Entropy Analysis of Packing Algorithms for Malware Detection

MUNKHBAYAR Bat-Erdene (MB)

“ MNSEC-2017”

Cyber Security Conference **2017-09-29.**

Contents

Introduction

Goals of research

- ***Classify*** known/unknown ***single-layer packing, re-packing and multi-layer packing algorithms*** of a given packed executable using similarity and supervised learning with symbolic representation

Malware

- The number of malware is increasing
- Malicious Software (Malware) authors are producing packed malware to avoid anti-malware system

Packers

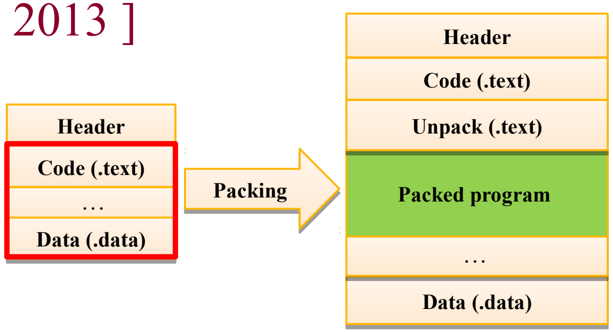
- A software program that compresses and encrypts other executable files

Packed malware

- A form of packed executable presents a significant challenge to analyze malware

Motivation

- Over 80% of malwares appear to be created using a packing algorithm to circumvent anti-malware systems [Osaghae et al. 2016, Jacob et al. 2012, and Bat-Erdene et al. 2013]

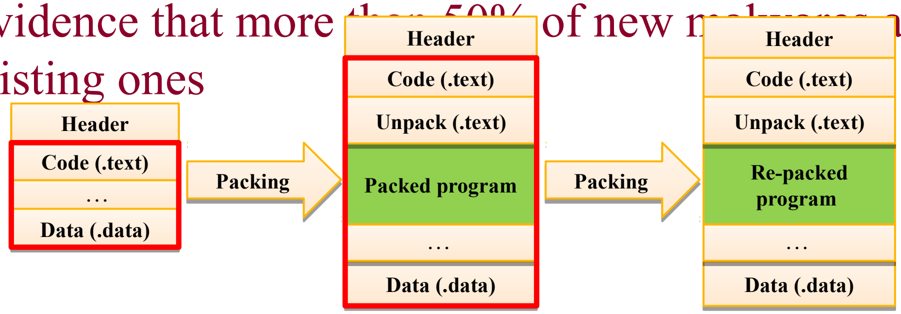


❖ Problem

➢ Cannot detect unknown/new packed malware

➢ Need to unpack packed malware

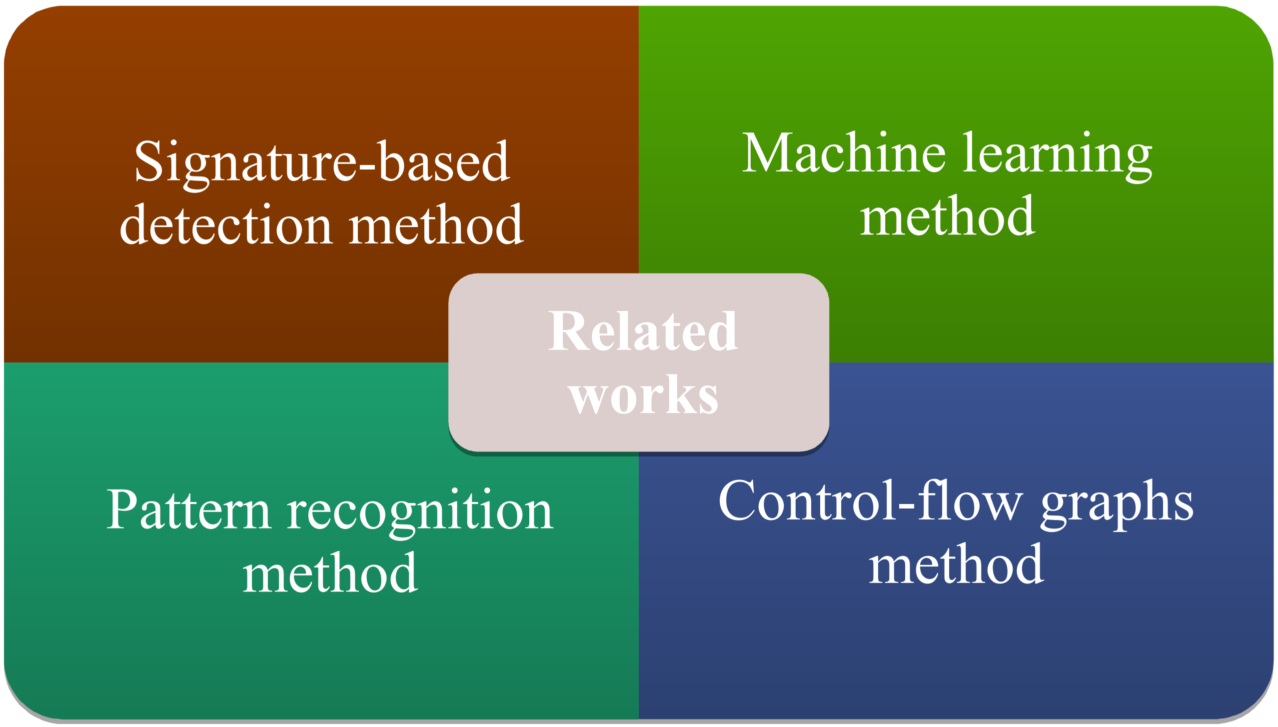
- There is the evidence that more than 50% of new malwares are simply re-packed versions of existing ones



Related Works

Related works 1/3

We conducted a study on previous related works in the following categories:



All method of which can be employed to detect single-layer packing algorithms and single-layer packed malware

Related works 2/3

1. Signature-based detection method:

- Uses pattern matching
- Searches for known patterns of data belonging to malwares in executable programs or other types of files
- Maintains and updates a blacklist of signatures

2. Machine learning method:

- A branch of artificial intelligence
- Machine learning is programming computers to optimize a performance criterion using example data or past experience
  - This method presented a vector of n-grams to represent malicious and benign files, and a comprehensive evaluation of classifiers

Related works 3/3

3. Pattern recognition method

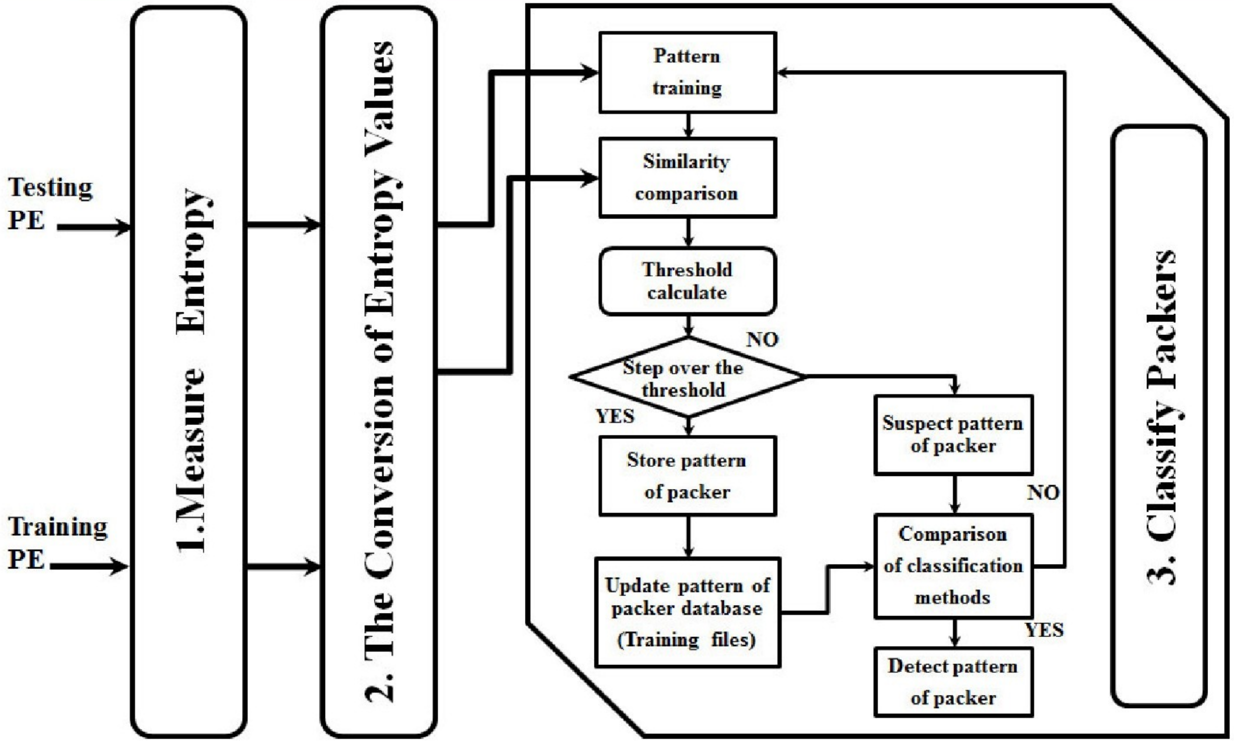
- Pattern recognition is a branch of machine learning
- Machine learning focuses on the recognition of patterns and regularities in data
  - Pattern recognition systems are in many cases trained from labeled "training" data (super vised learning)
  - But when no labeled data are available other algorithms can be used to discover previousl y unknown patterns (unsupervised learning)

4. Control-flow graphs method

- A control-flow graph (CFG) is a directed graph representation of a program and usually a sparse graph
- CFGs include all possible control paths in a program

Main Mechanism

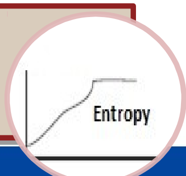
Proposed main mechanism



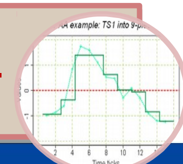
Our mechanism consists of 3 steps

- Measure entropy patterns
- Extract patterns of symbolic representation using entropy patterns
- Classify SAX patterns

Entropy analysis



Symbolic representation



Classification

1. Measure Entropy Pattern

- Measuring entropy pattern determines the entropy value of packed executable in unpacking process

1. We executed a given single-layer packed, re-packed, or multi-layer packed executable and let it conduct unpacking process

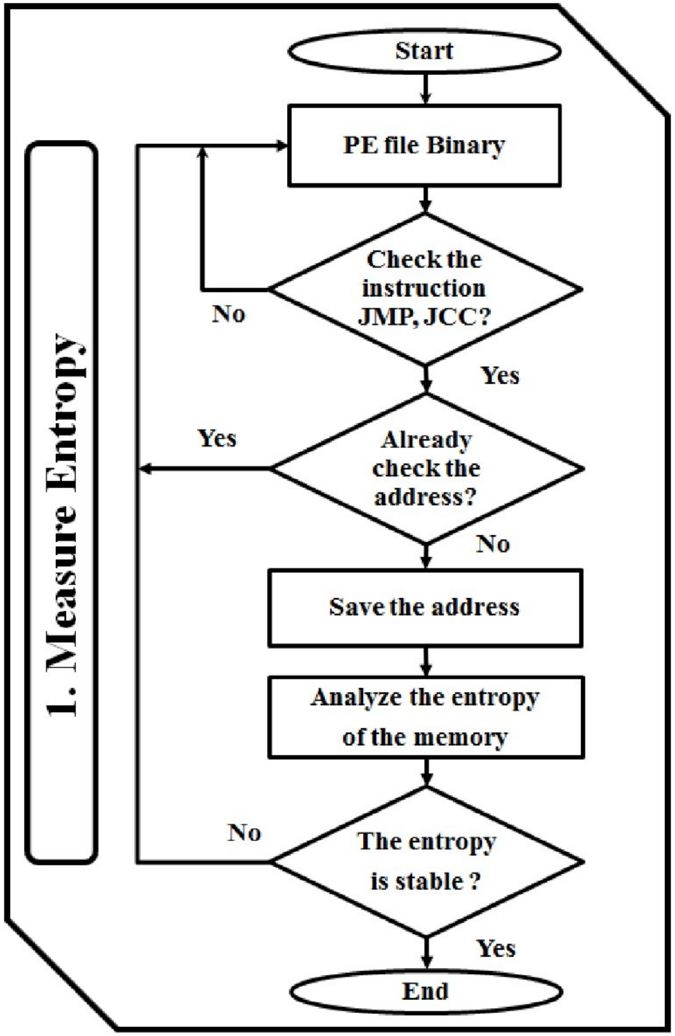
2. During an unpacking process, packed instructions are unpacked by a decompression module

3. We measured entropy to determine changes in memory space

4. We measured entropy score to find the OEP

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

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.



1.1. Entropy Analysis

- Entropy can be used to evaluate a compression algorithm
  - The packed executable is completely unpacked only if original entry point (OEP) is found
- 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.
- Entropy analysis is conducted by measuring a specific memory space
- We use entropy analysis to detect the existence of packing algorithm

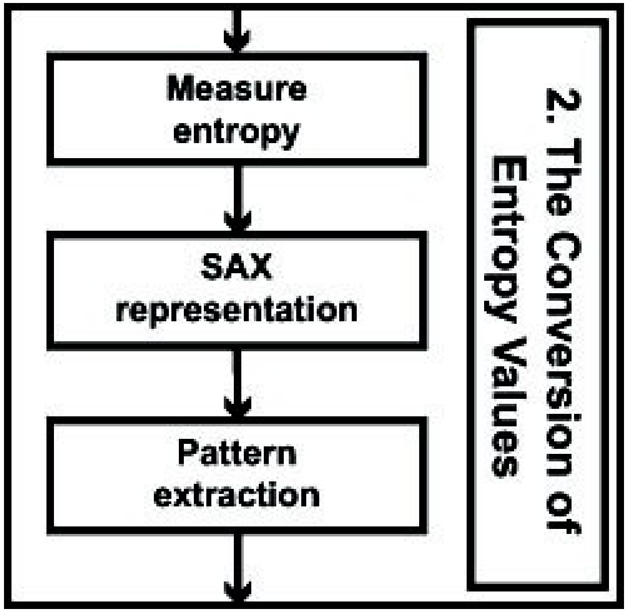
Algorithm 1 Finding the OEP during unpacking

Input: The input is a single-layer packed, re-packed or multi-layer packed executable. The output is an entropy sequence. The packed executable, an instruction pointer, and entropy of unpacked code are represented by P, IP and E, respectively. We assume that an executable is categorized as either packed or native.

Output: Locate OEP of P.

```
// Initialization runs the executable
Find an entry point and the all section of P.
Set a break point to the entry point.
Set R to the range of the all section.
// Start analysis
while the PROCESS is not terminated do
  IP ← a current instruction pointer
  // Measure entropy in only this condition
  if IP is for a JMP instruction then
    Measure entropy of R.
  else
    Continue this loop.
  end if
  // Check if unpacking is complete
  if E_min ≤ Measured entropy ≤ E_max then
    // Check if it jumps onto the unpacked code
    if Jump into R from outside of R is true then
      OEP ← The next instruction address
      Break this loop.
    else
      Continue this loop.
    end if
    Continue this loop.
  end if
end while
```

2. Convert Symbolic Representation



- Read entropy patterns
- Convert to symbolic representation
- Extract the symbolic entropy patterns of packing algorithm using SAX
- Compare with existing symbolic represen-tation and scan similarity of packing al-gorithms

2.1. Symbolic Representation

- A symbolic representation allows for a dimensionality reduction and index es using a lower-bounding distance measure of the true distance
- SAX is one of the most competitive methods in the literature
- Lin et al. defined the symbolic representation of time-series as the Symboli c Aggregate approXimation (SAX)

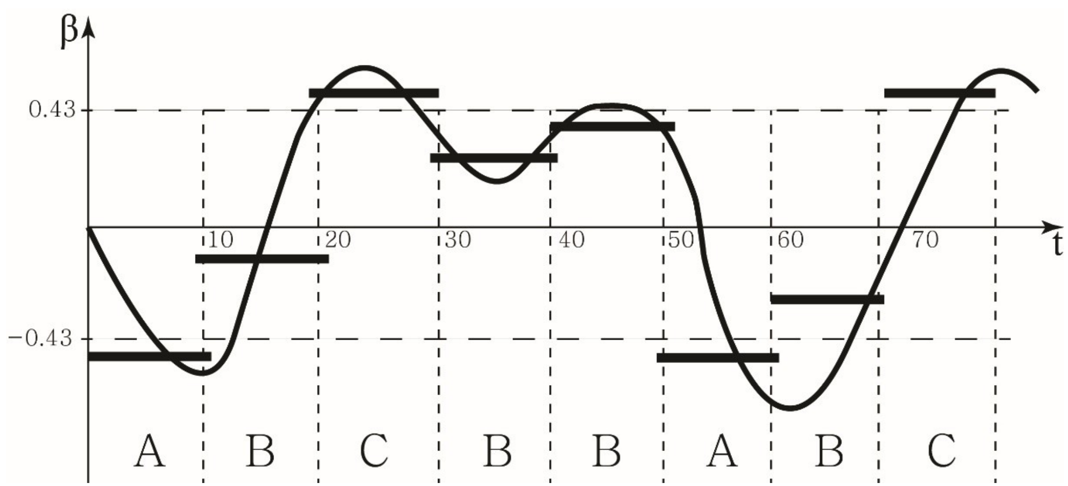


Figure 3.3: The entropy pattern is discretized by first obtaining a PAA approximation, and then by using predetermined breakpoints (β) to map the PAA coefficients into SAX symbols.

2.2. Symbolic Aggregate approXimation (SAX)

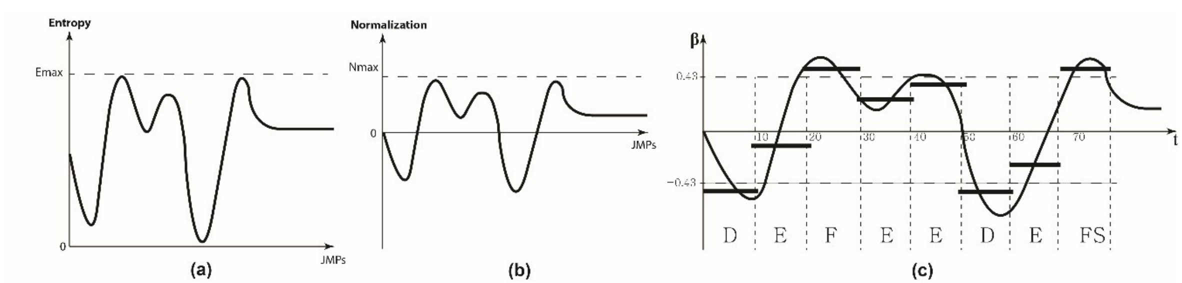
- SAX is the first symbolic representation for time-series data mining [Lin et al.]

SAX is applied as follows:

- Scale and normalize time-series;
- Reduce the dimensionality of the time-series using the Piecewise Aggregate Approximation (PAA) *(Lin et al. and Keogh et al.)*
- Discretize PAA representation of the time-series that is achieved by determining the number and location of breakpoints *(Yi et al. and Keogh et al.)*

SAX reduces numerical data to a short string (characters)

- Thousands of data points of numerical, continuous data becomes ' ABCEDEFGH'



2.3. SAX analysis

- The SAX method approximates time-series x of length *n* into vector $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_M)$ of any arbitrary length M ($M < n$, typically $M << n$), where each $\tilde{x}_i$ is calculated through the following formula:

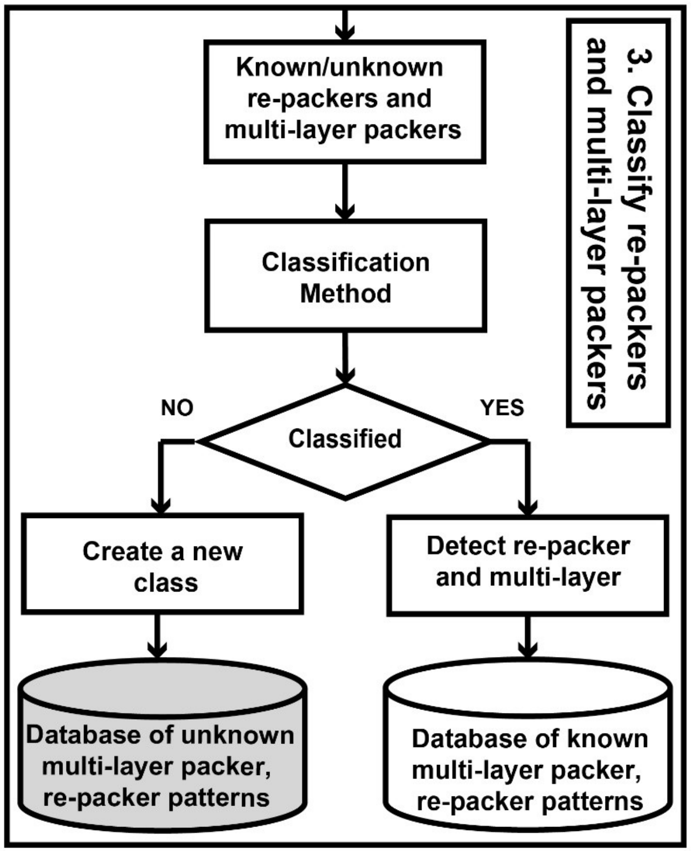
$$\tilde{x}_i = \frac{1}{r}\left[\sum_{j=r(i-1)+1}^{ri} (\tilde{x}_j)\right]$$

where r is a ratio defined as $r = \frac{n}{M}$.

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

Illustration of conversion into symbolic representation: SAX.

3. Classification



- Our method is a type of supervised classification method
- We detect known and unknown single-layer packing, re-packing, or multi-layer packing algorithm
- Compare existing symbolic representation patterns and scan similarity of single-layer packing, re-packing, or multi-layer packing algorithm
- Similarity: $F(x, y) = \frac{\sum_{i=1}^{n} \sqrt{x_i * y_i}}{\sqrt{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}}$

Note that the normalization of sequences is explicitly included and that F(x, y) = 1 if and only if x = y.
In general, $0 \leq F(x, y) \leq 1$.

3.1. Classification method

- Our proposed method includes two types of classification
  - The first one is a similarity measurement classification
  - A second one includes commonly used classification methods such as the Naive Bayes and Support Vector Machines

Classification categories

1. Supervised learning	2. Unsupervised learning

↓

To use

Flow graph based	Code graph based	Control Flow graph Generation	Markov chain graph	**Entropy analysis**

↓

Algorithms

Decision Trees, **Naive Bayes**, **SVM**, Bayesian Networks , Artificial Neural Networks, The Sequential Minimal Optimization (SMO) Algorithm, k−Nearest Neighbor (kNN) Algorithm, **Symbolic approach** etc

Single-layer Packing Algorith m Detection

Evaluation 1: Single-layer packing algorithm detection

- We proposed a method for detecting single-layer packing algorithms
- In these experiments, methods of similarity measurement, symbolic representation an d popular forms of classification were used on each single-layer packed executable

In this experiment the dataset con-tains

650 single-layer packed executables

- 326 of which were single-layer packed malwares
- 324 are single-layer packed benign executables

Packers

We used popular
19 packers
in the experiments

Experimental result

High accuracy of 95.35%

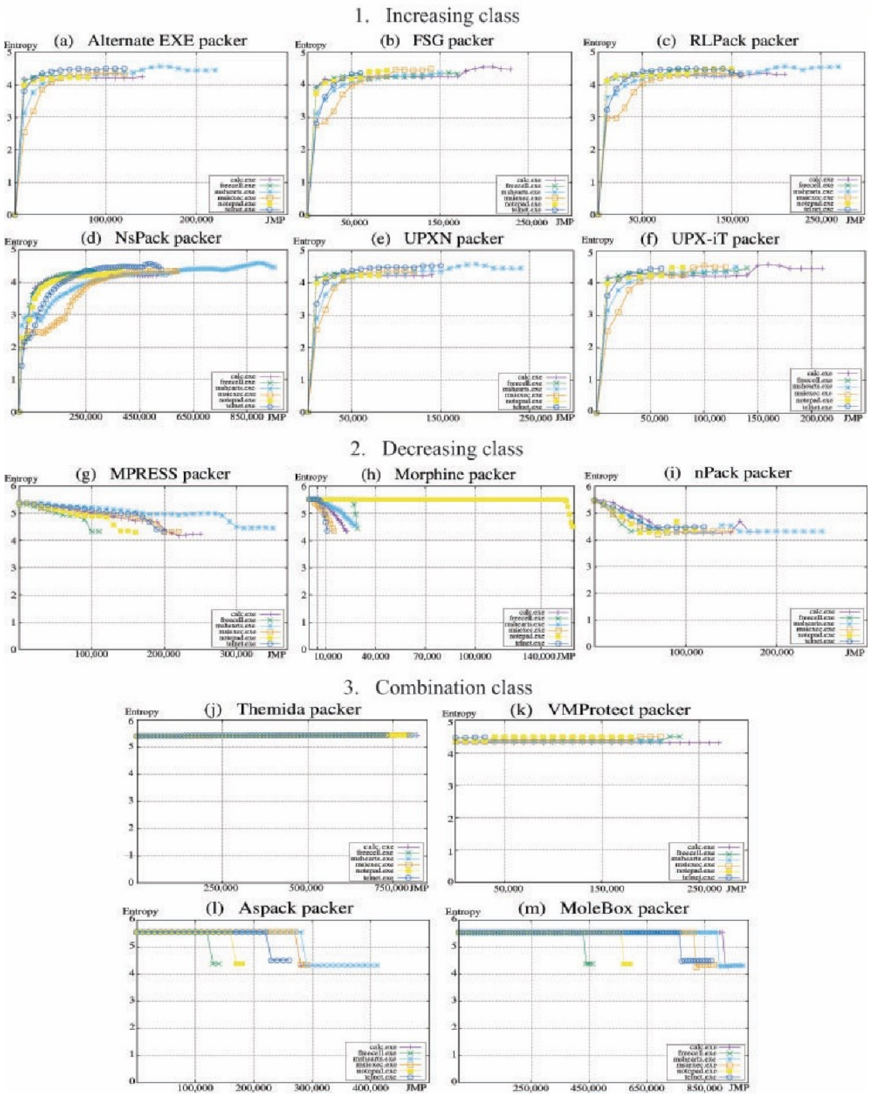
The SVM classification's ***accuracy is 95.5%*** which is higher than NB

We classified packing algorithms in four class based on their graphically visualized patterns

1. Increasing class 2. Decreasing class 3. Combination class 4. Constant class

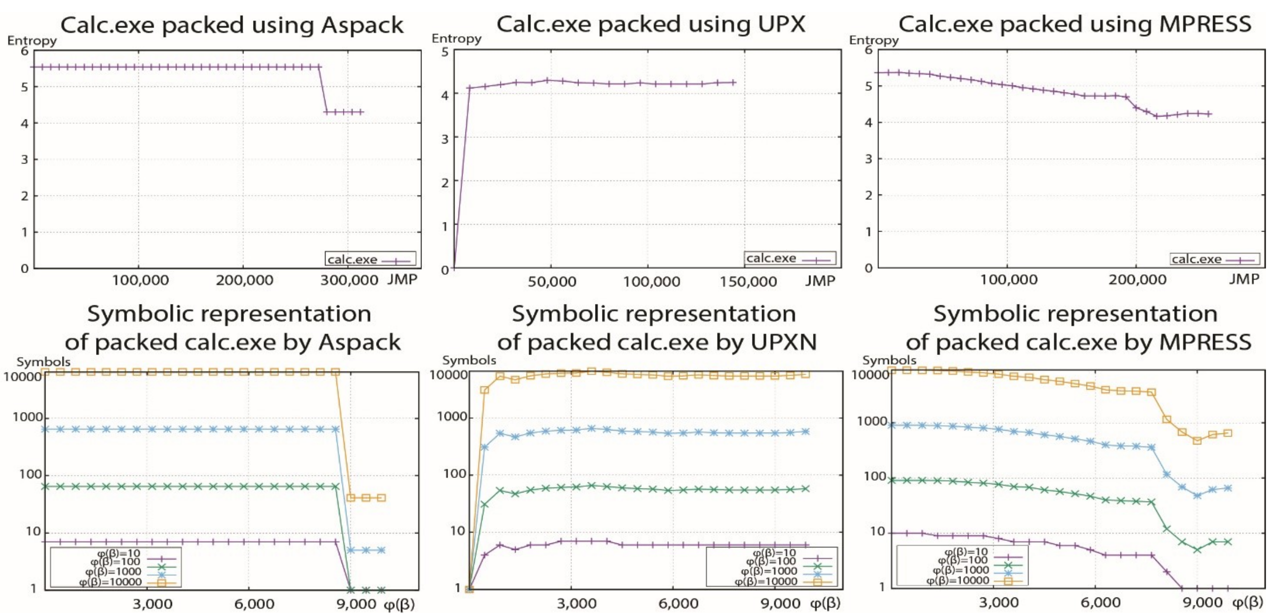
Evaluation 1: Single-layer packing algorithm detection

1. ***Increasing class***
   - Packing algorithms of the Increasing class initialize memory space, where unpacked code will be written, as zeros; it starts with zero entropy values
2. ***Decreasing class***
   - On the other side, packing algorithms of the Decreasing class does not initialize memory space before unpacking packed executables.
3. ***Combination class***
   - The combination class is divided in two classes, the increasing -to-constant and the decreasing-to-constant patterns.
4. ***Constant class***
   - Constant class encloses patterns of packing algorithms for benign packed executables. Entropy patterns of benign packed executables have constant values.



Results of experiments using SAX and entropy analysis (Single-layer packing algorithm)

- First, we present the benign ``calc. exe'' files single-layer packed using the 19 packing algorithms.
- Second, we assign four types of $\phi(\beta)$ values to the packed ``calc.exe'' executables converted using SAX.
  - In this example, $\phi(\beta)=10$, $\phi(\beta)=100$, $\phi(\beta)=1000$, and $\phi(\beta)=10000$ where $n=100000$, $\phi(\beta)$ is the number of symbols, and the entropy value is mapped to the character symbols, ``abcdefghijklmnopqrstuvwxyz''



Results of experiments using SAX and entropy analysis (Single-layer packing algorithm)

Num.	PACKERS	$\mathcal{T}_r\%$	$\mathcal{F}_r\%$	$\mathcal{A}\%$	$\mathcal{P}\%$	$\mathcal{R}\%$
1.	Alternate_EXE	100.0	0.0	100.0	100.0	100.0
2.	FSG	100.0	3.1	98.2	96.0	100.0
3.	RLPack	90.5	8.1	90.2	86.4	90.5
4.	NsPack	95.2	4.8	96.1	95.2	95.2
5.	UPXN	90.5	5.4	92.2	90.5	90.5
6.	UPX-iT	94.5	1.8	94.6	94.8	94.8
7.	MPRESS	100.0	0.0	100.0	100.0	100.0
8.	Morphine	100.0	0.0	100.0	100.0	100.0
9.	nPack	100.0	10.0	94.1	91.7	100.0
10.	Themida	96.3	5.9	96.0	92.3	96.3
11.	VMProtect	96.2	2.9	95.0	92.3	96.2
12.	Aspack	95.2	4.8	96.1	95.2	95.2
13.	MoleBox	100.0	0.0	100.0	100.0	100.0
14.	Petite	92.3	11.5	90.9	91.7	92.3
15.	ASProtect	91.9	9.4	91.9	92.6	92.7
16.	Mew	100.0	0.0	100.0	100.0	100.0
17.	Yoda's Crypter	92.3	4.7	93.5	88.9	92.3
18.	PELock	88.5	9.7	88.3	88.5	88.5
19.	tELock	96.2	8.8	93.3	89.3	96.2
	AVERAGE	**95.77**	**4.78**	**95.35**	**94.13**	**95.83**

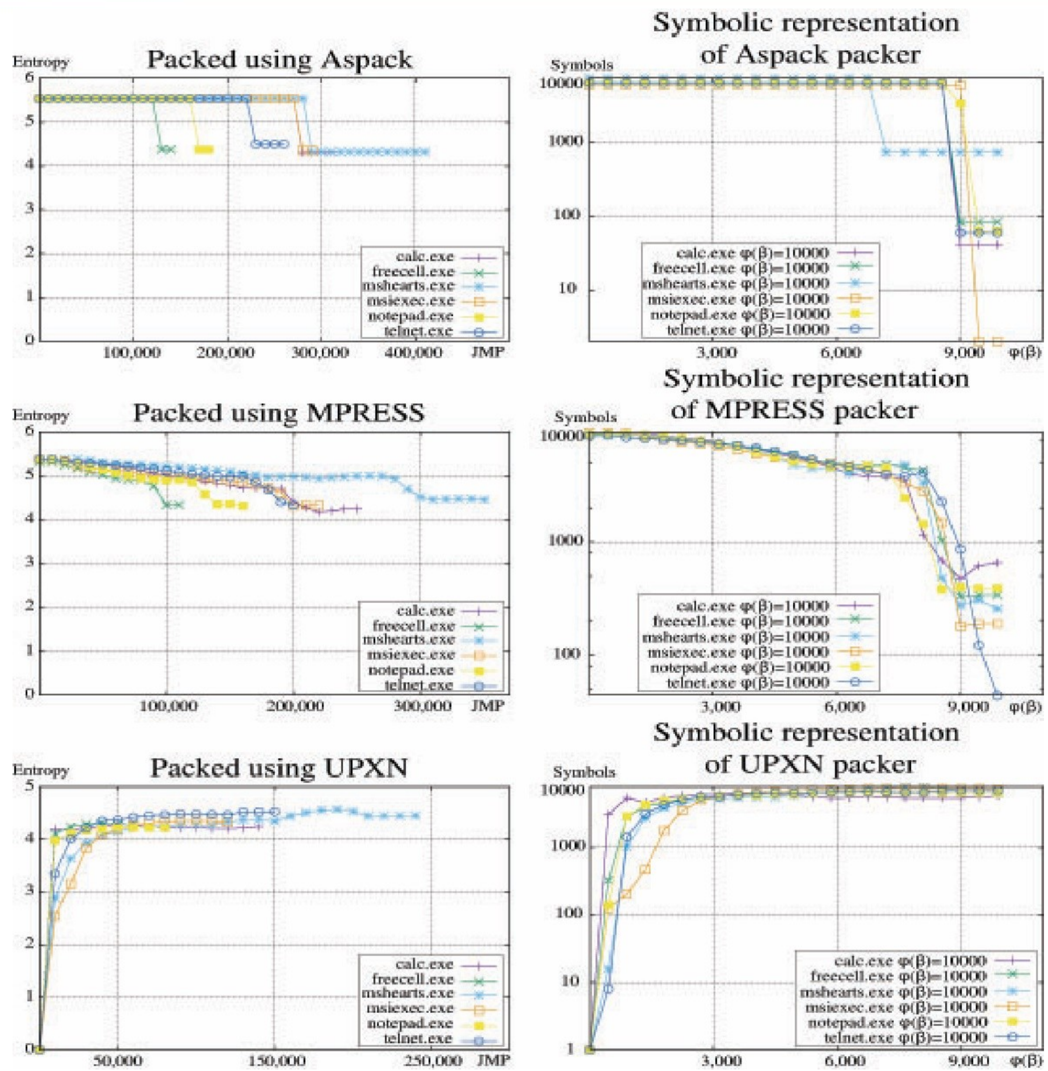
Detailed accuracy of each single-layer packer using the fidelity similarity classification dataset.

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**

Accuracy rates of supervised learning classifier.

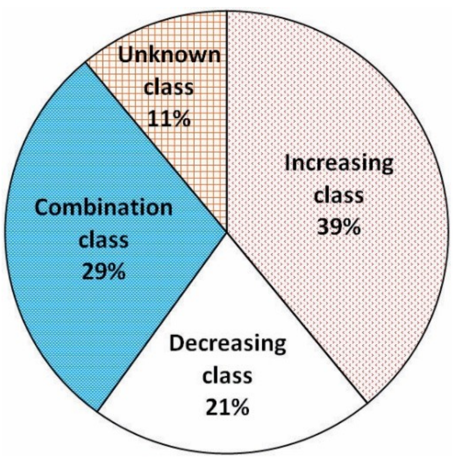
Results of experiments using SAX and entropy analysis (Single-layer packing algorithm)

Experimental results of entropy patterns of three popular packers converted into symbolic representations.

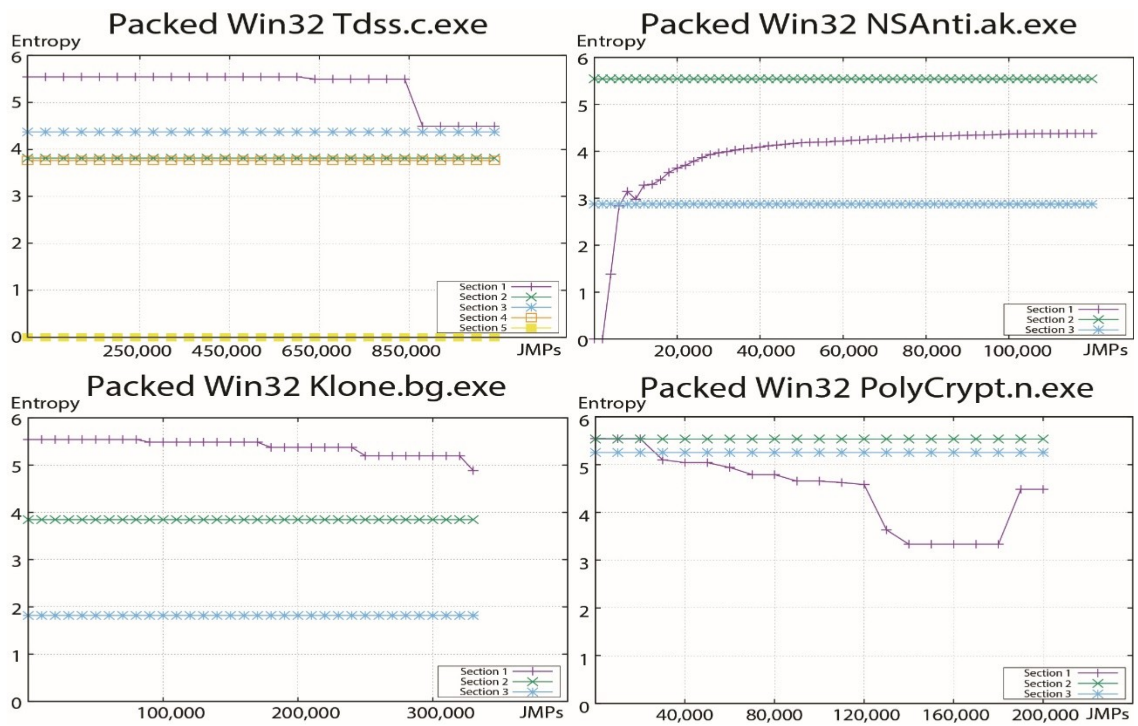


Results of single-layer packed malware detection

- We conducted the experiments using 326 single-layer packed malware executables classified into four classes
- We can classify 89% of the single-layer packed malware into classes of known pac king algorithms (classes A, B, and C), and the remaining 11% into the class of unkn own packing algorithms



Classification of single-layer packed malware



Results of single-layer packed malware detection

1. The single-layer packed malware pattern of NSanti.ak looks very similar with the packer patterns of NsPack (98.6%) among class A

2. The single-layer packed malware pattern of Klone.bg looks very similar to the packer pattern of MPRESS (99.98%) among class B

3. The single-layer packed malware pattern of Tdss.c has a similarity with the packer pattern of Molebox (99.98%) among class C

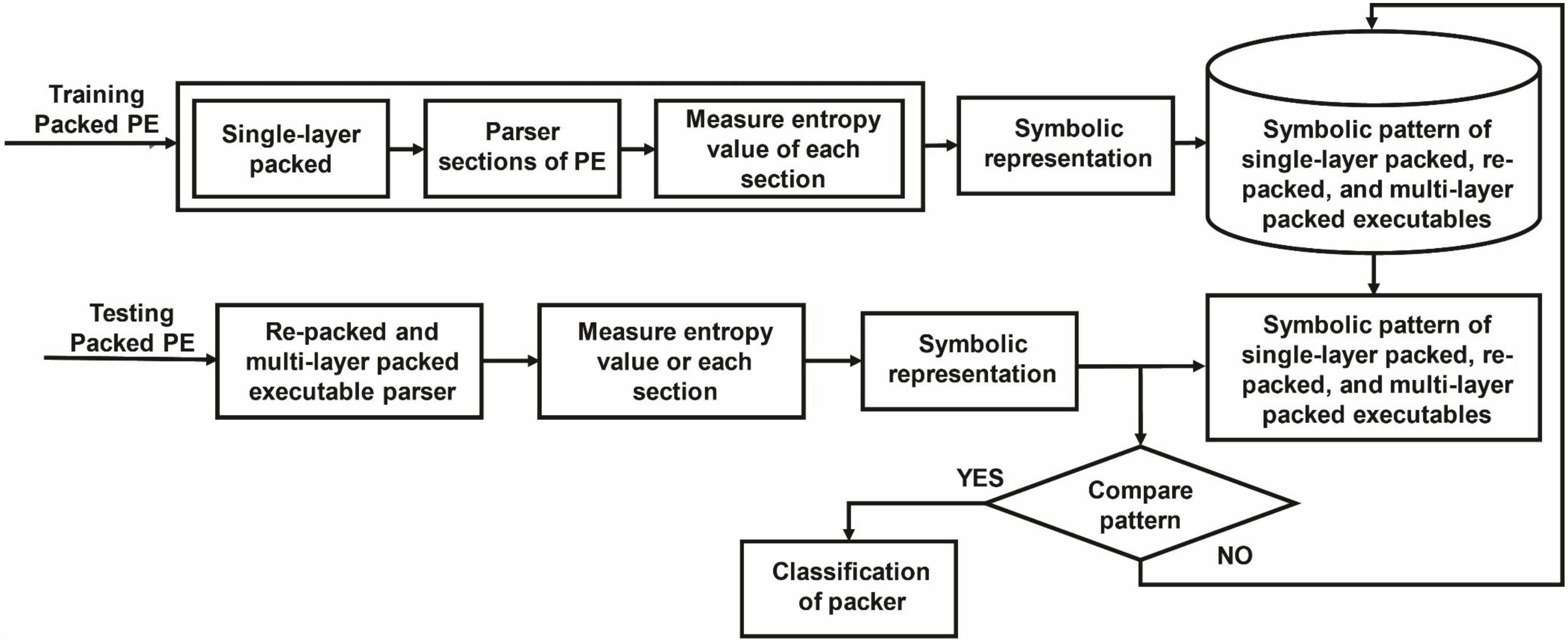
Class C: Patterns of packing algorithms	Pattern of Tdss.c
Aspack	84.95%
Molebox	**99.98%**
Class A: Patterns of packing algorithms	Pattern of NSanti.ak
Alternate_EXE	83.57%
FSG	86.54%
NsPack	**98.60%**
RLPack	83.93%
UPXN	81.36%
Class B: Patterns of packing algorithms	Pattern of Klone.bg
MPRESS	**99.98%**
nPack	80.93%
Morphine	75.78%

Detection of packing algorithms from packed malware

Re-packing or Multi-layer Packing Algorithm Detection

Re-packing and multi-layer packing algorithm detection

- The one more idea of this thesis is to measure the entropy values while unpacking re-packed or multi-layer packed executables



Re-packing or multi-layer packing algorithm detection method.

Classier for 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*
  - *Constant class*

- We shows the fidelity performance of experiments on the single-layer packed, re-pa cked, or multi-layer packed executables using
  - *Aspack*
  - *Alternate EXE*
  - *nPack*
  - *NsPack*
  - *RLPack*
  - *VMProtect packing algorithms*

Classier 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

Fidelity similarity for re-packing and multi-layer packing algorithms

Evaluation 2: Re-packing and multi-layer packing algorithms detection

- The dataset used in this experiment contains six benign executables for packing algorithms
  - 2196 re-packed and multi-layer packed benign executables
  - 19 popular packers

In this experiment the dataset contains	Packers	Experimental result
2196 re-packed and multi-layer benign packed executables	We used popular ***19 packers*** in the experiments	High accuracy of 95.35%

Evaluation 2: Re-packing and multi-layer packing algorithms detection

- Yoda's Cryptor packing algorithm can re-pack or multi- layer pack an executable, re-packed or multi-layer packed executables would not work

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

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

♦ is re-packed benign executables. For example: **Alternate_Exe + Alternate_Exe**; ★ is two way packed multi-layer packed executables. For example: **NsPack + Aspack** and **Aspack + NsPack**; ■ is one way packed multi-layer packed executables. For example: **Alternate_Exe + NsPack**; **Failed** is executable not packed with re-packing or multi-layer packing algorithm.

Experimental results of packed executables with the single-layer packers, re-packers, and multi-layer packers

Results of experiments using SAX and entropy analysis

- We packed each executable one time, two times, and combination times using 19 packing algorithms
- We extract entropy pattern of packed notepad.exe by 19 packing algorithms
- We scale entropy pattern of each packed notepad executable
- We calculate the number of symbols ϕ() for converting using SAX

			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	♦	Failed	Failed	Failed	Failed	Failed	Failed	Failed	★/not exe
2.		RLPack	■	♦	Failed	■	★/not exe	★	■	Failed	■
3.		NsPack	■	■	♦	Failed	★/not exe	★	★	■	Failed
4.		nPack	■	Failed	Failed	♦	★/not exe	Failed	Failed	Failed	★/not exe
5.		Themida	Failed	★	★	★/not exe	♦	■/not exe	Failed	Failed	★/not exe
6.		VMProtect	Failed	★	★	Failed	Failed	♦	Failed	Failed	Failed
7.		Aspack	■/not exe	Failed	★/not exe	■	Failed	■/not exe	♦	Failed	★/not exe
8.		MEW	Failed	■	Failed	■	■	■	Failed	♦	★/not exe
9.		Yoda's Crypter	★/not exe	Failed	Failed	★/not exe	★/not exe	■/not exe	★/not exe	★/not exe	♦/not exe

♦ is re-packed benign executables. For example: **Alternate_Exe + Alternate_Exe**; ★ is two way packed multi-layer packed executables. For example: **NsPack + Aspack** and **Aspack + NsPack**; ■ is one way packed multi-layer packed executables. For example: **Alternate_Exe + NsPack**; **Failed** is executable not packed with re-packing or multi-layer packing algorithm.

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

Results of experiments using SAX and entropy analysis

- We used features of single-layer packed, re-packed, or multi-layer packed executab les to create the operation of each re-packed or multi-layer packed executables, suc h as
  - the number of sections
  - the size of the section
  - name of the section

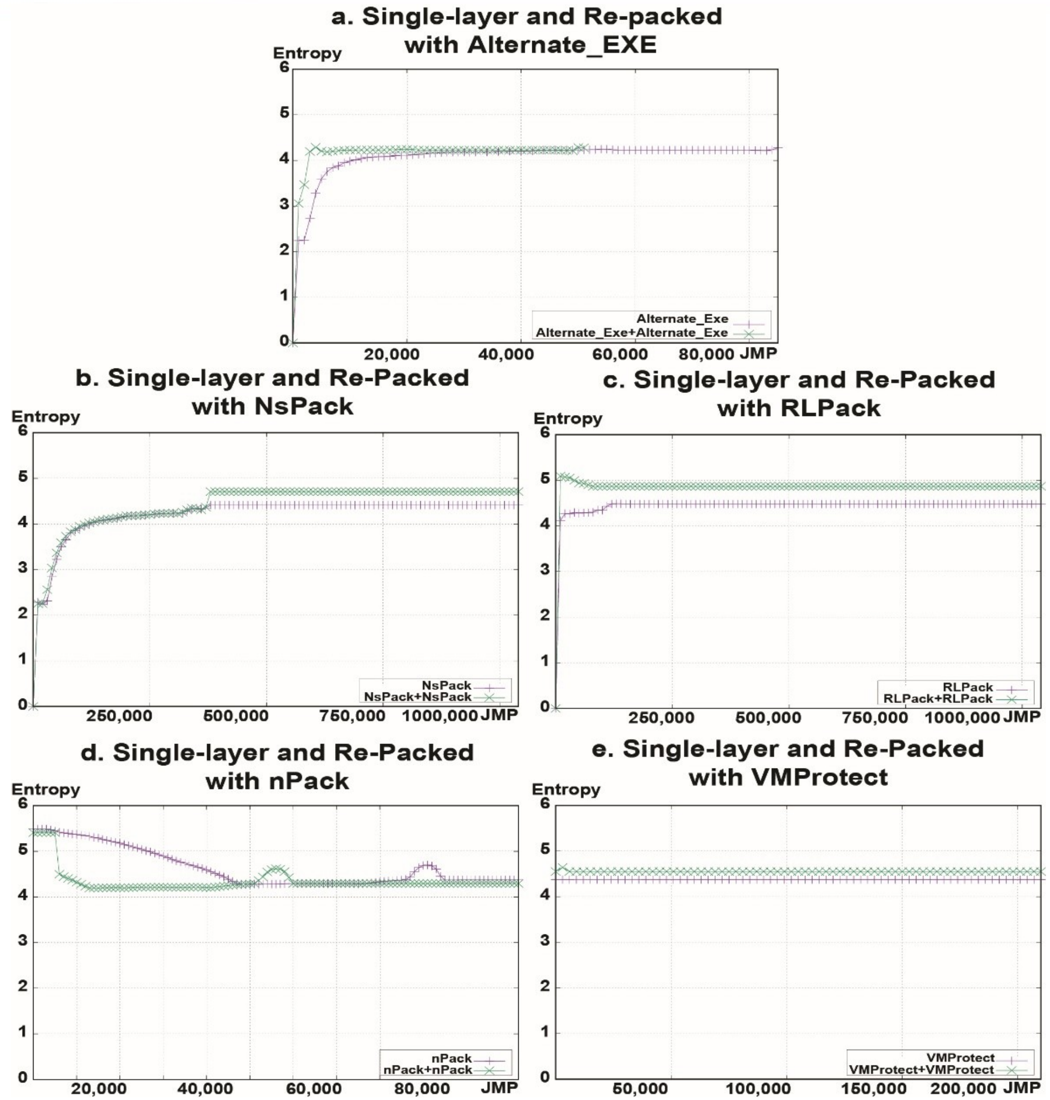
- Next, we found that the nine re-packed or multi-layer packed executable's entropy patterns of 8 packing algorithms
  - *New class* includes MEW, Yoda's Cryptor;
  - *Increasing class* includes Alternate EXE, NsPack, RLPack;
  - *Decreasing class* consists of nPack;
  - *Combination class* consists of VMProtect, Themida and Aspack;
  - *Consant class* includes TELock

Results of experiments using SAX and entropy analysis

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
x-axis is ``JMP'' instruction numbers

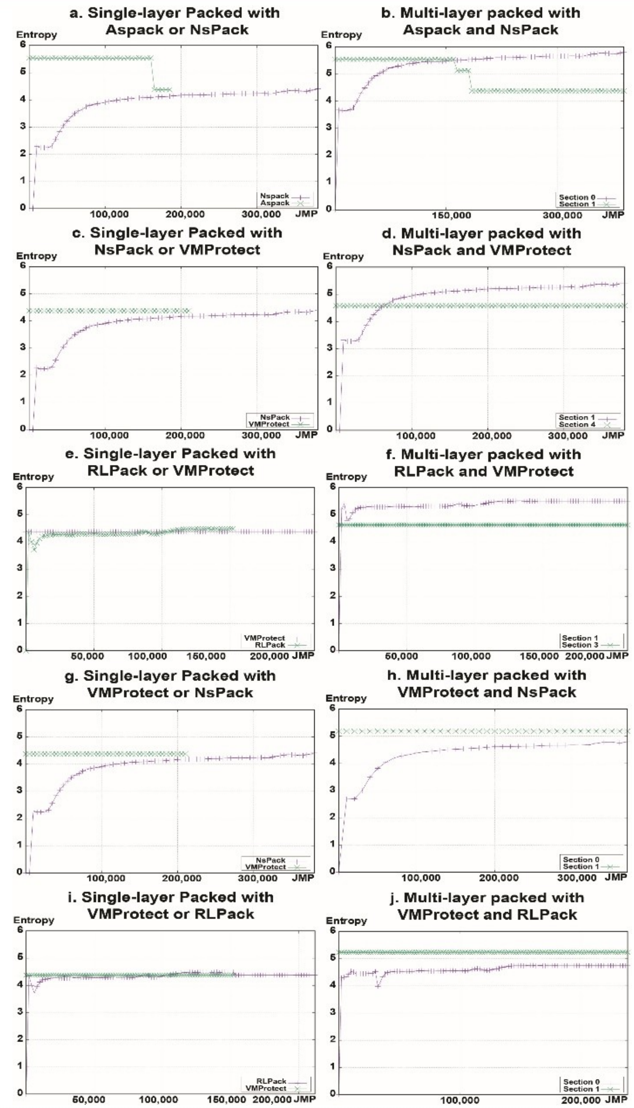


Results of experiments using SAX and entropy analysis

Entropy patterns of single-layer packed and multi-layer packed executable of Notepad.exe using two packers

(a) NsP or Asp;
(b) NsP and Asp;
(c) NsP or VMP;
(d) NsP and VMP;
(e) RLP or VMP;
(f) RLP and VMP;
(g) VMP or NsP;
(h) VMP and NsP;
(i) VMP or RLP;
(j) VMP and RLP

y-axis is entropy values
x-axis is ``JMP'' instruction numbers



Results of experiments using SAX and entropy analysis

- The average accuracy using re-packers and multi-layer packer are 98.5% and 97.5%, respectively
- 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

	N	Packing Algorithm	$\mathcal{T}_r$%	$\mathcal{F}_r$%	$\mathcal{A}$%	$\mathcal{P}$%	$\mathcal{R}$%
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	$\mathcal{T}_r$%	$\mathcal{F}_r$%	$\mathcal{A}$%	$\mathcal{P}$%	$\mathcal{R}$%
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**

Conclusion

- This is the first work to classify single-layer packed, re-packed and multi-layer pac ked executables using entropy pattern of packing algorithms

- We presented a novel technique for the detection of single-layer packing, re-packing or multi-layer packing algorithms using
  - SAX representations of the entropy values
  - The similarities in the sequence of SAX symbols in each packer

- We produced a highly accurate single-layer packer, re-packer and multi-layer packe r classification system on real life data

Future work: We will extract symbolic patterns from new packed malware, examine r e-packed or multi-layer packed malware packing algorithms

- To use additional supervised classification methods for re-packer and multi-layer pa cker classification and detection

Thank you for listening!

Contact: munkh0724@gmail.com
munkhbayar@korea.ac.kr