A novel compression-based approach for malware detection using PE header

Conference Paper · May 2013

DOI: 10.1109/IKT.2013.6620051

2013 5th Conference on Information and Knowledge Technology (IKT)

A Novel Compression-Based Approach for Malware Detection Using PE Header

Zahra Khorsand, Ali Hamzeh
Computer Science and Engineering Dept., ECE School
Shiraz University
Shiraz, Iran
{khorsand,ali}@cse.shirazu.ac.ir

Abstract— **Recently, various invasions of malwares and their incurred damages threaten the usability and privacy of computer systems. Due to the dramatic growth of these attacks, malware detection has been brought up as an important topic in computer security. Since traditional signature based techniques embedded in commercial anti-viruses have failed to detect new and obfuscated malwares, machine learning algorithms have been used to detect behavior patterns of malwares via features extracted from programs. In this paper, we propose two methods based on compression models as heuristic malware detection techniques. The main advantage of our approach is eliminating the feature extraction step which is vital and expensive for machine learning based approaches. Also, this study focuses on solving the problem of memory space requirement of these models by applying more compatible input data without changing the raw nature of the programs or the compression algorithm. To evaluate the effectiveness of the proposed methods, several experiments are conducted. The experimental results of both methods show promising improvement of accuracy to support the main idea.**

Keywords-component; malware detection; machine learning; data compression; classification

I. INTRODUCTION

Any malicious codes or contents which specially designed to interfere with normal functionality of computer systems, steal sensitive information without user's consent or gain illegitimate access to private computer systems or networks are considered as malwares [1]. Due to the serious economic damages caused by malicious attacks, malware detection has been considered as an important challenge in computer security communities.

Currently, commercial anti-malwares have been widely used. These malware detectors commonly use a signature-based scan which looks up for a signature – unique byte-sequences of the program –in their databases. These scans are fast and accurate; however the generation process of a unique signature for every new malware needs a significant amount of effort on malware analysts. Unfortunately, this effort can be easily wasted by a plain modification in the code applied by a simple obfuscated technique. This problem lies in the high dependency of the signature with the byte patterns of the codes.

To deal with this problem, several machine learning methods have been employed to learn the behavioral patterns of programs instead of byte patterns. Therefore, the structural data called features which reflect programs' behaviors are commonly used in these methods. Indeed, the features are reduced representation of input data that are relevant to learning task. Thus, through transforming input data into features, most of the input information is ignored. Moreover, in most cases, some error-prone preprocessing such as unpacking and disassembling should be applied prior to these transformations. These problems appear since machine learning algorithms expect structured data as an input, while, in malware detection field; they encountered executable files which are unstructured.

In the area of text categorization and spam filtering - which also work with unstructured text documents - these difficulties could be overcome by the aid of compression based models [2, 3, 4, 5, 6, 7].Since no prior assumptions about language and data encoding are considered in these models, they can be employed as predictor or classifier for any unstructured data sequences including text documents or executable programs. The most remarkable advantage of compression-based methods over machine learning and data mining techniques is that they make full use of source data in byte/character-level. Moreover, they are easy to apply because they are constructed from raw executable files, without any need of fallible and time consuming preprocessing. However, the main reason of lower tendency towards compression based methods is their large memory requirements to maintain compression models. For instance, in the previous attempt to apply a compression based classifier to detect malwares, in training phase, instead of using the entire content of the programs to construct programs models, the first byte block of executable programs were used (after omitting the header and null bytes) [8].Although this study show promising result, using a subset of the programs leads generating inaccurate models. Even if a sufficient memory space has been provided to maintain the models constructed from the entire contents of the programs, slow running time is hindrance.

This study concentrates on alleviating this problem by employing the characteristics of programs as well as compression algorithms. For this purpose, we proposed two different compression based methods using effective input data to feed the models. In this study, we used PPM algorithm which is an established compression algorithm on natural languages texts due to its very high compression ratio [9]

The first proposed method is based on modeling the header of the executable programs instead of the entire content of them, which is not feasible due to the size of the models produced. One of the most important parts of PE file is its header. Although, this section of executable files is small, the contents are significant to detect malicious features. Indeed, one of the prevalent approaches in malware analysis is investigating the header of the PE files, since the header provide an image of executable file and contains structural features which change along with the body contents of the program. In addition, due to the small volume of the header section, the compression algorithms are able to model the header completely without any need to omit some parts which causes constructing inaccurate models. The results show that this approach has superior performance than the PPM models constructed from the body content of programs, especially when there exists some memory space constraints.

The second method focuses on using a new representation of programs which is more effective than binary representation where obtaining this representation does not involve any error prune processes. We used ASCII-character sequences named string within the programs as an input for the models which fulfill our ideal condition. The experiments indicate that this representation can construct more compact and accurate models comparing to the binary content of programs.

The rest of the paper is organized as follows: In section II, we provide a brief overview of related works. We discuss about the details of compression based classifier in Section III. In section IV, we describe our proposed methods in details. We report the results of experiments in Section V. Finally, we conclude our research and discuss the future work.

II. Related Works

While using compression models in text categorization is a standard approach, the idea of using this technique in spam filtering and malware detection has still been under investigation [7, 8, 10].

Most relevant work to our study is the work byY.Zou and M.Inge who proposed the compression based classifier as a malware detector for the first time [8]. They model Benign and malware programs using the Prediction by Partial Matching (PPM) algorithm [11] which is one of the adaptive lossless compression algorithms. So, the concept class of a new executable program is determined based on the model which compresses the program with the higher compression ratio. To solve the problem of memory requirement for models, they decide to omit the header of executable files and use a subset of body content of each program. They achieve remarkable results and state that this technique can efficiently detect new malwares. In their next research, they attempt to improve this technique by changing the input data into op code sequence of programs [10]. By this transformation, they model the entire benign/malware programs using their op code sequence. In this approach, the running time improves by 25% and their results, especially the true negative rate, is improved. However, obtaining op-code sequence of programs is an error-prone preprocessing and undermines the basic idea of using the compression models.

In most malware detection studies, programs are converted into a reduced representation such as most relevant byte n-grams [12], op code sequences [13], interpretable strings [14], and API call sequences [15]; and then, a few machine learning techniques including feature selection and classification processes this new representation of programs.

For instance, J.Z. Kolter and M.A. Maloof in [12] used binary content of programs which is similar to our method in this aspect. They first generate all n-grams of all programs in their dataset, and then, selected top n-grams with the highest information gain [16] as binary features. They applied several machine learning classifiers such as instance-based learner[17], Naive Bayes [18], support vector machine [19], decision trees [20] and boosted version of each one on their resulted dataset [21]. They concluded that this methodology can be used to detect previously undiscovered malwares.

III. A Compression Based Classifire

Although compression based classification is not a straightforward approach for a classification task, due to the nature of the models created by this technique, they can be employed as a probabilistic text classifier based on character-level data in the wide range of source data such as text documents, biological sequences and binary sequences [2, 5, 7, 8]. These models are fast to construct without any need of time consuming preprocessing such as tokenizing or word-based feature extracting. In this study, PPM algorithm has been used for modeling, among several compression algorithms:

A. Prediction by Partial Matching (PPM)

One of the well-known data compression algorithms is prediction by partial matching (PPM).Due to its strong performance, the PPM algorithm has been widely used as a standard lossless compression method for natural language texts since its introduction in mid-1980s [11].

PPM method is based on an encoder contains a statistical model which is designed by arithmetic coding protocols. The model stores a map that keeps the frequency occurrence of each symbol. The encoder encodes each input symbol with probability provided by this map.

Basically, the model estimates the conditional probability of next symbol by considering its immediately preceding N symbols. The map stores every possible strings of 0, 1, ..., N symbols, in other words, it maintains every possible 0 to N-order context. Each entry of the map for string s keeps the count of every symbol x that immediately follows s. Hence, the conditional probability of x in the context s is $\frac{C(x|s)}{C(s)}$, where $C(x|s)$ is the number of times that x follows s and $C(s)$is the frequency occurrence of s in the training text. Using these contexts' statistics, the probability distribution of training text which is required by arithmetic encoder will be achieved.

There are two practical issues in constructing models. First, the map which is kept by model can grow very large; hence in real applications, N is small, typically 3 to 7. Next problem arises when the algorithm encounters a symbol which is novel to the current context. This problem which is named zero-frequency problem has been solved in PPM algorithm using the idea of partial matching instead of exact matching.

The algorithm works as follows: the model starts with an empty map and every time the algorithm sees a new context, encounters a zero-frequency symbol; it tries to find a match for a string with one symbol less than the input string. This process repeats until a match for the truncated string is found. However, the probability is discounted by a fixed amount which is called escape probability. In the case where the process reaches 0-order context and a match is not found, the algorithm assumes the distribution of symbols over the text is uniform and assigns a fixed probability to the symbol.

The algorithm keeps the map updated for the frequency of each symbol which appears after each context. In the case of 0-order contexts, the counts are just the number of times which each symbol has been repeated within the entire text. The process of constructing the map is fast because PPM algorithm is a one-pass compression method which means the process is done in a single pass over the text.

B. Classification via Compression

The basic idea of using compression based classifiers arises from the fact that compressing a message is only possible when some statistical patterns occur during the message. Therefore, the more number of similar patterns appear in two messages, it is more probable that these two are in the same class[22, 23, 24].

The compression based classifier is constructing from the models. Each model is representing one class of dataset. As in malware detection case, the algorithm first builds two models; the first one is constructed from malware training set and the second is constructed from benign training set .In formal description, where minimum cross-entropy were used as classification criterion[2, 3],given $X = x_1x_2 \dots x_d$ which contains d symbols, average number of bits required to encode X using two k-order PPM model M_+ and M_- is computed as follows:

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

$$P(X|M_-) = -\sum_{i=1}^{d} \log p(x_i|x_{i-k}^{i-1}M_-). \quad (2)$$

For classifying a new file, the file will be compressed using both two models. The class of the file is determined by the model that can compress it better as follows:

$$\text{Class(X)} = \arg \min_{c\in\{+,-\}} P(X|M_c)$$

$$= \arg \min_{c\in\{+,-\}} - \sum_{i=1}^{d} \log p\left(x_i \middle| x_{i-k}^{i-1}, M_c\right). \quad (3)$$

IV. Our Proposed methods

As mentioned earlier, the main disadvantage of compression based methods is their large storage complexity. Due to the adversary nature of malwares, pruning techniques which apply to compression models lose their effectiveness. To address this problem, this study proposes two different methods in order to detect malwares. Considering both structure of Portable Executable (PE) files and properties of PPM models, we build compression-based classifiers which leads to more efficient scheme of malware/benign programs. In the first method, the PPM models are constructed from the header of programs instead of their body content. The second method works with string occurs in the body of programs which represent more compact information about programs. Remark that we do not make any change in the PPM compression algorithm; we simply provide more suitable compact data to train these models. Following sections introduced the proposed methods in more details.

A. Header Classifire

The header classifier uses header region of PE files as a source for training PPM models. The PE file is the format of executable binaries, object codes and DLLs which is used in Windows operating systems. A PE file has two different regions named header and body. The header contains the information needed for the Windows OS loader to manage the file. It contains several parts; each has its own structure. Some of malware analysis methods focus on detecting abnormalities or extracting structural features from these parts. While most of them cannot exploit all header information efficiently, compression based models is one of the best choices to utilize these information.

Due to the memory problem of compression algorithms, the models trained with small datasets which contains 500 to 2000 files. Fortunately, the models become relatively well-trained despite their small train set. In practice, due to the fact that the models cannot be constructed via all binary content of files, a small subset of files is used for modeling, though using this trick has its own limitation. As we know, in a malware, a small number of codes are malicious and the rest are benign, therefore by cutting the programs into a small subset, it is probable that these parts were not seen. Using these misleading subsets in malware model generate a model which is similar to the benign one and consequently, cause to decrease the effectiveness of models.

With this interpretation, choosing an appropriate subset of files is a key point to gain an efficient model. Applying the header as a small but important unit of an executable files in compression models is a good option. Due to small size of headers, all content of headers can participate in modeling therefore, it does not have the limitations of the models constructed from a subset of the programs.

Furthermore, since the header of files is commonly contains 200 to 500 meaningful bytes (all bytes except null ones), we can gain more accurate models by increasing the order of PPM models, which is not feasible for large input data.Fig.1 shows the algorithmic view of the first method.

B. String-based Classifire

By changing the input content of our models to a more effective one, we can improve the accuracy of models. Note that this transformation should not involve any error-prone preprocessing such as disassembling. The proposed transformation, which also satisfies above condition, is using all strings exist within the executable files. By using this representation of programs as an input, the alphabet size of PPM models can be reduced to 128 symbols, which in this case the models can be simpler. In fact, by this reduction in symbol

```
Algorithm: headerClassifier ($PPMH_+$,$PPMH_-$ ,TS, DHS)
Input:  $PPMH_+$: Malware Header PPM Model
        $PPMH_-$: Benign Header PPM Model
        TS: Test set of executable instances
        DHS: default header size
Output: CL (TS)$\in \{+,-\}^n$: classification of TS instances
        which is + for malwares and – for benign instances; n
        is the size of Test set.

1   for each instance ts in TS does
2       if ts is PE file then tsh = getHeader (ts)
3       else tsh = getblock (ts, DHS) endif
4       ths' = deleteNullBytes (ths)
5       mb = Bits (ths', $PPMH_+$)
6       bb = Bits (ths', $PPMH_-$)
7       if mb < bb then CL (ts) = +
8       else CL (ts) = −endif
9   end
```

Figure 1. Header Classifire Algorithm

```
Algorithm: stringbasedClassifier ($PPMS_+$,$PPMS_-$ ,TS,
        DBS, DSS)
Input:  $PPMS_+$: Malware Header PPM Model
        $PPMS_-$: Benign Header PPM Model
        TS: Test set of executable instances
        DBS: default block size
        DSS: default string size
Output: CL (TS)$\in \{+,-\}^n$: classification of TS instances
        which is + for malwares and – for benign instances; n
        is the size of Test set.

1   for each instance ts in TS does
2       tss = getAllStrings (ts, DSS)
3       tss' = getBlock (tss, DBS)
4       mb = Bits (tss', $PPMS_+$)
5       bb = Bits (tss', $PPMS_-$)
6       if mb < bb then CL (ts) = +
7       else CL (ts) = −endif
8   end
```

Figure 2. String based classifire

list, a natural pruning will be applied to the models. Consequently, in the same memory storage space, the string models are more accurate than the binary ones, since more useful information can be used in these models.

It is important to remark that in this representation, we describe a string as a sequence of ASCII characters with the least size of 4 bytes. We do not filter the strings with a natural language word collection but we believe that if this filtering applies to strings, the classifier will be improved. Fig. 2 presents the detail of the string based classifier.

V. Experiments And Results

The performance of the proposed methods is assessed in three steps; (1) first, a comparison between header classifier and string based classifier and original method proposed in [8], which we alternatively named it byte-based classifier, is discussed and then (2) a comparison between the string and original method is given and (3) finally, a comparison between header classifier along with string/byte based classifier and Kolter's method [12] are given. The statistics of the datasets as well as the details of the setup of these experiments and their results are presented in the following sections.

A. The Dataset

In this study, as described, there is no restriction about the type of files. Indeed, the dataset contains mixture of standard PE and non-PE files. Besides, the files can be packed or non-packed. Even in the header classifier, which involves extracting the header of PE files, in the case where the file has no PE format, the first byte-block of file was used instead of the header block.

1) The Benign Dataset:

The benign dataset used in the experiments, consists of 850 "EXE" and 750 "DLL" files collected from "Program Files" and "System32" folder of a standard Windows XP machine, which randomly partitioned into an 1100-file set for training set and a 500-file set for test set. The sizes of benign files vary from 1 KB to 15 MB.

2) The Malware Dataset:

The malware dataset is obtained from "VX Heavens Virus Collection" database downloaded from: http://vx.netlux.org. Where 1600 malwares are randomly selected from all available malwares in the database which include different types such as backdoor,constructor,worm,Trojan,virus and hack tools with their sizes ranging from 1 KB to 11 MB. Similar to the benign dataset; the collection is randomly partitioned into 1100-file and 500-fileas training and test set, respectively.

B. The Experimental Environment

All the experiments were conducted under the environment of Windows 7operating system with Intel(R) Core(TM) i5-2410M@2.30 GHz processor and 4 GB of RAM. The java source code of PPM algorithm implementation which is used in [5] is available in: http://www.cs.technion.ac.il/~ronbeg/vmm. The heap size for all the experiments was set to 2GB memory space.

C. Comparison between header classifier and string/byte based classifiers

The main aim of this set of experiments is to assess the effectiveness of header classifier compared to the string/byte based classifier. All three methods were trained by the same training set described in section 5.1 while, at most,2 GB heap size was used to maintain the trained PPM models.

As stated in the algorithm shown in Fig.1, the header PPM models are trained with the header of PE files and in the case where the file is not a PE file; the first byte-block of file is used. To choose the byte-block size, the size of headers of the collection is inspected. The header sizes were different in PE files; the most frequent header sizes were 512, 1024 and 2048 bytes. We used 1024 bytes as the size of block when the file is a non-PE file. After removing null bytes, the header sizes with useful information commonly range from 200 to 500 bytes.

To train the PPM models by original compression based method, the header of PE files was removed and the first byte-block of the body of files, after removing the null bytes, was

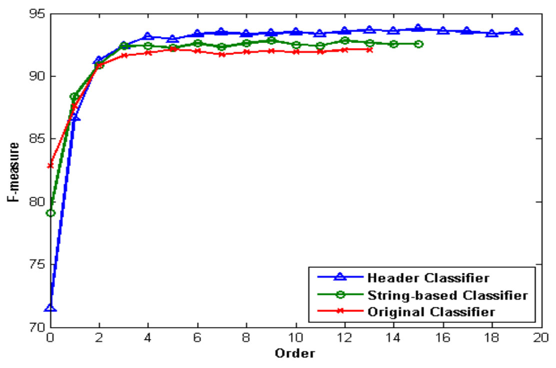

Figure 3. F-measure results for detecting malwares vs. benign programs as order of PPM models increases in which the models constructed from reading 512-bytes block from each dataset files

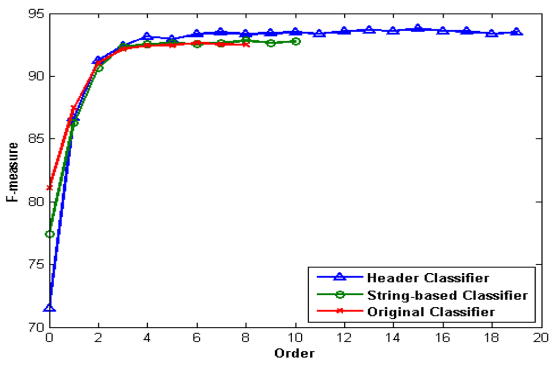

Figure 4. F-measure results for detecting malwares vs. benign programs as order of PPM models increases in which the models constructed from reading 1024-bytes block from each dataset files

used to train PPM models. The size of this block in the first set of experiment is 512 bytes and 1024 bytes in the second one. Finally, for the string based classifier, as shown in Fig.2, the default block sizes were set to 512 and 1024 bytes for the first and second set of experiments, respectively.

The f-measure related to the three PPM based classifier (header classifier and string/byte based classifier) by changing the order of PPM models is reported in Fig.3-4. All the three methods examined different PPM-orders until the memory fails to store the models. As it was expected, increasing the order of the PPM models in the methods causes improvement of the f-measure which represents the efficiency of method. In the orders which the curve are not drown, the PPM models of the classifiers grow larger than the heap size, therefore measuring the f-measure of the classifiers in the next orders are impossible. As shown in the Fig.3-4, the maximum order for the header, string and original classifier is 19, 15 and 13 in the first experiment and 19, 10 and 8 in the second one.

As figures3and 4demonstrate, the header classifier has superior f-measures compared with the string/byte based classifier. This observation suggests that the header information is more significant for the classification when the small subset of files is provided. Another conclusion is that while the input data for header models is very smaller than the inputs of the other two models, modeling the headers as a complete unit of programs is one of the reasons to have more accurate models. Besides, small input data also leads to have more accurate PPM models by increasing the order of PPM model; while it is not applicable in the string/byte based classifiers

D. Comparsion between the string and byte basedclassifers

To compare the string and byte based classifiers, 6 set of experiments with the same preprocessing steps mentioned in previous section were conducted; each method used 512, 1024, 2048, 3072, 4096 and 5120 byte-blocks. The f-measure of string/byte classifiers, by increasing the order of the PPM models is presented in figures 5 to 10.

These figures indicate that the string based PPM models are more accurate than the byte based ones as the order of the models increases. Notice that when the order of models are 0-2, the original method performs better than the string based method, however in this range of orders, both of the methods are not trained completely yet, which yield relatively low f-measures (between 70% to 90 %).As shown, the volume of byte models reaches earlier than the string based models to 2 GB heap size. This evidence suggests that the string input creates more compact models. One reason to this observation is decreasing the alphabet size of PPM models from 256 to 128 symbols. In general, the curves prove that string based method provide more compact, but useful input data for PPM models than the original method which leads creating more efficient models than the byte based models.

E. Comparison between Kolter's method and compression based methods

To investigate the efficiency of compression based approach compared to the machine learning technique, the method given in [12]is implemented which is based on the byte n-grams. The common property between compression based method and Kolter's work is that this method also works with the raw binary executable files without any transformation to other representations which is called error-prone preprocessing. As suggested in [12], the size of n-grams is set to 4. To have fair comparison with the compression methods, the same constraint of memory space (2GB) is considered for Kolter's method.

Due to this restriction, the algorithm cannot be applied to all 2200 files of the training set (1100 benign and 1100 malware) to generate the 4-grams, therefore we used 30'000 first bytes of each file; this volume is at least 6 times larger than the contents which used in modeling of the compression based method. In the next step, 500 most relevant n-grams are selected by measuring their information gain as binary features. In the classification phase, three well-known machine learning classifier are applied: decision tree [20], SVM [19] and adboostM1 [25]. Java implementations of these classifiers are provided in Waikato Environment for Knowledge Acquisition (WEKA) [26].

The best results achieved by the four methods and their relative parameter settings are presented in the tables I and II. As the results show, in their best parameter settings, each three compression based methods outperform the Kolter's method when using the same memory space constraint.

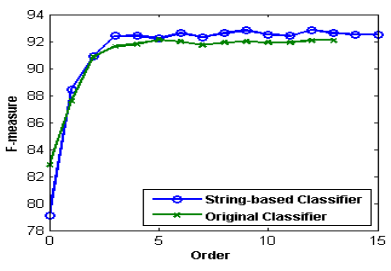

Figure 7. F-measure results for detecting malwares vs. benign programs as oreder of PPM models increases in which the models constructed from reading 512-bytes block from each dataset files

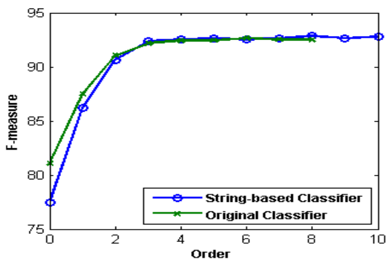

Figure 8. F-measure results for detecting malwares vs. benign programs as oreder of PPM models increases in which the models constructed from reading 1024-bytes block from each dataset files

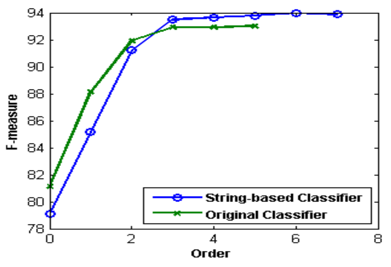

Figure 9. F-measure results for detecting malwares vs. benign programs as oreder of PPM models increases in which the models constructed from reading 2048-bytes block from each dataset files

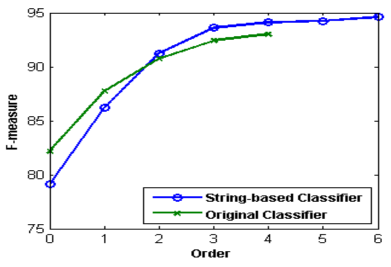

Figure 10. F-measure results for detecting malwares vs. benign programs as oreder of PPM models increases in which the models constructed from reading 3072-bytes block from each dataset files

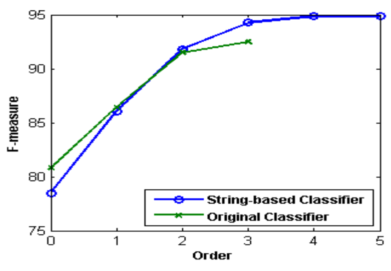

Figure 5. F-measure results for detecting malwares vs. benign programs as oreder of PPM models increases in which the models constructed from reading 4096-bytes block from each dataset files

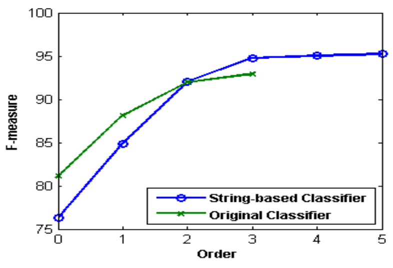

Figure 6. F-measure results for detecting malwares vs. benign programs as oreder of PPM models increases in which the models constructed from reading 5120-bytes block from each dataset files

Although it may well be true that by comparing these methods, we cannot extend this result to all the other machine learning methods, it is important not to overlook that this results indicate the compression based methods can be more effective than some machine learning techniques for their simplicity and efficiency in malware detection domain.

TABLE I. PARAMETER SETTINGS FOR COMPARING THE FOUR METHODS IN THEIR BEST SETTINGS WITH THE CONSTRAINT OF 2GB MEMORY SPACE

Method	**Parameter Settings for the Best Results**				
	Max byte[a]	***Ngram size***	***PPM Order***	***Feature Size***	***Classifier***
Kolter's	30000	4	-	500	SVM
Original	3072	-	4	-	PPM Based
Header based	header[b]	-	15	-	PPM Based
String[c] based	5120	-	5	-	PPM Based

a. Max byte read from each file of dataset

b. default header size is 1024 byte

c. default string length = 4 character

TABLE II. RESULTS OF THE FOUR METHOD IN WHICH USE THEIR BEST PARAMETER SETTINGS WITH THE CONSTRAINT OF 2 GB MEMORY SPACE

Method	**TP rate**	**FP rate**	**TN rate**	**FN rate**	**Accuracy**	**Precision**	**Recall**	**Specificity**	**F-measure**
Kolter's	0.92	0.11	0.88	0.08	90.2	88.8	92.0	88.4	90.4
Original	0.94	0.09	0.91	0.05	92.9	91.3	94.8	91.0	93.0
Header	**0.96**	0.09	0.90	**0.03**	93.6	91.1	**96.6**	90.6	93.8
String	**0.96**	**0.05**	**0.94**	0.04	**95.2**	**94.5**	96.0	**94.4**	**95.2**

VI. Conclusion And Future works

In this study, our research focus on developing a malware detector based on compression algorithms. The idea arises from the fact that compression algorithms commonly work with plain texts as messages; they make full use of the contents of messages. Since the program files are raw unstructured data type like texts, this approach is more suitable than the machine learning techniques which usually work with structured data types called features .In these approaches, raw data must be converted to features in order to be more useable in a learner classifier. Therefore, during this conversion, most of the information will be ignored. Moreover, these techniques commonly involve some error-prone preprocessing such as disassembling and unpacking.

To avoid these problems, we apply compression based classifiers based on PPM models. While the models are fast to construct and easy to apply, they suffer from the memory space complexity. To address this problem, we proposed two methods which exploit the properties of PPM models and the structure of programs.

In the first method, we apply PPM to the header part of programs instead of their entire content. The header of the programs contains important information which is highly relevant to the class of programs (benign/malware).Besides, due to small size of input data for the PPM models, we can increase the order of PPM models, and consequently more accurate models will be achieved.

In the second method, we construct the PPM models from the string content of a program. With this transformation, a new representation of programs will be provided to models, which is more compact than the original representation (byte sequences). Since the strings construct from ASCII-character sequences, the alphabet used in PPM models decrease from 256 to 128, therefore a natural pruning will be apply to models.

The experiments demonstrate that both proposed methods outperform the byte-based method since both of them provide more significant data to construct the PPM models without any needs of error-pruned transformations. Besides, we can conclude that the compression based technique can compete perfectly with the standard machine learning approaches in malware detection domain; indeed this approach is an interesting avenue which needs further investigation in future. Therefore, we are going to develop our methods in the following directions: 1) Examine other compression algorithms in the structure of compression based classifiers to find the best one for malware detection field. 2) Investigate other classification criteria when using the compression models which is more suitable with the compression algorithm. 3) Improve the compression based classifiers by finding more appropriate input data to construct the models.

References

[1] Nash, Troy. An Undirected Attack Against Critical Infrastructure. Technical Report, US-CERT Control Systems Security Center, 2005.

[2] Frank, Eibe, Chang Chui, and Ian H. Witten. "Text categorization using compression models." (2000).

[3] Teahan, William John. "Text classification and segmentation using minimum cross-entropy." (2000).

[4] Teahan, William, and D. Harper. "Using compression-based language models for text categorization." In *Proceedings of 2001 Workshop on Language Modeling and Information Retrieval*. 2001.

[5] Begleiter, Ron, Ran El-Yaniv, and Golan Yona. "On prediction using variable order Markov models." J. Artif. Intell. Res. (JAIR) 22 (2004): 385-421.

[6] Marton, Yuval, Ning Wu, and Lisa Hellerstein. "On compression-based text classification." *Advances in Information Retrieval* (2005): 300-314.

[7] Bratko, Andrej, Bogdan Filipič, Gordon V. Cormack, Thomas R. Lynam, and Blaž Zupan. "Spam filtering using statistical data compression models." *The Journal of Machine Learning Research* 7 (2006): 2673-2698.

[8] Zhou, Yan, and W. Meador Inge. "Malware detection using adaptive data compression." In *Proceedings of the 1st ACM workshop on Workshop on AISec*, pp. 53-60. ACM, 2008.

[9] Blelloch, Guy E. "Introduction to data compression." Computer Science Department, Carnegie Mellon University (2001).

[10] Gong, Tao, Xiaobin Tan, and Ming Zhu. "Malware Detection via Classifying with Compression." In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pp. 1765-1768. IEEE, 2009.

[11] Cleary, John, and Ian Witten. "Data compression using adaptive coding and partial string matching." *Communications, IEEE Transactions on* 32, no. 4 (1984): 396-402.

[12] Kolter, J. Zico, and Marcus A. Maloof. "Learning to detect and classify malicious executables in the wild." *The Journal of Machine Learning Research* 7 (2006): 2721-2744.

[13] Santos, Igor, Borja Sanz, Carlos Laorden, Felix Brezo, and Pablo Bringas. "Opcode-sequence-based semi-supervised unknown malware detection." *Computational Intelligence in Security for Information Systems* (2011): 50-57.

[14] Ye, Yanfang, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. "SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging." *Journal in computer virology* 5, no. 4 (2009): 283-293.

[15] Shankarapani, Madhu K., Subbu Ramamoorthy, Ram S. Movva, and Srinivas Mukkamala. "Malware detection using assembly and API call sequences." *Journal in computer virology* 7, no. 2 (2011): 107-119.

[16] Hand, David J., Heikki Mannila, and Padhraic Smyth. Principles of data mining. MIT press, 2001.

[17] Aha, David W., Dennis Kibler, and Marc K. Albert. "Instance-based learning algorithms." Machine learning 6, no. 1 (1991): 37-66.

[18] Mitchell, Tom M. "Bayesian Learning." Machine learning 414 (1997).

[19] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20, no. 3 (1995): 273-297.

[20] Mitchell, Tom M. "Decision tree learning." Machine learning 414 (1997).

[21] Schapire, Robert E., and Yoav Freund. Boosting: Foundations and Algorithms. MIT Press (MA), 2012.

[22] Moffat, Alistair. "Implementing the PPM data compression scheme." Communications, IEEE Transactions on 38, no. 11 (1990): 1917-1921.

[23] Cormack, Gordon. "Data Compression Models for Prediction and Classification." *NATO SECURITY THROUGH SCIENCE SERIES D-INFORMATION AND COMMUNICATION SECURITY* 15 (2008): 142.

[24] Knoll, Byron. "Text Prediction and Classification Using String Matching." Department of Computer Science, University of British Columbia.

[25] Freund, Yoav, and Robert Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting." In Computational learning theory, pp. 23-37. Springer Berlin/Heidelberg, 1995.

[26] Garner, Stephen R. "Weka: The waikato environment for knowledge analysis." In Proceedings of the New Zealand computer science research students conference, pp. 57-64. 1995.