entropy

Article

Packer Detection for Multi-Layer Executables Using Entropy Analysis

Munkhbayar Bat-Erdene, Taebeom Kim, Hyundo Park and Heejo Lee *

Department of Computer Science and Engineering, Korea University, 02841 Seoul, Korea; munkhbayar@korea.ac.kr (M.B.-E.); ktb88@korea.ac.kr (T.K.); hyundo95@korea.ac.kr (H.P.)

* Correspondence: heejo@korea.ac.kr; Tel.: +82-2-3290-3638

Academic Editor: Raúl Alcaraz Martínez
Received: 31 January 2017; Accepted: 13 March 2017; Published: 16 March 2017

Abstract: Packing algorithms are broadly used to avoid anti-malware systems, and the proportion of packed malware has been growing rapidly. However, just a few studies have been conducted on detection various types of packing algorithms in a systemic way. Following this understanding, we elaborate a method to classify packing algorithms of a given executable into three categories: single-layer packing, re-packing, or multi-layer packing. We convert entropy values of the executable file loaded into memory into symbolic representations, for which we used SAX (Symbolic Aggregate Approximation). Based on experiments of 2196 programs and 19 packing algorithms, we identify that precision (97.7%), accuracy (97.5%), and recall (96.8%) of our method are respectively high to confirm that entropy analysis is applicable in identifying packing algorithms.

Keywords: re-packing algorithms; original entry point (OEP); multi-layer packing; piecewise aggregate approximation (PAA); symbolic aggregate approximation (SAX); entropy analysis

1. Introduction

1.1. Background

Nowadays, malware creates distress and significant financial loss by violating privacy of computer users. Unfortunately, connived (indulged) on their previous success attackers develop their malware so that harder to detect [1,2]. Following Yan et al.'s [3] understanding, we consider packer as "a program that produces a number of data blocks to form a compressed and encrypted version of the original executable". Packing helps to evade from anti-virus (AV) by diminishing the size or transforming the appearance of executable binary [2,4–7]. Overall, "a packer is a program that transforms an executable binary into another form, and packing is becoming one of the widely used technique." According to recent studies over 80% of malware are obfuscated with packers and compression techniques, Osaghae et al. [8], Jacob et al. [9], Bat-Erdene et al. [2] and Brosch et al. [10]. Generally, to hide the original behavior of the malware attackers use different packing algorithms to generate a greater number of malware options. Nowadays, Aspack [11], MEW [12], ASProtect [13], NsPack [14], Themida [15], RLPack [16], VMProtect [17], and Alternate_EXE [18] are widely used packers. The identification and classification of packing techniques are becoming vital for revealing an intention and a real behavior of the packing algorithms [2]. Besides, quickly detecting and correctly unpacking packers allow us to efficiently and accurately unpack a packed executable file and conduct further analysis.

1.2. Multi-Layer Packing

Malware, malicious software (e.g., viruses, worms, or Trojan horses), challenges computer systems in the form of the packed executables and is becoming a growing problem to computer systems. Symantec Research Laboratories (Osaghae et al. [8], Al-Anezi et al. [19], Santos et al. [20] and McAfee [21]), over 80% of malware appears to be produced using a packer to circumvent anti-malware systems; furthermore, more than 50% of new malware are re-packed versions of existing malware [19,20,22]. If the packed malware [23,24] is re-packed or multi-layer packed a detection of its infection through signature matching is impossible [2,25–32]. In Figure 1 we present three main parts of the packing structure.

- Single-layer packing algorithm. A previous study of Bat-Erdene et al. [2] was devoted to single-layer packing algorithms.
  $F_i(P)$—Single-layer packed benign and malware executables; where F_i is packer, P is benign or malware executable.
- Re-packing algorithm.
  $F_i(F_i(P))$—Re-packed benign and malware executables; where F_i are same packers; P is benign or malware executable.
- Multi-layer packing algorithm.
  $F_j(F_i(P))$—Multi-layer packed benign and malware executables; where F_i and F_j are different packers; P is benign or malware executable.

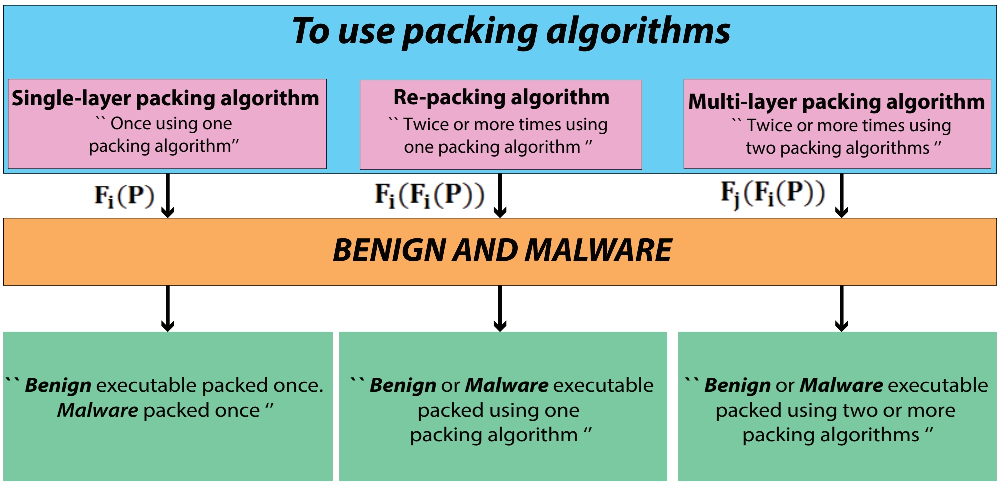
To use packing algorithms
Single-layer packing algorithm
`` Once using one packing algorithm"
Re-packing algorithm
`` Twice or more times using one packing algorithm "
Multi-layer packing algorithm
`` Twice or more times using two packing algorithms "
$\mathbf{F_i(P)}$
$\mathbf{F_i(F_i(P))}$
$\mathbf{F_j(F_i(P))}$
BENIGN AND MALWARE
`` Benign executable packed once. Malware packed once "
`` Benign or Malware executable packed using one packing algorithm "
`` Benign or Malware executable packed using two or more packing algorithms "

Figure 1. Structure of single-layer packed, re-packed, and multi-layer packed executables.

To avoid detection systems, attackers use many different packers in one malware family. Packing algorithms are the double-edged sword; they are extensively used in packing malware (Jeong et al. [33], Ugarte-Pedrero et al. [25], Bat-Erdene et al. [2]) and are also responsible for protecting genuine software from reverse engineering. A packed executable has an encoded data section. When packed executable running on the memory, data section's size and data changed. An entropy measurement of running programs enables us to assess whether a given executable is packed. Identification of packing algorithms is necessary for recognizing hidden malware. Anti-malware systems need the ability to work with a large number of packers and be ready for new ones since more and more packing algorithms are created each day. A system capable of automatically detecting packing algorithms is indispensable, yet there is no complete database. Existing automatic systems mainly concentrate on detecting the malware itself rather than on methods/approaches for developing them. Currently, single-layer packing, re-packing, or multi-layer packing algorithms are used extensively in malware development to assist the malware to remain undetected. We, therefore, examined a method for detection and classification of re-packing or multi-layer packing algorithms and classify them by creating simple patterns of packers. Through our method a user can detect a re-packer or multi-layer packers with high accuracy, as confirmed by our experiments.

1.3. Main Contributions

Our contributions are three-fold, as described below:

1. We propose a method based on entropy analysis to detect executables re-packed or multi-layer packed two or more times.
2. We develop a holistic method for identifying re-packer or multi-layer packer and determine their packing algorithms of given executables, irrespective of whether the re-packer or multi-layer packer of the executable are already known. This, to the best of our knowledge, is the first method to detect re-packer or multi-layer packer using machine learning. Considering experiment results, we claim that our approach is practically efficient and simpler than any other known methods.
3. We introduce a data conversion method, which significantly reduces the space complexity by reducing data size by from 1/2 to 1/10000 times.

The article is structured as follows. The next section discusses related works. Section 3 defines the entropy analysis, packer complexity type, the structure of the proposed method and a symbolic representation conversion. Section 4 describes the classifier, similarity measurement, and incremental aggregate analysis. Section 5 describes the evaluation result of re-packed or multi-layer packed executables and classification techniques. Finally, the article ends with summaries of main findings of this paper.

2. Related Work

Various methods have been developed for identification of packed malware. Devi et al. [34], for example, proposed classified single-layer packed and non-packed executables using a pattern recognition technique for the detection of packed malware binaries. This approach aims to extract the best set of features from the Windows Portable executable files to pass it to their classification model. The classification model functions once an executable is classified as a single-layer packed. Then a second phase concludes whether it is the single-layer packed benign or malware.

Perdisci et al. [35] applied various pattern recognition techniques to classify executables into single-layer packed and non-packed categories. Their method used publicly available unpacking tools and signature-based anti-malware systems to distinguish between particular kinds of malware and benign executables. The weakness of these techniques is that re-packed or multi-layer packed files cannot be detected and unpacked since their method is not for packer identification. In contrast, our method identifies and unpacks single-layer packed, re-packed, or multi-layer packed executable files using entropy analysis. Types of packing algorithms are extracted from packed Portable Executable (PE) files. The packed files described in this paper are in PE format (Guo et al. [5] and Pietrek et al. [36,37]), which is the format used in the most Microsoft Windows operation systems.

The thwart program analysis based automated malware detection, malware authors gradually adopt code protection techniques. Although, Ugarte-Pedrero et al. [38] proposed these techniques that are initially designed to counter reverse engineering and effectively resist many program tampering attempts, they are becoming a standard measure of malware detection circumvention. Lyda et al. [39] presented an encrypted and single-layer packed malware detection technique based on entropy analysis to analyze packed PE files via byte distribution. Their methodology computes entropy at benign model level, in which entropy is computed based only on the occurrence frequency of certain bytes of an executable without considering how these bytes were produced. However, this technique is not useful in detecting re-packers or multi-layer packers. Sun et al. [40] proposed “a packer classification method that applies pattern recognition techniques using randomness profiles, a unique feature set extracted from packed executables.” The researchers presented a packer classification approach by analyzing the performance of various statistical classifiers and employing statistical classification algorithms including naive Bayes and k-nearest neighbor. However, their proposed technique did not apply for detecting unknown packers, re-packers, and multi-layer packers. On the contrary, our classification technique has the ability to extract the packing techniques from packed PE files because we use a generic unpacking algorithm.

3. Entropy Analysis for Detecting Single-Layer Packing, Re-Packing, or Multi-Layer Packing Algorithm

In order to detect the re-packing or multi-layer packing algorithm, we first extract the entropy pattern by unpacking the executable, and then we compare features of the re-packing or multi-layer packing algorithms with those of single-layer or non-packed executables. In this part, we briefly describe the operations involved in the re-packing and multi-layer packing process of packed executables, using entropy analysis and the symbolic representation.

3.1. Packer Complexity Type

Packers are programs that encode executable files and restore the original executable when the packed files are loaded into memory. There are three kinds of packers (Figure 1):

- *Single-layer packing algorithm*: These packers represent the simplest case. Single-layer packing uses only one packer to pack a given binary. This packing technique changes the size, number of sections and name of an executable (Figure 2).
- *Re-packing algorithm*: These packers contain re-packed unpacking layers, each one executed sequentially to unpack each of the sections. The re-packing algorithm uses the same packer two times to pack a given binary and utilizes compression techniques similar to those of a single-layer packer, but changes the size.
- *Multi-layer packing algorithm*: These packers contain multiple unpacking layers, each one executed sequentially to unpack the following routine. Once the original code has been reconstructed, the last transition transfers the control back to it. This packing uses a combination of potentially different packers to pack a given binary, and extensively facilitates the generation of a large number of packed binaries from the same input binary. Multi-layer packing algorithms change the size, number of sections and name of a single-layer packed executable.

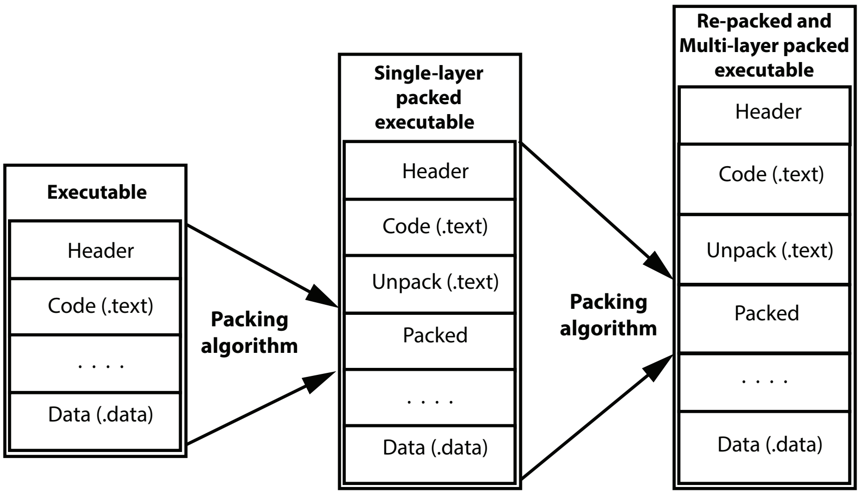

Figure 2. Single-layer packing, re-packing, and multi-layer packing process of a PE file.

3.2. Structure of Packer Detection Algorithm

The main idea of this paper is to measure the entropy values while unpacking re-packed or multi-layer packed executables.

Figure 3 illustrates a process of packer algorithm classification including steps such as unpacking the re-packed and multi-layer packed executables, conversion of the entropy patterns into symbolic representation and comparison of the symbolic representations. Detecting packing algorithms straightaway through the entropy pattern is difficult due to the large size of the data that makes the process time-consuming, generates many errors and hinders the analysis. Therefore, we extracted unique symbolic representation pattern based on entropy patterns. Then,we classified symbolic representation pattern using similarity classification techniques. Figure 4 illustrates the process of conversion of the entropy pattern into the symbolic representation. Figure 4 shows three primary graphs of symbolic representation process. The first graph is illustrates the original entropy pattern, Figure 4a. The second graph demonstrates a normalization of the entropy pattern, Figure 4b. The last graph shows a conversion into symbolic representation, Figure 4c.

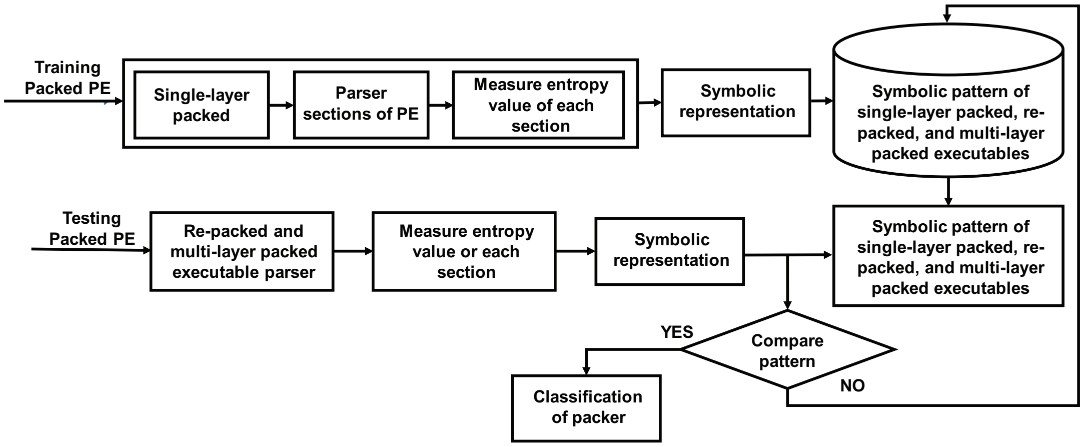
Training Packed PE
Single-layer packed
Parser sections of PE
Measure entropy value of each section
Symbolic representation
Symbolic pattern of single-layer packed, re-packed, and multi-layer packed executables
Testing Packed PE
Re-packed and multi-layer packed executable parser
Measure entropy value or each section
Symbolic representation
Symbolic pattern of single-layer packed, re-packed, and multi-layer packed executables
YES
Compare pattern
NO
Classification of packer

Figure 3. Re-packing and multi-layer packing algorithm detection method.

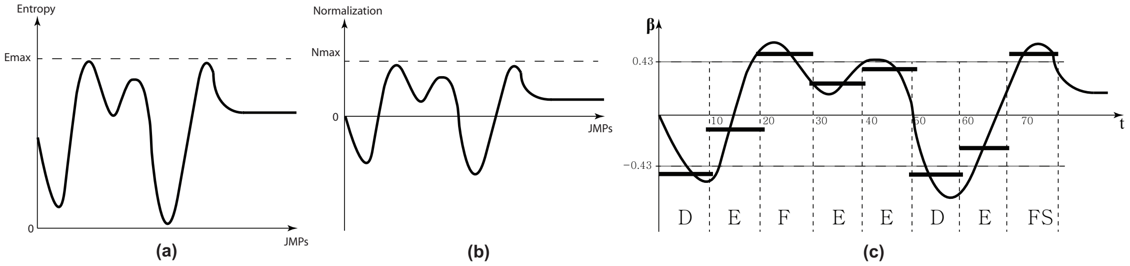

Figure 4. Entropy pattern conversion into the symbolic representation. (**a**) Original entropy pattern; (**b**) Normalized of entropy pattern; (**c**) Symbolic representation of normalized entropy pattern.

3.3. Entropy Analysis and Measurement

Our approach involves the unpacking of re-packed or multi-layer packed executables and identifying their OEPs. Entropy is a method for measuring uncertainty in a series of information units (Jeong et al. [33], Bat-Erdene et al. [2] and Vapnik et al. [41,42]). Measuring the entropy pattern during unpacking process refers to determining the entropy value of re-packed or multi-layer packed executable. The measured entropy of the original information is lower than that of the compressed information. Packers then use decryption or loading stubs to "unpack" the program before resuming normal execution at the original entry point (OEP) of the program. OEP is the address of the first instruction of the decompressed code. During the unpacking process, we utilize a decompression module to unpack packed instructions. The paused process continues to execute the next instruction if the unpacking process is incomplete. The end of the unpacking process can reveal the unpacked code, which was encoded in the data section. This analysis determines whether the unpacking process is complete by measuring the entropy of all sections of the re-packed or multi-layer packed executable. Furthermore, measures the memory space according JMP instruction address. The algorithm used in this procedure is provided in Algorithm 1. Shannon's formula devised to measure the information entropy is as follows:

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

where $H(x)$ is value of the measured entropy value; *P(i)* is the probability of the i-th unit of information in the series of n variables of event x. 2, 10, or Euler's number is usually used as the base number of the logarithm (b).

Algorithm 1: Unpacking packed executable.

```
Input: Re-packed RP or Multi-layer packed executable (MP).
InP-Instruction pointer, UnE-Entropy value, ME-Measure entropy
Output: Entropy values. OEP of RP and MP
To find all section and entry point of RP and MP.
To set a breakpoint to the OEP.
To set Rn to the range of the all section of RP and MP.
// Start analysis.
while the PROCESS is not finish do
    InP ← a current InP
    // ME in only this condition.
    if InP is for a JMP then
        ME of Rn.
    end
    if $-\xi \leq ME \leq +\xi$ then
        // The entropy value is stable
        if Jump into Rn from outside of Rn is true then
            OEP ← The next instruction address
            Break this loop.
        end
    end
end
```

3.4. Conversion into Symbolic Representation (SAX)

Recent literature review concedes that SAX is one of the viable methods that utilizes a similarity measurement, which is easy to compute, see lookup table as shown in Table 1. Lin et al. [43] defined the symbolic representation of time-series as the Symbolic Aggregate approXimation (SAX), see Table 2. In our research SAX is applied as follows, Figure 5:

- Scale and normalize time-series;
- Reduce the dimensionality of the time-series using the Piecewise Aggregate Approximation (PAA) by a method demonstrated in Lin et al. [43] and Keogh et al. [44,45]. The dimensionality reduction technique is analogous to Fourier transformation and Wavelets;
- Discretize PAA representation of the time-series that is achieved by determining the number and location of breakpoints, as demonstrated by Yi et al. [46] and Keogh et al. [44].

The lower-bound distance between two symbolic strings can be proven by simply pointing the existing proofs of PAA representation. First, we transform the entropy values into the PAA representation; Next, we symbolize the PAA representation into SAX. Symbolic Aggregate Approximation is based on the fact that a normalized time-series have high Gaussian distribution [2,43,47].

Table 1. A lookup table where the breakpoints divide a Gaussian distribution in an arbitrary number of equiprobable regions.

β_i	*a*				
	3	**4**	**5**	**6**	**7**
β_1	−0.43	−0.67	−0.84	−0.97	−1.07
β_2	0.43	0	−0.25	−0.43	−0.57
β_3		0.67	0.25	0	−0.18
β_4			0.84	0.43	0.18
β_5				0.97	0.57
β_6					1.07

Table 2. Symbolic representation: SAX.

Variable	A Series Data
X	A time series $X = x_1, x_2, ..., x_n$
$\overline{X}$	A PAA of a time series $\overline{X} = \overline{x_1}, \overline{x_2}, ..., \overline{x_M}$,
$\widetilde{X}$	SAX of time series $\widetilde{X} = \widetilde{x_1}, \widetilde{x_2}, ..., \widetilde{x_M}$
M	The number of PAA segments representing time series X, where $M \leq n$
a	Alphabet size. a is integer, where $a > 2$

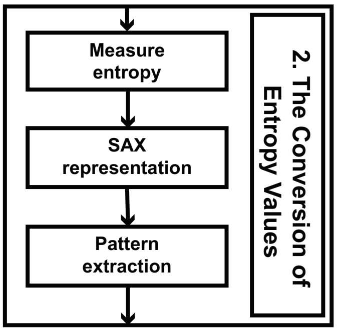

Figure 5. Structure of the pattern extraction method.

In Algorithm 2, we present the process of converting the entropy value into SAX. The entropy values from unpacking the re-packed or multi-layer packed executables are abbreviated as *En*, the scaling values of the original entropy pattern-$\overline{En}$, the normalized scaling entropy values-$\overline{En}_{norm}$, the breakpoints-β, the number of symbols-$\phi(\beta)$, the sequence of symbols-G, and the SAX pattern as $\widetilde{X}$ respectively.

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

where r is n/M; n is the length of the string and M is the length of the original time-series. In other words, we divide data into M equally sized frames to reduce the time-series from n dimensions to M dimensions, see Table 2. We normalize each entropy values to have a mean of zero and a standard deviation of one before converting it to PAA representation. Breakpoints ($\mathcal{B}$), a sorted list of numbers, $\mathcal{B} = \beta_1, \beta_2, \ldots, \beta_{a-1}$, where $\beta_{i-1} < \beta_i$. It divides the area under N(0, 1) Gaussian curve into equal areas. The size of the alphabet is a random integer a, greater than 2.

$$from \quad \beta_i \quad to \quad \beta_{i+1} = \frac{1}{a} \tag{3}$$

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

Breakpoints are determined from statistical lookup table, as shown in Table 1 Lin et al. [43]. We transform the original entropy values into PAA representations, $\overline{Q}$ and $\overline{S}$ using Formula (2), we can then obtain a lower bounding approximation using the Euclidean distance ($\mathcal{ED}$) [44]:

$$\mathcal{D}(\overline{Q},\overline{S}) = \sqrt{\frac{n}{N}} \cdot \sqrt{\sum_{i=1}^{M}(\overline{Q}_i - \overline{S}_i)^2}. \tag{4}$$

Algorithm 2: Conversion into SAX

```
Input: $En$, $\overline{En}$, $\overline{En}_{norm}$, $\beta$ and $\phi(\beta)$.
// Extract symbolic presentation pattern, which will be detect re-packing algorithm and
  multi-layer packing algorithms.
Output: $Gn$ and $\widetilde{X}$.
$\overline{En} \leftarrow$ scale $En$
// Scale the entropy values for SAX.
$\overline{En}_{norm} \leftarrow$ Normalize $(\overline{En})$
Loop $i = 0; i < \phi(\beta); i < i + 1$
if $\beta_{i-1} < \beta_i$ then
    $\phi(\beta) \leftarrow$ Divide $(\overline{En}_{norm})$
    // Divide normalized entropy values using $\phi(\beta)$.
    $G \leftarrow$ Convert $(\overline{En}_{norm})$
    $\widetilde{X} \leftarrow$ new SAX pattern $(Gn)$
    // New unique symbolic pattern
    // Classify $\widetilde{X}$ using fidelity similarity measurement for detection re-packing or multi-layer
      packing algorithms.
end
End Loop
```

4. Classifier

Our proposed method used fidelity, similarity measurement, for classification. This classification is the simplest and more intuitive and is based on the concept of similarity. We provide a detailed explanation in Section 4.1. We used a method, to generate high-accuracy patterns for detecting re-packing or multi-layer packing algorithms. We designed a classifier using several approaches to classification. In this work, we performed machine learning classification based on simple patterns. The main concept of the experiment is detecting re-packer or multi-layer packers. We arranged available re-packer or multi-layer packers by determining their nearest similar simple patterns and placing them into the families with analogous patterns. We created a new database of families if the family of similar patterns did not exist. Figure 6 demonstrates how we utilized the similarity measurements of classification methods to classify the re-packing or multi-layer packing algorithms. In our proposed method we used similarity classification such as fidelity. We extracted entropy patterns of known/unknown single-layer packing algorithms, as shown in the previous works [2,47]. We used entropy pattern of single-layer packing algorithm to detect re-packing and multi-layer packing algorithms. We classified re-packing or multi-layer packing algorithms in the five classes based on their graphically visualized patterns, including New class, Increasing class, Decreasing class, Combination class, and Constant class. Straightly using entropy patterns to detect re-packing or multi-layer packing algorithms is tough to perform further analysis because of a significant amount of entropy values and the number of errors incurred. Therefore, we used classification methods, which significantly reduce complexity.

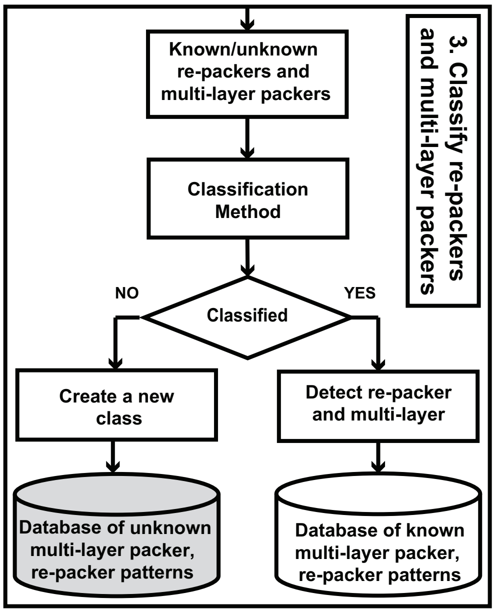
Known/unknown re-packers and multi-layer packers
Classification Method
Classified
NO
YES
Create a new class
Detect re-packer and multi-layer
Database of unknown multi-layer packer, re-packer patterns
Database of known multi-layer packer, re-packer patterns
3. Classify re-packers and multi-layer packers

Figure 6. Structure of a classifier.

4.1. Fidelity Coefficient Similarity Measurement

We use the fidelity [2] similarity measures to characterize the similarity between the entropy values of each packed executable. For the given sequences $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$ of random positive values $(x_j, y_j > 0)$, the fidelity $F(x, y)$ is defined as:

$$F(x,y) = \frac{\sum_{i=1}^{n} \sqrt{x_i \cdot y_i}}{\sqrt{\sum_{i=1}^{n} x_i}\sqrt{\sum_{i=1}^{n} y_i}} \tag{5}$$

where $0 \leqslant F(x, y) \leqslant 1$.

When we extract symbolic representation patterns from entropy patterns of training and testing packed executables, we perform fidelity similarity measurement on the patterns. Table 3 shows the fidelity performance of experiments on the single-layer packed, re-packed, or multi-layer packed executables using Aspack, Alternate_EXE, nPack, NsPack, RLPack and VMProtect packing algorithms. We extract symbolic representation patterns of re-packer and multi-layer packer. Then we compare symbolic representation patterns of re-packer or multi-layer packer with that of training packed executables, including patterns of single-layer packers. When we unpack the single-layer packed executable, we define OEP from the first section. When unpack the multi-layer packed executable, define OEP from several sections. Therefore, we compare combination section of single-layer packer with sections of multi-layer packer (Table 3).

Table 3. Fidelity similarity for re-packing and multi-layer packing algorithms.

Single-Layer Packer	**Re-Packer**	$F(x,y)$
Alternate_Exe	Alternate_Exe + Alternate_Exe	0.9920
nPack	nPack + nPack	0.9908
NsPack	NsPack + NsPack	0.9982
RLPack	RLPack + RLPack	0.9914
VMProtect	VMProtect + VMProtect	0.9999
Single-Layer Packer	**Multi-Layer Packer**	$F(x,y)$
Aspack	Section 1	0.9949
NsPack	Section 0	0.9821
NsPack	Section 1	0.9965
VMProtect	Section 4	1.0000
RLPack	Section 1	1.0000
VMProtect	Section 3	1.0000
VMProtect	Section 1	1.0000
NsPack	Section 0	0.9961
VMProtect	Section 1	1.0000
RLPack	Section 0	0.9908

4.2. Incremental Aggregate Analysis

When detecting packing algorithms using the similarity measurement is hard we use an incremental aggregate analysis [47]:

$$\Psi(W) \equiv (W_0, W_{\max}, W_{\min}, \sigma_1, \cdots, \sigma_m), \tag{6}$$

where $W_0 = w_1$ is the initial entropy value; $W_{\max} \equiv \max\{w_1, \cdots, w_n\}$ and $S_{\min} \equiv \min\{w_1, \cdots, w_n\}$; $m \equiv n/Z$, with Z as the coarse-graining parameter ($Z > 1$) and represents the incremental change in the coarse-grained entropy values:

$$\sigma_j \equiv \begin{cases} 1 & (w_{(j+1)Z} - w_{jz} > 0) \\ 0 & (w_{(j+1)Z} - w_{jz} = 0) \quad (j = 1, \cdots, l) \\ -1 & (w_{(j+1)Z} - w_{jz} < 0) \end{cases} \tag{7}$$

5. Assessment of the Classification Method

The dataset used in this experiment contains six benign executables for packing algorithms, 2196 re-packed and multi-layer packed benign executables and 19 popular packers. The data samples of the six benign executables are selected from Windows System 32 files. We conducted around 2500 experiments on re-packing and multi-layer packing algorithm detection. Table 4 demonstrates the combination of single-layer packing, re-packing, or multi-layer packing algorithms for the experiment. First, we extracted re-packed executables from single-layer packed executables. We selected orange color for the re-packed executables; Second, we extracted multi-layer packed executables from a combination of single-layer packers and indicated by blue and green colors. The data sample consisted of training and testing packed executables in the ratio of 50:50, where from a sample of 2196 re-packed and multi-layer packed executables 1098 were training sets. We compared performances of different classification techniques by assessing their correctness in the prediction of the actual classification of the packers. Before introducing the metrics, we defined the classification of the re-packers and multi-layer packers. Positive if the packer is predicted to be in the Increasing class, and negative if it is not. Let $\mathcal{ACC}$ denote the accuracy of classification based on the percentage of tested set packers that are correctly identified by the classifier. Where accuracy $\mathcal{ACC}$ indicates the overall effectiveness. In Formula (8) accuracy $\mathcal{ACC}$ indicates the overall effectiveness.

$$\mathcal{ACC} = \frac{(TPR + TNR)}{(P + N)} = \frac{(TPR + TNR)}{n}. \tag{8}$$

Table 4. Experimental results of packed executables with the single-layer packers, re-packers, or multi-layer packers.

			1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**	**11.**	**12.**	**13.**	**14.**	**15.**	**16.**	**17.**	**18.**	**19.**
N			**FIRST PACKER**																		
		PACKERS	**Alternate_Exe**	**FSG**	**RLPack**	**NsPack**	**UPXN**	**UPX-iT**	**MPRESS**	**Morphine**	**nPack**	**Themida**	**VMProtect**	**Aspack**	**Molebox**	**Petite**	**ASProtect**	**MEW**	**Yoda'sCrypter**	**PELock**	**tELock**
1.	**SECOND PACKER**	**Alternate_Exe v2.000**	Yes	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Yes	Failed	Failed
2.		**FSG v2.0**	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Yes	Failed	Yes	Failed	Failed	Failed	Yes	Failed	Failed
3.		**RLPack v1.2**	Yes	Failed	Yes	Failed	Yes	Yes	Failed	Failed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Failed	Yes	Yes	Yes
4.		**NsPack v3.7**	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Failed	Failed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Failed	Yes	Yes
5.		**UPXN v301**	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed
6.		**UPX-iT v1.0**	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed
7.		**MPRESS v1.27**	Failed	Failed	Failed	Failed	Yes	Yes	Failed	Failed	Failed	Yes	Yes	Failed	Yes	Failed	Failed	Failed	Failed	Yes	Failed
8.		**Morphine v1.6**	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed
9.		**nPack v1.1.300.2006**	Yes	Failed	Failed	Failed	Yes	Yes	Yes	Failed	Yes	Yes	Failed	Failed	Yes	Failed	Yes	Failed	Yes	Yes	Failed
10.		**Themida v2.4**	Failed	Yes	Yes	Yes	Yes	Failed	Failed	Failed	Yes	Yes	Yes	Failed	Yes	Yes	Yes	Failed	Yes	Yes	Yes
11.		**VMProtect v1.7**	Failed	Failed	Yes	Yes	Failed	Yes	Yes	Failed	Failed	Failed	Yes	Failed	Failed	Yes	Yes	Failed	Failed	Failed	Failed
12.		**Aspack v2.28**	Yes	Failed	Failed	Yes	Yes	Yes	Yes	Failed	Yes	Failed	Yes	Yes	Yes	Yes	Yes	Failed	Yes	Yes	Failed
13.		**Molebox v2.6.1**	Yes	Failed	Yes	Yes	Yes	Yes	Yes	Failed	Yes	Yes	Yes	Yes	Failed	Yes	Yes	Yes	Yes	Yes	Yes
14.		**Petite v2.3**	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Yes	Failed	Failed	Failed	Failed	Failed	Yes	Yes	Failed
15.		**ASProtect v.1.23**	Yes	Failed	Failed	Failed	Yes	Yes	Failed	Failed	Yes	Yes	Yes	Failed	Yes	Failed	Failed	Failed	Yes	Failed	Yes
16.		**MEW v1.2**	Failed	Failed	Yes	Failed	Failed	Failed	Yes	Failed	Yes	Yes	Yes	Failed		Yes	Yes	Yes	Yes	Yes	Failed
17.		**Yoda's Crypter v1.3**	Yes	Yes	Failed	Failed	Yes	Yes	Yes	Failed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
18.		**PELock v2.0**	Failed	Yes	Failed	Failed	Yes	Yes	Failed	Failed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Failed	Yes	Failed	Yes
19.		**tELock v0.98**	Failed	Failed	Failed	Failed	Yes	Yes	Failed	Failed	Yes	Yes	Yes	Failed	Yes	Yes	Failed	Yes	Yes	Yes	Failed

Gray is re-packed benign executables; for example: **Alternate_Exe + Alternate_Exe**. **Green** is two way packed multi-layer packed executables; for example: **NsPack + Aspack** and **Aspack + NsPack**. **Blue** is one way packed multi-layer packed executables; for example: **Alternate_Exe + NsPack**. **Yes** is executable packed with re-packing or multi-layer packing algorithm. **Failed** is executable not packed with re-packing or multi-layer packing algorithm.

Table 5. Experimental results of the re-packing and multi-layer packing algorithms.

			1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**
N			**FIRST PACKER**								
		PACKERS	**Alternate_Exe**	**RLPack**	**NsPack**	**nPack**	**Themida**	**VMProtect**	**Aspack**	**MEW**	**Yoda'sCrypter**
1.	**SECOND PACKER**	**Alternate_Exe**	Yes	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Yes/not exe
2.		**RLPack**	Yes	Yes	Failed	Yes	Yes/not exe	Yes	Yes	Failed	Yes
3.		**NsPack**	Yes	Yes	Yes	Failed	Yes/not exe	Yes	Yes	Yes	Failed
4.		**nPack**	Yes	Failed	Failed	Yes	Yes/not exe	Failed	Failed	Failed	Yes/not exe
5.		**Themida**	Failed	Yes	Yes	Yes/not exe	Yes	Yes/not exe	Failed	Failed	Yes/not exe
6.		**VMProtect**	Failed	Yes	Yes	Failed	Failed	Yes	Failed	Failed	Failed
7.		**Aspack**	Yes/not exe	Failed	Yes/not exe	Yes	Failed	Yes/not exe	Yes	Failed	Yes/not exe
8.		**MEW**	Failed	Yes	Failed	Yes	Yes	Yes	Failed	Yes	Yes/not exe
9.		**Yoda's Crypter**	Yes/not exe	Failed	Failed	Yes/not exe	Yes/not exe	Yes/not exe	Yes/not exe	Yes/not exe	Yes/not exe

Gray is re-packed benign executables; for example: **Alternate_Exe + Alternate_Exe**. **Green** is two way packed multi-layer packed executables; for example: **NsPack + Aspack** and **Aspack + NsPack**. **Blue** is one way packed multi-layer packed executables; for example: **Alternate_Exe + NsPack**. **Yes** is executable packed with re-packing or multi-layer packing algorithm. **Failed** is executable not packed with re-packing or multi-layer packing algorithm.

Result of Experiments

Following our previous work, we utilized SAX representation method to extract re-packing or multi-layer packing algorithm patterns. The results of experiments on eight packing algorithms selected from 19 possible packing algorithms (Table 5) demonstrated that some executables did not execute the files when the executables were packed re-packed and multi-layer packed. For example Alternate_Exe, RLPack, NsPack, nPack, Themida, VMProtect, Aspack, and MEW. Although Yoda's Cryptor packing algorithm can re-pack or multi-layer pack an executable, re-packed or multi-layer packed executables would not work. We used in the experiment six benign executables, e.g., calc.exe and notepad.exe, to detect re-packing or multi-packing algorithms. We packed each executable one time, two times, and combination times using 19 packing algorithms. The experiment was conducted as follows:

1. We depute packed benign notepad.exe from six packed benign executables.
2. We extract entropy pattern of packed notepad.exe by 19 packing algorithms including the singe-layer packer, the re-packer, and the multi-layer packer.
3. We scale entropy pattern of each packed notepad executable.
4. We calculate the number of symbols $\phi(\beta)$ for converting using SAX.

Overall, we measure the similarity of all packers and extracted patterns of the packing algorithms through SAX:

$$\phi(\beta) = \frac{E}{M}, \tag{9}$$

$\phi(\beta)$ is the number of symbols used for extracting the packing algorithm pattern. In Figure 7 we demonstrate SAX pattern of packed Notepad.exe with Aspack packer, and also show the combination of $\phi(\beta)$ for converting the entropy pattern into SAX. The results of experiments denote that our proposed method is not only useful for identifying re-packing or multi-layer packing algorithms, but also applicable to re-packed and multi-layer packed executables. For instance, first, we used features of single-layer packed, re-packed, or multi-layer packed executables to create the operation of each re-packed or multi-layer packed executables, such as the number of sections, and the size and name of the section. Second, we found that the nine re-packed or multi-layer packed executable's entropy patterns of 8 packing algorithms shown in Figures 8 and 9 and classified into five classes [47]:

- *New class* includes MEW, Yoda's Cryptor;
- *Increasing class* includes Alternate_EXE, NsPack, RLPack;
- *Decreasing class* consists of nPack;
- *Combination class* consists of VMProtect, Themida and Aspack;
- *Constant class* includes TELock.

Experimental results for entropy patterns of single-layer packing and multi-layer packing are demonstrated in Figure 9. To detect the multi-layer packing algorithm we used two single-layer packing algorithms. For instance, Figure 9a demonstrates entropy patterns of single-layer packing of Notepad.exe benign executable with two single-layer packing algorithms including Aspack(Notepad_Apack.exe) and NsPack(Notepad_NsPack.exe). Figure 9b illustrates entropy pattern of multi-layer packing of Notepad.exe; first, with Aspack packing algorithm, and next with NsPack algorithm (Notepad_NsPack_Aspack.exe). Next, we compare and measure fidelity similarities of single-layer and multi-layer packing algorithms (Table 3). In Figure 8 we illustrate entropy patterns of single-layer packing and re-packing. First, we identify re-packing algorithm based on the single-layer packing algorithm. For instance, entropy patterns of Notepad.exe benign executable packed with the single-layer packing and re-packing algorithms, is demonstrated in Figure 8c. The pattern of Notepad_RLPack.exe is very similar to the pattern of Notepad_RLPack_RLPack.exe because they are using same packing algorithms. Then, we measure fidelity similarity of single-layer packing and re-packing algorithms (Table 3).

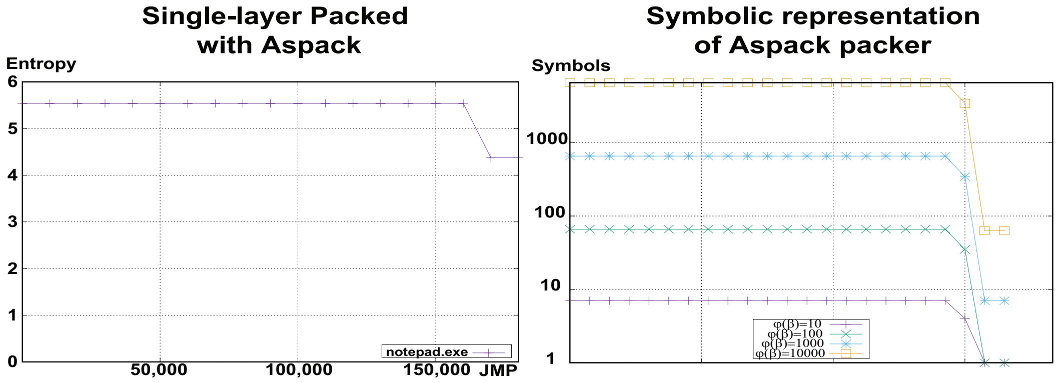

Figure 7. Entropy patterns converted into SAX using different values of $\phi(\beta)$ for the Aspack packers.

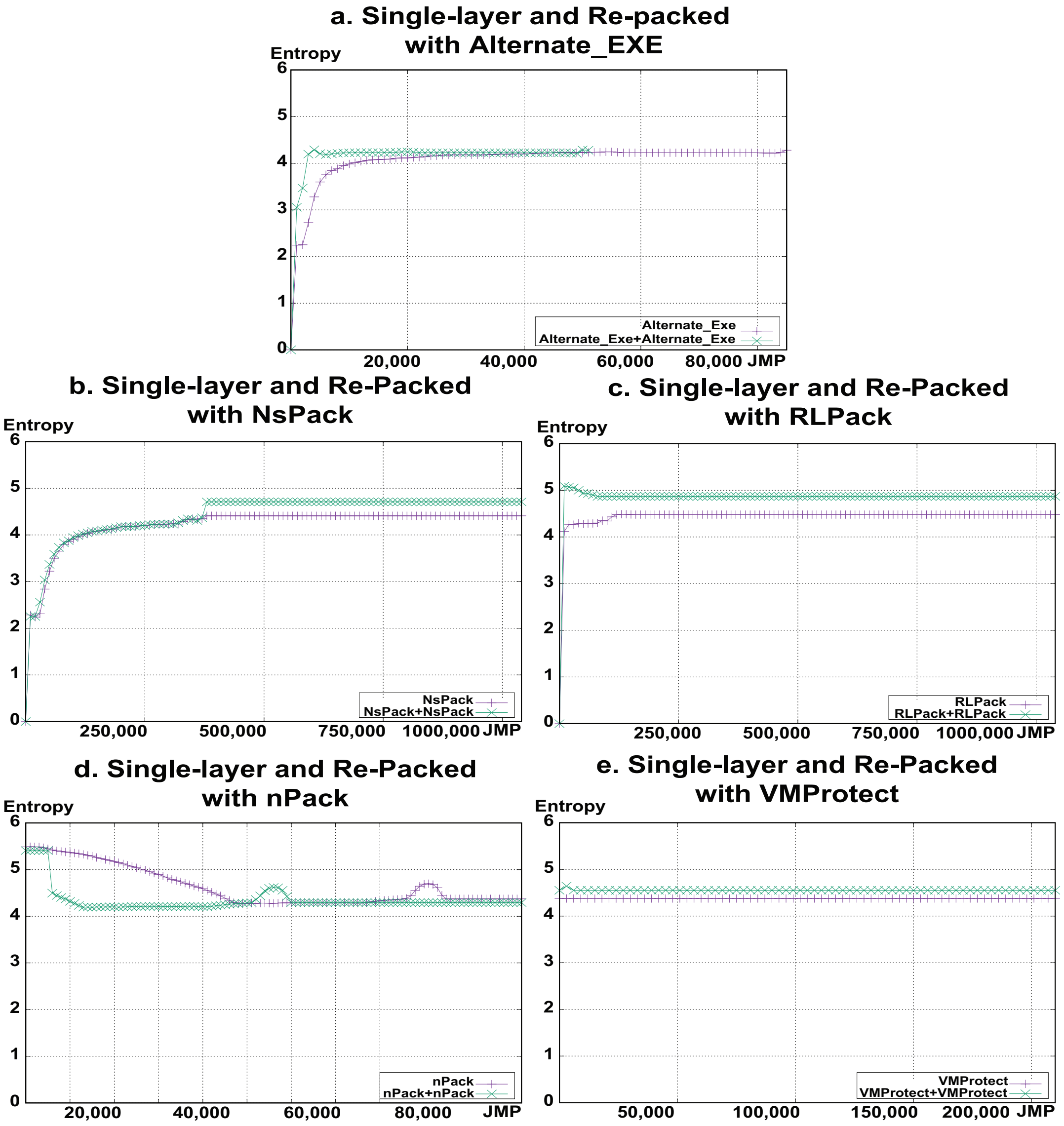

Figure 8. Entropy patterns of single-layer packed and re-packed executable of Notepad.exe when a packer is (**a**) Alternate_EXE; (**b**) NsPack; (**c**) RLPack; (**d**) nPack; (**e**) VMProtect. y-axis is entropy values and x-axis is "JMP" instruction numbers.

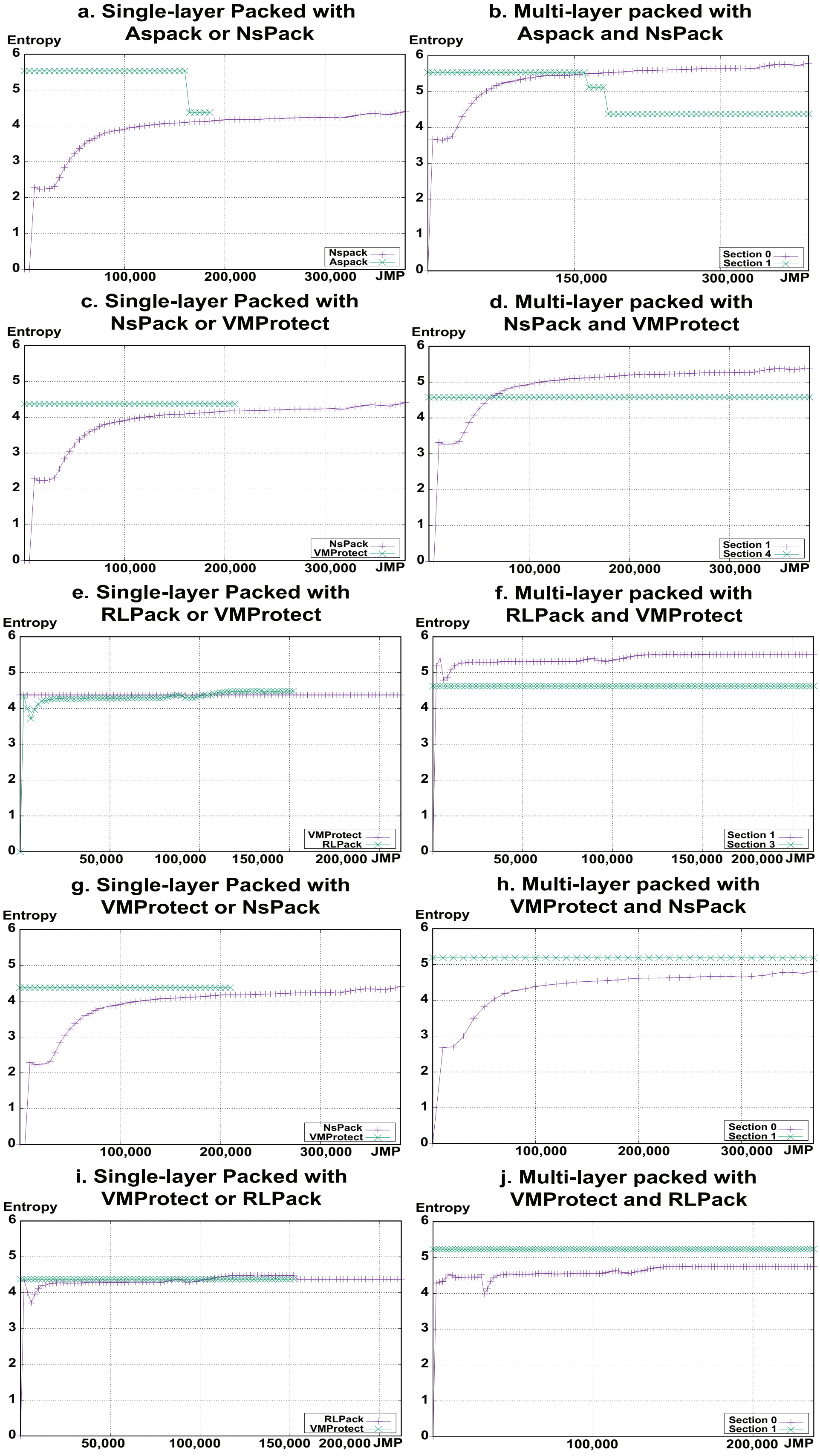

Figure 9. Entropy patterns of single-layer packed and multi-layer packed executable of Notepad.exe using two packers. (**a**) NsPack or Aspack; (**b**) NsPack and Aspack; (**c**) NsPack or VMProtect; (**d**) NsPack and VMProtect; (**e**) RLPack or VMProtect; (**f**) RLPack and VMProtect; (**g**) VMProtect or NsPack; (**h**) VMProtect and NsPack; (**i**) VMProtect or RLPack; (**j**) VMProtect and RLPack. *y*-axis is entropy values and *x*-axis is "JMP" instruction numbers.

Table 6 shows results of an operation for single-layer packed, re-packed, or multi-layer packed executables using few features of classification. In other words, we verify two ways for detecting re-packers or multi-layer packers using the features of executables and their entropy patterns. Symbols are used to express the entropy patterns. Then, the patterns of eight re-packing and multi-layer packing algorithms converted using SAX, Table 7. Based on the fidelity similarity method we detect re-packing and multi-layer packing algorithms of any packed executables. The average accuracy using re-packers and multi-layer packer are 98.5% and 97.5%, respectively (Table 7). The accuracy of both VMProtect and MEW re-packing and multi-layer packing algorithms is 100%, the minimum accuracy is 95.8%, which relates to the RLPack multi-layer packing algorithm. An average true positive rate ($\mathcal{T}_r$) of 97.8%, a false positive rate ($\mathcal{F}_r$) of 1.8%, precision ($\mathcal{P}$) of 98.2%, and recall ($\mathcal{R}$) of 96.2% for detection of re-packing algorithms. Average true positive rate ($\mathcal{T}_r$) is 95.5%, false positive rate ($\mathcal{F}_r$) is 2.3%, the precision ($\mathcal{P}$) is 97.7%, and the recall ($\mathcal{R}$) is 96.8% for detection multi-layer packing algorithms.

Table 6. Operation for single-layer packing, re-packing, and multi-layer packing algorithms.

Type of Packers	Operation		Name of Operation	Packers
	Section Number	**Size**		
Single-layer packers	Equivalent	Increment	NESI	Themida
		Decrement	NESD	-
	Increment	Increment	NISI	nPack; VMProtect
		Decrement	NISD	-
	Decrement	Increment	NDSI	Aspack; MEW; NsPack
		Decrement	NDSD	Alternate_Exe; RLPack
Re-packers	Equivalent	Increment	NESI	-
		Decrement	NESD	-
	Increment	Increment	NISI	Aspack + Apack; nPack + nPack RLPack + RLPack; Themida + Themida VMProtect + VMProtect
		Decrement	NISD	-
	Decrement	Increment	NDSI	Nspack + Nspack
		Decrement	NDSD	Alternate_Exe + Alternate_Exe
Multi-layer packers	Equivalent	Increment	NESI	-
		Decrement	NESD	Alternate_Exe + nPack
	Increment	Increment	NISI	nPack + Aspack; NsPack + VMProtect RLPack + VMProtect
		Decrement	NISD	-
	Decrement	Increment	NDSI	Alternate_Exe + NsPack; Aspack + NsPack Aspack + RLPack; Alternate_Exe + RLPack MEW + NsPack; nPack + MEW RLPack + MEW; nPack + RLPack RLPack + NsPack; Themida + MEW VMProtect + Aspack; VMProtect + MEW VMProtect + RLPack
		Decrement	NDSD	-

Table 7. Detailed accuracy of re-packer and multi-layer packer using the fidelity.

	N	Packing Algorithm	$\mathcal{T}_r$%	$\mathcal{F}_r$%	$\mathcal{ACC}$%	$\mathcal{P}$%	$\mathcal{R}$%
Re-Packer	1.	Alternate_EXE	96.0	1.2	99.0	98.8	96.3
	2.	ASPACK	96.0	4.0	97.5	96.0	95.2
	3.	MEW	100.0	0.0	100.0	100.0	100.0
	4.	NPACK	100.0	0.8	98.4	99.2	100.0
	5.	NSPACK	98.5	2.3	96.7	97.7	95.5
	6.	RLPACK	95.8	4.2	97.0	95.8	90.8
	7.	THEMIDA	96.0	1.9	99.0	98.1	92.3
	8.	VMPROTECT	100.0	0.0	100.0	100.0	100.0
		AVERAGE	**97.8**	**1.8**	**98.5**	**98.2**	**96.2**
	N	**PACKERS**	$\mathcal{T}_r$%	$\mathcal{F}_r$%	$\mathcal{ACC}$%	$\mathcal{P}$%	$\mathcal{R}$%
Multi-Layer Packer	1.	Alternate_EXE	93.7	3.0	97.2	96.9	96.3
	2.	ASPACK	93.0	2.0	96.8	97.9	96.3
	3.	MEW	94.5	1.3	98.8	98.6	98.0
	4.	NPACK	95.0	5.5	97.3	94.5	96.8
	5.	NSPACK	98.5	2.3	95.9	97.7	94.8
	6.	RLPACK	93.0	3.5	95.8	96.4	95.8
	7.	THEMIDA	96.0	0.5	98.1	99.5	96.7
	8.	VMPROTECT	100.0	0.0	100.0	100.0	100.0
		AVERAGE	**95.5**	**2.3**	**97.5**	**97.7**	**96.8**

6. Conclusions

We proposed a new technique for re-packing or multi-layer packing algorithms detection using SAX symbolic representation; and classify re-packing, or multi-layer packing algorithms in five classes using scaling of entropy patterns with symbolic representation patterns. We assessed our proposed method on a large data set, consisting of 2196 benign single-layer packed, re-packed, and multi-layer packed executable files and more than 1098 known benign re-packed and multi-layer packed executable files.

The proposed method unpacked and identified re-packed and multi-layer packed executables successfully, while maintaining high accuracy. Our work demonstrates that algorithms produce a highly accurate re-packer and multi-layer packer classification based on real life data. The results of experiments denote that the technique we proposed is not only useful for identifying re-packing or multi-layer packing algorithms, but also applicable to re-packed and multi-layer packed executables.

Further research may extract symbolic patterns from new re-packed or multi-layer packed malware and use supervised classification methods for re-packer and multi-layer packer classification and detection.

Acknowledgments: This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0190-16-2011, Development of Vulnerability Discovery Technologies for IoT Software Security).

Author Contributions: Munkhbayar Bat-Erdene initiated the main idea and participated in the annotation of the data. Munkhbayar Bat-Erdene programmed the experiments, and Heejo Lee supervised the study. Munkhbayar Bat-Erdene first drafted the manuscript. Heejo Lee, Munkhbayar Bat-Erdene, Taebeom Kim and Hyundo Park reviewed and further developed the manuscript and contributed to the final version. Munkhbayar Bat-Erdene and Taebeom Kim developed a custom debugger forWindows 32bit and 64bit PE programs. They focused on CPU instructions of unpacking behaviors and changing the entropy values. The debugger automatically measures entropies of each section at the run time, and it detects an original entry point when entropy values are stable. After detection of the original entry point, the debugger dumps memories, and we verify whether or not the unpacking process is finished. Hyundo Park contributed in making the experimental hypothesis and validating the experiment results. Furthermore, he reviewed the paper from beginning to the end. All authors read and approved the final version of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Choi, H.; Zhu, B.B.; Lee, H. Detecting Malicious Web Links and Identifying Their Attack Types. In Proceedings of the 2nd USENIX Conference on Web Application Development, Portland, OR, USA, 15–16 June 2011; pp. 125–136.
2. Bat-Erdene, M.; Kim, T.; Li, H.; Lee, H. Dynamic classification of packing algorithms for inspecting executables using entropy analysis. In Proceedings of the IEEE International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE), Fajardo, PR, USA, 22–24 October 2013; pp. 19–26.
3. Yan, W.; Zhang, Z.; Ansari, N. Revealing Packed Malware. *IEEE Secur. Priv.* **2008**, *6*, 65–69.
4. Wright, C.S. Packer Analysis Report Debugging and Unpacking the NsPack 3.4 and 3.7 Packer. Available online: https://www.sans.org/reading-room/whitepapers/malicious/packer-analysis-report-debugging-unpacking-nspack-34-37-packer-33428 (accessed on 14 March 2017).
5. Guo, F.; Ferrie, P.; Chiueh, T.C. A study of the packer problem and its solutions. In Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID), Cambridge, MA, USA, 15–17 September 2008; pp. 98–115.
6. Yara Rules Project. 2016. Available online: https://github.com/Yara-Rules/rules/ (accessed on 14 March 2017).
7. GitHub. 2017. Available online: https://github.com/m-dwyer/packer-malware (accessed on 14 March 2017).
8. Osaghae, E.O. Classifying Packed Programs as Malicious Software Detected. *Int. J. Inf. Technol. Electr. Eng.* **2016**, *5*, 22–25.
9. Jacob, G.; Comparetti, P.M.; Neugschwandtner, M.; Kruegel, C; Vigna, G. A static, packer-agnostic filter to detect similar malware samples. In Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Heraklion, Greece, 26–27 July 2012; pp. 102–122.
10. Brosch, T.; Morgenstern, M. *Runtime Packers: The Hidden Problem*; Black Hat: Washington, DC, USA, 2006.
11. ASPack Software. Available online: http://www.aspack.com/aspack.html (accessed on 14 March 2017).
12. Northfox's Dungeon. MEW Software. Available online: https://web.archive.org/web/20070831063728/http://northfox.uw.hu/index.php?lang=eng&id=dev (accessed on 14 March 2017).
13. ASPack Software. Available online: http://www.aspack.com/asprotect32.html (accessed on 14 March 2017).
14. NsPack 3.7. Available online: http://nspack.download-230-13103.programsbase.com/ (accessed on 14 March 2017).
15. Oreans Technologies. Available online: http://www.oreans.com/themida.php (accessed on 14 March 2017).
16. RLPack 1.2. Available online: http://rlpack.software.informer.com/1.2/ (accessed on 14 March 2017).
17. VMPsoft. Available online: http://vmpsoft.com/products/vmprotect/ (accessed on 14 March 2017).
18. Alternate_EXE Packer. Available online: http://www.alternate-tools.com/pages/c_exepacker.php?lang=ENG (accessed on 14 March 2017).
19. Al-Anezi, M.M.K. Generic packing detection using several complexity analysis for accurate malware detection. *Int. J. Adv. Comput. Sci. Appl.* **2016**, doi:10.14569/IJACSA.2014.050102.
20. Santos, I.; Ugarte-Pedrero, X.; Sanz, B.; Laorden, C.; Bringas, P.G. Collective classification for packed executable identification. In Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference; Perth, Australia, 1–2 September 2011; pp. 23–30.
21. McAfee. The Good, the Bad, and the Unknown. Available online: http://www.techdata.com/mcafee/files/MCAFEE_wp_appcontrol-good-bad-unknown.pdf (accessed on 14 March 2017).
22. Cesare, S.; Xiang, Y; Zhou, W. Malwise—An effective and efficient classification system for packed and polymorphic malware. *IEEE Trans. Comput.* **2013**, *62*, 1193–1206.
23. Offensive Computing. Available online: http://www.offensivecomputing.net/ (accessed on 14 March 2017).
24. VX Heavens. 2017. Available online: http://vxheaven.org/ (accessed on 14 March 2017).
25. Ugarte-Pedrero, X.; Santos, I.; Sanz, B.; Laorden, C.; Bringas, P.G. Countering entropy measure attacks on packed software detection. In Proceedings of the 2012 IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 14–17 Janaury 2012; pp. 164–168.
26. Shafiq, M.Z.; Tabish, S.M.; Mirza, F; Farooq, M. Pe-miner: Mining structural information to detect malicious executables in realtime. In Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID), Saint-Malo, France, 23–25 September 2009; pp. 121–141.

27. Liu, L.; Ming, J.; Wang, Z.; Gao, D; Jia, C. December. Denial-of-service attacks on host-based generic unpackers. In Proceedings of the International Conference on Information and Communications Security (ICICS), Beijing, China, 14–17 December 2009; pp. 241–253.
28. Meijer, B.R. Rules and algorithms for the design of templates for template matching. In Proceedings of the 11th IAPR International Conference on Pattern Recognition, Conference A: Computer Vision and Applications, The Hague, The Netherlands, 30 August–3 September 1992; pp. 760–763.
29. Cesare, S.; Xiang, Y. Classification of malware using structured control flow. In Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing, Brisbane, Australia, 1 January 2010; pp. 61–70.
30. Burges, C.J. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.* **1998**, *2*, 121–167.
31. Martignoni, L.; Christodorescu, M.; Jha, S. Omniunpack: Fast, generic, and safe unpacking of malware. In Proceedings of the Computer Security Applications Conference (ACSAC), Miami Beach, FL, USA, 10–14 December 2007; pp. 431–441.
32. Kang, M.G.; Poosankam, P.; Yin, H. Renovo: A hidden code extractor for packed executables. In Proceedings of the ACM Workshop on Recurring Malcode, Alexandria, VA, USA, 2 November 2007 ; pp. 46–53.
33. Jeong, G.; Choo, E.; Lee, J.; Bat-Erdene, M.; Lee, H. Generic unpacking using entropy analysis. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software (MALWARE), Nancy, France, 19–20 October 2010; pp. 98–105.
34. Devi, D.; Nandi, S. Detection of packed malware. In Proceedings of the First International Conference on Security of Internet of Things, Kollam, India, 17–19 August 2012 ; pp. 22–26.
35. Perdisci, R.; Lanzi, A.; Lee, W. Classification of packed executables for accurate computer virus detection. *Pattern Recognit. Lett.* **2008**, *29*, 1941–1946.
36. An in-Depth Look Into the Win32 Portable Executable File Format—Part 2. Available online: http://www.delphibasics.info/home/delphibasicsarticles/anin-depthlookintothewin32portableexecutablefileformat-part2 (accessed on 14 March 2017).
37. Pietrek, M. Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. Available online: https://msdn.microsoft.com/en-us/library/ms809762.aspx (accessed on 14 March 2017).
38. Ugarte-Pedrero, X.; Balzarotti, D.; Santos, I.; Bringas, P.G. SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In Proceedings of the IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 659–673.
39. Lyda, R.; Hamrock, J. Using entropy analysis to find encrypted and packed malware. *IEEE Secur. Priv.* **2007**, *5*, 40–45.
40. Sun, L.; Versteeg, S.; Boztaş, S.; Yann, T. Pattern recognition techniques for the classification of malware packers. In Proceedings of the 15th Australasian conference on Information security and privacy, Sydney, Australia, 5–7 July 2010; pp. 370–390.
41. Vapnik, V. *The Nature of Statistical Learning Theory*; Springer: Berlin/Heidelberg, Germany, 1995.
42. Vapnik, V.; Chervonenkis, A.Y. *Theory of Pattern Recognition: Statistical Problems of Learning*; Nauka: Moscow, Russia, 1974.
43. Lin, J.; Keogh, E.; Lonardi, S; Chiu, B. A symbolic representation of time series, with implications for streaming algorithms. In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, CA, USA, 13–13 June 2003; pp. 2–11.
44. Keogh, E.; Kasetty, S. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Min. Knowl. Discov.* **2003**, *7*, 349–371.
45. Keogh, E.; Chakrabarti, K.; Pazzani, M.; Mehrotra, S. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.* **2001**, 3, 263–286.
46. Yi, B.K.; Faloutsos, C. Fast time sequence indexing for arbitrary Lp norms. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt, 10–14 September 2000.
47. Bat-Erdene, M.; Park, H.; Li, H.; Lee, H.; Choi, M.S. Entropy analysis to classify unknown packing algorithms for malware detection. *Int. J. Inf. Secur.* **2016**, doi:10.1007/s10207-016-0330-4.