Generic Packing Detection using Several Complexity Analysis for Accurate Malware Detection

Dr. Mafaz Mohsin Khalil Al-Anezi
Computer Sciences
College of Computer Sciences and Mathematics,
Mosul University, Mosul, Iraq

***Abstract*— The attackers do not want their Malicious software (or malwares) to be reviled by anti-virus analyzer. In order to conceal their malware, malware programmers are getting utilize the anti reverse engineering techniques and code changing techniques such as the packing, encoding and encryption techniques. Malware writers have learned that signature based detectors can be easily evaded by "packing" the malicious payload in layers of compression or encryption. State-of-the-art malware detectors have adopted both static and dynamic techniques to recover the payload of packed malware, but unfortunately such techniques are highly ineffective. If the malware is packed or encrypted, then it is very difficult to analyze. Therefore, to prevent the harmful effects of malware and to generate signatures for malware detection, the packed and encrypted executable codes must initially be unpacked. The first step of unpacking is to detect the packed executable files.**

The objective is to efficiently and accurately distinguish between packed and non-packed executables, so that only executables detected as packed will be sent to an general unpacker, thus saving a significant amount of processing time. The generic method of this paper show that it achieves very high detection accuracy of packed executables with a low average processing time.

In this paper, a packed file detection technique based on complexity measured by several algorithms, and it has tested using a packed and unpacked dataset of file type .exe. The preliminary results are very promising where achieved high accuracy with enough performance. Where it achieved about 96% detection rate on packed files and 93% detection rate on unpacked files. The experiments also demonstrate that this generic technique can effectively prepared to detect unknown, obfuscated malware and cannot be evaded by known evade techniques.

Keywords—Packed Executables; Malware Detection; compression algorithms

I. INTRODUCTION

As a consequence of the arms race between virus writers and anti-virus vendors, sophisticated code obfuscation techniques are commonly implemented in computer viruses. Executable code polymorphism, metamorphism, packing, and encryption, have been proven very effective in evading detection by traditional signature-based anti-virus software. Traditional signature-based anti-virus software needs updating the virus database regularly, and the virus detection relying on the known virus database is a passive protection technology without the capacity of detecting the new unknown virus, the virus deformation, and packed virus. Among these techniques, executable packing is the most common due to the availability of several open source and commercial executable packers [21][14].

According to [9][5][7], over 80% of computer viruses appear to be using packing techniques. Moreover, there is evidence that more than 50% of new viruses are simply re-packed versions of existing ones, see Fig.1. It has been reported that among 20, 000 malware samples collected in April 2008, more than 80% were packed by packers from 150 different families. This is further complicated by the ease of obtaining and modifying the source code of various packers. Currently, new packers are created from existing ones at a rate of 10 to 15 per month [7].

Although executable packing is very popular among virus writers, it is also applied for encrypting benign executables. Programmers of benign software apply packing to their applications mainly to make the resulting executables smaller in terms of bytes, and therefore faster to distribute through the network, for example. Also, packing makes reverse-engineering more difficult, thus making it harder for hackers to break the software license protections. As a matter of fact, there exist many commercial executable packing tools that have been developed mainly for protecting benign applications from software piracy. However, the percentage of packed benign executables is low (perhaps as low as 1%, although we were not able to find any study that can confirm this estimate, which is based solely on our experience) [14].

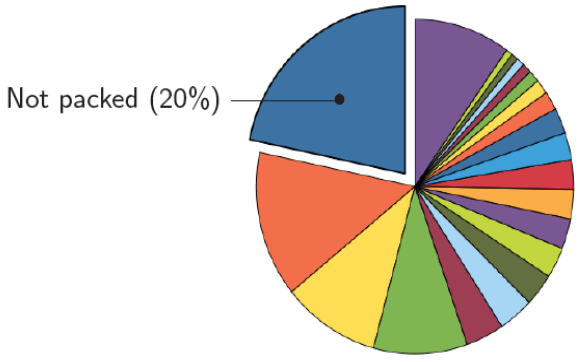

Fig. 1. Malware and packing, 80% of new malware are packed with various packers, 50% of new malware samples are simply repacked versions of existing malware [8].

An executable packing tool is a software that given a program P generates a new program P′ which embeds an encrypted version of P and a decryption routine. When P′ is executed, it will decrypt P on the fly and then run it. Assuming P contains known malicious code, signature based anti-virus would (likely) be able to detect it. However, if P has been packed the anti-virus will try to match the signature of P on P′. As the malicious code of P is encrypted in P′, no match will be found. Therefore, P will evade detection and infect the victim machine, if P′ is executed [14].

PE (Portable executable) file format is a standard Windows executable file format, which plays a very important role in the Windows operating system. PE files are widely used in Win32 executable programs including EXE, DLL, OCX, SYS, SCR and so on. PE viruses are designed in the way making use of the characteristics of PE file structure, and are portable on different hardware platforms, which is a serious security threat to the Windows operating system [21].

The very first step in the unpacking of packed file is to detect packed executable files. Recently, many researchers and analysts have focused on packed file detection techniques. In this paper, however, a new lightweight packed PE file detection technique based on the analyze the complexity of PE files by several algorithms. Packed PE files were analyzed using the proposed technique. It was found that nearly every type of packed PE file has higher complexity than it in unpacked status.

The methods always used are intelligent or it depends on database, but there are two major problems in them. Firstly, masses of malicious and benign codes as training data set are difficult to collect. Secondly, it would consume a lot of time to train the classifiers, and so the efficiency of the detection of unknown virus is dissatisfactory and difficult to use in practice.

A few generic and automatic unpacking techniques have been proposed to unpack packed binaries without specific knowledge of the packing technique used, e.g., OmniUnpack, Justin, Renovo, PolyUnpack and others [7].

The objective is to accurately distinguish between packed and non-packed executables, so that only the executables detected as packed will be sent to a computationally expensive general unpacker for hidden code extraction, before being sent to the antivirus software.

Therefore, the classification system here helps in improving virus detection while saving a significant amount of processing time. This paper do not focus on the improvements in virus detection accuracy achieved after unpacking, because this has already been studied in other researchs, for example. Instead, it focus on the accuracy and computational cost related to the classification of packed executables into the two classes packed and non-packed.

II. Related Works

Most Packer Detection Methods can be summed up by: Signature based (Executable code signatures) and Heuristics (Entropy Checks, Import Address Table, Other Checks (not exclusive to packers)).

Coogan et al. [3] proposed an automatic static unpacking mechanism. It uses static analysis techniques to identify the unpacking code that comes with a given malware binary, then uses this code to construct a customized unpacker for that binary. This customized unpacker can then be executed or emulated to obtain the unpacked malware code.

Exeinfo PE [4] is an ongoing work for packed PE file detection and PE header information extraction. It shows the entrypoint, file offset, compiler information and the unpack information of the input file.

Renovo [6] utilizes a virtual machine. By using a virtual machine, they run a packed executable and record memory writing operations on shadow memory. When execution flow reaches one of checked bits of the shadow memory, all the checked memory bits are dumped. Shadow memory is changed to extract hidden code from packed executables with multiple hidden layers. With this mechanism, Renovo can find hidden layers as well.

OmniUnpack [8] monitors the program execution and tracks written, as well as written-then-executed, memory pages. When the program makes a potentially damaging system call, OmniUnpack invokes a malware detector on the written memory pages. If the detection result is negative, execution is resumed. If new type of malware appears, the dangerous system calls they defined on their paper could not match.

OllyDbg [12] is a debugger that emphasizes binary code analysis, which is useful when source code is not available. It traces registers, recognizes procedures, API calls, switches, tables, constants and strings, as well as locates routines from object files and libraries. According to the program's help file, version 1.10 is the final 1.x release. Version 2.0 is in development and is being written from the ground up. The software is free of cost, but the shareware license requires users to register with the author. OllyDbg is only available in 32-bit binaries. OllyDbg shows the message box that the input file is packed when the file is detected as a packed or encrypted file.

PEiD [13] is most commonly used with signature-based packers, cryptors and compilers for PE file detection. At present, it can detect more than 600 different signatures in PE files. PEiD is unique in some regard when compared to other identifiers. Its detection rates are pretty good among the current identifiers. Moreover, it has a plugin interface that supports plugins such as Generic OEP Finder and Krypto ANALyzer. Finally, it is free and easy to use.

Robert, et al.[16] present an encrypted and packed malware detection technique based on entropy analysis. In their paper, they analyzed packed PE files via the byte distribution. A set of metrics are developed that analysts can use to generalize the entropy attributes of packed or encrypted executable and thus distinguish them from native (non-packed or unencrypted) executables. As such, this methodology computes entropy at a naive 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. Entropy analysis examines the statistical

variation in malware executables, enabling analysts to identify packed and encrypted samples quickly and efficiently.

PolyUnpack [17] performs static analysis over a packed executable to acquire a model of what its execution would look like if it did not generate and execute code at runtime. When the first instruction of a sequence not found in the static model is detected, the unknown instruction sequence is written and the execution of the packed executable is halted.

Hump-and-Dump [20] is a different approach from other research. Hump-and-Dump tries to find the OEP. Using a characteristic of unpacking, it counts the number of loops used in unpacking. When the number of loops is greater than a threshold and no more big loops are used for the period of a threshold, the address of the loop end point is the OEP.

III. PACKER

A packer is proposed to reduce file size at first. A packed executable file is a file applied packer. This packed executable file operates functionally same as original file. Fig. 2 illustrates packing operation of packer [9].

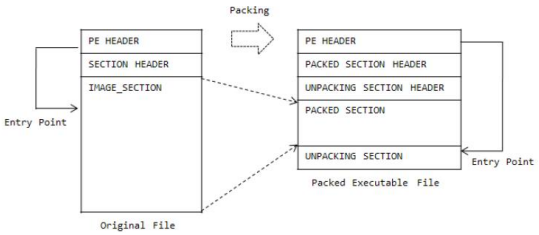

Fig. 2. Packer structure [9].

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. Fig. 3 illustrates the procedures of execution 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 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 are proposed, that is packed executable files are considered as malware.

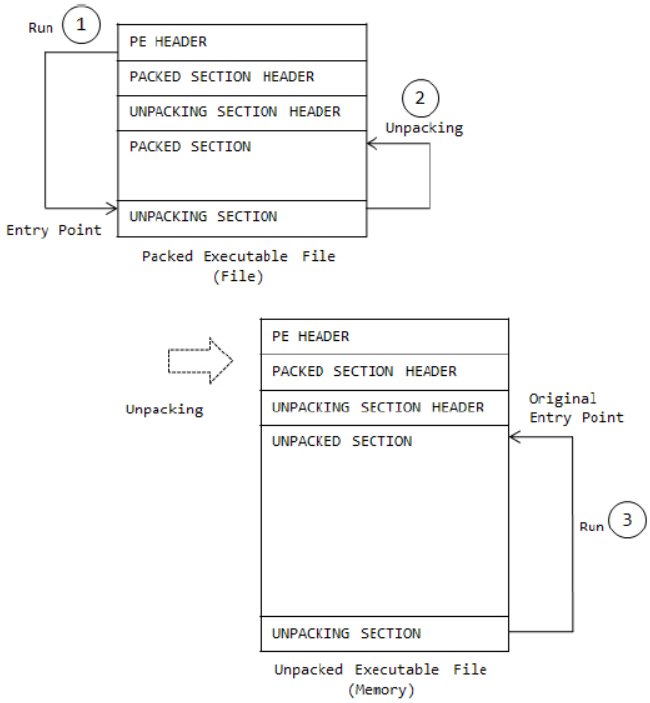

Fig. 3. Execution operation of packed executable file [9].

Among some packers, the most widely used packer is the UPX, ASPack, Themida, and so on. In next section, only will describe characteristics of UPX packer because it is used to pack the dataset in this paper.

A. UPX

The UPX(Ultimate Packer for eXecutables) was released in March 1998. That is the first beta version. And then recently the version 3.07 was released in September 2010, the UPX is created by the Markus Oberhumer and Laszlo Molnar. And that is distributed in GPL(General Public License). The UPX offers the more high compression ratio than the Winzip or GZIP, see Fig4. And the decompress speed is faster than the others compression applications. The compression speed of the UPX is about 10MB/sec on the Pentium 133 and about 200MB/sec on the Athlon 2000. Also the UPX supports many file formats and various platforms. As explained earlier, the

UPX compression ratio is superior other applications and is a commonly used algorithm. However, the UPX is already a widely known packing algorithm so, the packed binary as the UPX is able to unpack [19][11].

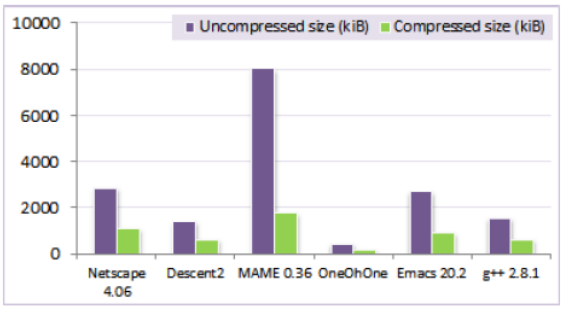

Fig. 4. UPX Compression Ratio [18]

IV. PE FILE AND PE VIRUS

PE (Portable Executable) file is an important executable file format of Windows operating system . ALL win32 executable (except VxDs and 16-bit DLLs) are PE file format. Files of 32bit DLLs, COM files, OCX controls, Control Panel Applets (CPL files) and NET executables are all PE format. The portability of PE file format means that the file format can be used on all Win32 platforms, and PE loader can recognize and use the file format in all win32 platforms. PE viruses take advantages of the PE file format to spread themselves among different Win32 platforms. The data structure of PE file in memory is consistent with that on disk. PE file uses a flat address space in which all code and data are merged into a large structure. PE loader maps the disk file to the virtual address space by the mechanism of mapping file to the memory [21]. All of the data structures of PE file are defined in WINNT.H.

A. PE virus

PE virus is a computer virus that can infect PE format file in Windows operating system. Most of PE viruses are written with Win32 assembly language. PE virus has no data section. Variables and data are all put in code section. There are several key technologies of Win32 virus like Virus address relocation, Obtaining API Address, Searching target files, Mapping files to the memory, The general process of virus infection, Returning to the host program [21], and others.

As mentioned before, signature is a specific bytes string. But 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 In particular, the packing of malware is the very first problem that an analyst should address. If it is impossible to unpack a packed executable file, the analysis is impossible because the codes cannot be understood [9].

V. FILE ANALYSIS

Security researchers need to find ways to fight malware, i.e., they need to obtain malware samples, analyze them to gain an understanding of malware tactics and weaknesses, and use that understanding to develop effective countermeasures [1].

A. Static Analysis

Static analysis is a generic term referring to analysis methods that do not involve executing the program to be analyzed, for the sake of brevity henceforth called a specimen. Static analysis can be used to gather a variety of information about a specimen, e.g., high-level information such as its file size, a cryptographic hash, its file format, imported shared libraries. Cryptographic hashes can be used to identify a specimen. Packer signatures or its entropy may be used to determine whether it might be runtime packed.

Static analysis has several advantages over dynamic approaches. As static methods do not involve executing a potentially malicious specimen, there is a lesser risk of damaging the system that analysis is performed on. Given availability of the right tools, it is also possible to perform the analysis on a platform that differs from the platform that the specimen is designed to run on, further mitigating the risk of damaging the analysis platform (e.g., by accidentally executing it). Furthermore, static analysis typically covers the whole specimen and not just those code paths that are executed for a set of inputs, like dynamic analysis.

B. Dynamic analysis

Dynamic analysis is a way of analyzing an unknown program by executing it and observing its behavior. When executing potentially hostile code, careful consideration must be given to securing the analysis environment, so as not to risk its destruction or even damage to other computer systems on the same network.

VI. UNPACKING TRADITIONAL METHODS AND THEIR LIMITATIONS

1) Signature-based Unpacking method.

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 [9]. PEiD is an example of signature-based packer[2][13].

2) Algorithm-based Unpacking method

Use of specific unpacking routines to recover the original code (i.e., one routine per packing algorithm). Their limitations are [8]:

- Every new packer requires a dedicated unpacking algorithm.
- New packers are created from existing ones at a rate of 10-15 per month.

3) Generic Unpacking method

Emulation/tracing of the execution until the unpacking routine terminates (e.g., PolyUnpack and Renovo). Their limitations are [8]:

- Unpacking is slow and is not suitable for end-user environments.
- Effectiveness depends on the fidelity of the emulation environment (packers leverage anti-emulation techniques).

4) Heuristic

This involves searching through the code in a file to determine whether that code takes actions that appear to be actions typical of a packed file. The more packed like code that is found, the more likely that a packed is present. Heuristics approach of detection to provide protection against new and unknown packer, but it is inefficient and inaccurate where it is usually resulting in false positives. And there is a difficult to describe a heuristic which will work on all kinds of computer systems.

VII. Proposed Method and Goals

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

To achieve these goals, the arbitrator of 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, this is easily the judge that executable file is packed or not. A complexity concept can measure the information quantity more correctly than entropy concept.

The programs in Portable Executable (PE) 32-bit and 64-bit Microsoft Windows operating systems format is used. And in order to classify an executable program, binary static analysis is used to extract information. This information allows us to translate each executable into a sequence string of bytes. Then apply complexity measures techniques to distinguish between packed and non-packed executables.

Figure 5 shows how the classifier may be used to improve virus detection accuracy with low overhead, compared to a system where all the executables are directly sent to the general unpacker. Once a PE executable is received, the classification system performs a static analysis of the PE file in order to measure the complexity of it. After that, the complexity obtained from the PE executable is compared with a threshold. If the executable is classified as packed, it will be sent to the general unpacker for hidden code extraction, and the hidden code will then be sent to the anti-virus scanner. On the other hand, if the executable is classified as non-packed, it will be sent directly to the anti-virus scanner. It is worth noting that the PE file classifier may erroneously label a non-packed executable as packed. In this case the general unpacker will not be able to extract any hidden code from the received PE file. Nonetheless, this is not critical because if no hidden code is extracted, the AV scanner will simply scan the original non-packed code. The only cost paid in this case is the time spent by the general unpacker in trying to unpack a non-packed executable. On the other hand, the PE classifier may in some cases classify a packed executable as non-packed. In this case, the packed executable will be sent directly to the anti-virus scanner, which may fail to detect the presence of malicious code embedded in the packed executable, thus causing a false negative. However, this PE file classifier has a very high accuracy and is therefore able to limit the false negatives due to these cases.

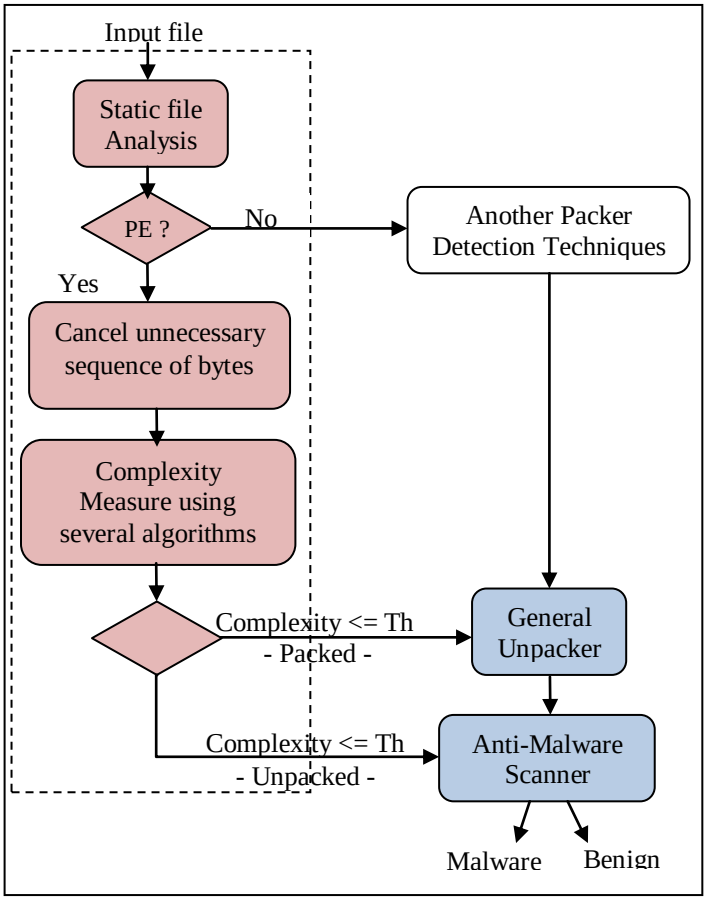

Fig. 5. Overview of classification Method and operations of Anti-Malware Scanner

The complexity concept is proposed by to complement the entropy concept to more exact measure information quantity. The complexity C(X) of a finite string X will be defined as the length of the shortest string of X. In other words, C(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 [9]. Complexity function is defined as

$$C(X) = \min \{X\} \quad (1)$$

For example, the finite string X as

$$\underbrace{111111\ldots..1}_{10;000\text{times}}$$

then, this X can represented as follow program.
print 10,000 times a '1'

However, a serious problem of complexity concept is incomputable, 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 [9].

$$\text{Compress}(X) = (X`) \quad (2)$$

where X` is the compressed string of string X. Thus, various compression algorithms are used to measure the complexity. The definition of compression algorithm is reduced input size to best smallest output size using their algorithm. Therefore, the complexity can measured for the input file using compression algorithm for classification.

Almost of packed executable file is compressed or encrypted, so to classify packed executable file is done when file has high complexity. But the difference value between file length's before and after compress is lower in the case of packed file than in the unpacked file. So if the complexity lower than Th value it will be packed else, unpacked. So setting Th and choice the compression algorithm are important for accuracy and performance.

Three steps are considered for implementation:

1) Scan the sequence string bytes of input file for unnecessary bytes and cancel them.

2) Compress the bytes string throughout the step1 by several compression algorithms and entropy.

3) Measure the Complexity of string throughout the step 2. By the follow operation.

- C = Length of X / Length of Compress(X) in the case of compression algorithms are used, where X is input string,
- C= 8- entropy, in the case of entropy is used,
- C <= Th : packed executable file, else: unpacked executable file.
- And, it is packed executable file, if the decision of at least 5 compression algorithms and entropy is packed, else unpacked executable file.

VIII. The used Compression Algorithms

The entropy and compression algorithms used to measure the complexity in this paper can be summarized as following:

A. Entropy Analysis

In information theory, entropy is a measure of uncertainty in a series of an information unit. Information is compressed by following a logical sequence. First, some repeated patterns are found in the information, and then the redundancies of the patterns are used to reduce the size of the information. That is, the number of patterns of the information is reduced by compression and a series of bits becomes more unpredictable, which is equivalent to uncertainty. Therefore, the measured entropy of compressed information is higher than of the original information. Shannon's formula is devised to measure information entropy, as follows [5]:

$$H(x) = -\sum_{i=1}^{n} p(i) \cdot \log_b p(i) \quad (3)$$

where H(x) is the measured entropy value and p(i) is the probability of an ith unit of information in event x's series of n symbols. The base number of the logarithm can be any real number greater than 1. However, 2, 10, and Euler's number e are chosen in general. We choose b=2 so this formula generates entropy scores as real numbers; when there are 256 possibilities, they are bounded within the range of 0 to 8.

B. LZO

Lempel–Ziv–Oberhumer (LZO) is a lossless data compression algorithm that is focused on decompression speed. It is a portable lossless data compression library written in ANSI C. It offers pretty fast compression and very fast decompression. Decompression requires no memory.

LZO is a data compression library which is suitable for data de-/compression in real-time. This means it favours speed over compression ratio [10].

It is a block compression algorithm—it compresses and decompresses a block of data. Block size must be the same for compression and decompression. The LZO library implements a number of algorithms with the following characteristics:

- Decompression is simple and *very* fast.
- Requires no memory for decompression.
- Compression is pretty fast.
- Requires 64 KiB of memory for compression.
- Includes compression levels for generating pre-compressed data which achieve a quite competitive compression ratio.
- Algorithm is thread safe.

C. Deflate

Deflate is a data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding. Deflate is widely thought to be implementable in a manner not covered by patents. This has led to its widespread use, for example in gzip compressed files, PNG image files and the ZIP file format for which Katz originally designed it [23][22].

Compression is achieved through two steps:

- The matching and replacement of duplicate strings with pointers.
- Replacing symbols with new, weighted symbols based on frequency of use.

D. LZW

LZW compression is named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch. It is the foremost technique for general purpose data compression due to its simplicity and versatility. Typically, you can expect LZW to compress text, executable code, and similar data files to about one-half their original size. LZW also performs well when presented with extremely redundant data files, such as tabulated numbers, computer source code,

and acquired signals. Compression ratios of 5:1 are common for these cases. LZW is the basis of several personal computer utilities that claim to "double the capacity of your hard drive". LZW compression is always used in GIF image files, and offered as an option in TIFF and PostScript [19].

E. Gzip

gzip is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding. DEFLATE was intended as a replacement for LZW and other patent-encumbered data compression algorithms which, at the time, limited the usability of compress and other popular archivers. "gzip" is often also used to refer to the gzip file format[23].

Although its file format also allows for multiple such streams to be concatenated (zipped files are simply decompressed concatenated as if they were originally one file), gzip is normally used to compress just single files. Compressed archives are typically created by assembling collections of files into a single tar archive, and then compressing that archive with gzip. The final .tar.gz or .tgz file is usually called a tarball.

gzip is not to be confused with the ZIP archive format, which also uses DEFLATE. The ZIP format can hold collections of files without an external archiver, but is less compact than compressed tarballs holding the same data, because it compresses files individually and cannot take advantage of redundancy between files (solid compression) [23].

F. QuickLZ

QuickLZ is the world's fastest compression library, reaching 308 Mbyte/s per core. It can be used under a commercial license if such has been acquired or under GPL 1, 2 or 3 where anything released into public must be open source [15]. It characterize by:

- Simple to use and easy to integrate. Get done in minutes and continue developing!
- Streaming mode for optimal compression ratio of small packets down to 200 - 300 bytes in size.
- Auto-detection and fast treatment of incompressible data.

IX. Experimental Results

The dataset used in this paper consists of 250 benign unpacked programs that were randomly gathered from the system files of windows XP operating system, then these files are packed using UPX. Each set of unpacked .exe files, and packed .exe files are enter alone in the classifier and the last decision is not depend on one the other.

Table 1 and figure 6 show the higher detection rate (True Positive TP = 0.96) of unpacked files is for the Totality Algs, this mean that the False Positive is (FP = 0.04). While the lower detection rate (TP = 0.83) of unpacked files is for the Qlz, this mean that the higher False Positive is (FP = 0.17).

Table 2 and figure 7 show the higher detection rate (True Negative TN = 0.97) of unpacked files is for the Entropy, this mean that the lower False Negative is (FN = 0.03). While the lower detection rate (TN = 0.9) of unpacked files is for the Qlz, this mean that the higher False Negative is (FN = 0.1).

TABLE I. 250 Unpack .exe file

Algorithm	Unpack	Pack	Detection Rate
Entropy	57	193	0.228
LZO	233	17	0.932
QLZ	207	43	0.828
Gzip	235	15	0.94
Deflate	235	15	0.94
LZW	239	11	0.956
Totality Algs	240	10	0.965

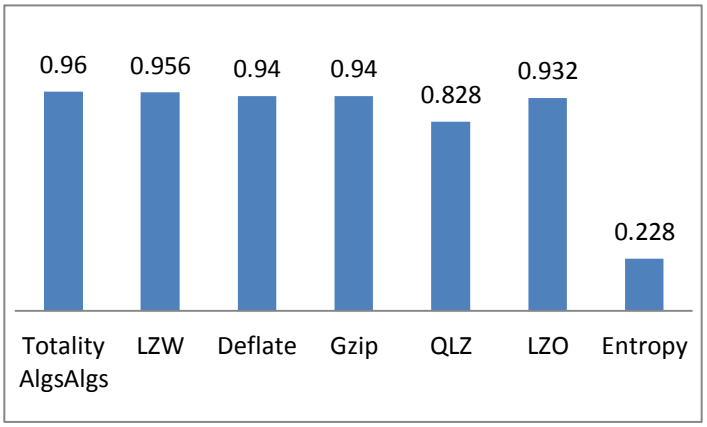

Fig. 6. 250 UnPack .exe file

TABLE II. 250 Pack .exe file

Algorithm	Unpack	Pack	Detection Rate
Entropy	7	244	0.976
LZO	25	225	0.9
QLZ	15	235	0.94
Gzip	16	234	0.936
Deflate	17	233	0.932
LZW	9	241	0.964
Totality Algs	18	232	0.928

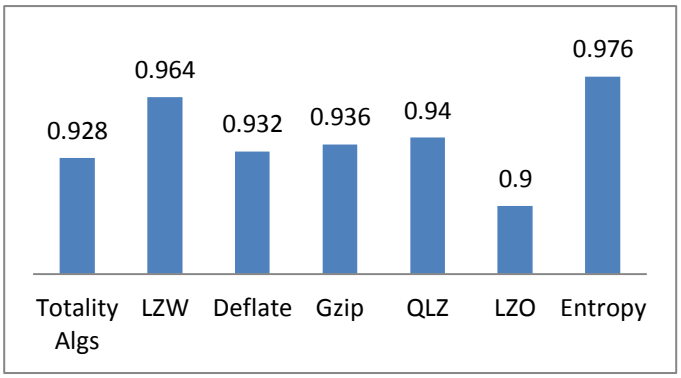

Fig. 7. 250 Pack .exe file

X. CONCLUSION

The main goal in this paper, is to classify a packed and unpacked executable file in simple manner and achieve 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.

Complexity is measured using entropy and five known compression algorithms. And the advantage of using complexity analysis is that it offers a convenient and quick technique for analyzing a sample at the binary level and identifying suspicious PE file (packed and encrypted Executables). This Generic unpacking has low-overhead by using existing hardware mechanisms, and it is characterized by fast, detect unknown packers, and resilient to anti-debugging.

For future works to enhance the detection rates use an artificial technique to segment the PE file and eliminate the less important segments, and further to add this packing/unpacking detection step to unpacking system.

REFERENCES

[1] Bohne L., 2008, "Pandora's Bochs: Automatic Unpacking of Malware", Diploma thesis, Laboratory for Dependable Distributed Systems University of Mannheim.

[2] Choi Y., Kim I., Oh J., and Ryou J., 2009, "Encoded Executable File Detection Technique via Executable File Header Analysis", International Journal of Hybrid Information Technology, Vol.2, No.2, pp. 25-36.

[3] Coogan K., Debray S., Kaochar T., and Townsend G., "Automatic static unpacking of malware binaries". In WCRE '09: Proceedings of the 2009 16th Working Conference on Reverse Engineering, pages 167–176, Washington, DC, USA, 2009. IEEE Computer Society, Iraq Virtual Science Library.

[4] Kang M. G., Poosankam P., and Yin H.. "Renovo: a hidden code extractor for packed executables". In WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode, pages 46–53, New York, NY, USA, 2007. ACM.

[5] L. Limin, M. Jiang, W. Zhi, G. Debin, and J. Chunfu, "Denial-of-Service Attacks on Host-Based Generic Unpackers", 2010.

[6] Martignoni L., Christodorescu M., and Jha S., "OmniUnpack Fast, Generic, and Safe Unpacking of Malware", ACSAC 2007, Iraq Virtual Science Library.

[7] Noh H., 2009, "Complexity-based Packed Executable Classification with High Accuracy", Master Thesis, School of Engineering, Information and Communications University, Korea.

[8] Oberhumer M., http://www.oberhumer.com/opensource/lzo, Version: 2.06, Date: 12 Aug 2011.

[9] Oberhumer M., Molnar L. & Reiser J., 1996-2010, "The Ultimate Packer for eXecutables", UPX, http://upx.sourceforge.net

[10] OllyDbg homepage, http://www.ollydbg.de/

[11] PEiD homepage, http://www.peid.info/

[12] Perdisci R., Lanzi A., and Lee W., 2008, "Classification of Packed Executables for Accurate Computer Virus Detection", Elsevier, Iraq Virtual Science Library.

[13] QuickLZ 1.5.x, http://www.quicklz.com/index.php, 2013

[14] Robert, Lyda, et al, "Using Entropy Analysis to Find Encrypted and Packed Malware", IEEE Security and Privacy, Apr. 2007, Iraq Virtual Science Library.

[15] Royal P., Halpin M., Dagon D., Edmonds R., and Lee W, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware". In ACSAC'06: Proceedings of the 22nd Annual Computer Security Applications Conference, pages 289–300, Washington, DC, USA, 2006. IEEE Computer Society, Iraq Virtual Science Library.

[16] Shin D., Im C., Jeong H., Kim S., and Won D., 2011, "The new signature generation method based on an unpacking algorithm and procedure for a packer detection", International Journal of Advanced Science and Technology Vol. 27, February, pp 59-78.

[17] Steven W. Smith , "The Scientist and Engineer's Guide to Digital Signal Processing", copyright ©1997-1998.

[18] Sun L., Ebringer T., and Boztas S., "Hump-and-dump: efficient generic unpacking using an ordered address execution histogram". 2nd Int'l CARO Workshop, May 2008.

[19] Tian Z., Sun X. and Yang H., 2011, "A Scheme of PE Virus Detection Using Fragile Software Watermarking Technique", International Journal of Digital Content Technology and its Applications. Volume 5, Number 2, February, pp. 158-164.

[20] Wagner C., "Data Compression- DEFLATE Algorithm", Spring Semester 2011.

[21] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/, 2013.