A Thesis for the Degree of Master

Complexity-based Packed Executable Classification with High Accuracy

Hanyoung Noh

School of Engineering

Information and Communications University

2009

Complexity-based Packed Executable Classification with High Accuracy

Complexity-based Packed Executable Classification with High Accuracy

Advisor : Professor Kwangjo Kim

by

Hanyoung Noh

School of Engineering
Information and Communications University

A thesis submitted to the faculty of Information and Communications University in partial fulfillment of the requirements for the degree of Master of Science in the School of Engineering

Daejeon, Korea
December 16, 2008
Approved by

Professor Kwangjo Kim
Major Advisor

Complexity-based Packed Executable Classification with High Accuracy

Hanyoung Noh

We certify that this work has passed the scholastic standards required by the Information and Communications University as a thesis for the degree of Master

December 16, 2008

Approved:

Chairman of the Committee
Kwangjo Kim, Professor
School of Engineering

Committee Member
Soontae Kim, Assistant Professor
School of Engineering

Committee Member
Doo Ho Choi, Ph.D
ETRI

M.S. Hanyoung Noh
20072030

Complexity-based Packed Executable Classification with High Accuracy

School of Engineering, 2009, 35p.
Major Advisor : Professor Kwangjo Kim.
Text in English

Abstract

Signature-based anti-virus scanner that utilizes specific bytes string is one of the popular malware detection methods. This method shows high efficiency and low false rate in detecting malware. However, it can't successfully detect the malware when packer method is applied to the malware. Packer compresses or encrypts the target file and pads compressed or encrypted file to additional section. Therefore, it changes the bit structure of the malware. When packed executable file is executed, compressed or encrypted file is decompressed or decrypted on memory. Then, the program that is loaded on the memory is executed. This means that function of the packed executable program is same as an original program.

Signature-based anti-virus scanner employs two methods to overcome the difficulty of detecting packed malware: packed executable classification and generic unpacking. Packed executable classification decides whether the target program is packed or not. Then, generic unpacking is performed depends on the decision of a packed executable classification. Various generic unpacking methods have been researched [13, 14, 15, 22], but these generic unpacking methods consume long time

and computing resources. Thus, accuracy of a packed executable classification is the critical issue in efficient malware detection. Because a false positive case happens, unnecessary time and resource is wasted performing generic unpacking on a non-packed executable file. On the other hand, a false negative case happens, malware detection would fail.

Previous packed executable classification methods show high false rate and can be easily evaded. PEiD[11] can be easily evaded by false signature. Bintropy[20] shows about 10% false rate. This method also can be evaded by monotonous padding bytes. Finally, Roberto *et al.* show high classification accuracy rate about 97%, but this method can be evaded by PE headers modification.

In this thesis, we propose highly accurate classification method which is resistant against evasion. Moreover, our method is efficient enough to be applied to practical anti-virus scanner. Our proposed classification method is based on complexity since packed executable file is usually compressed or encrypted.

We use LZO compression algorithm to measure the complexity which is the fastest compression algorithm as far as we know. Moreover, we propose a block dividing algorithm for exact complexity measurement which is previous step of complexity measurement. This block dividing step removes unnecessary part of complexity measurement. In our experiment, our proposed method shows accuracy about 97%. This rate is about 10% more accurate than Bintropy. In addition, our proposed method cannot be evaded by known evasion techniques, and it is efficient to be applied to practical anti-virus scanner.

Contents

List of Figures

List of Tables

List of Abbreviations

DLL Dynamic Link Library

EAT Export Address Table

IAT Import Address Table

LZW Lempel-Ziv-Welch

LZO Lempel-Ziv-Oberhumer

PE Portable Executable

List of Notations

B Block String without unnecessary string

C Complexity value

CC Collision count

$Compress(X)$ the length of compressed X

CT Collision threshold

$H(X)$ the entropy of random variable X

$K(X)$ the Kolmogorov complexity of random variable X

$\min(X)$ the minimum length of input string using specific algorithm

$p(x)$ distribution of x

t_h Threshold

W Window

W' Temporary window

x a random variable in X

X a finite string

I. Introduction

1.1 Overview

An enormous amount of economic loss was caused by spread of malware, which exceeds $13 billion dollar in 2007 [5], and this is still seriously increasing.

To defeat the malware, anti-virus scanner, firewall, intrusion detection system, honeypot and *etc.* are proposed. Among the proposed methods, anti-virus scanner is commonly used since it has low false rate and high efficiency. Two types of host-based anti-virus scanner exist: behavior-based and signature-based [6]. The behavior-based anti-virus scanner is usually used to detect unknown malware such as zero-day worm, but it has high false rate. On the other hand, signature-based anti-virus scanner is more popularly used to detect malware since it has false rate. In this thesis, we focus on signature-based anti-virus scanner.

The signature-based anti-virus scanner detects the malware by signature which exists in malware as a specific bytes string, so it has low false-negative rate. If no signature is matched with the target, anti-virus scanner will classify an input file as non-malware. However, malware maker uses various evasion techniques such as control-flow obfuscation, source obfuscation, instruction virtualization, and packer which combine all evading techniques. In fact, the packer is originally proposed to reduce file size, but malware maker misuse packer to hide its malicious intention. As we mentioned before, signature is a specific bytes string. When packer technique is applied to specific file, that file will have different file structure in comparison to the original file. It means that malware makers can generate variant of malware using packers.

Thus, anti-virus scanner cannot detect variant of malware by original signature. Recently, almost 92% malwares are found to be protected by packers [24]. To overcome those difficulties, generic anti-virus scanner will follow Figure 1.1 operation.

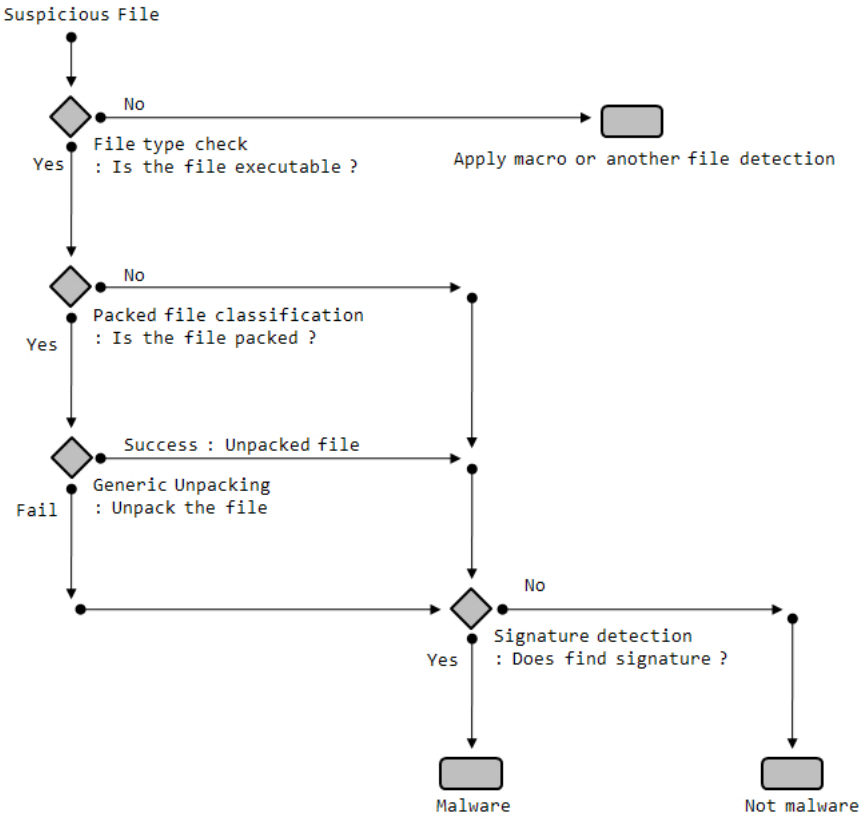

Figure 1.1: Operation of Signature-based Anti-virus Scanner

Before the concept of packer was proposed, we did not require specific algorithm such as packed file classification or generic unpacking procedures. When suspicious file is input, anti-virus scanner checks the file type, and signature detects whether the suspicious file is malware or not. However, packer changed entire structure of anti-virus scanner.

Let us assume that suspicious file is input into anti-virus scanner. Then, a file type should be checked, and packed file classification pro-

cedure is necessary. If suspicious file is packed, that goes into generic unpacking procedure. The generic unpacking unpacks the packed executable file. Finally, suspicious file is detected by signature. When signature is found, it means that suspicious file is malware.

Therefore, anti-virus scanner should have specific mechanism that could judge whether the packer protects the malware or not. There have been many research works proposed for unpacking methods [13, 14, 15, 22]. Nevertheless, these proposed unpacking methods have been low unpacking success rate, and requires longer time to unpack.

To detect malware with high accuracy and efficiency, we need more exact packed executable classification method which is pre-step of generic unpacking method. If packed file classification conducts false classification, it wastes unnecessary time and computing resources, and user's terminal could be infected by malware.

However, packed file classification is not popularly researched [20, 21], and those works that are already proposed to have various weaknesses. [20] has high false classification rate, [21] can be evaded easily even if it performs with high classification rate.

1.2 Our Goals

Our main goals are to achieve high accuracy on packed file classification with appropriate performance for practical anti-virus scanner. In addition, our proposed method is not evaded by avoidance techniques. Ultimately, our goal is that reduce the malware infection.

To achieve these goals, we approach the packed executable file classification based on complexity, not signature, entropy, or characteristics. Since packed executable file that is compressed or encrypted usually has high complexity, we can easily judge that executable file is packed or not. A complexity concept can measure the information quantity more

correctly than entropy concept.

We implement our proposed method using Lempel-Ziv-Oberhumer (LZO) compression algorithm [18] which is the fastest compression algorithm as far as we know. In our experiment, our proposed method achieves our goals: high accuracy about 97% with enough performance in our experiment, and cannot be evaded by known evade techniques.

1.3 Organization

The remaining parts of this thesis are organized as follows: Chapter II give portable executable file structure and definition of packer in briefly to explain previous related works. In Chapter III, we describe previously proposed unpacking why we need high accuracy of packed file classification method. In addition, we explain limitations and detail of previous proposed packed executable classification methods. In Chapter IV, we present our proposed method based on complexity. We describe the implementation and experiment result of proposed method. Finally, we summarize and conclude this thesis in Chapter VI.

II. Preliminaries

2.1 Portable Executable File Format

In this thesis, we introduce this portable executable(PE) file format for two reasons: understanding operation of packer and explaining Roberto *et al.* [21] what they use the values of PE headers.

The PE file format is designed for Win32-based systems, it supposed to be across all 32-bit operating systems of Microsoft. Users can use same PE format executable on any version of Windows: Windows NT, Windows 95, Windows XP, and Win32s. Moreover, it can support 64-bit operating systems such as Windows Vista by setting a value of PE header. Executable files(.exe), object code(.obj), and Dynamic Link Library(DLL)(.dll) are kind of PE file formats.

PE file format is constructed with IMAGE_DOS_HEADER, MS-DOS Stub Program, IMAGE_NT_HEADER, IMAGE_SECTION_HEADERS, and IMAGE_SECTIONS. Figure 2.1 illustrates the PE file format.

When we execute the PE executable file, PE loader verifies the PE executable file and maps the file on the virtual memory. And then, loads the functions of import table using DLL files.

Every PE file begins with IMAGE_DOS_HEADER. This header needs for execution on old version Windows O/S. The only importance value is e_lfanew value which is a file offset of the PE header. This e_lfanew value points out the PE header.

The IMAGE_NT_HEADER is most important header of PE file format; it has core information for PE loader. This header begins with PE signature means Portable Executable. This header consists with two

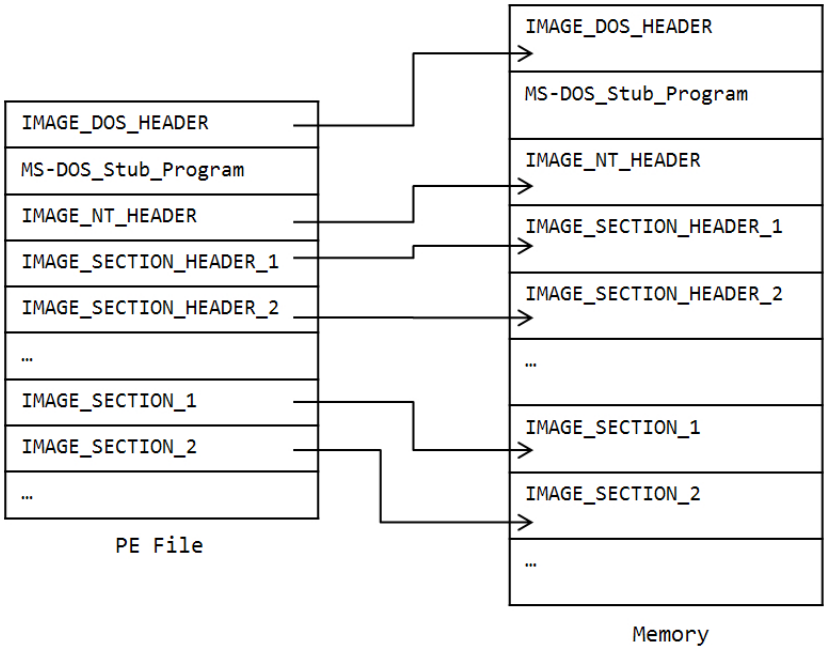

Figure 2.1: File and memory of PE Structure

headers: IMAGE_FILE_HEADER and IMAGE_OPTIONAL_HEADER. IMAGE_FILE_HEADER has the number of sections value which is used in Roberto *et al.* IMAGE_OPTIONAL_HEADER has the DataDirectory structure which has Import Address Table(IAT) and Export Address Table(EAT). This IAT is a list of used external functions of DLL files. When the PE file is loaded, PE loader allocates the external functions using IAT.

Next is IMAGE_SECTION_HEADRS. This header consists with section name, characteristics of section such as permission of section: Readable/Writable/Executable, and IMAGE_SECTION follows IMAGE_SECTION_HEADERS. Usually, Microsoft compiler makes sections like .text, .data, and .rsrc.

This PE format specification is defined in winnt.h file, and [16, 17] describe this PE format more detailed.

2.2 Packer

A packer is proposed to reduce file size at first. A file applied packer that we called 'packed executable file'. This packed executable file operates functionally same as original file. Figure 2.2 illustrates packing operation of packer.

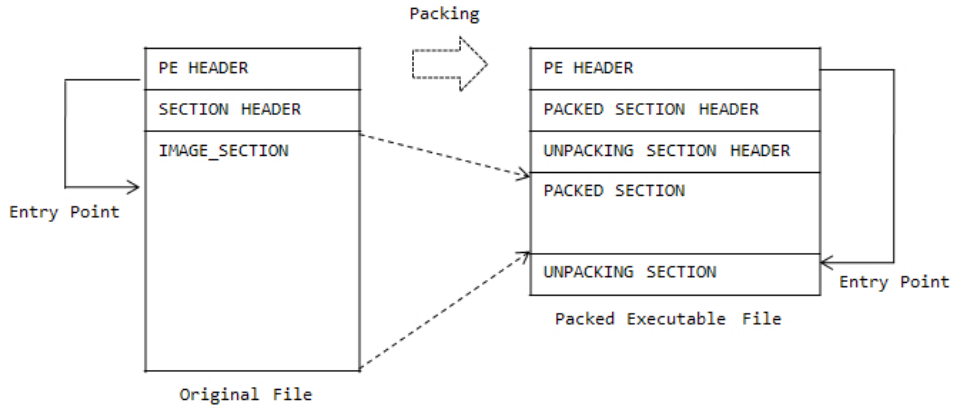

Figure 2.2: Packer structure

In packing procedure, a packer compresses or encrypts the IMAGE_SECTION of input file that is the packed data, and then insert additional UNPACKING_SECTION_HEADER and UNPACK_SECTION which can decompress or decrypt the packed data. Lastly, Packer modifies the entry point to start instruction of UNPACKING_SECTION. Those are all of packing process. Therefore, packed executable file has smaller size than original file size and same functional operation as original file.

Now, we explain the execution of packed executable file. Follow Figure 2.3 illustrates the procedures of execution of packed executable

file.

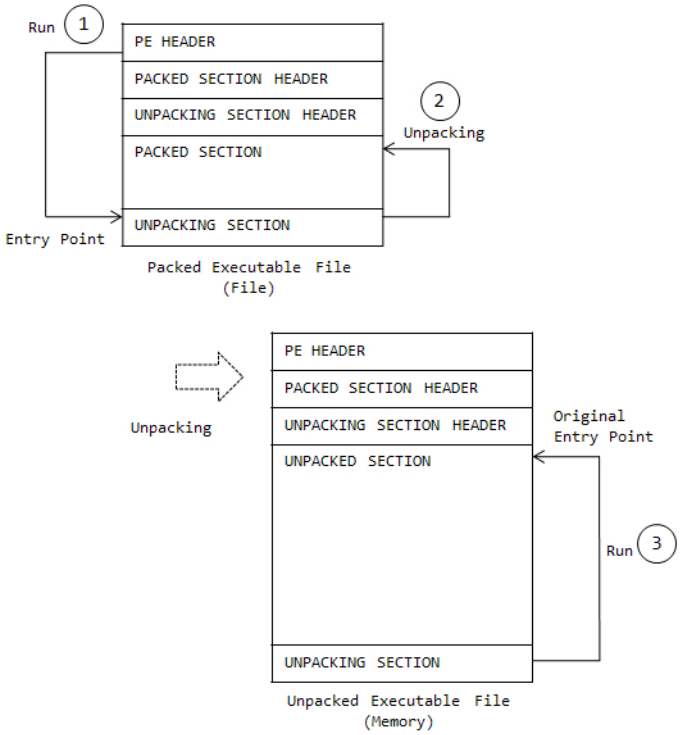

Figure 2.3: Execution operation of packed executable file

When a packed executable file is executed, PE loader loads the packed file to virtual memory, and then the instruction of UNPACKING_SECTION that is indicated by entry point is executed. Next, UNPACKING_SECTION decompresses PACKED_SECTION which is original section(s). Lastly, UNPACKED_SECTION is executed on virtual memory. That is why operation of packed executable file is functionally same.

However, a packed executable file has a different bytes structure with original file. Namely, packed executable file has a different signature with original file. Therefore, anti-virus scanner does not consider packer that cannot detect the packed executable file by a signature of original

file.

There exists various packers such as UPX, FSG, ASPack, Morphine, Exestealph, Pecompact, Yodacrypt, MEW, Packman, Upack, RLPack, Icrypt, EXE Smasher, Themida, and *etc.* Also, these packers have lots of versions, and manual packers which malware makers made exists. Malware maker is able to generate variant of malware using lots of packers to evade anti-virus scanner.

For instance, there exist one malware and three packers. Malware maker generates three variant malwares using three packers. If malware maker applies packers to three variant malwares repeatedly, lots of variant malwares can be generated. In this way, malware maker makes variant malwares using various packers. As a matter of fact, 92% of malwares are packed executable [24] in 2006. Of course, there exists that usage of packer for protection of commercial programs from malicious reverse engineering, but this *normal* usage is less than 2% (in fact, there is no study about *normal* usage of packer). Thus, anti-malware methods such as 'exepacker blacklisting [9]' are proposed, that is packed executable files are considered as malware.

III. Related Work

In this chapter, we review previously proposed methods about unpacking and packed executable classification. Firstly, we describe why we need high accuracy of packed executable classification method throughout the review the limitations of previous unpacking methods. And then, we review the shortcomings of proposed packed executable classification methods.

3.1 Unpacking Methods

Unpacking is the recovering the original file from a packed executable file. Simply, we can unpack a program by executing and dumping the program on virtual memory. However, it is infeasible in anti-virus scanner since the packed executable could be malware. It means that executes the malware on user's computer. Therefore, this executing and dumping unpacking method cannot apply to anti-virus scanner. Excepting dumping unpacking method, we describe the safe unpacking methods which can be applied anti-virus scanner.

3.1.1 Signature-based Unpacking method

Signature-based unpacking method is using optimized unpacking algorithm which is determined by the signature that specific length of bytes string of packed executable file. It is fast and exact unpacking method, because using optimized unpacking algorithm. However, it must need signature update and analysis of optimized unpacking algorithm. It cannot respond to the new proposed packer and modified packer.

There exist a large number of packer programs such as commercial, open-source packer, and manual packer. Therefore, it is a difficult problem that deals with various packers using this signature-based unpacking method. Moreover, to make optimized unpacking method, we have to analyze the packed executable file. It needs more than hour times and patience of human.

In this respect, we cannot apply signature-based unpacking method to anti-virus scanner mainly.

3.1.2 Algorithm-based Unpacking method

Algorithm-based unpacking method is proposed by Miroslav *et al.* [15], it overcomes the long time of analysis of packed executable file. Authors suggest the decompression framework that is used characteristic of packer which are used famous or similar compression algorithm such as LZ77, Lempel-Ziv-Welch(LZW), *etc.* This unpacking method can unpack by applying to decompress algorithm to packed executable file's packed section. For example, when packed executable files are packed by LZW compression algorithm, authors can unpack all the packed executable files using one LZW decompression algorithm. In this way, Miroslav *et al.* can reduce time of development of unpacking algorithm using compression algorithm.

Nevertheless, Miroslav *et al.* have limitations that need analysis of development unpacking of new compression algorithm. Moreover, a little modification of packing algorithm needs an implementation of new decompression algorithm.

3.1.3 Generic Unpacking method

Previous signature-based and algorithm-based unpacking method need human resource for analysis of unpacking algorithm and decompress

algorithm. However, [13, 14, 22] are generic unpacking methods which does not need any human resource for new packer.

PolyUnpack [22] can unpack the suspicious file using static and dynamic analysis. In their experiment, PolyUnpack takes average 1,020 seconds for unpacking. In fact, this time includes VMware's startup and shutdown time.

Renovo [14] achieved higher success rate and efficiency than PolyUnpack using instruction points moving where to newly-written memory region which is a unique feature of packed executable file. Renovo has about 9 times faster about 40 seconds than PolyUnpack about 365 seconds in their experiment.

These PolyUnpack and Renovo can unpack the packed executable files without human analysis, but it has huge to apply anti-virus scanner.

On the other hands, OminUnpack [13] is something different with PolyUnpack and Renovo. This OmniUnpack locates on the kernel of operating system, and it detects the signature of malware when a program is executed. In this way, this OmniUnpack can detect malware since the program is unpacked when execution time. OmniUnpack has higher efficiency than PolyUnpack and Renovo, but it still needs more improvement of efficiency. This unpacking method is about 5 times lower than anti-virus scanner. Moreover, it operates on the kernel that is another overhead.

3.2 Packed Executable Classification

Packed Executable Classification is previous step of unpacking procedure. We review proposed unpacking methods in previous chapter; unpacking methods need not only long time for unpacking, but also unpacking success rate is not good. Therefore, this packed executable classification step is important.

When non-packed executable file is classified to packed executable file, then non-executable file goes to unpacking step. And then, generic unpacker needs unnecessary time and user's resource, because non-executable file cannot be unpacked. On the contrary, packed-executable file is classified to non-packed executable file, then packed executable file is detected by anti-virus scanner. However, anti-virus scanner cannot find the signature of malware from packed executable file since it is packed. Therefore, high-accuracy packed executable classification can reduce unnecessary time and user's computing resources and archive high malware detection rate.

There are three types of proposed packed executable classification methods: PEiD, Bintropy, Roberto *et al.*

3.2.1 PEiD [11]

Packer has a unique signature that their own. This signature is almost same as malware signature that anti-virus scanner used. The most famous signature-based classification method is PEiD [11] program; PEiD supports plug-in and external signature database. It is fastest classification method since using only specific length of byte string, and it has low false positive rate.

However, signature-based classification method has limitations: it needs periodic signature updates since the number of new proposed packers is around 15 per month and modified packers(manual packer) are easily programmed using open-source packer. In addition, malware maker can modify the signature of packer easily. It is possible that insert fake signature to pretend different signature of packer, not original signature. Figure 3.1 shows a change of signature of packer by RLPack.

RLPack is a kind of packer that can insert a 'fake signature'. Applied RLPack executable file can be classified as another packer applied

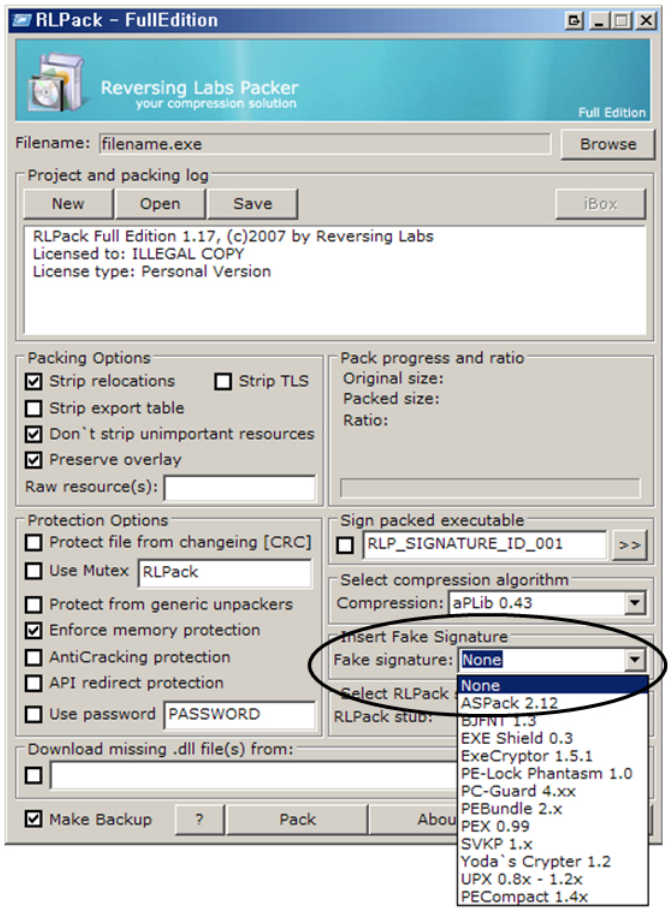

Figure 3.1: Insert the fake signature by RLPack [19]

executable file. However, it means that it is possible insert unknown signature to packed executable file, and then signature-based classification method cannot find a matched packer signature. Summing up, this signature-based classification has high efficiency and low false positive rate, but it needs signature updates and can be evaded by fake signature.

3.2.2 Bintropy [20]

Bintropy is based on entropy packed executable classification method. This entropy-based method can overcome the limitations of signature-based classification method which needs of periodic signature database update and fake signature evasion, because entropy-based method uses entropy measure and classifies file as packed or not. This entropy concept use that packed executable files are usually compressed or encrypted. In the other words, ordinary compressed or encrypted file has been high-entropy, so if high entropy file can be classified to packed executable file.

The entropy concept is proposed by Shannon [3] in 1948. Definition of entropy is that a measure of the uncertainty of input string. Let X be a finite or countable set, let x a random variable in X with distribution $P(X = x) = p(x)$. Then the entropy of random variable x is defined as III.1.

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{III.1}$$

Bintropy authors assume that the packed executable file has high entropy, therefore Bintropy can classify the file as packed or not by entropy value. Bintropy reads whole file once and calculate the entropy, so it is fast. This Bintropy has a merit that does not need signature of packer update which is a limitation of signature-based classification method.

However, This method has two shortcomings. First is that Bintropy has higher false rate around 10% [21] than other classification methods. Second is that Bintropy can be evaded by simple evasion techniques such as adding monotonous bytes string to the section. For example, malware maker adds long null byte string to the packed executable file, that packed executable file has low entropy rate which can be classified

to non-packed executable file by Bintropy method.

3.2.3 Roberto *et al.* [21]

Roberto *et al.* use values and entropy of PE format header. Following is the feature values of Roberto *et al.*, and these values described in PE file format section of preliminaries.

- Number of standard sections
- Number of non-standard sections
- Number of Executable sections
- Number of Readable/Writable/Executable sections
- Number of entries in the IAT
- Entropy of the PE header
- Entropy of the code sections
- Entropy of the data sections
- Entropy of the entire PE file

Entropy values are calculated using III.1 and other values are counted from PE format header. Authors define the standard sections: .text, .data, and .rsrc., and others are non-standard sections. In addition, number of entries in the IAT is important value since usually packed executable files have a few numbers of IAT entries than ordinary executable files. In general, executable files use dynamic implicit linking, but packed executable files use dynamic explicit linking method.

For instance, UPX packed executable file has .UPX0, .UPX1, and .rsrc sections; this packed file has one standard section and two non-standard sections, and less than 10 IAT entries. Conversely, Ordinary

executable file has .text, .data, and .rsrc sections, and has been larger than 100 IAT entries.

In this way, Roberto *et al.* have classified the suspicious file as packed executable or not with high accuracy 98.91% in their experiment.

However, there is a blind point that feature values of PE format header could be modified. Each name of section and permission of each section is not absolute value. Therefore, all feature values could be modified by malware makers to evade Roberto *et al.* For example, malware maker modifies the UPX packer has .text, .data, and .rsrc section names as same as ordinary executable file. Also, this packer pads the monotonous bytes string to each section to reduce rate of entropy. And then, Roberto *et al.* cannot classify the new packed executable file.

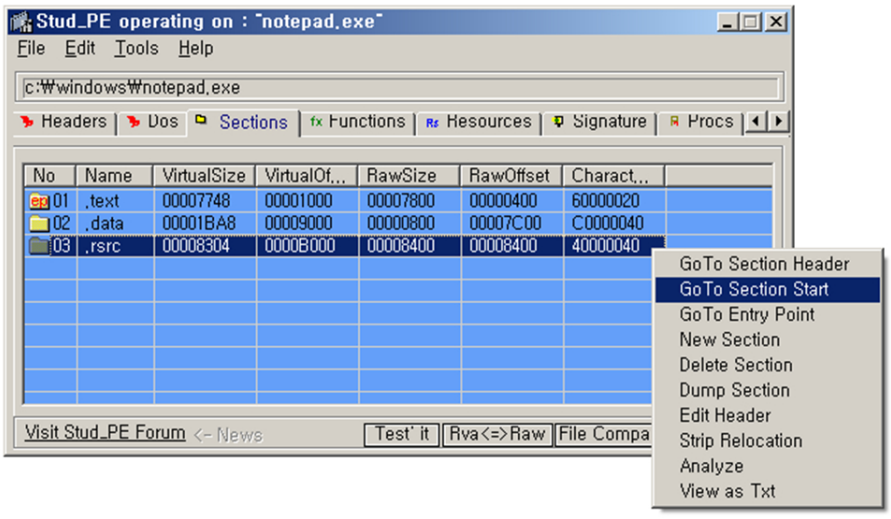

Figure 3.2: Modification of PE Header by Stud_PE [4]

Figure 3.2 is the StudPE program which is one of the PE header modification tools. StudPE can modify the name of section, number of section, size of section, permission of section, and context of section. It means that we can evade Roberto *et al.* by fake packed executable files using StudPE.

IV. Proposed Method

4.1 Our Approach

In related work, we review the previously proposed unpacking and packed executable classification methods. In this thesis, we propose a packed executable file classification method. We have three goals: archive high accuracy, non-evade technique, and efficiency that can apply practical anti-virus scanner. These goals ultimately contribute to the anti-virus scanner that reduces malware infection with a little overhead.

In Bintropy [20], authors assumed packed executable files has higher entropy than non-packed executable files, but Bintropy has high false rate about 10%. In this thesis, we assume packed executable files have high complexity than non-packed executable files since packer frequently used compression algorithms or encryption algorithms for packing. Both compressed data and encrypted data are known as that has high complexity. Underlying this assumption, we propose complexity-based packed executable file classification method.

Our approach is based on the complexity, not signature, entropy, and characteristics. As a matter of fact, an entropy concept is a kind of complexity concept, but there is some difference. In this section, we describe what complexity is, how to measure the complexity, and how to classify the file using complexity.

The complexity concept is proposed by Kolmogorov, Chaitin and Solomonoff [10] to complement the entropy concept to more exact measure information quantity. The Kolmogorov complexity(simply, complexity) $K(X)$ of a finite string X will be defined as the length of the

shortest string of X. In other words, $K(X)$ is the length of the shortest computer program that represents X and then stops. The computer program can be programming languages or any others. Complexity function is defined as

$$K(X) = \min\{X\} \tag{IV.1}$$

For example, the finite string X as

$$\underbrace{11111....1}_{10{,}000 times}$$

then, we can represent this X as follow program.

print 10,000 times a '1'

However, a serious problem of complexity concept is incomputable [8], since finding optimal algorithm that makes the shortest length output program from input string x is infeasible. A good news is compression algorithm as same as the complexity concept.

$$Compress(X) = (X') \tag{IV.2}$$

where X' is the compressed string of string X. Thus, various compression algorithms are used to measure the complexity [23]. The definition of compression algorithm is reduced input size to best smallest output size using their algorithm. Therefore, we can measure the complexity of the input file using compression algorithm for our classification method.

However, we do not need whole function of compression algorithm to measure the complexity. Compression algorithm consists of calculation of compressed length, writing compressed data, and writing a dictionary or other additional data for decompressing. For complexity measure,

we do not need writing compressed data and writing additional data for decompressing procedures. In means that performance of measure the complexity is more efficient than whole compression algorithm.

In this way, our approach can classify the packed executable file using complexity C and threshold t_h by follow operation.

1. C = Length of X / Length of *Compress(X)*
2. $C > t_h$: packed executable file.
3. Else : non-packed executable file.

where X is input string. Almost of packed executable file is compressed or encrypted, we can classify the packed executable file when file has high complexity that over the t_h value. In our approach, setting t_h and choice the compression algorithm are important for accuracy and performance.

4.2 Implementation

We consider three steps for implementation of our approach. First step is a block dividing step for unnecessary bytes string of input file. Second step is that measure the complexity of string throughout the step 1. Last step is classification of input file, whether packed executable or not using t_h. Figure 4.1 illustrates overview of proposed method. We will describe all steps in detail.

4.2.1 Step 1 : Block Dividing

Step 1 is a block dividing step. In general, a PE file has lots of null bytes for filling the left of used section alignment which is a basic unit of file alignment. For example, compiler uses only 0x300 data space when file

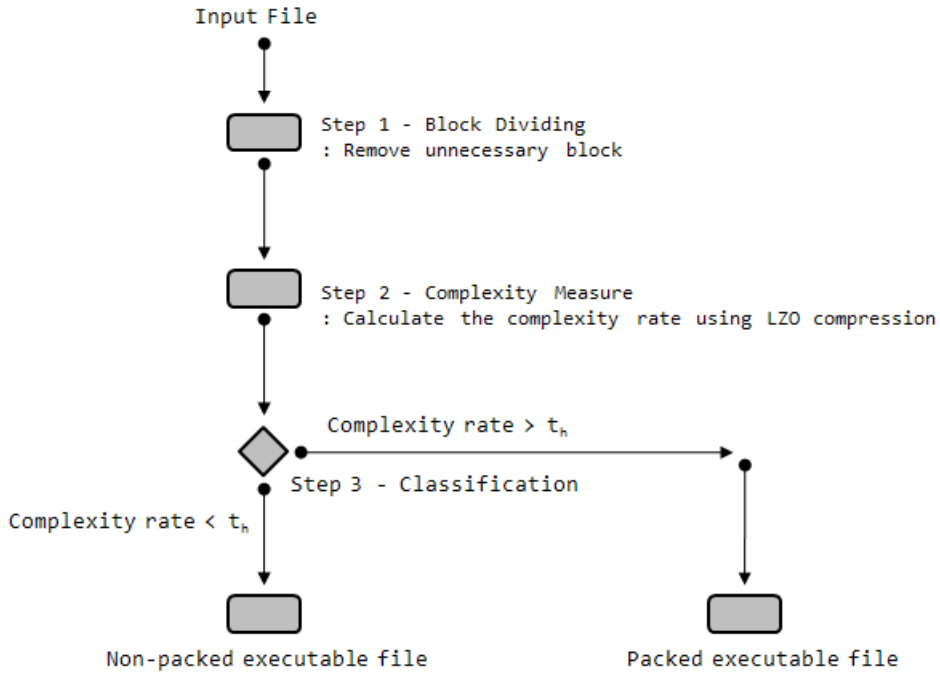

Figure 4.1: Overview of Proposed Method Implmentation

alignment is 0x400, then 0x100 spaces is filled with null bytes. Moreover, a PE file has Bitmap icon file which has low entropy value generally. Moreover, occasionally packed executable file has lots of null bytes, icon file with small size about less than 100KB. In this case, that null bytes padding or icon file can reduce the complexity in largely since size is small as could be reduced the complexity to evade the classification.

Simply put, these cases can interrupt our classification. Therefore, we should consider this block dividing step. To implement this step, we propose a simple algorithm. Following table 4.1 describes notation of our block dividing algorithm and Figure 4.2 illustrates the algorithm.

Dividing algorithm is simple, if the same value of W is found consecutively over CT times, then removes W until W is different with W'. When this dividing algorithm is ended, B consists only necessary string

Table 4.1: Notation of Block Dividing

B	Block string that without unnecessary string
W	Window contains n length string
W'	Temporary Window
CC	Collision count
CT	Collision Threshold

without unnecessary string such as icon, null bytes, or other monotonous strings. We will use the B string in next complexity measure step.

4.2.2 Step 2 : Complexity Measure

Step 2 is complexity measure step which is most important step of our proposed method. We implement the complexity measure using Lempel-Ziv-Oberhumer(LZO) compression algorithm [18] which is the fastest lossless compression algorithm as far as we know. The reason of fastest LZO compression algorithm selection is that not only accuracy is important, but also efficiency.

The LZO algorithm is fast lossless data compression and extremely fast decompression algorithm based on slide dictionary. Slide dictionary is developed for Lempel-Ziv(LZ) algorithm that is frequency-based compression algorithm. Using dictionary, LZ or LZO compression algorithm remove the redundancy of input string.

LZO package consists of various compression algorithms: LZO1, LZO1A, LZO1-99, LZOX and *etc.* We use LZOX-1 algorithm for our complexity measure step, because LZOX-1 algorithm is the fastest compression algorithm among the LZO compression algorithm. LZOX-1 algorithm has about 5 MB/s compression performances in [18]. In fact, it is slower than memcpy() function that has about 60 Mb/s performance, but 5 MB/s is enough performance in the view of anti-virus scanner

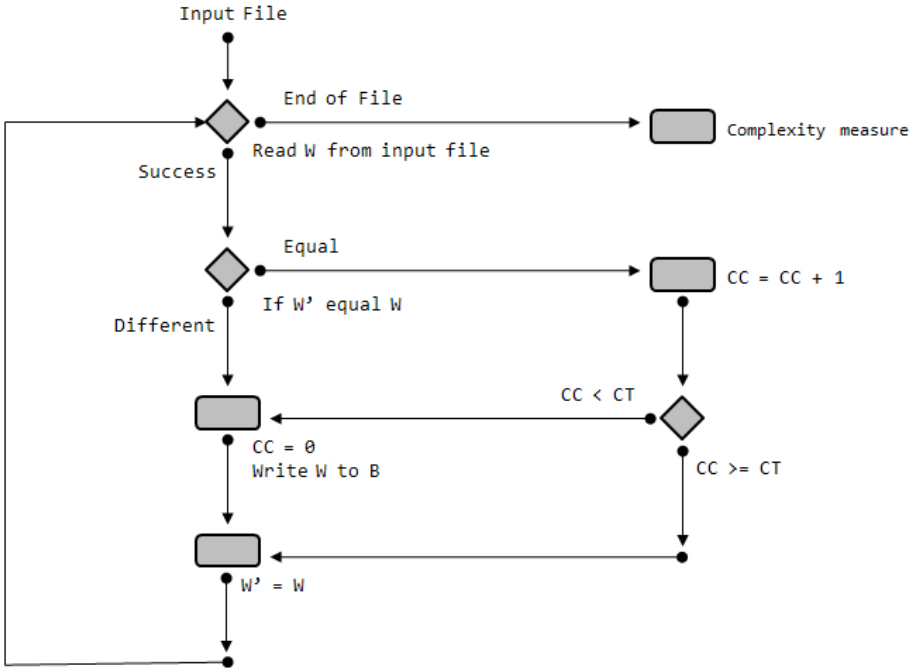

Figure 4.2: Block Dividing Algorithm

that has about 5∼10 MB/s.

In fact, we do not compress the input B string, but measure the complexity. It means that we do not need implementation of writing of compressed data and dictionary. Thus, we modified LZOX-1 compression algorithm simply, just calculates the length of compressed string B. This implementation result has achieved about 6 times faster than original LZO1-X compression algorithm in our experiment. When the end of calculation of length of compressed B, we calculate the complexity using equation IV.3

$$C = \frac{B'}{B} \tag{IV.3}$$

where C is complexity rate, B' is length of compressed B. In this way, we measure the complexity.

4.2.3 Step 3 : Classification

Last step is classification using complexity value C and threshold t_h. Generally packed executable file has high complexity rate than non-packed executable file, we can classify the input file as packed or not using t_h. In fact, this step is same as Bintropy. Only difference is that Bintropy uses entropy, but our classification uses complexity.

If $C > t_h$, then a file is classified as a packed executable file, else a file is classified as a non-packed executable file.

Therefore, setting the threshold t_h is significant to achieve high accurate classification.

V. Experiment & Analysis

5.1 Environment

We implement the program with Visual C++ 6.0 program and experiment environment is Windows XP SP3 on Intel Core 2 Duo CPU E6750 2.66GHz and 2GB RAMs. We gather 2332 sample files for experiment. 343 non-packed executable files from windows, windows/system, and windows/system32 directory on Windows XP Service Pack 2, and then we generated 2079 packed executable files using 6 packers(Aspack[1], FSG, Morphine, PeCompact[2], Upack[7], UPX[12]) except packing error occurred files.

To comparing, we implement the four programs which are Bintropy, Bintropy applied block dividing step, LZO complexity measure, and proposed method that is LZO complexity measure with block dividing step. Lastly, we implement a decision tree which is a kind of machine learning algorithm for threshold setting.

5.2 Threshold Setting

Threshold setting is important to achieve exact classification. In this thesis, we use a machine learning method, which is used by Roberto *et al.* [21], to set the threshold t_h. We make five threshold setting groups that each group consists of 50 non-packed executable files and 60 packed executable files. Table 5.1 shows machine learning result.

Firstly, the unit of Bintropy and Bintropy with Block is entropy and LZO and LZO with Block is complexity rate. 'with Block' means that applied our implementation step 2 which is the block dividing algorithm.

Table 5.1: Machine Learning Result

	Bintropy	Bintropy with Block	LZO	LZO with Block
Group1	6.466	6.604	0.694	0.808
Group2	6.604	6.732	0.692	0.810
Group3	6.530	6.623	0.686	0.810
Group4	6.501	6.592	0.700	0.800
Group5	6.181	6.650	0.693	0.799
Average	6.456	6.640	0.693	0.805

LZO with Block implementation is our proposed method which has step 1 and step 2.

In this table 5.1, block dividing step applied implementation has higher value than others. Because monotonous bytes remove in input file that means file has more complexity than before. We will use average threshold values for classification in our experiment.

5.3 Analysis

We have experiment using t_h and remaining samples of threshold setting groups. Before the classification, we calculate the average entropy and complexity of target files by each implementation. Following table 5.2 shows average of calculation.

Table 5.2: Average entropy and complexity of target files

	Bintropy	Bintropy with Block	LZO	LZO with Block
Non-packed	5.275	5.945	0.546	0.676
Packed	7.209	7.506	0.873	0.945
Gap	1.934	1.561	0.327	0.269

At first glance, the gap between non-packed and packed file is smaller than without applying block dividing algorithm. It looks that applying

block dividing algorithm makes difficult the separation between packed and non-packed files.

Table 5.3: Standard Deviation of target files

	Bintropy	Bintropy with Block	LZO	LZO with Block
Non-packed	0.930	0.537	0.116	0.086
Packed	0.758	0.515	0.112	0.058
Average	0.844	0.526	0.114	0.072

However, this table 5.3 shows smaller average standard deviation value of files applied block dividing algorithm than without block dividing algorithm. It means that throughout block dividing algorithm, the gap between the packed and non-packed file group is closed than before, but each group is closed to average of each group. In this way, we can easily separate between non-packed file and packed file using the block dividing algorithm.

Table 5.4: Result of Classification

	Bintropy	Bintropy with Block	LZO	LZO with Block
Success	89.9%	93.1%	92.2%	97.2%
False	10.10%	6.92%	7.83%	2.84%
False positive	9.65%	6.66%	7.57%	2.52%
False negative	0.45%	0.26%	0.26%	0.32%

Table 5.4 shows average classification result when we applied a threshold of each group to each implementation. This result shows that our proposed method has higher accuracy than Bintropy, and block dividing step also aids classification of implementations.

Table 5.5: Packed classification methods comparison table

	PEiD [11]	Bintropy [20]	Roberto *et al.* [21]	Proposed method
Based on	Signature	Entropy	Characteristics	Complexity
Accuracy	High but need signature	Low (about 90%)	High (about 98%)	High (about 97%)
Evasion technique	Fake signature	Add null bytes	PE header Modification	Nothing

5.4 Comparison

Table 5.5 shows the brief comparison of packed executable classification methods. First of all, our proposed method is based on complexity, not signature, entropy, characteristics. In case of no avoid technique, Bintropy show about 90% accuracy. This accuracy is lower than our proposed method about 97%. This accuracy is almost same as Roberto *et al.*, but our proposed method does not have any known evasion techniques such as padding null bytes or PE header modification.

In addition, our proposed method has high efficiency that is about 30 MB/s, and it needs smaller time than 2.5 μ second per one file classification in our experiment. This efficiency is enough for applying practical anti-virus scanner.

VI. Conclusion

To overcome packer technique, signature-based anti-virus scanner adopts packed executable file classification and generic unpacking methods. Generic unpacking method is relatively widely researched, but this method has several limitations that is low unpacking rate and uses long time and computing resource for unpacking. Therefore, packed executable file classification method is important. Because, when wrong classification is occurred, unnecessary unpacking step is needed or malware cannot be detected. There are three packed file classification methods are proposed. However, these methods have limitations, too. Previous packed file classification methods can be easily evaded or have high false rate.

In this thesis, we propose a high accuracy packed executable file classification method based on complexity. This complexity concept is presented to measure the information quantity more exact than entropy concept. In short, complexity is that shortest output string using optimal algorithm from input string. However, complexity is incomputable since finding optimal algorithm is infeasible. Therefore, many researchers use compression algorithms to measure the complexity such as LZ77, LZW, and *etc.*

We implement our proposed method using LZO compression algorithm which is fastest compression algorithm as far as we know. In addition, we propose a block dividing step, which is removing monotonous block, for exact complexity measure. In our experiment, we use the threshold that sets by machine learning. Our experiment result shows high classification accuracy about 97%, it is higher classification rate than previous entropy-based classification method. Moreover, our proposed method cannot be evaded by evasion techniques as far as we know,

and has enough performance applying to practical anti-virus scanner. Our proposed method contributes low usage of computing resource and high malware detection.

복잡도에 기반한 정확한 실행압축 파일 구분 기법

노한영

악성코드를 탐지하기 위한 대표적인 기법은 파일의 일부 바이트를 서명으로 사용하는 서명 기반의 탐지 기법이다. 서명 기반의 탐지 기법은 정확하고 빠르다는 장점을 가지고 있어서 가장 널리 사용되고 있다. 악성코드 제작자는 이를 회피하기 위하여 파일의 구조를 변경하는 실행압축 기법을 사용한다. 실행압축 기법은 파일의 일부 또는 전체를 암호화 또는 압축을 하고, 이를 복호화 또는 압축해제 하기 위한 코드를 추가로 덧붙인다. 프로그램이 실행이 되면 먼저 복호화 또는 압축해제 코드가 실행이 되고 메모리 상에 복호화, 압축 해제가 이루어진 후에 프로그램이 실행이 된다.

서명 기반의 악성 코드 탐지 프로그램은 이에 대응하여 실행압축 여부를 판단하고, 실행압축을 해제하는 단계를 추가하였다. 현재까지 실행압축 해제 단계는 많이 연구되었지만 아직까지도 그 실행압축 해제율과 속도는 빠르지 않다. 그렇기 때문에, 실행압축 해제 단계의 전 단계인 실행압축 판단 여부의 정확도는 더 중요하다. 실행압축 판단이 잘못되면, 필요치 않은 실행압축 해제 시간이 소모되거나 악성코드 탐지에 실패하게 된다. 본 논문에서는 사용자의 자원 사용을 낮추고, 악성코드 탐지율을 높이기 위하여 실행압축 여부를 판단하는 기법에 대하여 연구하였다.

기존에 연구된 실행압축 여부를 판단하는 기법들은 그 정확도가 높지 않거나, 또는 간단한 회피 기법으로 회피가 가능하다는 단점을 가지고 있다. 본 논문에서는 정확도가 높고 회피 기법을 갖지 않으면서도, 실제 악성코드 탐지 프로그램에 적용할 수 있는 속도를 가지는 기법을 제안한다. 제안 기법은 파일의 복잡도를 측정하여 실행압축

을 판단하는 기법으로, 실행 압축된 파일은 압축되어있거나, 암호화가 되어있어 복잡도가 높다는 특징을 사용한다.

실제 구현에서는 복잡도 측정을 위하여 압축 기법을 사용하였으며, 그 중에서도 빠른 실행 속도를 위하여 알려진 압축 알고리즘 중에서 가장 빠른 Lempel-Ziv-Oberhumer(LZO) 압축 기법을 사용하였다. 그리고, 정확한 복잡도 측정을 위하여 불필요한 부분을 제거하는 단계를 복잡도 측정 전 단계에 추가하였다. 구현 프로그램의 수행 결과 기존에 제안되었던 Bintropy 기법보다 정확도가 약 10% 가 향상이 되었다. 그리고, 알려진 회피기법으로 회피가 불가능하며, 수행 속도는 일반적인 악성코드 탐지 프로그램에 적용하여도 무리가 없는 빠른 수행 속도를 보여주었다.

References

1. ASPack Software. ASPack and ASProtect.
   http://www.aspack.com/

2. Bitsum Technologies. PECompeact2.
   http://www.bitsum.com/pec2.asp

3. C. E. Shannon, "A mathematical theory of communication", *Bell System Technical Journal*, vol. 27, July and October, 1948, pp. 379–423 and 623-656.

4. CGSoftLabs, Stud_PE
   http://www.cgsoftlabs.ro/

5. Computereconomics, "Annual Worldwide Economic Damages from Malware Exceed $13 billion", June, 2007.
   http://www.computereconomics.com/article.cfm?id=1225

6. Daniel J. Sanok, Jr, "An analysis of how antivirus methodologies are utilized in protecting computers from malicious code", *InfoSecCD '05: Proceedings of the 2nd annual conference on Information security curriculum development*, 2005, pp. 142–144.

7. Dwing, UPack, 2008, http://wex.cn/dwing/mycomp.htm

8. G. Chaitin. *The Limits of Mathematics.*, Springer-Verlag, 1998.

9. Gabor Szappanos, Exepacker blacklisting, *Virus Bulletin*, Oct. 2007, pp.14–19.

10. Grunwald, Peter and Vitanyi, Paul, "Shannon information and Kolmogorov complexity", *arXiv:cs.IT/0410002*, 2004.
http://arxiv.org/abs/cs.IT/0410002

11. Jibz, Qwerton, snaker, and xineohP, PEiD, http://www.peid.info/

12. Markus F.X.J. Oberhumer and Laszlo Molnar and John F. Reiser, UPX, http://upx.sourceforge.net/.

13. Martignoni, L. and Christodorescu, M. and Jha, S., "OmniUnpack: Fast, Generic, and Safe Unpacking of Malware", *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, Dec. 2007, pp. 431–441.

14. Min Gyung Kang and Pongsin Poosankam and Heng Yin, "Renovo: a hidden code extractor for packed executables", *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode*, 2007, pp. 46–53.

15. Miroslav Vnuk, "Pavol Navrat, Decompression of Run-Time Compressed PE-Files", *Studies in Informatics and Control*, Vol. 15, No. 2, June 2006.

16. M. Pietrek. "An in-depth look into the Win32 Portable Executable file format"
http://msdn.microsoft.com/msdnmag/issues/02/02/PE/

17. M. Pietrek. "An in-depth look into the Win32 Portable Executable file format, part 2"
http://msdn.microsoft.com/msdnmag/issues/02/03/PE2/

18. Oberhumer.com, LZO Compression algorithm,
http://www.oberhumer.com/opensource/lzo/.

19. Reversing Labs Packer, RPack
http://rlpack.jezgra.net/index.html

20. Robert Lyda and James Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware", *IEEE Security and Privacy*, vol. 5, no. 2, March 2007, pp. 40–45.

21. Roberto Perdisci and Andrea Lanzi and Wenke Lee, "Classification of packed executables for accurate computer virus detection", *Pattern Recogn. Lett.*, vol. 29, no. 14, June 2008, pp. 1941–1946.

22. Royal, P. Halpin, M. Dagon, D. Edmonds, R. Wenke Lee, "PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware", *Proceedings of 2006 Annual Computer Security Applications Conference (ACSAC)*, Dec. 2006, pp. 289–300.

23. Taras Kowaliw, "Measures of complexity for artificial embryogeny", *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008, pp. 843–850

24. T. Brosch and M. Morgenstern. "Runtime packers: The hidden problem?", *Black Hat USA 2006*,
https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf, 2006.

25. Wayne J. Radburn, PEview, http://www.magma.ca/ wjr/

Acknowledgement

There have been a lot of people without whom this thesis could not have been completed. First of all, I cannot thank you enough, or I do not know how to express my thanks to Prof. Kwangjo Kim, my advisor. Furthermore, thanks for effort and interest of advisory committee members who Prof. Soontae Kim and PhD. Doo Ho Choi. Also, I thank all members of our laboratory: Hyunrok Lee, Zeen Kim, Kyusuk Han, Konidala Munirathnam Divyan, Dang Nguyen Duc, Jangsung Kim, Minhea Kwak, Hwewon Park, Hyeran Mun, Sungmok Shin, Myounghan You, and Imsung Choi who give me advices and encouragement. Moreover, I also thanks Hyunkyoung Park for helpful as a staff member.

Especially, I would like to give my special thanks to my family: my parents and my brother. Without their help I am not able to complete this thesis.

Curriculum Vitae

Name : Hanyoung Noh

Date of Birth : May 26, 1983

Sex : Male

Nationality : Korean

Education

2002.3–2007.2 Computer Science
Sungkyunkwan University (B.S.)

2007.2–2009.2 Engineering
Information and Communications University (M.S.)

Career

2007.03–2007.12 Graduate Research Assistant
Research and Development of Next Generation DRM
SK telecom

2007.03–2008.02 Graduate Research Assistant
Development of Sensor Tag and Sensor Node Technique for RFIDUSN
Electronics and Telecommunications Research Institute(ETRI)

2008.01–2008.12 Graduate Research Assistant
Research on the Next Generation Standard of EPC-global
ETRI

2007.03–2008.06 Graduate Research Assistant
Research on ID-based Cryptography Technique and Applications for 4G Mobile Network
Samsung-ICU Research Center

2007.03–2008.02 Graduate Research Assistant
Standards Development for Cyber Security Policy on Digital Measurement & Control System
Korea Institute of Nuclear Safety(KINS)

2008.02–2008.12 Graduate Research Assistant
Development of Secure Sensor Network for Surveillance and Reconnaissance
SNR

2008.03–2009.02 Graduate Research Assistant
A Study on Cyber Security Assessment using Penetration Test of Digital I&C Systems
KINS

Publications

(1) 2007. 6 노한영, 이현록, 김광조, "하드웨어 고유 번호 기반 소프트웨어 보호 방식", 2007년도 한국정보보호학회 하

계학술발표대회, pp.385-390, 2007. 6.22, 단국대학교 천안캠퍼스 제3과학관.

(2) 2007. 12 이현록, 노한영, 박재범, 김광조, "난독화 서버를 이용한 소프트웨어 보호방식", CISC-W'07 Proceedings vol.17, no.2, pp.410-413, 2007.12.1, 상명대학교, 서울.

(3) 2008. 1 Hanyoung Noh, Jangseong Kim, Chanyeob Yeun and Kwangjo Kim, "New Polymorphic Worm Detection based on Instruction Distribution and Signature", Proc. Of SCIS 2008, Jan. 22-25, 2008, Miyajaki, Japan.

(4) 2008. 6 노한영, 김광조, "오탐지 정확도를 개선한 실행압축 판단 기법 연구", CISC-S'08 Proceedings vol.18, no.1, pp.384-387, 2008.6.26, 순천향대학교, 천안.

(5) 2008. 12 노한영, 김장성, 김광조, "포렌식을 고려한 휴대폰 데이터 보안", CISC-W'08 Proceedings vol.19, no.1, pp.66-69 , 2008.12.6, 고려대학교, 서울