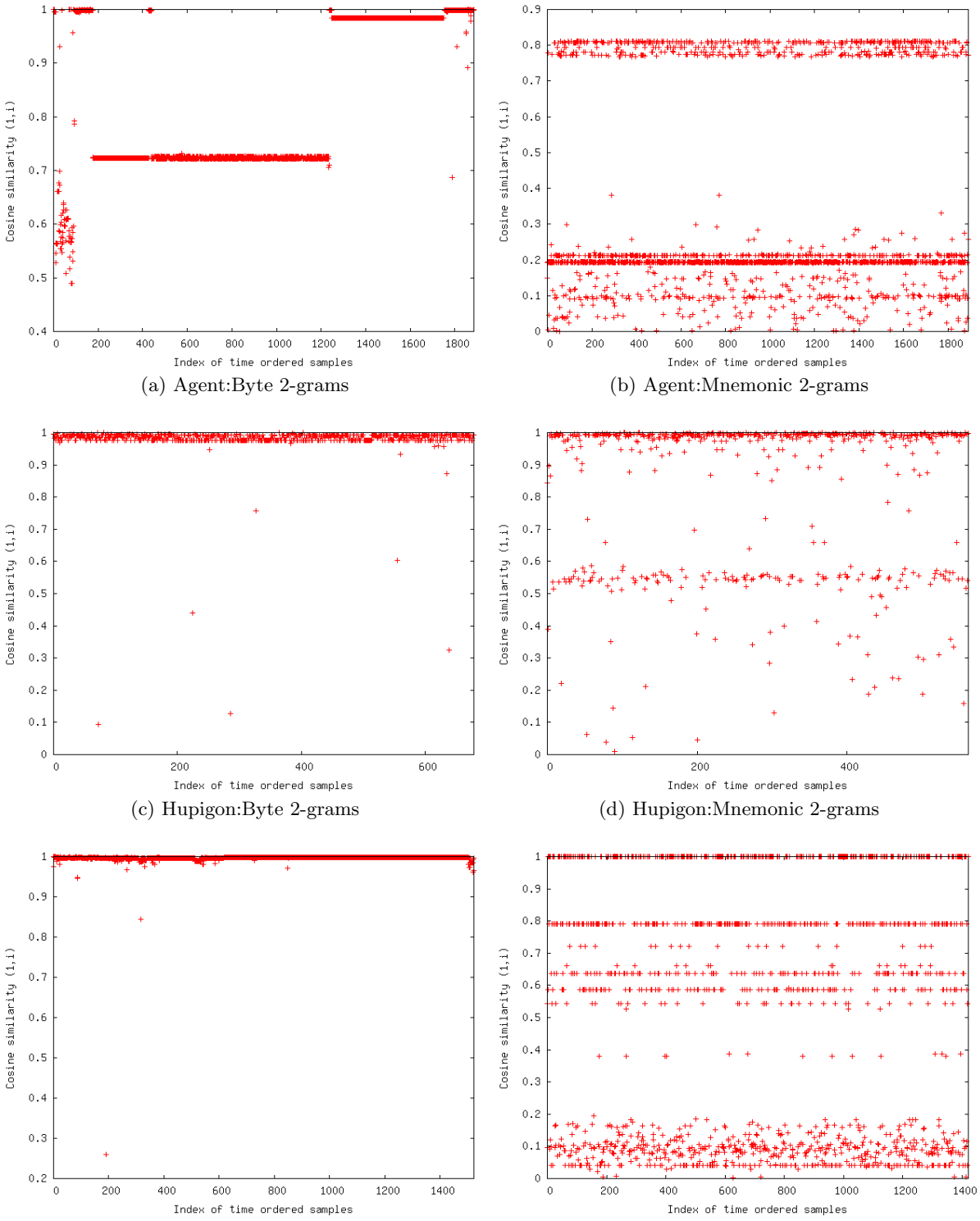
We expected relative similarity to decrease with time for mnemonic 2-grams as well since addition of new features, modification of existing features and removal of old features will change the malware code significntly over time. The results we obtained were otherwise. Figures 2 (b), (d), and (e) show the relative similarity plots for the three malware families in the first treatment. The results of the other treatment were almost identical. The plots show flat split bands in all families with the number and tightness of bands varying with family. An intersting thing to observe is that different bands occur throughout, without any interleaved regions. This indicates that, at the code level, there are more than one type of malware in each family all of which were generated throughout the time period considered. Even though, there are few byte level differences in Hupigon and Pcclient, there are different types of malware at code level in both families. The plots for mnemonic 2-grams with PE header date (month/year) on the x-axis are shown in the appendix in Figure 4. The plots in Figure 4 also give a sense of most active periods of a family.

Agent is the most intriguing family with byte level differences showing as interleaved bands in Figure 2 (a) but at the code level different bands exist simultaneously and throughout (Figure 2 (b)). This implies that at the code level malware exists in different forms that do not change significantly with time. This also shows the importance of the choice of type of features in computing relative similarity. The byte level features in Agent, for example, show sudden drifts whereas mnemonic level features do not show any significant drift. Another important point is what we call similar malware executables at byte level may be different malware programs at mnemonic level. This can be explained in light of the fact that different instructions may use the same registers and addresses, leading to a small change at byte level but a signifant change at the mnemonic level.

5.4 Tracking metafeatures

There were tens of thousands of byte 2-grams and mnemonic 2-grams in each malware family (Table 5). Hence it was not possible to track each one of them individually. We chose to track two metafeatures of mnemonic 2-grams to get a sense of how the raw features drift over time. The two metafeatures are described below:

- *Term frequency of the mnemonic 2-gram with highest document frequency*: This metafeature was obtained by first determining the mnemonic 2-gram that occured in most samples in a family. Then the term ferquency of this mnemonic 2-gram was tracked in sam-



(a) Agent:Byte 2-grams

(b) Agent:Mnemonic 2-grams

(c) Hupigon:Byte 2-grams

(d) Hupigon:Mnemonic 2-grams

(e) Pcclient:Byte 2-grams

(f) Pcclient:Mnemonic 2-grams

Figure 2: Relative similarity with byte 2-grams and mnemonic 2-grams for three malware families

ples over time. More precisely, let F be the set of all the features in all the malware samples of the dataset and $F_S \in F$ be the set of features occuring in the malware sample S. Then, the projection function Π can be written as

$$\Pi = \max_{DF}(F_S))$$

where $\max_{DF}$ returns the feature with highest docu-

ment frequency. Now, the integer valued metafeature function F_M is $TF(\Pi(F_S))$, the term frequency of the output of Π.

- *Number of unique mnemonic 2-grams*: Each malware sample consists of a subset of all mnemonic 2-grams. Since, any mnemonic 2-gram can occur more than once in a sample, we tracked the number of *unique* mnemonic 2-grams in malware samples over time. Here, the projection function Π returns the set F_S and the integer valued metafeature function F_M is $|F_S|$, the size of of the set F_S.

The dataset and the number of mnemonic 2-grams for this study was the same as the first treatment of the relative similarity study. The results are shown in Figure 3. The concept drift in the first metafeature (term frequency) is shown in Figures 3(a), 3(c) and 3(e) for the three malware families. Each of these plots show split flat bands of samples throughout. This is in agreement with the observation from mnemonic 2-grams relative similarity study. We can conclude that at the code level in each family there are more than one kind of samples throughout the time interval of samples in the dataset and that these different kinds of samples in each family do not change significantly over time. The drift in the second metafeature (number of mnemonic 2-grams) is shown in Figures 3(b), 3(d) and 3(e). The number of unique mnemonic 2-grams in Agent (Figure 3(b)) is mostly between 4,000 and 5,000 and this tight band remains flat throughout. The band in Hupigon is spread out but it does indicate two kinds of samples at around 5,000 and 20,000. Pcclient is similar to Agent with a tight band at around 2,500.

The two metafeatures give a general trend of the type of drift in the malware families. The families under consideration show insignificant drift. This implies the performance of machine lerning based malware detectors can be robust for such families.

5.5 Retraining

The purpose of this study was to detect drift on the basis of the difference in performance of classifiers trained on original training data and new training data. We split the dataset into original (O) and new (N) based on timestamp. As discussed in Section 4.3, we made sure that samples in a family with the same timestamp did not end up in different training sets. This implied a particular cutoff timestamp could not be used to determine the two training sets. The training sets were variable length to accomodate all samples with one timestamp in one of the training sets. There were a total of 1,825 samples in the original training dataset comprising a mix of agent, hupigon, pcclient and benign samples. There were 1,851 samples in the recent training dataset also comprising of a mix of agent, hupigon, pcclient and benign samples. The test dataset had a mix of 2,325 samples. Benign samples were Windows XP system DLL files. TF-IDF of mnemonic 2-grams were used as features, as in the relative similarity experiment. The features in the training data were selected using a document frequency threshold of 100 to keep the number of features manageable. Four classifiers were chosen for the experiment: IB3, J48, SVM and Naive Bayes.

The result of this study is shown in Table 6. The classifiers were trained and tested with binary (malware,benign) as well as multiclass (agent,hupigon,pcclient,benign) data. As can be seen from the table, with the exception of Binary IB3, the accuracy of classifiers trained on the two datasets is comparable. This is another evidence indicating negligible drift since the performance can be comparable only if the distribution of mnemonic 2-grams does not change significantly in samples over time.

Classifier	Binary		Multiclass	
	O	N	O	N
IB3	70.77	44.87	93.16	91.39
J48	99.78	99.69	97.29	96.25
NB	98.62	99.00	94.79	95.01
SMO	99.87	99.95	97.63	96.90

Table 6: Accuracy of different classifiers trained on original and recent dataset

6. LIMITATIONS AND FUTURE WORK

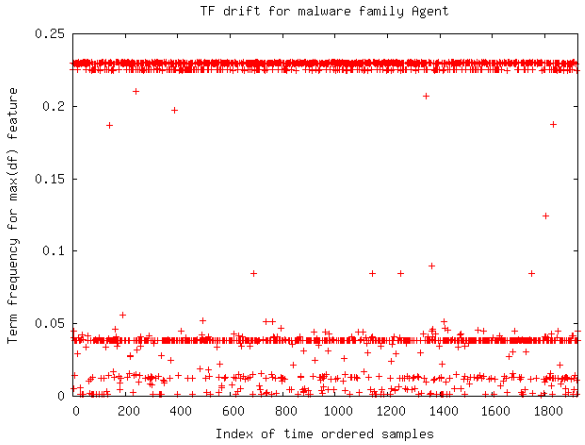
A limitation of our study of concept drift in malware families is that it tracked drift in static features only. Many types of behavioral features have been proposed for clustering and classifying malware [3, 2]. The number of behavioral features in a malware is usually far less than the number of static features like n-grams. Hence, it may be easier to track the behavioral features individually. However, the extraction of behavioral features is less efficient than extracting n-grams thus making static features more suitable in a stream classification setting. Moreover, most malware families are formed on the basis of some behavioral similarity. Hence, behavioral features may not show significant temporal drift compared to n-grams since the same behavior may have an obfuscated implementation resulting in different n-grams.

Also our study was limited to three malware families. We did not attempt to conduct an exhaustive study with a large number of malware families since such a study is not only infeasible but also the conclusions may not be an indicator of presence/absence of drift in future families. A desired characteristic for choosing families in a drift study is they should have a long temporal profile, i.e., they should have been active over several years. Although the data set in our study was limited to three families, each family had a long temporal profile (Table 2).

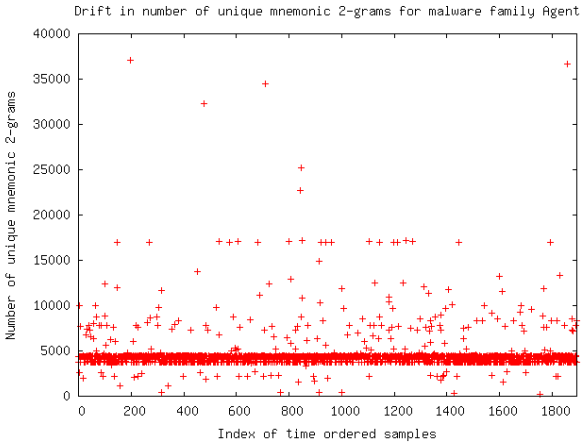
The results of our experiment give some evidence of the impact of unpacking on malware binaries. We saw in Section 5.3 that what we thought were different unpacked binaries may have originated from a small subset of original binaries. A more controlled study to determine the nature of the impact of unpacking is needed to confirm the relationship between unpacking, evolution and drift in malware.

7. CONCLUSIONS

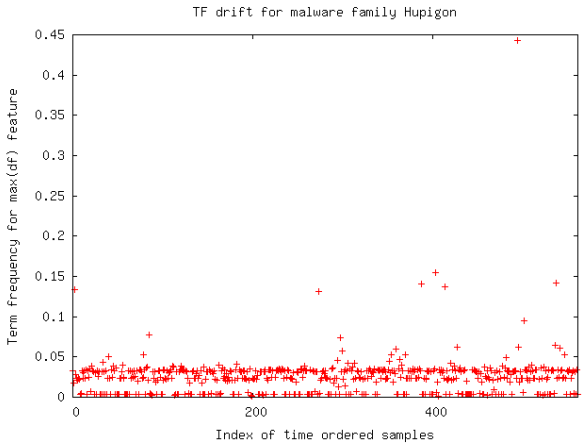
We proposed two measures to track concept drift in malware when the number of features is large. We studied concept drift in three real world malware families and found negligible drift in mnemonic 2-grams. The results of three different experiments—relative similarity, metafeatures, and retraining—provide evidence in favor of negligible drift in mnemonic 2-grams. Though, our study was limited to three malware families and two kinds of features, it gives a direction for further exploration of drift in malware under differ-



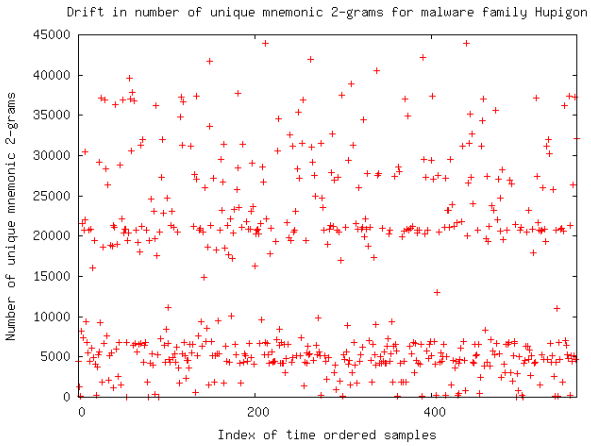
(a) Agent:Term frequency



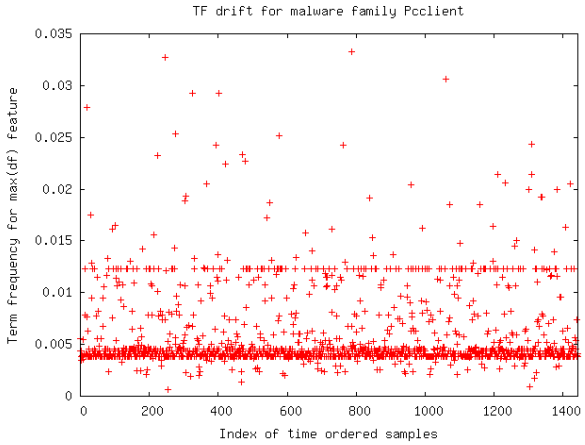
(b) Agent:Unique mnemonic 2-grams



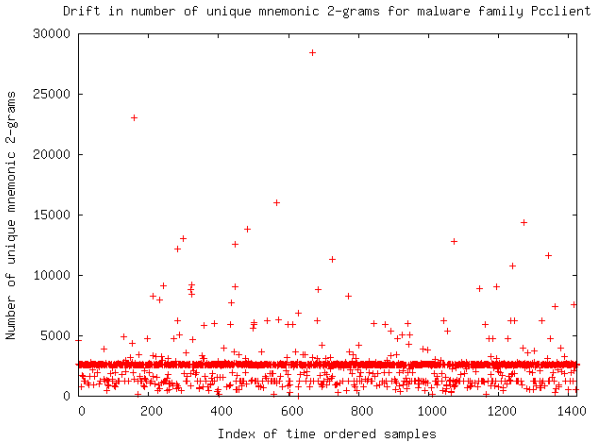
(c) Hupigon:Term frequency



(d) Hupigon:Unique mnemonic 2-grams



(e) Pcclient:Term frequency



(f) Pcclient:Unique mnemonic 2-grams

Figure 3: Metafeature drift for three malware families

ent malware evolution models. The negligible drift in certain types of features can be exploited by deploying classifiers based on these features thus making them more robust to evolving malware.

8. ACKNOWLEDGEMENTS

This research work was sponsored in part by funds from Air Force Research Lab and DARPA (FA8750-10-C-0171)

and from Air Force Office of Scientific Research (FA9550-09-1-0715).

9. REFERENCES

[1] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Proc. of the 28th Annual Intl. Computer Software and Applications Conference*, 2003.

[2] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, pages 178–197. Springer-Verlag, 2007.

[3] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Network and Distributed System Security Symposium (NDSS)*, 2009.

[4] S. Choi, S. Cha, and C. Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[5] C. Collberg and J. Nagra. *Surreptitious software: obfuscation, watermarking, and tamperproofing for software protection*. Addison-Wesley Professional, 2009.

[6] S. Delany, P. Cunningham, and B. Smyth. Ecue: A spam filter that uses machine learning to track concept drift. In *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*, pages 627–631. IOS Press, 2006.

[7] S. Delany, P. Cunningham, and A. Tsymbal. A comparison of ensemble and case-base maintenance techniques for handling concept drift in spam filtering. In *Proceedings of the 19th International Conference on Artificial Intelligence (FLAIRS 2006)*, pages 340–345, 2006.

[8] S. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4):187–195, 2005.

[9] A. Dries and U. Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2(5-6):311–327, 2009.

[10] F. Fdez-Riverola, E. Iglesias, F. Díaz, J. Méndez, and J. Corchado. Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications*, 33(1):36–48, 2007.

[11] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer, 2001.

[12] J. Friedman and L. Rafsky. Multivariate generalizations of the wald-wolfowitz and smirnov two-sample tests. *The Annals of Statistics*, pages 697–717, 1979.

[13] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. *Advances in Artificial Intelligence–SBIA 2004*, pages 66–112, 2004.

[14] GDataSoftware. G data malware report. `http://www.gdatasoftware.com/uploads/media/G_Data_MalwareReport_H1_2011_EN.pdf`, 2011.

[15] F. Guo, P. Ferrie, and T. Chiueh. A study of the packer problem and its solutions. In *Recent Advances in Intrusion Detection*, pages 98–115. Springer, 2008.

[16] P. Hall and N. Tajvidi. Permutation tests for equality of distributions in high-dimensional settings. *Biometrika*, 89(2):359–374, 2002.

[17] M. Hayes, A. Walenstein, and A. Lakhotia. Evaluation of malware phylogeny modelling systems using automated variant generation. *Journal in Computer Virology*, 5(4):335–343, 2009.

[18] D. Helmbold and P. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–45, 1994.

[19] R. Hogg, J. McKean, and A. Craig. Introduction to mathematical statistics, 2005.

[20] M. Karim, A. Walenstein, A. Lakhotia, and L. Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1):13–23, 2005.

[21] M. Kelly, D. Hand, and N. Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 367–371. ACM, 1999.

[22] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

[23] R. Klinkenberg and I. Renz. Adaptive information filtering: Learning drifting concepts. In *Proc. of AAAI-98/ICML-98 workshop Learning for Text Categorization*, pages 33–40, 1998.

[24] J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2721–2744, 2006.

[25] A. Kuh, T. Petsche, and R. Rivest. Learning time-varying concepts. In *Proceedings of the 1990 Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 183–189, 1990.

[26] M. Lehman. Laws of software evolution revisited. *Software process technology*, pages 108–124, 1996.

[27] P. Li, L. Liu, D. Gao, and M. Reiter. On challenges in evaluating malware clustering. In *Recent Advances in Intrusion Detection*, pages 238–255. Springer, 2010.

[28] M. Masud, T. Al-Khateeb, K. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Cloud-based malware detection for evolving data streams. *ACM Transactions on Management Information Systems (TMIS)*, 2(3):16, 2011.

[29] M. M. Masud, L. Khan, and B. Thuraisingham. A hybrid model to detect malicious executables. In *Proc. of the IEEE Intl. Conf. on Communications (ICC 2007)*, pages 1443–1448, 2007.

[30] T. R. Microsoft Protection Center and Response. Malware encyclopedia. `http://www.microsoft.com/security/portal/Threat/Encyclopedia/Browse.aspx`, 2011.

[31] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici. Unknown malcode detection using OPCODE representation. In *Intelligence and Security Informatics*, volume 5376 of *Lecture Notes in Computer Science*, pages 204–215. Springer Berlin, 2008.

[32] C. Nachenberg. Computer virus-coevolution. *Communications of the ACM*, 50(1):46–51, 1997.

[33] N. Rosenblum, B. Miller, and X. Zhu. Extracting compiler provenance from program binaries. In *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 21–28, 2010.

[34] B. Rubinstein, B. Nelson, L. Huang, A. Joseph, S. Lau, S. Rao, N. Taft, and J. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference*, pages 1–14, 2009.

[35] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proc. of S&P 2001: IEEE Symposium on Security and Privacy*, pages 38–49, 2001.

[36] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[37] P. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson Addison Wesley, 2006.

[38] A. Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 2004.

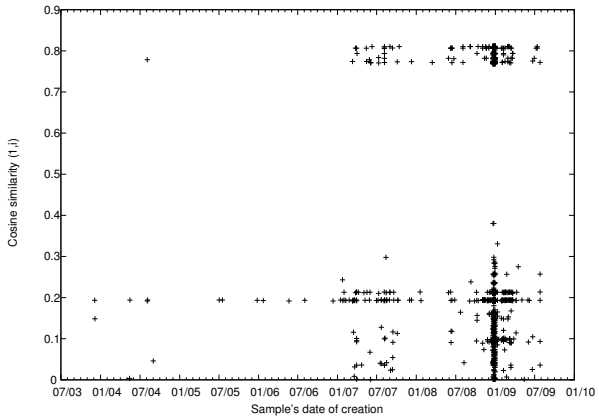
[39] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[40] T. Yamamoto, M. Matsushita, T. Kamiya, and K. Inoue. Measuring similarity of large software systems based on source code correspondence. *Product Focused Software Process Improvement*, pages 179–208, 2005.

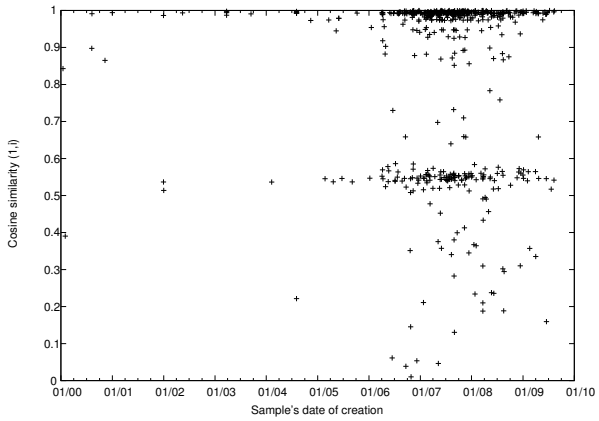
[41] I. Zliobaite. Learning under concept drift: an overview. Technical report, Vilnius University, Lithuania, 2009.

APPENDIX

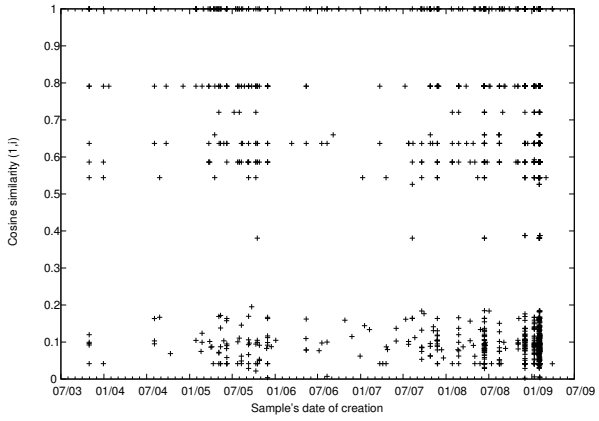
Figure 4 shows the results of the plot in Figures 2 (b), (d), and (e) with PE header date (month/year) on the x-axis.



(a) Agent:Mnemonic 2-grams



(b) Hupigon:Mnemonic 2-grams



(c) Pcclient:Mnemonic 2-grams

Figure 4: Relative similarity with respect to PE header date for mnemonic 2-grams for three malware families