Dynamic Classification of Packing Algorithms for Inspecting Executables using Entropy Analysis

Munkhbayar Bat-Erdene, Taebeom Kim, Hongzhe Li and Heejo Lee

Div. of Computer & Communication Engineering
Korea University
Seoul, Republic of Korea
{munkhbayar, ktb88, hongzhe, heejo}@korea.ac.kr

***Abstract*—Packing is widely used for bypassing anti-malware systems, and the proportion of packed malware has been growing rapidly, making up over 80% of malware. Few studies on detecting packing algorithms have been conducted during last two decades. In this paper, we propose a method to classify packing algorithms of given packed executables. First, we convert entropy values of the packed executables loaded in memory into symbolic representations. Our proposed method uses SAX (Symbolic Aggregate Approximation) which is known to be good at large data conversion. Due to its advantage of simplifying complicated patterns, symbolic representation is commonly used in bio-informatics and data mining fields. Second, we classify the distribution of symbols using supervised learning classifications, i.e., Naive Bayes and Support Vector Machines. Results of our experiments with a collection of 466 programs and 15 packing algorithms demonstrated that our method can identify packing algorithms of given executables with a high accuracy of 94.2%, recall of 94.7% and precision of 92.7%. It has been confirmed that packing algorithms can be identified using entropy analysis, which is a measure of uncertainty of running executables, without a prior knowledge of the executable.**

***Index Terms*—Symbolic Aggregate Approximation (SAX), Piecewise Aggregate Approximation (PAA), Entropy Analysis, Original Entry Point (OEP), Packing Algorithms.**

I. Introduction

A. Background on packing malware

Malware not only violates the security and privacy of computer users, it also incurs a significant amount of financial loss. Attackers continuously make their malware harder to detect and analyze Choi *et al.* [1]. Although the development of anti-virus and other malware analysis tools can mitigate this situation to some extent, the evolution of malware, such as the booming growth of polymorphic and metamorphic malware, is making its analysis more and more difficult.

One popular and widely used technique is packing. Using different packing algorithms, attackers can generate a great number of malware options, and can hide the original behavior of the malware, which makes malware harder to be analyzed and to be detected from analysis tools such as anti-malware software. A packer is basically a program that produces a number of data blocks forming a compressed or encrypted version of the original executable, as described Yan *et al.* [2].

Some packers use more sophisticated technique to evade detection. As mentioned above, statistics have shown that over 80% of malware is packed, as demonstrated by Lyda *et al.* [3] and Yan *et al.* [2]. Popular packers include ASPACK, FSG, MEW, MPRESS, NSPACK, ACPROTECT, MOLEBOX, PECOMPACT, PELOCK, TELOCK, THEMIDA, UPX, VM-PROTECT and YODA CRYPTER, many of which are readily available on the Internet.

As packers are making executables harder to be analyzed the identification and classification of packing techniques is becoming extremely important in revealing real behavior and the intention of packed program. This information not only helps the detection and analysis of malware, but also allows functions and characteristics of benign packed programs to be analyzed. In addition, quickly and correctly identifying packers allows us to easily and correctly unpack a packed executable file and retrieve the original payload for further malware analyses.

B. Motivation

Malware is becoming a growing problem in modern computer systems. Malicious software, also known as malware (e.g., viruses, worms, or Trojan horses), mostly in the form of packed executables, presents a significant challenge to computer systems. According to Symantec Research Laboratories Fanglu *et al.* [4] and Lyda *et al.* [3], over 80% of malware appear to be created using a packing algorithm to circumvent anti-malware systems. If the malware is packed, it becomes impossible to detect its infection through signature matching.

Moreover, there is an evidence that more than 50% of new malware are simply re-packed versions of existing malware. Many different packers are used throughout one malware family to avoid detection systems. Although packing algorithms are widely used for packing malware Jeong *et al.* [5], they are also applicable for protecting legitimate softwares from reverse engineering. A packed executable has an uncertain data section at run time. From entropy measurement of running programs, we can know whether the given executable is packed or not.

A tremendous number of packing algorithms are created every year, however there is no complete database to detect them. The detection of packing algorithms is necessary for recognizing hidden malware, and preventing such malware from deluding anti-malware systems. Anti-malware systems need to deal with a lot of packers and be prepared for new ones every day.

An automatic system in detecting packing algorithms is indispensable. A vast range of packing algorithm varieties makes a packer detection process time consuming. All existing automatic systems are mainly concentrated on detecting malware, but not on methods of developing them. Nowadays, packing algorithms are used extensively in malware development to help malware unrevealed.

To lay a foundation for this, we studied and discovered a new technique for identification and classification of packing algorithms by creating simple patterns of packers. Furthermore, we studied the use of entropy based detection, which is dynamic in nature, and more reliable than signature based techniques. In this paper we propose a method for identifying

978-1-4799-2533-9 ©2013 IEEE

packing algorithms of given either known or unknown packed executables, and classify them by popular classification techniques such as Naive Bayes and the Support Vector Machine. We can detect the use of a packer and distinguish unknown packers different from existing 15 packers in the experiment. Results of this method demonstrate high accuracy in detecting packing algorithms.

C. Main contributions

Our contributions are two-fold, which are described as follows:

1) We developed a holistic method for identifying packing algorithms of given executables, no matter whether the packer of the executable is already known or not. That, to the best of our knowledge, is the first method to find a systematic way for classifying a packer used in an executable. Our approach was proven to be effective in practice and simpler than any other known methods.
2) We introduce a ***data conversion*** method, to transform numbers into symbols, in order to greatly reduce the space complexity while preserving the accuracy of detection. Our method proved that the data size can be reduced by from 1/2 to 1/10000 times. The method implies a packer classification algorithm, which converts entropy patterns (numbers/numeric values), consisting from large data set, into symbols using a symbolic representation.

The structure of this paper is as follows: Section II describes previous work related to packer classification and pattern recognition techniques. Section III defines the structure of the proposed method with 466 packed executables i.e., a symbolic representation conversion method classifier. Section IV describes the evaluation result of the proposed method using packed executables and different classification techniques. Finally, Section V summarizes the main points of this paper.

II. Related Works

Manual analysis of packers is an early solution to malware exposure. In terms of efficiency, manual analyses are too costly. Despite its expense, however, such an approach is mainly used in practice to reduce the false negatives of a signature. Popular researches in detection and classification of packing algorithms include works by Perdisci *et al.* [6], Li *et al.* [7] and Cesare *et al.* [8]. Perdisci *et al.* proposed a pattern recognition technique for the fast detection of packed executables.

The authors applied various pattern recognition techniques to classify executables into two categories, packed and non-packed. Their technique used publicly available unpacking tools and signature based anti-malware systems to distinguish between specific kinds of malware and benign executables. If an executable is classified as being packed, it will be sent to a universal unpacker for hidden code extraction, and the hidden code will then be sent to an anti-virus scanner.

On the other hand, if the executable is classified as non-packed, it will be sent directly to an anti-malware scanner. Nine features are combined for classification, i.e., the number of standard and non-standard sections; the number of executable sections; the number of readable, writable, and the number of entries in the PE file's Import Address Table (IAT); and the PE header entropy; code section entropy, data section entropy, and file entropy.

This system achieves very high accuracy (above 95%) using classifiers such as Naive Bayes, J48 Decision Tree, Bagged-J48, K-Nearest-Neighbors (KNN), and a Multi-Layer Perceptron (MLP). The best results were obtained using the MLP classifier, which had the best accuracy of 98.91% for the test dataset. However, this approach did not perform the classification of different packers into different families.

The weakness of this technique is that unknown packed files cannot be unpacked because the researchers used universal-unpacking tools as an unpacking mechanism. Moreover, this approach utilizes static analysis, and the researchers did not propose a technique for packer identification. In fact, our classification system extracts the types of packing algorithms from packed PE files, and our unpacking method unpacks any packed executable files using entropy analysis.

Li Sun *et al.* [7] showed an effective packer classification framework that applies pattern recognition techniques on automatically extracted randomness profiles of the packers. As a replacement for signature matching approaches, the authors presented a packer classification problem by analyzing the performance of various statistical classifiers. They also tested various statistical classification algorithms, including k-nearest neighbor, best-first decision tree, sequential minimal optimization and naive Bayes.

All four classifiers were extremely effective, with three of the four achieving an average true positive rate of around 99% or above; However, Naive Bayes was least effective with a true positive rate of around 94%. The k-nearest neighbor classifier with k = 1 obtained the best overall performance. Its true positive rate is 99.6%, and its false positive rate is 0.1%.

The system also reveals that the low randomness profile of the packed file normally produced by the PE header and unpacking stub contains important packer information. It is therefore very useful in distinguishing between families of packers. This approach belongs to signature based packer detection techniques because the distance between the test file and each packer signature is measured. The shorter the distance is, the more likely the file is packed with this packer.

Signature based packer detection has a weakness in terms of measuring distance. For instance, it can not measure every packed files because some packers protect the packed section after packing the given files. Li Sun et al. *did not propose a technique for detecting unknown packers. Through this method, the detection of non-signature based packer and packed files is impossible. While on the contrary our classification technique extracts packing algorithms from packed executables.*

Cesare *et al.* [8] proposed a novel malware classification method by constructing control flow graph based signature. The similarity between structured graphs can be quickly determined using string edit distances to classify malware effectively. What is similar to our work is that they also detect OEP (Original Entry Point) of packed executables by the use of entropy analysis when unpacking the packed malwares. However, their classification method is merely based on similarity distance and pre-defined threshold which is relatively vague and may be not effective enough for classification. That is why we are using classic classification models such as Naive Bayes and SVM to improve our classification results. Whats more, even though they showed a relatively good result in terms of identifying and classifying malware and its variants, they did not actually perform the packing algorithms classification as we did which is so important for the analysis of packed malware. Jeong *et al.* [5] proposed a mechanism to find the original entry point (OEP) using entropy analysis. They did not actually perform the packing algorithms classification.

III. Method for Packing Algorithm Detection

A. Structure of Packing Algorithm Detection

A main concept of our paper is measuring entropy values while unpacking packed executables by themselves. Detecting packing algorithms straightaway through entropy pattern is problematic. Because of the big (huge amount of) data and bursting of errors, the process is time consuming and difficult to analyze. Hence, we extracted simple patterns from Entropy patterns through symbolic representation. Then we classified the simple patterns through Naive Bayes and SVM classification algorithms, which is a key point in our proposed method of detecting packing algorithms. In general, we classified symbolic representations using supervised learning classification techniques. Figure 1 shows the three main parts of our method. The first part is measuring the entropy patterns, the second is converting them into symbolic representation, and the last is classification.

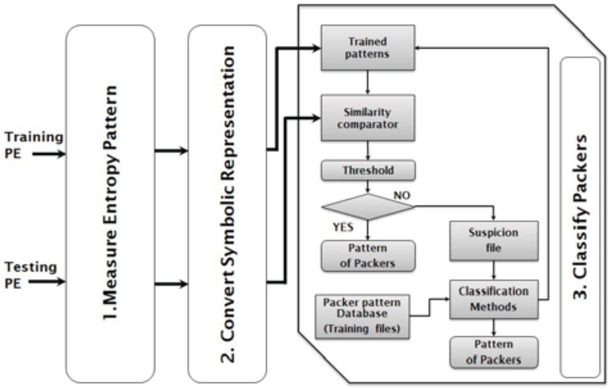

Fig. 1. A packing algorithm detection method

1) Measuring the entropy pattern: Measuring the entropy pattern determines the entropy value of the packed executables during the unpacking process. Therefore, we first execute a given packed executable, and let it conduct the unpacking process. While the unpacking process, packed instructions are unpacked using a decompression module, and intuitively, the measured entropy of the memory space will change. Finally, the end of the unpacking can be detected by monitoring the cessation of entropy changes. The executable is executed and keeps running until JMP instruction is encountered. Entropy analysis is conducted by measuring a specific memory space. It decides whether or not unpacking process is completed by measuring the entropy value of the sections of packed executable, and checking if a JMP instruction jumps to an instruction address in the sections. The packed executable is completely unpacked only if OEP is found. Otherwise during execution we measure the entropy value to determine the OEP. The address of the first instruction of the decompressed code is called the original entry point (OEP). Conversely, if the unpacking process is incomplete, the paused process continues to execute the next instruction. Therefore, we use entropy analysis to detect the existence of packing algorithms.

2) Entropy analysis : Information theory, or entropy, is a method for measuring uncertainty in a series of information units Jeong *et al.* [5]. Entropy can be used to evaluate a compression algorithm. In technical terms, entropy measures the level of difficulty, or the probability of independently predicting information of the series. Data compression is the process of encoding information to represent the information in a fewer number of bits. Information is compressed by following a logical sequence. First, some repeated patterns are found in the information, and the redundancies of the patterns are then used to reduce the amount of information. Therefore, compressed information has a smaller number of patterns than before. In other words, the number of patterns in the information is reduced through compression, and the series of bits become more unpredictable, which is equivalent to uncertainty. Therefore, the measured entropy of the compressed information is higher than that of the original information. Shannon's formula was devised to measure the information entropy as follows:

$$H(x) = -\sum_{i=1}^{n} p(i) \cdot log_b^{p(i)} \tag{1}$$

where $H(x)$ is the measured entropy value and *p(i)* is the probability of an i^{th} unit of information in the series of n symbols of event x. The base number of the logarithm can be any real number greater than 1. However, 2, 10, or Euler's number, *e*, is generally chosen. We used two constants, termed E_{min} and E_{max} to determine if unpacking is completed, but they have not been given values yet. The entropy is also in the range between E_{min} and E_{max}.

B. Conversion into Symbolic Representation

An orderless entropy pattern is then converted into a simple symbolic pattern. To extract a packing algorithm and represent its pattern through a symbolic representation, we use the entropy analysis shown in Figure 2. The symbolic aggregate approximation (SAX) is the first symbolic representation for time series data mining.

Symbolic representation allows for a dimensionality reduction and indexing with a lower-bounding distance measure of the true distance. One can take advantage of the generic time series data mining model, as well as a host of other algorithms, definitions, and data structures that are only defined for discrete data, including hashing, Markov models, and suffix trees. SAX is one of the most competitive methods in the literature.

SAX utilizes a similarity measurement that is easy to compute because it is based on pre-computed distances obtained from lookup tables. We introduce a detailed explanation of this in Section III-B1.

1) Classification of symbolic aggregate approximation (SAX) : Similarity search is a fundamental problem in computer science, and has many application area such as multimedia databases, bio-informatics, pattern recognition, text mining, computer vision, data mining, and machine learning. Time series are data types that appear in many medical, scientific and financial applications. Time series data mining includes many tasks such as classification, clustering, similarity search, motif discovery, anomaly detection and so on.

One key to the successful performance of these tasks is the use of representation methods that can represent a time series efficiently and effectively Keogh *et al.* [9]. Of all the symbolic representation methods in the literature on time series data mining, the symbolic aggregate approximation (SAX) stands out as one of the most powerful method Jessica *et al.* [10]. The main advantage of this method is that the similarity measurement that it uses is easy to compute owing to the use of statistical lookup tables. In this paper we present an improved similarity measurement for using SAX.

This measurement has same advantages as the original similarity measurement used in SAX. SAX is based on the fact that a normalized time series has a ***highly Gaussian distribution***, and thus by determining the breakpoints that correspond to the size of the alphabet, one can obtain equally sized areas under a Gaussian curve Jessica *et al.* [10].

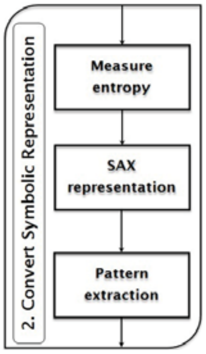

Fig. 2. A structure of the pattern extraction method

SAX is applied as follows: First, the time series are normalized. Second, the dimensionality of the time series is reduced using piecewise aggregate approximation (PAA) Keogh *et al.* [9]. Third, the PAA representation of the time series is discretized, which is achieved by determining the number and location of the breakpoints. The breakpoint locations are determined using statistical lookup tables such that these breakpoints produce equally sized areas under a Gaussian curve.

In algorithm 1, we presented process of converting entropy value into SAX. The interval between two successive breakpoints is assigned to a symbol of the alphabet, and each segment of the PAA that lies within that interval is discretized by that symbol. As a final step, similarity measurement of SAX's is performed. The SAX method approximates time series X of length n into vector $\overline{X}$ = ($\overline{x_1}$, $\overline{x_2}$, . . . , $\overline{x_M}$) of any arbitrary length M ($M<n$, typically $M<<n$), where each $\overline{x_i}$ is calculated through the following formula:

$$\overline{x_i} = \frac{1}{r}\left[\sum_{j=r(i-1)+1}^{ri}(x_i)\right], \tag{2}$$

where r is a ratio defined as,

$$r = \frac{n}{M}. \tag{3}$$

Here, M is the length of the original time service (entropy pattern), n is the length of the string (the number of frame or symbols). Simply stated, to reduce the time series from n dimensions into M dimensions, the data are divided into M equally sized frames. SAX is the first symbolic representation of a time series with an approximate distance function that lower bounds the Euclidean distance.

In SAX, the data are first transformed into a PAA representation, and the transformed PAA representation is then symbolized into a sequence of discrete strings. As said in Lin *et al.* [10] there are two important advantages of SAX that are mentioned in definitions one and two:

Definition 1: **Dimensionality Reduction** occurs when data of very high dimensionality are converted into data of much lower dimensionality such that each of the lower dimensions conveys much more information. Thus, the dimensionality reduction of PAA is automatically carried over to a symbolic representation Keogh *et al.* [9] and Yi *et al.* [11].

Algorithm 1: An algorithm for converting symbolic representation

```
Require: An algorithm is required for converting entropy
    values into symbolic representation. a number of symbols,
    entropy of unpacked code, breakpoints, symbolic value
    and normalized entropy of unpacked code are abbreviated
    as $\phi(\beta)$, $E$, $\beta$, '$X$ and '$E$ respectively, in the following
    pseudocode.
Ensure: Extract symbolic unique pattern, which will be
    used in detecting packing algorithms.
    {Convert entropy values into symbolic representation}
    We detect to packing algorithm using of the symbolic
    representation, the SAX.
    if Breakpoint $\beta_{i-1} < \beta_i$ and '$E \leftarrow$ normalized of $E$ are
    true then
        $\phi(\beta) \leftarrow$ calculate a number of symbols with breakpoints
    else
        Continue this loop.
    end if
    if '$X \leftarrow$ convert '$E$ into SAX true then
        $\phi(\beta) \leftarrow$ analyze '$X$ using breakpoints
        We extract new unique a symbolic pattern from a
        entropy pattern.
        Calculate to similarity measurement, accuracy and
        recall.
    else
        Continue this loop.
    end if
```

Definition 2: **Lower Bound** is a number less than or equal to every number in a given set of entropy values. In other words, it allows distance to be defined on in the original time series (original entropy pattern). Hence, the lower bounding distance between two symbolic strings can be proved by simply pointing to the existing proofs for PAA representation itself.

Given the normalized data set (entropy values) have highly Gaussian distribution, we can simply determine the $breakpoints$ that will produce a, equal-sized areas under Gaussian curve.

Definition 3: **Breakpoints** ($\mathcal{B}$) are a sorted list of numbers, $\mathcal{B}$=β_1, β_2, . . . , β_{a-1}, such that $\beta_{i-1} < \beta_i$ divides the area under a N(0,1) Gaussian curve into equal areas. The size of the alphabet is also an arbitrary integer a greater than 2 (e.g., for the letters = (a, b, c), a = 3). According to Lin *et al.* [10], these breakpoints are determined by looking them up in a statistical table.

$$from \quad \beta_i \quad to \quad \beta_{i+1} = \frac{1}{a} \tag{4}$$

(β_0 and β_a are defined as $-\infty$ and ∞, respectively)

Definition 4: **Euclidean distance** ($\mathcal{ED}$) is the most common distance measure for time series [12] analysis. Given two time series Q and S of the same length n, defines their Euclidean distance.

$$\mathcal{ED}(Q, S) = \sqrt{\sum_{i=1}^{n}(Q_i - S_i)^2} \tag{5}$$

If we transform the original subsequence into SAX representations, We use the following equation.

$$\mathcal{EDS}(\overline{Q},\overline{S}) = \sqrt{\frac{n}{N}}\sqrt{\sum_{i=1}^{M}(\overline{Q}_i - \overline{S}_i)^2} \qquad (6)$$

C. Classifier

There are two categories of classification methods. First, supervised classification uses ground truth data in the form of simple sets. A maximum likelihood classifier is a typical supervised classification method. Second, unsupervised classification utilizes only spectral features without the use of ground truth data. Clustering is an unsupervised classification, in which a group of spectral values will be regrouped into a few clusters with spectral similarity.

Our proposed method includes two types of classification. The first one is a similarity measurement classification. We introduce a detailed explanation of it in Section III-D. A second one includes commonly used classification methods such as the Naive Bayes classifier, Support Vector Machines and J48 Decision Trees. By applying our method, we generated patterns with high accuracy of detecting known and unknown packing algorithms.

However, the classification of known and unknown packing algorithms turned out to be a critical factor in our research. A classifier can be designed using various approaches for classification. Roughly speaking, there are three different methods that can be used Meijer *et al.* [13].

The first method is the simplest and most intuitive, and is based on the concept of similarity. Template matching is the one of such examples. The second method is probabilistic. This includes Naive Bayes (NB), support vector machine (SVM), J48 decision tree, Bagget-j48, RandomForest, RandomWood, k-nearest-neighbors (KNN) and Multi-Layer Perceptron (MLP).

The k-nearest neighbor (KNN), Parzen window, and branch-and-bound (BnB) classifying methods are well known among researchers. The third method is the direct construction of decision boundaries by optimizing certain error criteria. Examples of this are Fisher's linear discriminant, multilayer perceptions, decision tree and support vector machines.

In this paper, we performed supervised classification based on simple patterns. The main part of the experiment is detecting unknown packers. We arranged available unknown packers by determining their nearest similar simple patterns and placed them into families with patterns they are analogous with. If a family of similar patterns did not exist we created a new database of families. Figure 3 illustrates our use of the first and second methods. We practiced using NB and SVM, which are both supervised learning classifiers, to classify the packing algorithms.

1) Naive Bayes (NB) classification : The naive Bayes classifier is an effective and efficient classification algorithm. Classification is a fundamental issue in machine learning and data mining fields. The NB classifier is both a supervised learning method and a statistical method for classification. The NB classifier is based on the Bayes rule of conditional probability, which enables us to create classifications of known and unknown packing algorithms using Bayes theorem.

2) Support vector machine (SVM) classification : Support vector machine is a supervised learning method used for classification and regression Vapnik [14] and Yapnik *et al.* [15]. SVM is a widely used machine learning method introduced by Vapnik [14]. SVM is a powerful, state of the art algorithm with strong theoretical foundations based on Vapnik's theory. A notable characteristic of an SVM is that its computational complexity is independent of the dimensionality of the kernel space, where the input feature space is mapped Burges *et al.* [16]. Thus, the curse of dimensionality is bypassed. An SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the output. In other words, given the data from two classes as two sets of feature vectors in an n-dimensional space, SVM constructs an optimal hyperplane that separates a set of one class instances from a set of the other class instances and maximizes the margin between the two data sets. That is, if two parallel hyperplanes are constructed, one on each side of the optimal hyperplane and pass through the nearest data point in each data class, the distance between the parallel hyperplanes needs to be as far apart as possible while still separating the data into two classes. This also has an influence on the classification for packing algorithm detection, and SVMs tend to have a high performance.

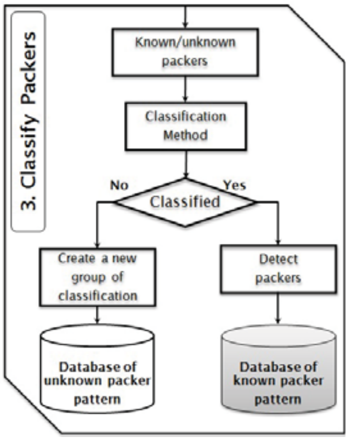
3. Classify Packers
Known/unknown packers
Classification Method
No
Yes
Classified
Create a new group of classification
Detect packers
Database of unknown packer pattern
Database of known packer pattern

Fig. 3. The structure of a classifier

D. Similarity Measurement

A similarity coefficient $\Phi(x,y)$ measures the strength and direction of the linear relationship between two symbolic representations of packing algorithm.

$$\Phi(x,y) = \frac{\sum_{i=1}^{n}(x_i * y_i)}{\left[\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}\right]}. \qquad (6)$$

The value is always between $[-1;1]$, where 1 indicates a strongly positive relation; 0, means no relation; and -1, means a strongly negative relation. Similarity is the most widely used coefficient. A similarity measure can be considered a measure for representing the similarity of two sequences x and y under the following comparison: This measure accepts an amplitude shift.

IV. Effectiveness Evaluation of the Classification

In this section, we describe the experimental results of our analysis. As previously stated, we proposed a method for detecting packing algorithms. The dataset used in this experiment contains 466 packed executables, 246 of which are packed malware files [17], and the remaining 220 are packed benign executables. The malware samples were collected from a malicious Web site ***offensive computing*** [18] and ***VX heavens*** [17].

The data sample of the 246 packed malware executables are a collection of viruses, Trojans, adware, spyware, and a few variants [18]. The data samples of the 220 packed benign executables are a collection of Windows system files and normal executables. We used fifteen popular packers in the experiments. We conducted around 6,800 experiments on packing algorithm detection.

In these experiments methods of similarity measurement, symbolic representation and popular forms of classification were used on each packed executable. The data samples were split into training and testing samples at a ratio of 50:50. The training data set consists of 110 data samples from 220 packed executables, with the rest being packed benign executables. Similarly, the testing data set consists of 123 samples from 246 packed malware, with the remaining being packed malware.

A. Evaluation Metrics

When comparing the performances of different classification techniques, it is important to assess how well they are able to correctly predict the actual classifications of the packers. Several metrics are conventionally used to numerically quantify the effectiveness of the classification performance. To introduce the metrics, let us define the classification of packers. A packer is positive if it is predicted to belong to a specific class, and is negative if it is predicted to belong to another class.

Suppose that for a test set with n packers, the sets of positive and negative packers for the classification are known, and P and N are the numbers of positive and negative packers respectively, such that

$$n = P + N. \tag{8}$$

Using the four important counts defined below:

- TP represents a true positive, which is the number of detected packers, correctly identified as a correct class of the packing algorithm;
- FP represents a false positive, which is the number of detected packers that do not belong to a classification but were incorrectly identified;
- TN represents a true negative, which refers to the number of detected packers, incorrectly identified as classifications;
- FN represents a false negative, which is the number of detected packers that belong to a classification but were incorrectly identified as classifications.

Then, the numbers of positive and negative detected packer P and N are defined by

$$P = TP + FN, \tag{9}$$

and

$$N = FP + TN. \tag{10}$$

Let $\mathcal{A}$ denote the accuracy of classification as the percentage of test set packers that are correctly identified by the classifier. That is,

$$\mathcal{A} = \frac{(TP + TN)}{n} = \frac{(TP + TN)}{(P + N)}. \tag{11}$$

The accuracy $\mathcal{A}$ provides the overall effectiveness. However, this measure has one limitation. Suppose that a test set contains a large number of negative packers and a very small number of positive packers, and that we use a classifier labeling every class as negative (no matter what the input data are). That is, TN is very high, and TP is very low.

Despite the classifier being very primitive, it will achieve very high classification accuracy on this data set. The true ($\mathcal{T}_r$) and false ($\mathcal{F}_r$) positive rates are introduced to measure the proportion of positive packers that are correctly identified and the proportion of negative packers that are incorrectly identified, respectively. For each class, $\mathcal{T}_r$ is calculated as

$$\mathcal{T}_r = \frac{TP}{P} = \frac{TP}{(TP + FN)}, \tag{12}$$

and $\mathcal{F}_r$ is defined by the following formula:

$$\mathcal{F}_r = \frac{FP}{N} = \frac{FP}{(FP + TN)}. \tag{13}$$

Two other fundamental ways to measure the classification effectiveness are precision and recall. Precision indicated the proportion of packers classified as positive, which are classified correctly, and recall shows the proportion of positive packers that have been correctly identified. Therefore, for each class, precision $\mathcal{P}$ is defined as

$$\mathcal{P} = \frac{TP}{(TP + FP)}, \tag{14}$$

and recall $\mathcal{R}$ is defined as

$$\mathcal{R} = \frac{TP}{(TP + FN)}. \tag{15}$$

B. Classification Result from an Entropy Analysis

All packers are measured based on their similarity. As an example, a graph and result of the MPRESS packer are also given in Table I. Table II shows the similarity between the experiment result of MPRESS and popular five packers from fifteen packing algorithms.

TABLE I
EXPERIMENTAL RESULTS OF THE MPRESS PACKER

MPRESS	calc	freecell	mshearts	msiexec	notepad	telnet
calc	1	0.9688	0.9709	0.9604	0.9913	0.9130
freecell	0.9688	1	0.9923	0.9865	0.9797	0.9669
mshearts	0.9709	0.9923	1	0.9742	0.9800	0.9411
msiexec	0.9604	0.9865	0.9742	1	0.9724	0.9667
notepad	0.9913	0.9797	0.9800	0.9724	1	0.9300
telnet	0.9130	0.9669	0.9411	0.9667	0.9300	1

We extracted packing algorithm patterns using the SAX representation method.

TABLE II
SIMILARITY BETWEEN THE EXPERIMENTAL RESULTS OF MPRESS AND OTHER PACKERS

MPRESS	NPACK ”calc.exe”	ASPACK ”calc.exe”	RLPACK ”calc.exe”	UPXN ”calc.exe”	NSPACK ”calc.exe”
”calc.exe”	0.7278	0.5599	-0.4034	-0.2093	-0.4276

However, first we present the benign ”calc.exe” files packed with fifteen packing algorithms. Second, packed ”calc.exe” executables converted by SAX using four types of $\phi(\beta)$ values. In the example above with $\phi(\beta) = 10$, $\phi(\beta) = 100$, $\phi(\beta) = 1000$, and $\phi(\beta) = 10000$ where n=100000, the data series is mapped to the character symbol 'abcdefghijklmnopqrstuvwxyz' where $\phi(\beta)$ is the number of symbols.

Value M and $\phi(\beta)$ have reverse relationship. The lower the value of M goes, the greater the beta becomes. When the value M is low, the accuracy of converting entropy values to symbolic representations will be high, which can be seen from equation(16).

$$\phi(\beta) = \frac{Entropies}{M} \tag{16}$$

We show the combination of $\phi(\beta)$ for converting entropy pattern into symbolic representation in Figure 4.

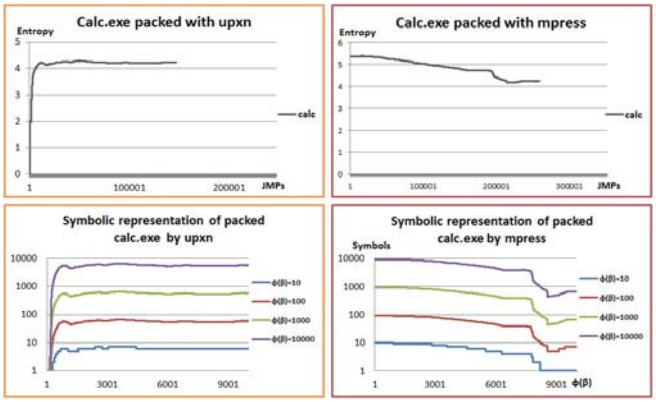

Fig. 4. Entropy patterns converted into symbolic representations using different $\phi(\beta)$

In other words, $\phi(\beta)$ evaluates the number of symbols used for extracting packing algorithm pattern. For instance (in fact), we packed calc.exe file by MPRESS packer. Next, we extracted entropy pattern by unpacking the file through measuring/analyzing its entropy value. Then we converted the entropy pattern into a simple pattern through symbolic representation. The simple pattern, which is expressed by symbols such as $\phi(\beta)$= $(10, 100, 1000, 10000)$ by using SAX algorithm, is presented in Figure 4. We show that the accuracy is independent from the number of symbols, $\phi(\beta)$, as shown in Table III.

TABLE III
RESULTS OF MPRESS PACKER CONVERTED BY SAX

MPRESS $\phi(\beta)$	$\mathcal{T}_r\%$	$\mathcal{F}_r\%$	$\mathcal{A}\%$	$\mathcal{P}\%$	$\mathcal{R}\%$
10	100.0	3.3	98.0	95.2	100.0
100	100.0	0.0	100.0	100.0	100.0
1000	100.0	0.0	100.0	100.0	100.0
10000	100.0	0.0	100.0	100.0	100.0

In Table IV, the patterns of fifteen packing algorithms converted by SAX are shown. The different packers, detected at an average accuracy of 94.2%. Table IV shows the detailed accuracy of the sample dataset. The detection results of the samples are presented as a confusion matrix. As shown in Table IV, the accuracy of the MPRESS and MEW packing algorithms are both 100%, whereas the minimum accuracy refers to 88.5% which relates PELOCK packing algorithm.

Unlike other packing algorithms, we predict that the PELOCK packing algorithm uses a protecting system (hiding packing algorithm). Table IV shows an average true positive rate of 94.7%, a false positive rate of 6.6%, a precision of 92.7%, and a recall of 94.7%. Finally, in Table V, we present two classifications of packing algorithms based on symbolic representation data.

TABLE IV
DETAILED ACCURACY OF EACH PACKER USING A MIXED SAMPLE DATASET

PACKERS	$\mathcal{T}_r\%$	$\mathcal{F}_r\%$	$\mathcal{A}\%$	$\mathcal{P}\%$	$\mathcal{R}\%$
NPACK	100.0	10.0	94.1	91.7	100.0
MPRESS	100.0	0.0	**100.0**	100.0	100.0
NSPACK	95.2	4.8	96.1	95.2	95.2
UPXN	90.5	6.7	92.2	90.5	90.5
RLPACK	90.5	10.0	90.2	86.4	90.5
ASPACK	95.2	4.8	96.1	95.2	95.2
PETITE	91.3	12.5	90.9	91.7	91.3
ASPROTECT	91.7	9.4	91.1	92.0	91.7
FSG	100.0	3.1	98.2	96.0	100.0
MEW	100.0	0.0	**100.0**	100.0	100.0
THEMIDA	92.3	5.9	93.3	92.3	92.3
YODA CRYPTER	92.3	5.9	93.5	88.9	92.3
VMPROTECT	96.2	2.9	95.0	92.3	96.2
PELOCK	88.5	14.7	88.3	88.5	88.5
TELOCK	96.2	8.8	93.3	89.3	96.2
AVERAGE	**94.7**	**6.6**	**94.2**	**92.7**	**94.7**

Tables III shows some experimental results of the MPRESS packer. The experimental results of the UPXN and RLPACK packing algorithms graphed in Figure 5 look similar. Figures 5 shows three packing algorithms deputed from fifteen packing algorithms i.e., UPXN, RLPACK, and MPRESS. The lower graphs show the conversion into symbolic representation for the corresponding packing algorithm.

TABLE V
DETAILED ACCURACY OF EACH CLASSIFIER

Classification	$\mathcal{T}_r\%$	$\mathcal{F}_r\%$	$\mathcal{A}\%$	$\mathcal{P}\%$	$\mathcal{R}\%$
Naive Bayes	98.0	1.5	90.4	91.8	98.0
Support vector machine	95.7	2.3	95.5	90.0	95.7
AVERAGE	**96.8**	**1.9**	**92.9**	**90.9**	**96.8**

The MPRESS packing algorithm shows a totally different pattern from the UPXN and RLPACK packers. However, we easily extracted a new symbolic pattern from each packing algorithm. The example graphs in Figure 5 shows each packing algorithm converted into a symbolic pattern with $\phi(\beta) = 10000$.

The experimental results for these notorious packing algorithms imply that our proposed method is useful for identifying packing algorithms. The results have also shown that the proposed method is applicable to packed malwares.

As shown in Table V, the accuracy of the NB classification is 98.0% which is higher than SVM classification. We can therefore classify packing algorithms using entropy analysis and symbolic representation with a high accuracy.

V. CONCLUSION

This paper discussed the use of packers, and presented a novel technique to detect packing algorithms using SAX representations and similarities of the sequence of SAX symbols in each packer. In this method, the low randomness profile of the packing algorithm is extracted and then passed to a pattern classifier.

Our work demonstrates that the randomness profile combined with strong pattern recognition algorithms produces a highly accurate packer classification system on real life data. The proposed system was tested on a large data set, including

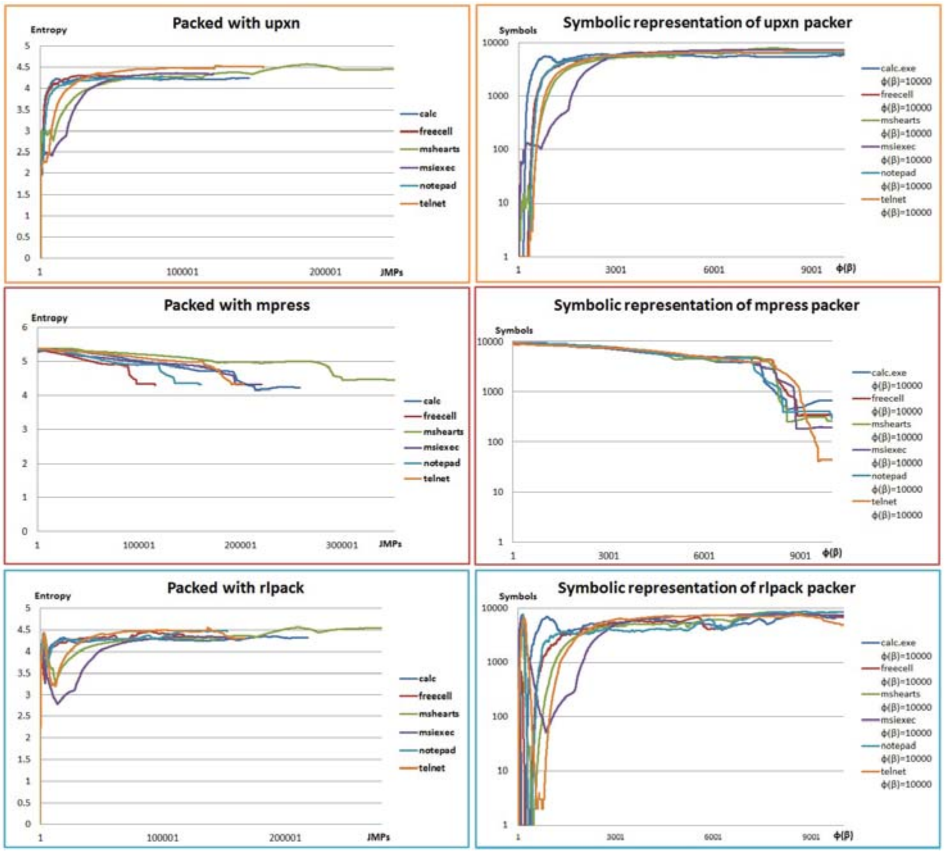

Fig. 5. Experiment result of entropy patterns converted into symbolic representations

220 benign packed files and more than 246 packed malware from the wild.

Our method classifies packing algorithms using NB and SVM classifiers with symbolic representation patterns. In future work, we will extract the symbolic patterns of new packed malware, multi-packing algorithms and use additional classification method for packer classification.

VI. Acknowledgments

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea and Microsoft under IT/SW Creative research program supervised by the NIPA (National IT Industry Promotion Agency) (H0503-12-1034). Additionally, this research was supported by the Public Welfare & Safety Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2012M3A2A1051118)

References

[1] H. Choi, B. B.Zhu, and H. Lee., "Detectiong maliciuos web links and identifying their attack types," *USENIX Int'l Conf. on Web Aplication Development (Web Apps)*, 2011.

[2] Y. Wei, Z. Zheng, and A. Nirwan., "Revealing packed malware," *IEEE Security Privacy*, pp. 65–69, 2008.

[3] L. Robert and H. James., "Using entropy analysis to find encrypted and packed malware," *IEEE Security and Privacy*, pp. 40–45, 2007.

[4] G. Fanglu, F. Peter, and C. Tzi-Cker., "A study of the packer problem and its solutions," in *RAID '08: Proc. of the 11th International Symposium on Recent Advances in Intrusion Detection*, pp. 98–115, 2008.

[5] G. Jeong, E. Choo, J. Lee, M. Bat-Erdene, and H. Lee., "Generic unpacking using entropy analysis," *IEEE Malware*, pp. 114–121, Octember 2010.

[6] R. Perdisci, A. Lanzi, and W. Lee., "Classification of packed executables for accurate computer virus detection," *Pattern Recognition Letters*, pp. 1941–1946, 2008.

[7] L. Sun, S. Versteeg, S. Boztass, and T. Yann., "Pattern recognition techniques for the classification of malware packers," in *Proc. of Information Security and Privacy on 15th Australasian Conference, ACISP 2010*, pp. 370–390, 2010.

[8] S. Cesare and Y. Xiang., "Classification of malware using structured control flow," *AusPDC '10 Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing*, pp. 61–70, 2010.

[9] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani., "Locally adaptive dimensionality reduction for similarity search in large time series databases," *ACM Transactions on Database Systems*, pp. 188–228, 2002.

[10] L. Jessica and K. Lonardi., "A symbolic representation of time series with implications for streaming algorithms," *DMKD'03: Proc. of the 8th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 2–11, 2003.

[11] B. Yi and C. Faloutsos., "Fast time sequence indexing for arbitrary lp norms," *VLDB'00: Proc. of the 26th International Conference on Very Large Data Bases*, pp. 385–394, 2000.

[12] K. Keogh, E., "A survey and empirical demonstration," *ACM SIGKDD Int'l. Conf on Knowledge Discovery and Data mininig*, pp. 349–371, 2003.

[13] B. Meijer, "Rules and algorithms for the design of templates for template matching," *Conference A: Computer Vision and Applications*, pp. 760–763, 1992.

[14] V. Vapnik., "The nature of statistical learning theory," *Springer-Verlag*, 1995.

[15] V. Vapnik and A. Chervonenkis, "Theory of pattern recognition: Statistical problems of learning," *Nauka*, 1974.

[16] C. J. C. Burges., "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, pp. 121–167, 1998.

[17] A. Baranovich., "Vx heavens." http://vx.netlux.org.

[18] G. T. I. S. Center., "Offensive computing." http://offensivecomputing.net, 2005.