Detection of Packed Malware

Dhruwajita Devi and Sukumar Nandi
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039, Assam, India
{dhruwajita.devi, sukumar}@iitg.ernet.in

ABSTRACT

Packing is the most popular obfuscation technique used by malware writers' community in present scenario. The traditional signature-based anti-virus software had played a major role in malware detection, until the dawn of the trend of packed malware. Hence to evade detection of the malwares, a malicious writer relies on packers' softwares; which transforms the binary appearance of the programs without affecting its execution semantics. Therefore the biggest challenge today for malware detection techniques is to figure out whether a given binary is packed or not.

In this paper, we apply pattern recognition technique for detection of packed malware binaries. The objective of our approach is to take out the best set of features from Windows Portable executable files in order to pass it to our classification model. The classification model works in two phases, in the first phase it classifies the packed and non-packed executables. Once an executable is classified as packed, the second phase of classification concludes whether it is packed benign or packed malware executable. We worked with the UPX packer for this approach and have been able to achieve more than 99.9% accuracy in the first phase of classification. We achieved more than 95% accuracy in the second phase of classification as well.

Keywords

Portable Executable, Packer, Packed, Non-Packed

1. INTRODUCTION

In order to evade detection by the anti-virus (AV) software and the reverse engineers, the malware writers' community uses the obfuscation technique of packing malwares. Polymorphism, metamorphism, packing, and encryption, etc. are some of these techniques. Though several open source and commercial executables packers are available in the market, Executable Packing is the most popular of all [14]. Packing means an executable file packed inside another executable file [19]. A Packer or executable packing tool is basically software which generates a new program while an executable program is fed as an input. The new program embeds an encrypted version of the input executable and a decryption routine. At the time of execution, the outer executable unpacks the contents of the inner executable into memory and the inner most executable is the original executable.

Further packing is the technique of compression and decompression. It combines both the compression and/or decompression code into a single executable file [18]. The decompression code is needed because at the time of runtime the packed executable must have to be unpacked or decompressed.

There exist sophisticated packers which use more advanced technique to gain victory over reverse-engineering, bypass and disable unpackers. One of the techniques is Multi-layer packing. It uses a combination of potentially different packers, and then generates a large number of packed binaries from the same input binary [2]. Multi-layer packing itself could be used as an indication of malware. Another technique is Anti-unpacking. It falls into two major categories: passive and active. These techniques are designed to make it difficult to uncover the logic of an unpacker. Passive anti-unpacking techniques are intended to make disassembly harder, which in turns makes it difficult to identify and reverse the unpacking algorithm. Active techniques are intended to protect the running binary against having the fully unpacked image intercepted and extracted. It can be further classified into three subcategories: Anti-dumping, anti-debugging, and anti-emulating. Enigma and Themida are some of the packers which promote their use of all of these techniques.

In this paper, pattern recognition techniques are applied for detection of packed malware binaries. The proposed framework works in two phases, in the first phase it classifies the packed and non-packed executables. Once an executable is classified as packed, the second phase of classification finds packed benign or packed malware executable. The UPX packer is used in the proposed frame work and has been able to achieve more than 99.9% accuracy in the first phase of classification and 95% accuracy in the second phase of classification.

In this paper, section 2 includes the related work. Section 3 provides a brief description of UPX packer. The proposed model is described in section 4. Experiments are included in section 5 and in its subsection. Section 6 and 7 include the comparison part and the conclusion respectively.

2. RELATED WORK

In paper [3], the approach captures an intrinsic nature of hidden code execution where the original code should be present in memory. The code is executed at some point at run-time. Hence, this approach monitors the runtime execution of the code. In paper [2], the Symantec Researchers develop software called Justin, is the abbreviation for Just-In-Time AV scanning. This approach basically uses some heuristics e.g. dirty page execution, stack pointer check and command-line argument access etc. to identify the end of unpacking of an executable at the time of execution. As soon as it detects the unpacking it invokes AV scanning against the process image at that time. By this way it detects the ingoing malicious activity. In paper [4], OmniUnpack monitors execution of a program in real-time to detect packing of an executable. After the executable gets unpacked it gives the malicious payload directly to the detector software. PolyUnpack [5] uses the technique of comparing the memory image with the static code model. All the above mentioned approaches are basically dealing with the runtime analysis of malware.

In paper [11] the author uses reverse engineering technique to determine the functioning of NsPack packer. They analyze the implementation and the unpacking technique of the respective packer using one of the famous reverse engineering tool OllyDbg.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SecurIT'12, August 17-19, 2012, Kollam, Kerala, India
Copyright 2012 ACM 978-1-4503-1822-8/12/08... $15.00

Paper [7], presents a method for detecting malware in which it presents the correlation between the semantics of the malware and its API calls. In this paper a base signature for an entire malware class is constructed rather than for a single specimen of malware. The signature is capable of detecting variants that belong to that class even if it is unknown and advanced.

Paper [17] describes the importance of Reverse engineering tool to enhance malware detection. These tools help analyzing the logic flow and internal data structures of an executable.

In [18], an accurate and real time PE-Miner framework is reported. This framework automatically extracts distinguishing features from PE files to detect zero-day malware. According to [20] Reverse engineering has become an important approach. Thus RE helps to analyze a program's logic flow and internal data structures, such as system call functions. This paper concentrates on static unpacking using some of the famous unpacker tools and plug-in.

In [8], the authors provide a tool viz. Bintropy that establishes entropy formula to calculate the amount of statistical variation of bytes in a data stream. It divides the PE file into blocks of 256 bytes and computes the entropy of each block. The authors use simple statistical inference to compute a 99.99% confidence interval, on the value of the average block entropy and maximum block entropy for a given dataset of packed binaries. They use threshold values to classify PE binaries. If a PE file is found to have average and maximum entropy above the respective threshold, it is classified as packed binary.

The approach in paper [6], deals with detection of packer families. Once an executable is detected as packed, it decides to which family of packer the packed executable belongs to. Here, sliding window randomness test with trunk pruning method is used based on *n*-gram analysis.

In paper [12], Kolter describes data mining and machine learning technique to classify malicious executables. They use *n*-gram analysis to distinguish between viruses and benign executables, but not between packed and non-packed.

In paper [1], the authors propose a pattern recognition technique to classify packed and non-packed binaries. Once classified the non-packed binaries are directly sent to the anti-virus program. The detected packed binaries are first sent to Universal Unpacker to get them unpacked. It is easy for any traditional malware detector to identify an unpacked binary whether it is malware or benign.

Paper [21], the model proposed by the authors, works in two phases. In the first phase, it extracts various features of portable executables and in the second phase it analyses the extracted features. It comes up with set of features, which can be used to identify whether a given binary is packed or not by UPX Packer.

3. BRIEF DESCRIPTION OF UPX PACKER

It has been proven by many researchers that 80% of the malwares are packed. The most popular packer is the UPX packer software. Different packers have different packing techniques based on their codes. Just for under-standing purpose a very brief explanation of UPX packer works given next. When a PE file is compressed by UPX, it merges all the sections of the binary into a single section, but it does not include the resource section [2]. As reported in [2], the combined data i.e. the code and the data section is compressed into a single section of the resulting packed binary and stored in section UPX1.

The figure 1 is a pictorial representation of how an executable looks like after packed by UPX packer. The resulting sections are PE header followed by an empty space. It is called section UPX0 which reserves the address range. The address range is needed when the binary will be unpacked and loaded into memory at the time of runtime. The next section is the packed data which is followed by unpacker code. The combination is called section UPX1. Finally ended up with the resource section i.e. .rscr. The entry point in the PE header will be changed to point directly to the Unpacker Code. If there is no resource section in the input binary the resulting UPX binary will not have .rsrc section. If the original binary had a resource section which contains some essential dlls (e.g.: kernel32.dll), the resulting UPX binary will also have a resource section [2].

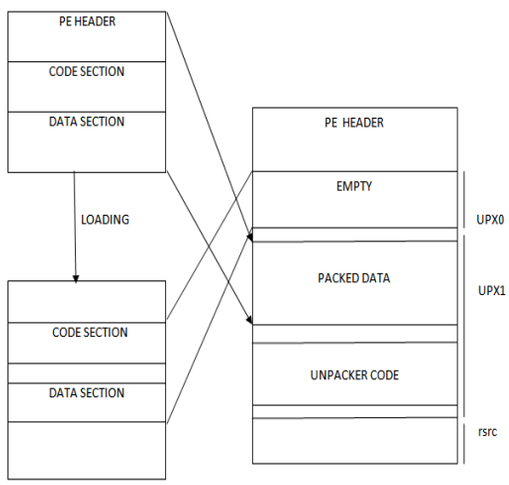

Figure 1. Functioning of UPX.

To visualize dependency we can use tool like dependency walker [18] etc. The first two sections are set to read/write/execute, regardless of what they were before, and are not changed at run time. Therefore UPX is NX compatible. The third section is simply set to read/write, since no execution should happen within that section. After unpacking, the header of the resulting binary is write-enabled by UPX. It writes protects the header again and changes the first two sections of the section table to read-only. This ensures compatibility with some application programs. This type of application programs check in-memory section table, instead of the actual section attributes. This is because these sections are supposed to be non-writable.

4. THE PROPOSED MODEL

The objective of the proposed model is to detect packed malwares vide classification model. Non-packed malicious executables could easily be detected by AV or similar kind of programs. However, it is hard to detect a malicious executable when it is in packed. Therefore, it is essential to detect an executable; whether it is packed or not before detecting if it is a malicious one. The Figure 2 shows classification model for the detection of packed malware and packed benign executables.

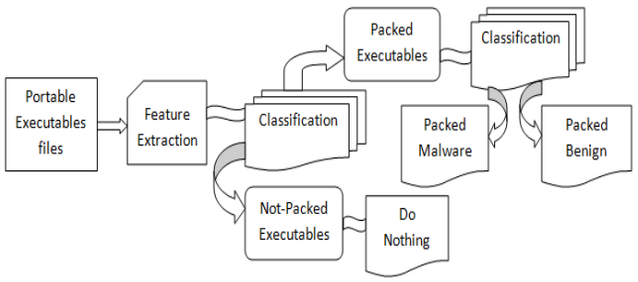

Figure 2. The Proposed Model.

The classification model works in two phases. In the first phase, it classifies the Packed and non-packed executables. If the executables are packed, we perform the second phase of classification. It classifies the input packed executables as packed malware or packed benign executables respectively.

Windows portable executable file has a large number of features. The features of all the executables are same although the differences lie in terms of values only. We extracted the features of benign as well as packed portable executable files as many as possible. The difference between the features of packed and non-packed executable files is obvious. After a number of observations, we have selected the best set of features among them. The results are shown in the following experimental section.

5. EXPERIMENTS AND RESULTS

We collected significant number of malwares that were downloaded from http://offensivecomputing.net/ and the benign executables were captured from a newly installed fresh windows XP SP2 machine. It was a total number of 4075 portable executable files. Among them 2954 were malicious programs and 1121 were benign executables. Initial experiments were performed to get some fundamental knowledge on benign and packer executables. We dump these files using Dumpbin [16] software and extracted some of the features. It is a GUI version of Microsoft's Dumpbin command-line tool and is freely available.

After performing the initial experiments, UPX packer is used to pack collected executable samples to carry forward the experiments with proposed classification model. Further these files were dumped and differences were again analyzed by extracting the same features using Dumpbin.

We developed a Portable Executable feature extraction program in C language. Perl scripting language is partially used. For classification, we used the well known data mining tool Weka. It is a popular suite of machine learning software written in Java. It was developed at the University of Waikato, New Zealand. Weka is free software available under the GNU General Public License [19]. The pattern recognition algorithms considered for classification are contained in Weka data-mining tool.

5.1 FIRST PHASE: DETECTION OF PACKED AND NON-PACKED EXECUTABLES

After several observations we came up with 2 features for the first phase of classification. The feature vector consisting of these 2 features was the best according to our analysis. These features are as follows:

Size of Headers (SOH): UPX packer wraps the whole executable into the packed data along with the unpacker code in UPX1. We know that size of the header contains size of the PE Header and the section (object) table. That is why the size of header of the resultant PE after being packed by UPX is generally greater than or sometimes equal to the size of executable which is not packed.

Size of Raw Data (SORD): UPX packers change the RAWSIZE of each packed section to zero. The size in memory remains unchanged, because the program still has to execute normally and to be unpacked at its original location. If the RAWSIZE is null, it means the section is non-existent on disk.

5.1.1 Classification

Following algorithms from the data mining tool Weka 3.6.4 have been applied for our classification model. Many classification algorithms are available in the Weka tool itself; among them we used 9 classification algorithms for our first phase. We applied 10 fold cross validation technique for the classification purpose. It breaks the data into 10 sets of size n/10 which in turn train the data on 9 datasets and test on 1. This process repeats 10 times and takes a mean accuracy.

The Weighted Average, ROC curve and the detailed accuracy list are shown in Table 1.

Table 1. Weighted Average

Algorithms	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Bayesnet	0.999	0.001	0.999	0.999	0.999	1
LibSVM	0.999	0.001	0.999	0.999	0.999	0.999
IBK	1	0	1	1	1	1
AdaBoostM1	0.999	0.001	0.999	0.999	0.999	0.999
Decision Table	0.999	0.001	0.999	0.999	0.999	0.999
J48	1	0	1	1	1	1
Random Forest	0.999	0.001	0.999	0.999	0.999	0.999
Random Tree	0.999	0.001	0.999	0.999	0.999	0.999

To define the formulas for TP rate, FP rate, Precision, Recall and F-measure, let us assume a test set which consists of n records, where $n=P+N$. P is the number of positive records and N is the negative records respectively [6].

$$TP\ rate = \frac{TP}{P} = \frac{TP}{TP+FN} \text{ and } FP\ rate = \frac{FP}{N} = \frac{FP}{FP+TN}$$

$$Precision = \frac{TP}{TP+FP} \text{ and } Recall = \frac{TP}{TP+FN}$$

$$F-measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

Where, TP represents the correctly identified number of true positive records that belong to a specific class [6].

FP represents the false positives. As the name implies these are the numbers of negative records which are incorrectly identified to belong to a class, to which it does not belong indeed.

TN refers to true negative records which are correctly identified as other classes.

FN represents the false negatives. It refers that the number of positive records which are incorrectly identified to belong to a class, to which it does not belong indeed.

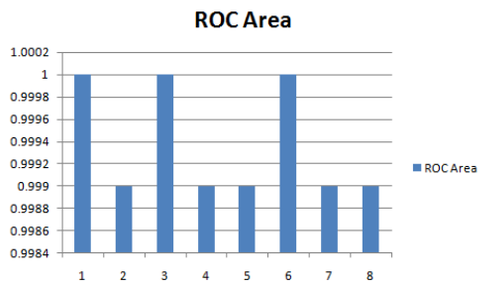

Figure 3. ROC Area.

The formula for accuracy is given below. Accuracy is the percentage of test set records that are correctly identified by the classifier [6]. That is,

$$Accuracy = \frac{TP + TN}{n} = \frac{TP + TN}{P + N}$$

Here, *P* and *N* are the number of positive records and negative records respectively.

Table 2. Accuracy List

Algorithms	Accuracy
Bayesnet	99.9214%
LibSVM	99.9214%
IBK	100%
AdaBoostM1	99.9214%
Decision Table	99.9214%
J48	100%
Random Forest	99.9214%
Random Tree	99.9214%

5.2 SECOND PHASE: DETECTION OF PACKED MALWARE AND PACKED BENIGN EXECUTABLES

In first phase of classification, we achieve 100% of accuracy for IBK and J48 algorithm and also approximately 100% of accuracy achieved for Bayesnet, LibSVM, AdaboostM1, Decision Table, Random Forest and Random Tree algorithm. Taking 100% of accuracy into consideration we proceed for second phase of classification by classifying packed executables into packed malware and packed benign. The feature set for this phase is given below.

Size of Code (SOC): Size of code is basically the sum of the sizes of all sections present in the executable. Our observation says that the SOC of malware is generally greater than the SOC of benign executables. This is because normally malware makes use of GetProcAddress, GetTickCount, socket and sendto etc. system calls to make its functioning easier. Malware which do not make use of such system functions would likely to be harder to write and result in a much larger code size. Hence even after packing, the size of these executables remains larger although its size gets reduced to some extent.

Size of Uninitialized Data (SOUID): We observe that the SOUID of packed malware is mostly greater than the size of the packed benign executables. Compressed sections usually have the UNINITIALIZED DATA flag enabled. It is because of the null size on disk. The loader takes the compressed sections and unpacks them to their original memory locations.

Size of Stack Reserve (SOSR): This feature actually depends on the memory architecture. We work on a windows xp sp2 platform. Therefore, in case of malware the maximum size of stack was Ox100000. The malware uses a huge number of stack space so that it does not crash out of low memory size.

Non Standard Section (NSS): Sections rather than the standard sections that reside in an executables are called non-standard sections. The standard sections names that are reported in [22] are related to Microsoft compilers. However, the same section names are commonly found in PE files generated using other compilers, e.g. .code section, .data section etc. While analyzing the packed executables by UPX packer, we observed that there are always two and two or three non-standard sections in case of packed benign and packed malware respectively.

5.2.1 Classification

We used the Weka 3.6.4 and 10 fold cross validation for the second phase of experiment. The combination of the four features obtained more than 95% of accuracy which is quite satisfactory. The list of weighted Average is shown in Table 3.

Table 3. Weighted Average

Algorithms	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Bayesnet	0.954	0.05	0.954	0.954	0.954	0.973
AdaBoostM1	0.952	0.052	0.952	0.952	0.952	0.969
J48	0.954	0.05	0.954	0.954	0.954	0.946
Logistic	0.952	0.053	0.952	0.952	0.952	0.967
MLP	0.952	0.053	0.952	0.952	0.952	0.961

Refer to figure 4 for the ROC curve and Table 4 for the Accuracy list respectively. From the accuracy list it is clearly visible that 95% accuracy is obtained in this phase of classification.

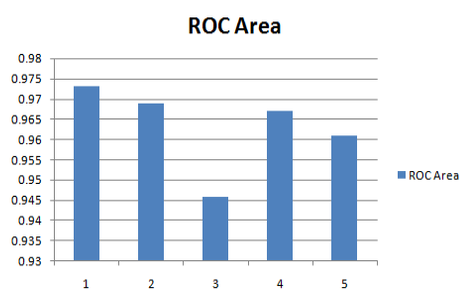

Figure 4. ROC Area.

Table 4. Accuracy List

Algorithms	Accuracy
Bayesnet	95.4023%
AdaBoostM1	95.2107%
J48	95.4023%
Logistic	95.2107%
MLP	95.2107%

6. COMPARISON

In this section, we give brief comparison between the work carried out in this paper and in paper [21]. The model that had been proposed in [21], worked in two phases. In the first phase, it extracts various features from a number of portable executable files and in the second phase, the model analyses the extracted features to figure out the best feature set, which was used to identify whether a given executables was packed or not by UPX Packer. Here, a maximum number of four features were taken for granted as the final output.

In this paper, we propose a model that comprised of two phases of classifications. In the first phase by means of two features as stated above, we have been able to achieve more than 99.9% accuracy in the first phase of classification. That explicitly classifies the packed and non-packed executables by UPX packer. The second phase of classification led to the classification of packed malware and packed benign executables by UPX packer. Considering a maximum number of four features, we have been able to achieve more than 95% accuracy in this phase of classification. For the first phase of the proposed framework, we have used only two features of the number of features reported in the second phase in [21].

7. CONCLUSION

The proposed framework first classified packed and non-packed executables. From the classified packed executable, it is further classified as the packed malware and packed benign executables. Therefore two phases of classifications are accomplished. In the first phase of classification, an average of more than 99.9% of accuracy and in the second phase an average accuracy of more than 95% of detection mechanism considering packed executable with UPX packer are achieved. This implies proper selection of feature sets in two phases.

8. ACKNOWLEDGMENTS

Authors acknowledge Dr. Neminath Hubballi for his contribution during discussion related to this work.

9. REFERENCES

[1] Roberto Perdisci, Andrea Lanzi,Wenke Lee. Classification of Packed Executables for Accurate Computer Virus Detection. Journal Pattern Recognition Letters archive Volume 29 Issue 14, October, 2008. Pages 1941-1946.

[2] Fanglu Guo, Peter Ferrie, and Tzi-cker Chiueh. A Study of the Packer Problem and Its Solutions. Proceeding RAID '08 Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection Pages 98 – 115, 2008.

[3] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. Renovo: A Hidden Code Extractor for Packed Executables, Proceeding WORM '07 Proceedings of the 2007 ACM workshop on Recurring malcode Pages 46-53.

[4] Lorenzo Martignoni, Mihai Christodorescu, Somesh Jha. OmniUnpack: Fast, Generic, and Safe Unpacking of Malware. Computer Security Applications Conference, 2007. ACSAC, 2007.

[5] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds,Wenke Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. Computer Security Applications Conference, 2006.

[6] Li Sun, Steven Versteeg, Serdar Boztas and Trevor Yann. Pattern Recognition Techniques for the Classification of Malware Packers. R. Steinfeld and P. Hawkes (Eds.):ACISP 2010, LNCS 6168, pp. 370390, 2010.

[7] V. Sai Sathyanarayan, Pankaj Kohli, and Bezawada Bruhadeshwar. Signature Generation and Detection of Malware Families. Y. Mu,W. Susilo, and J. Seberry (Eds.):ACISP 2008, LNCS 5107, pp. 336349, 2008.

[8] ROBERT LYD, Sparta, JAMES, HAMROCK, McDonald, Bradley. Using entropy analysis to find encrypted and packed malware. Journal IEEE Security and Privacy archive Volume 5 Issue 2, March 2007 Pages 40-45.

[9] M. Morgenstern and T. Brosch. Runtime packers: The hidden problem?, Blackhat USA 2005.

[10] M. Zubair Shaq, S. Momina Tabish, Fauzan Mirza, Muddassar Farooq. PE-Miner:Mining Structural Information to Detect Malicious Executables in Realtime. RAID 2009, LNCS 5758, pp. 121–141, 2009.

[11] Craig S Wright. Packer Analysis Report - Debugging and unpacking the NsPack 3.4 and 3.7 packer. 7th DOD/NBSComputers and Security Conference, volume 6, pages 22--35, September 1987.

[12] J. Zico Kolter , Marcus A. Maloof. Learning to Detect and Classify Malicious Executables in the Wild. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2004.

[13] Matt Pietrek. Peering Inside the PE: A Tour of the Win32 Portable Executable File Format, Article: http://msdn.microsoft.com/en-us/library/ms809762, March 1994.

[14] Goppit. Portable Executable File Format - A Reverse Engineer View, Code Breakers Journal - Aug 15, 2005.

[15] Michael Howard. Revealing Packed malware. Published by the IEEE ComPuter soCiety, 1540-7993/07/$25.00 2007.

[16] http://www.cheztabor.com/dumpbinGUI/

[17] http://www.dependencywalker.com

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

[19] http://www.cs.waikato.ac.nz/ml/weka/index_downloading.html

[20] Andrew Walenstein, Daniel J. Hefner and Jeffery Wichers. Header Information in Malware Families and Impact on Automated Classifiers. 2010 5th International Conference on Malicious and Unwanted Software, 2010.

[21] Dhruwajita Devi and Sukumar Nandi. PE File Features in Detection of Packed Executables. International Journal of Computer Theory and Engineering, Vol. 4, No. 3, June 2012.