Malware Detection Using Adaptive Data Compression

Yan Zhou
School of Computer and Information Sciences
University of South Alabama
Mobile, AL 36688
zhou@cis.usouthal.edu

Meador Inge
School of Computer and Information Sciences
University of South Alabama
Mobile, AL 36688
wmi601@cis.usouthal.edu

ABSTRACT

A popular approach in current commercial anti-malware software detects malicious programs by searching in the code of programs for *scan strings* that are byte sequences indicative of malicious code. The scan strings, also known as the signatures of existing malware, are extracted by malware analysts from known malware samples, and stored in a database often referred to as a virus dictionary. This process often involves a significant amount of human efforts. In addition, there are two major limitations in this technique. First, not all malicious programs have bit patterns that are evidence of their malicious nature. Therefore, some malware is not recorded in the virus dictionary and can not be detected through signature matching. Second, searching for specific bit patterns will not work on malware that can take many forms—obfuscated malware. Signature matching has been shown to be incapable of identifying new malware patterns and fails to recognize obfuscated malware. This paper presents a malware detection technique that discovers malware by means of a learning engine trained on a set of malware instances and a set of benign code instances. The learning engine uses an adaptive data compression model—prediction by partial matching (PPM)—to build two compression models, one from the malware instances and the other from the benign code instances. A code instance is classified, either as "malware" or "benign", by minimizing its estimated cross entropy. Our preliminary results are very promising. We achieved about 0.94 true positive rate with as low as 0.016 false positive rate. Our experiments also demonstrate that this technique can effectively detect unknown and obfuscated malware.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—Learning; I.5 [**Computing Methodologies**]: Pattern Recognition; I.7 [**Computing Methodologies**]: Document and Text Processing

AISec'08, October 27, 2008, Alexandria, Virginia, USA.

General Terms

Algorithms, Security

Keywords

malware detection, machine learning, statistical data compression

1. INTRODUCTION

Many current commercial anti-virus programs are based on pattern matching schemes that search for pre-defined bit patterns, oftern referred to as scan strings or signatures, that are indicative of malicious code. Although this type of software has been very effective in detecting known malicious code, there are some limitations. First, signatures of known malicious code are extracted by malware analysts and stored in a database often referred to as a virus dictionary. Code scanning essentially checks whether there is a match of stored signatures in scanned files. One problem with this reactive process is that a certain number of computers must be infected before a new virus pattern can be captured and stored for future use. The second problem with this approach is that not all malware has bit patterns that are indicative of the presence of malicious intent. Thus, some malware cannot be detected simply through signature matching. In addition, a virus dictionary often requires frequent update as new "viruses" are created and discovered, which "entails a significant amount of effort on the part of human virus experts" while "even the best experts have been known to select poor signatures." [10] Another limitation of signature based anti-virus software is their susceptibility to obfuscated malware. As malware programmers get more sophisticated, defining bit patterns of malicious programs has become more challenging. It is now common to see adversaries obfuscate their malicious code to make it unrecognizable in order to foil existing anti-virus software. An ideal anti-virus utility should be able to both detect existing malicious code in any forms and identify new patterns that have not been encountered before.

Several machine learning techniques have been investigated for identifying unknown malicious executables, including neural networks [10], naive Bayes [14], k-nearest neighbor, support vector machines, decision trees, and boosted classifiers [11]. Although the results are promising, all these techniques share a common constraint—construction of structured input—because standard learning algorithms expect structured input such as vectors. However, executables, like text documents, are naturally unstructured, therefore

must be preprocessed to form the structured input expected by the learning algorithms. Such preprocessing is normally done by extracting features (signatures, byte sequences, binary profiles, etc.) from executables and subsequently selecting the most relevant features for learning. Unfortunately, such preprocessing is error-prone, and may not be able to make full use of the source data as a result of feature selection. Thus, the outcome of all the previous attempts of using machine learning and data mining techniques for malware detection relied heavily on the specific feature extraction process used to create feature vectors from the executable code.

In this paper, we propose a new malware detection technique using a learning methodology that works on unstructured input, that is, raw executables, with an underlying statistical compression model. The general idea of this method is to construct two compression models, one from a collection of malicious code and one from a collection of benign code. A new executable is classified according to which of the resulting models compresses the code more efficiently. The proposed unstructured learning technique has a number of advantages over standard machine learning algorithms that only work on structured input. First, the compression algorithms work on the character-level raw executables rather than the preprocessed subset of the original code. For this reason, preprocessing and feature selection, both of which are highly prone to error, are unnecessary. Instead the algorithm is able to make full use of all features of executables. Another benefit of the character-level nature of the compression algorithms is that they are more robust to obfuscation. Adversaries often disguise the malicious code by producing polymorphic malicious code, for example, through packing, code reordering, and junk insertion [2]. This can cause a standard learning algorithm to fail to identify the disguised malicious code unless extra care and effort are expended to detect and deal with these obfuscations [2, 9, 12, 13, 15].

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides a review of the statistical compression model used to build the learning engine, and presents our malware detection technique. Our preliminary results are presented in Section 4. Section 5 concludes our work and discusses future directions.

2. RELATED WORK

Although malware detection is a well-established research field, there has been relatively little published research on using machine learning and data mining techniques to detect malicious code. In one of their early attempts, Kephart et al. [10] showed that nearest-neighbor classification with Hamming distance or edit distance performs poorly for detecting boot sector viruses because of the negligible difference a short string of viral code makes. Instead, they built a neural network to discriminate between infected and uninfected code. The input to the neural network is a set of trigrams—3-byte strings. They extracted a set of trigrams that appear frequently in viral boot sectors but rarely in legitimate ones and produced 25,000 distinct trigrams. In addition, they performed feature pruning in order to avoid having the classifier generalize poorly on new viral instances. As a result, a "cover" of trigrams, with at least one trigram representing each viral instance, is defined and a 4-cover was selected for use in their experiments. A 4-cover is a set of trigrams in which at least four different trigrams represent each viral instance. Each viral instance is now reduced to a 4-cover, with the presence of each trigram set to 1 and its absence set to 0, which defines the vector input to their neural network. Since no legitimate instances contain any of the trigrams extracted from the viral instances, the learning problem is ill-defined. To solve this problem, they introduced one artificial negative example for each trigram feature, in which that trigram feature has a value of 1 and the rest of the feature values are 0. They reported a false positive rate of 0.0002 measured on artificial boot sector instances, and a false negative rate of 0.15.

A recent work on learning to classify malicious executables compares the performance of several machine learning algorithms [11]. Kolter et al. collected 1,971 benign and 1,651 malicious executables as their training examples. They converted each executable to hexadecimal code and produced n-grams of byte code—by packing each 4-byte sequence—as features. The process resulted in over 255 million distinct n-grams. They performed feature selection using information gain and selected the 500 most relevant n-grams as features for classification. From the results of a small-scale study varying the sizes of n-grams and the numbers of features, it was concluded that 4-grams and 500 features appear to produce the best results. They experimented with four machine learning algorithms—naïve Bayes, decision trees, support vector machines, and boosting—and reported the results of running stratified ten-fold cross-validation. They concluded that boosted decision trees outperform all other learning methods they have tried. They also tried classifying malware based on payload functions, such as backdoors and mass-mailing, and achieved 0.9 AUC (areas under the ROC curve). In addition, they applied their trained malware detector to 291 new malicious code that were not used for training, and booted decision tree achieved 0.98 true-positive rate and 0.05 false-positive rate.

Our work is largely inspired by recent work on filtering e-mail spam using statistical data compression models [18, 1]. Spam filtering and malware detection share some common ground. Both are working against adversaries aiming to create unwanted disruptions of the normal use of computers. When machine learning techniques are involved, both need to handle unstructured input, either e-mail or executable code, and both have to be robust to adversarial attacks such as obfuscation to remain practical.

In an early attempt [18] of using statistical compression for filtering e-mail spam, a spam filtering strategy was proposed to use a practical entropy coding theory—huffman coding—to dynamically encode the feature space of e-mail collected over time, and apply online learning to adapt to new spam concepts. The feature space is defined over two Huffman prefix trees, one constructed from the spam corpus and the other from the legitimate corpus. Classification of an e-mail message is indicated by its score computed from the two prefix trees.

A relatively new idea of using context-based statistical data compression algorithms for spam filtering was recently proposed [1]. Bratko et al. proposed and investigated the use of character-level data compression models for spam filtering. The general idea is to construct two compression models, one from a collection of spam e-mails and one from a collection of legitimate e-mails, and then to classify a message according to which of the resulting models compresses the message more efficiently. Using statistical data

compression for spam filtering has a number of advantages over machine learning algorithms that use word-level models. First, the compression algorithms work on the character level rather than the word level. For this reason, preprocessing and feature selection, both of which are highly prone to error, are unnecessary. Instead the algorithm is able to make full use of all message features. Another benefit of the character-level nature of the compression algorithms is that they are more robust to obfuscation. Spammers often disguise spammy words by deliberately misspelling them or by inserting punctuation between characters. This can cause a word-level spam filter to misclassify e-mails containing such words unless extra care and effort are expended to detect and deal with these obfuscations. Bratko et al. [1] implemented their compression-based spam filter using the prediction by partial matching algorithm with escape method D (PPMD). Their experiments for the Trec 2005 spam track showed promising results. They also demonstrated that their filter was indeed quite robust to obfuscation.

3. MALWARE DETECTION USING STATISTICAL DATA COMPRESSION MODELS

In previous studies by several researchers, compression-based classification models have shown great success in the domain of e-mail spam filtering [18, 1]. The common ground between spam filtering and malware detection has open a new avenue for research in this area. As mentioned earlier, both research fields involve discriminating between legitimate and illegitimate instances, for example, spam versus legitimate e-mails and malware versus benign programs. The discriminating methods used in both areas need to deal with unstructured input, either plain text or raw executable code. Furthermore, for those discriminating methods to remain practical in the wild, they have to be robust to deliberate, adversarial attacks, such as obfuscated e-mail or malware, that aim to disguise the presence of the malicious intent of the adversary. In this section, we discuss the general idea of compression-based classification, review the statistical compression model—prediction by partial matching (PPM), as well as present the classification model that will be used to identify malware in our experiments.

3.1 Classification versus Data Compression

Data compression works by assigning a more compact representation to frequently recurring patterns in data. Likewise, many machine learning algorithms rely on identification of frequent data patterns to build classification models. In the domain of spam filtering and malware detection, statistical compression models, especially character-level compression models, are appealing for the following reasons:

- compression-based models can take raw data, such as e-mail and executables, as input, skipping error-prone preprocessing;
- compression-based models are potentially more robust to character-level adversarial attacks such as text obfuscation in e-mail and polymorphic/metamorphic viruses, and
- some of the best compression models, if implemented properly, are not CPU time demanding, thus suitable for operating in the real e-mail environment.

With that said, compression based classification models have drawbacks. First, runtime efficiency does come with a price of high memory demand. Although, some of the memory cost can be ameliorated by using pruning strategies. Drinic et al. show one technique that can reduce memory consumption by as much as 70% while still maintaining the same level of compression ratios [6]. Second, it is simply not intuitive to build high level abstract concepts directly at the character level. Finally, compression-based learning models have not shown superior performance against traditional machine learning approaches in other text classification problems. One reason for this is that machine learning methods are particularly adept at honing in on a few features that can uniquely indentify the class of an instance. There is no such abandonment of irrelevant features with compression-based approaches [7]. Nonetheless, the previous study [1] shows that using statistical data compression in spam filtering has produced very promising results. Our focus is on applying and investigating this approach in the domain of malware detection.

3.2 Prediction by Partial Matching

Statistical compression methods follow the basic protocol of arithmetic coding [17]. Given a sequence of symbols, the encoding for the i^{th} symbol depends on its probability in the current context. Different compression methods build different statistical models to predict the symbol probability. The best ones [3, 5, 4] among the context-based adaptive approaches give a good estimate of the true entropy of data by using symbol-level dynamic Markov modeling. For example, prediction by partial matching (PPM) [3] predicts the symbol probability conditioned on its k immediately proceeding symbols, forming a k^{th} order Markov model. A dynamic context length is used to look for the longest string match between the current sequence and the ones that occurred previously. More specifically, the algorithm first looks for a match of an order-k context. If such a match does not exist, it looks for a match of an order $k-1$ context, until it reaches order-0. Whenever a match is not found in the current context, the total probability is adjusted by what is called an *escape probability*. The escape probability models the probability that a symbol will be found in a lower-order context. If the symbol is not predicted by the order-0 model, a probability defined by a uniform distribution is predicted. Therefore, given an input $X = x_1 x_2 \dots x_d$ of length d, where $x_1, x_2, \dots, x_i$ is a sequence of symbols, its probability given a compression model M can be estimated as

$$p(X|M) = \prod_{i=1}^{d} p(x_i | x_{i-k}^{i-1}) \quad (1)$$

where $x_i^j = x_i x_{i+1} x_{i+2} \dots x_j$ for $i < j$.

In general the PPM algorithm maintains a map from contexts to probability distributions. The map is initially empty and as new contexts are seen during processing, new probability distributions are created. When a symbol is encountered under a context already existing in the map, the statistics are adjusted accordingly. In practice, there are several policies for how the symbol and escape probabilities are adjusted.

As a simple illustrative example consider the estimated probability produced for the sequence of symbols "macabre" with $k = 1$ under alphabet Σ, where $|\Sigma| = 256$ (Extended

ASCII). Assume that the model, M, has the context to probability distribution mapping in Table 1, that was obtained from processing "abracadabra". Note that the univeral probability is $\frac{1}{250}$ instead of $\frac{1}{256}$ since we already have statistics for the symbols $\{a, b, c, d, r, Escape\}$. Again, there are many different policies for how these probability distributions are maintained. For the purpose of this example it is only important that they somehow exist. Per Equation 1 we must compute

$$\begin{aligned} P(macabre|M) &= p(m|\lambda)p(a|m)p(c|a)p(a|c) \\ &\quad \cdot p(b|a)p(r|b)p(e|r) \\ &= \left(\frac{5}{13}\cdot\frac{1}{250}\right)\left(\frac{4}{13}\right)\left(\frac{1}{7}\right)\left(\frac{1}{2}\right) \\ &\quad \cdot \left(\frac{2}{7}\right)\left(\frac{2}{3}\right)\left(\frac{1}{3}\cdot\frac{5}{13}\cdot\frac{1}{250}\right) \\ &= 3.30279282097e^{-09} \end{aligned}$$

The third through six factors are trivial as we already have statistics for those symbols under their provided context. The first factor is calculated by multiplying the escape probability, $\frac{5}{13}$, under the empty context λ times the universal probability for m. The second probability is just the probability of a under λ, since we have no statistics on the context m. The last context is the escape probability, $\frac{1}{3}$, under r times the escape probability, $\frac{5}{13}$, under λ times the universal probability for e. Note that the escape probalities in the preceding formula slightly reduce the total probability when a symbol is not found in the current context.

Table 1: Frequency Table

context	distribution	
a	b	$\frac{2}{7}$
	c	$\frac{1}{7}$
	d	$\frac{1}{7}$
	Escape	$\frac{3}{7}$
r	a	$\frac{2}{3}$
	Escape	$\frac{1}{3}$
b	r	$\frac{2}{3}$
	Escape	$\frac{1}{3}$
c	a	$\frac{1}{2}$
	Escape	$\frac{1}{2}$
d	a	$\frac{1}{2}$
	Escape	$\frac{1}{2}$
λ	a	$\frac{4}{13}$
	b	$\frac{1}{13}$
	c	$\frac{1}{13}$
	d	$\frac{1}{13}$
	r	$\frac{1}{13}$
	Escape	$\frac{5}{13}$
Universal	$\frac{1}{250}$, for $c \in \Sigma \| c \notin \{a, b, c, d, r, Escape\}$	

3.3 Classification with Statistical Compression Models

To simplify the description of the compression-based classification algorithm, we assume a binary classification problem. Given a set of training data of a binary classification problem, the algorithm works as follows. First, it builds two compression models M_+ and M_-, one from each class, using the PPM algorithm with a k^{th} order Markov model. For each new instance $X = x_1 x_2 \ldots x_d$ of length d that needs to be classified, the average number of bits required to encode the instance is computed using the two compression models M_+ and M_-, respectively, as follows:

$$B(X|M_+) = -\sum_{i=1}^{d} \log p(x_i|x_{i-k}^{i-1}, M_+)$$

$$B(X|M_-) = -\sum_{i=1}^{d} \log p(x_i|x_{i-k}^{i-1}, M_-).$$

The classification of X is determined by the model that provides a better compression rate:

$$\begin{aligned} c(X) &= \arg\min_{c\in\{+,-\}} B(X|M_c) \\ &= \arg\min_{c\in\{+,-\}} -\sum_{i=1}^{d} \log p(x_i|x_{i-k}^{i-1}, M_c). \end{aligned}$$

The pseudocode for the algorithm is given in Algorithm 1.

Algorithm `PPM Classifier`
Input: Training set $\mathbf{T} = \mathbf{T}_+ \cup \mathbf{T}_-$, test set $\mathbf{P} = \{X_1, \ldots, X_n\}$, and the order of the Markov model in PPM, k
Output: Classification $c(X_i) \in \{+, -\}$ of $X_i \in \mathbf{P}$, for $i = 1, \ldots, n$.
$M_+ \leftarrow$ `CreatePPM`($\mathbf{T}_+$);
$M_- \leftarrow$ `CreatePPM`($\mathbf{T}_-$);
forall $X \in \mathbf{P}$ **do**
  $Bits(X, M_+) = ComputeBits(X, M_+)$;
  $Bits(X, M_-) = ComputeBits(X, M_-)$;
  $c(X) = \arg\min_{c\in\{+,-\}} Bits(X, M_c)$;
end

Algorithm 1: PPM-based classification

The PPM-based classification model has been extensively studied in the spam filtering domain [1]. Its performance in spam filtering is very promising. In this paper, we investigate its applicability to the problem of malware detection. Note that the applicability of compression-based classification algorithms to real classification problems may be hindered by the large demand on memory resources. The space requirement of the compression model increases exponentially with the order of the underlying Markov model, since essentially any context of length between 0 to the max order must be stored. In addition, the classification model is built at the character level, and thus provides no insights of the hidden (malware and spam) patterns.

4. EXPERIMENTAL SETUP AND PRELIMINARY RESULTS

In this section, we present the details of the setup of our experiments and our preliminary results on a collection of benign code and malware code executable on Windows XP plafforms. Note that the technique presented is general enough to be applicable to any platform environment.

4.1 Data Source

We collected a set of 2027 distinct Windows EXE and DLL files from the "Program Files" and "system32" folders on standard Windows XP machines, among which 452 (22%) were randomly selected and used for training and 1575 (78%) were used for testing. We also collected 1434 malicious code from the web site VX Heavens (http://vx.netlux.org), among which 439 (30%) were randomly chosen and used for training and 995 (70%) were used for testing. The collection of malicious code includes four types of malware: backdoor, worm, Trojan, and virus. We converted each executable to hexadecimal codes using a standard Unix hexdump utility. All headers and white spaces are removed.

4.2 Preliminary Results

The size of the executables in the training set varies from 431 bytes to about 5.5M bytes. Since memory usage increases polynomially as the size of code instances increases, we decided to use a subset of each instance in the training set to build the two compression models. We setup and ran three experiments. In the first experiment, we used only the first 2000 bytes in each hexadecimal code in the training data to build the two compression models for classification. In our second and third experiments, we used the first 10,000 bytes and 50,000 bytes in each instance in the training data, respectively.

We present our experimental results as ROC curves. The ROC curves in this paper are represented using the true positive rates and the false positive rates. ROC curves are typically used to show the tradeoff between sensitivity (true positive rate) and specificity (1- false positive rate). The total area under a ROC curve (AUC) is also commonly used as a metric to compare binary classifiers. Figures 1—3 show the ROC curves of the outcome of our three experiments. We also list, in Table 2, the accuracy, true positive rate (TPR), false positive rate (FPR) and AUC values from the three experiments in which 2000, 10,000 and 50,000 bytes of each executable in the training set are used for building the compression model, respectively.

Table 2: The accuracy, true positive rate (TPR), false positive rate (FPR) and AUC values from the three experiments in which 2000, 10,000, and 50,000 bytes in each training sample are used for training, respectively.

	Accuracy	TPR	FPR	AUC
Exp. 1 (2KB)	0.950	0.983	0.070	0.956
Exp. 2 (10KB)	0.961	0.927	0.017	0.955
Exp. 3 (50KB)	0.966	0.938	0.016	0.961

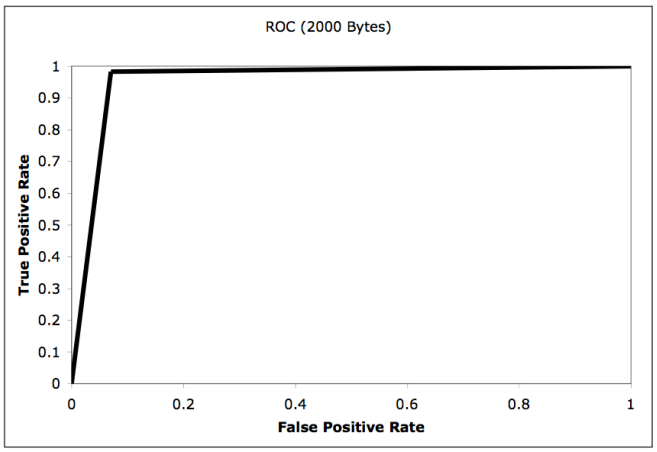

Figure 1: ROC curve for the first experiment in which only the first 2000 bytes are used for building the compression models.

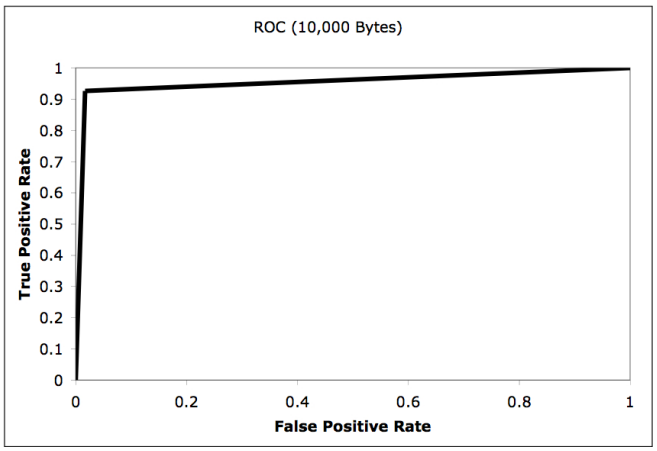

Figure 2: ROC curve for the first experiment in which only the first 10,000 bytes are used for building the compression models.

4.3 Discussions

In our experiment, we used a training set of a much smaller size compared to the size of the test set, thus some of the malicious code was not available for training. As a matter of fact, some malicious code in the test set is fundamentally different than the one our model was built on. This allows us to test the ability of the PPM-compression based classification algorithm to identify unknown malware or variants of the existing malware, for example obfuscated malware, that are not available for training. Our experimental results show that for less than 0.02 false positive rate, we can achieve close to 0.94 true positive rate. And if a false positive rate of 0.07 is tolerable, we can achieve more than 0.98 true positive rate.

There are several reasons that the compression-based classification algorithms work well in the domain of malware detection. First, compression-based classification algorithms form the decision boundary of a classification problem based

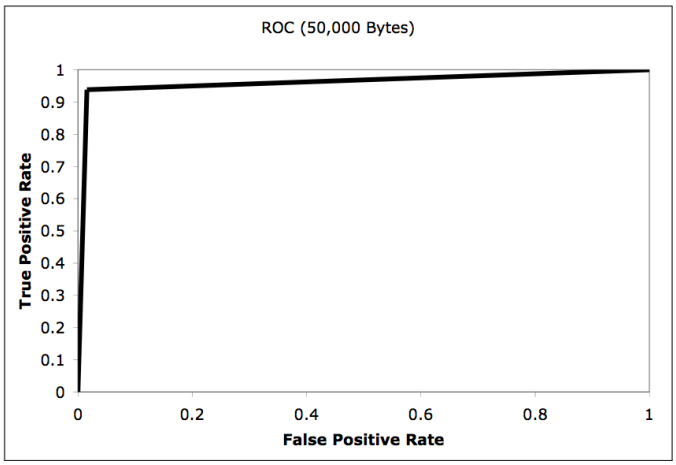

Figure 3: ROC curve for the first experiment in which only the first 50,000 bytes are used for building the compression models.

on the average number of bits used for encoding the executables. The idea is similar to classification by minimum cross-entropy [1] since the average number of bits required for encoding an instance is essentially the cross-entropy of the instance estimated using the PPM compression models. Second, by taking the raw executables as input, this type of learning algorithms can make use of full features of executables, some of which may be lost during feature pruning in standard learning processes. Third, techniques that adversaries use to make their malicious code unrecognizable to signature scanning can, in fact, help compression-based learning algorithms detect the malicious code. For example, one of the most common approaches to creating obfuscated viruses is code-reordering through the insertion of randomized jump instructions. The compression-based classification algorithms can easily discriminate this pattern from instruction sequences of legitimate code by statically ruling out the likelihood of observing the jump instructions in certain context. Another obfuscation technique that is often used is packing, which encrypts or compresses the data and code sections of an infected executable, and inserts a decryption routine that executes at runtime to recover the original data and code sections. This technique also makes the infected executables look different to the compression-based algorithms at the bit/character level. For the same reason, semantic nops (junk instructions that do not affect program behavior) can also be caught easily by the compression-based classifiers.

5. CONCLUSIONS AND FUTURE WORK

We propose to apply compression-based classification algorithms to detecting malicious code. These type of learning algorithms take the raw data, for example executable code, as input and form a decision boundary based on the average number of bits required to encode the data. For a binary classification problem, two compression models are built using the prediction by partial matching algorithm, one from each class. A new instance is classified according to the model that produces a better compression rate. We investigated the applicability of compression-based learning algorithms by running three sets of experiments on a collection of benign and malicious code executables found on the Windows platform. The preliminary results are very promising. Also note that this approach can be applied to data collected from any other platform.

The compression-based algorithms have some limitations. First, the applicability of these type of techniques to real applications is limited due to the fact that the underlying statistical compression models consume large amounts of resources. Second, compression-based classification models do not explicitly identify concept drift and adapt to it. Moreover, compression-based learning models are insensitive to subtle differences between different classes when the number of discriminative features is small [16].

Another area of exploration is to determine what type of methods a potential adversary could use to alter their malware in order to trick the compression-based classifier into assigning false-negatives. Attacks using words known to be considered good by a classifier are common in spam filtering and compression-based spam filters are known to be susceptible these attacks [8]. We plan to perform further analysis to determine what types of attacks could be successful in the compression-based malware classification domain.

In the future, we plan to enhance the compression-based classification models by pruning the underlying compression models and eliminating the features that are less important in defining the decision boundary of a classification problem. We also plan to compare our model with other state-of-art models that have been devised to detect malware. Furthermore, we plan to design a similar compression-based learning algorithm that is incremental and adaptive.

6. ACKNOWLEDGEMENT

The authors would like to thank Zach Jorgensen for his help, valuable ideas, and comments.

7. REFERENCES

[1] A. Bratko, G. Cormack, B. Filipič, T. Lynam, and B. Zupan. Spam filtering using statistical data compression models. *Journal of Machine Learning Research (JMLR)*, 7:2673–2698, December 2006.

[2] M. Christodorescu, S. Jha, J. Kinder, S. Katzenbeisser, and H. Veith. Software transformations to improve malware detection. *Journal in Computer Virology*, 3:253–265, 2007.

[3] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32(4):396–402, 1984.

[4] J. Cleary and I. Witten. Unbounded length contexts of ppm. *The computer Journal*, 40(2/3):67–75, 1997.

[5] G. Cormack and R. Horspool. Data compression using dynamic markov modeling. *The Computer Journal*, 30(6):541–550, 1987.

[6] M. Drinic, D. Kirovski, and M. Potkonjak. Ppm model cleaning. In *DCC '03: Proceedings of the Conference on Data Compression*, page 163, Washington, DC, USA, 2003. IEEE Computer Society.

[7] E. Frank, C. Chui, and I. Witten. Text categorization using compression models. In *IEEE Data Compression*

Conference (DCC-00), pages 200–209. IEEE CS Press, 2000.

[8] Z. Jorgensen, Y. Zhou, and M. Inge. A multiple instance learning strategy for combating good word attacks on spam filters. *Journal of Machine Learning Research (JMLR)*, 9:1115–1146, June 2008.

[9] S. Josse. Secure and advanced unpacking using computer emulation. *Journal in Computer Virology*, 3:221–236, 2007.

[10] J. Kephart, G. Sorkin, W. Arnold, D. Cheese, G. Tesauro, and S. White. Biologically inspired defenses against computer viruses. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI95)*, pages 985–996. Morgan Kaufman, 1995.

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

[12] C. Nachenberg. Us patent no. 5,826,013: Polymorphic virus detection module, 1998.

[13] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 289–300, Washington, DC, USA, 2006. IEEE Computer Society.

[14] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49, Los Alamitos, CA, 2001. IEEE Press.

[15] S. K. Udupa, S. K. Debray, and M. Madou. Deobfuscation: Reverse engineering obfuscated code. In *WCRE '05: Proceedings of the 12th Working Conference on Reverse Engineering*, pages 45–54, Washington, DC, USA, 2005. IEEE Computer Society.

[16] I. Witten. Applications of lossless compression in adaptive text mining. In *Proceedings of the 34th Annual Conference on Information Sciences and Systems (CISS-00)*, New Jersey, 2000.

[17] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. In *Communications of the ACM*, pages 520–540. June, 1987.

[18] Y. Zhou, M. S. Mulekar, and P. Nerellapalli. Adaptive spam filtering using dynamic feature space. In *ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 302–309, Washington, DC, USA, 2005. IEEE Computer Society.