A Fine-Grained Classification Approach for the Packed Malicious Code

Shanqing Guo[1], Shuangshuang Li[1], Yan Yu[2], Anlei Hu[3], and Tao Ban[4]

[1] School of Computer Science and Technology, Shandong University, China
[2] School of Computer Science, Nanjing University of Science & Technology, China
[3] DNSLAB, China Internet Network Information Center, Beijing, China
[4] National Institute of Information and Communications Technology, Japan
bantao@nict.go.jp, guoshanqing@sdu.edu.cn,
huanlei@cnnic.cn, yuyan@mail.njust.edu.cn

Abstract. Executable packing is the most common technique to evade detection by anti-virus software.Many signature-based unpackers have been presented to uncover hidden viruses,which make the signature-based anti-virus software successfully detect the packed malicious code. However,these universal unpackers are computationally expensive and scanning large collections of executables may take several hours or even days.In order to improve the computational efficiency, Machine learning techniques have recently been proven effective in solving the focused issues,but up to now,no methods can show what packing method has been used in it.In this paper we proposed a fine-grained detection method to detect whether a malicious code has been packed and which method is been used to.This method firstly extract a hex-string from the target object file and then apply a String-Kernel-Based SVM Classifier to implement the fast detection of packed malicious code.We also show that our system achieves very high detection accuracy of packed executables, so that only executables detected as packed will be sent to an universal unpacker, thus saving a significant amount of processing time.

Keywords: Computer Security, String-Based Kernel,Support Vector Machine, Packed Malicious Code,Computer Virus Detection.

1 Introduction

In these days,honeypot systems,operated in malware analysis groups,encounter with numerous of malware samples day by day.Unfortunately,large portions of such malware samples,at around 50%,are identified to be packed with PEiD tool [1]. Applying packing in malware can degrade the effectiveness of signature-based AV scanners,for it has no choice but to create a separate signature. Therefore, It is very significant to develop an efficient automated approach to identify packers and uncover hidden codes from so huge volumes of malwares. Some universal unpackers [2, 3] have been developed, and these tools are able to detect and extract (part of) the original code from the encrypted code without specific knowledge about the encryption algorithm. However, these universal unpackers introduce a high computational overhead, and the processing time may vary

T.W. Chim and T.H. Yuen (Eds.): ICICS 2012, LNCS 7618, pp. 497–504, 2012.

from tens of seconds to several minutes per executable. For example, the average time it takes to unpack a packed virus using the Renovo [3] unpacker is around 40 seconds. For this reason, we need to do some research about how to effectively distinguish the packed malicious code from the unpacked malicious code.

Machine learning techniques have recently been proven effective in solving the focused issues in this paper [4–6],but most of the existed works focus on how to detect whether a malware been packed. In this paper,based on the string-based kernel function, we proposed a fast fine-grained detection method to accurately distinguish whether a malware is packed, and if packed, what kind of packed toolkit has been used? After the packed toolkit has been identified, it can be directly sent to a special unpacking engineer to implement the hidden code extraction. Therefore, our classification system helps in improving virus detection while saving a significant amount of processing time.

The remainder of the paper is organized as follows. In Section 2 we present an overview of the related work. Section 3 introduce the Support Vector Machine(SVM) and a simple string-based kernel function.In Section 4 we briefly discuss the Portable Executable (PE) file format, and describe the features used for classifying PE executables. We then present and discuss the experimental results in Section 5,and summarize our work in Section 6.

2 Related Works

Although distinguishing between packed and non-packed executables is undecidable [3], several detecting methods have been proposed. The firstly proposed methods is the signature-based methods,and it also has developed many tools,like PEiD [1]. Signature-based detectors are fast and have relatively low false positives, but malware authors soon discovered that it was sufficient to use a slightly different packing algorithm each time to frustrate this kinds of regular expression matching algorithm, which make the signature-based detectors produce a high number of false negatives alerts.

The other related works is dynamic unpackers. Dynamic unpackers execute and monitor a program in memory, and detect attempts of executing dynamically decrypted code. To date, some dynamic unpackers have been proposed: Omniunpack[7], PolyUnpack[2], which try to monitors the execution of applications in memory and detects whether a PE file is packed. One important drawback of the dynamic unpackers is the performance overhead they introduce. Unfortunately, this performance overhead makes it impossible to install and run them as real-time systems on end-user machines. Therefore, [4, 8] divides the PE file into blocks of 256 bytes , and detect whether a file be packed by only computing the entropy of each block, the average, and the maximum block entropy. [9] do not limit the analysis to the PE file entropy and introduce and motivate the use of additional features that help in distinguishing between packed and non-packed executables.

To the best of our knowledge, the closest works to ours are [4, 8]. There exist many difference between [4, 8] and our works.On the one hand, our proposed

method is a fine-grained classification method,which can identify which packed tools has been used by a packed malware, but [4, 8] only can detect whether a malware is packed. On the other hand, we can show that our classification approach also has a low average processing time with very low false positive and false negative rates.

3 String-Kernel-Based SVM Classifier

Support vector machines (SVMs) are a set of related supervised learning methods that are used for classification and regression analysis. SVM maps the data points into a high dimensional feature space, where a hyperplane or set of hyperplanes was constructed to implement the task of classification. According to the form of the error function, SVM models can be classified into distinct groups. Here, We only focus on the standard soft-margin SVM problem (C-SVM). The main task in C-SVM is to solve the following quadratic optimization problem with respect to α:

$$\begin{array}{l} \min_\alpha \ \frac{1}{2}\alpha^T Q\alpha - 1^T\alpha \\ Subject\ to\ y^T\alpha > 0 \quad 0 \le \alpha_i \le C, i = 1, \cdots, m \end{array} \tag{1}$$

Where Q is the N*N positive semi-definite kernel matrix, $Q_{ij} = y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) = (x_i)^T(x_j)$ is the kernel function; and 1 is a vector of all ones.C-SVM predict the class label of a new data point x to be classified by the following decision function: $f(x) = sign(\sum_{i=1}^{m} y_i \alpha_i K(x_i, x) + b)$ Where b is a bias constant.

Traditional regular kernels for SVM work merely on numerical data, which is unsuitable for internet security where huge amount of string data is presented. Towards extending SVM for string data processing, many string-based kernel were proposed.In our implementation, we use $K = e^{\lambda d(i,j)}$ as the kernel function of the SVM for getting better results,here, $d(i, j)$ is the Levenshtein (or edit) distance [10]. Analog to the other substring kernel, the computational complexity of Levenshtein Distance is $O(\mid s \parallel t \mid)$. In the case that s and t have the same length, the complexity is $O(n^2)$.

4 JUMPS: JUst-in-tiMe Packer Scanning

4.1 Our Classification System

Figure 1 illustrates the processing procedure of our proposed system. Once a PE executable is received, our classification system performs a static analysis of the PE file and extract the first N bytes from its code section. After first N bytes were translated into a pattern vector, the obtained pattern vector representation of the PE executable is sent to the SVM classifier. If the executable is classified as packed, it will be sent to the universal 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

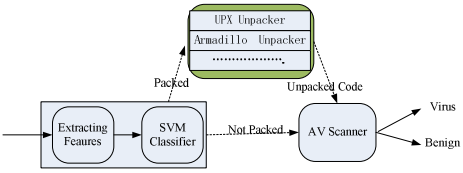

Fig. 1. Our Classifier System

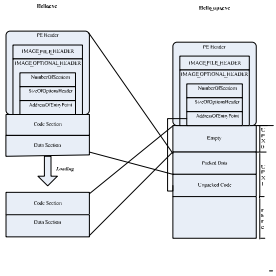

Fig. 2. The layout of the packed binary Hello.exe by UPS packer

scanner. It is worth noting that the PE file classifier may erroneously label a non-packed executable as packed. In this case the universal 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 universal 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.Figure 1 also shows that our 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 universal unpacker.

4.2 Features Extraction

Lets start with a simple example,named UPX, which arguably is among the most straightforward packers in use today. Fig.2 shows how UPX packs an example program Hello.exe. When UPX compresses a PE binary, it begins by merging all of its sections into a single section, with the exception of the resource section. The combined data is then compressed into a single section of the resulting packed binary. In Fig2, the code section and data section of hello.exe is compressed and stored in the Packed Data area of section UPX1 of the resulting binary Hello upx.exe.

The resulting binary Hello upx.exe contains three sections. The first section UPX0 is entirely virtual - it contains no physical data and is simply a place-holder.It reserves the address range when Hello.exe is loaded to memory. The second section contains the Packed Data, followed immediately by the Unpacker Code. The entry point in the PE header of Hello upx.exe is changed to point directly to the Unpacker Code. The third section contains the resource data, if the original binary had a resource section, and a partial import table, which contains some essential imports from kernel32.dll as well as one imported function from every DLL used by the original binary.

From this example, we can get a fact that the packing and unpacking process is simple. The packer modifies the entry point of the original file and inserting an unpacking routine.When the compressed binary is launched, the unpacker code is firstly executed to decompress or decrypt the original code and data into the UPX0 section, actually allocated in memory and performs some tasks normally carried out by the PE loader, such as import resolution. lastly, it transfers control to the original code, for example by jumping to the so-called Original Entry Point (OEP).

Our method is basically to leverage general behavior of unpacker codes dangled in packed binaries.According to the analysis of more than thirteen different packing techniques, there exists a one - to - one correspondence between the unpacker and the packer,so the unpacker can be used to uniquely identify the packer. By analyzing the execution of a packed malicious code, it can be known that the unpacker will be firstly executed, so under the help of the AddresssOfEntrypoint in the PE Header, we can locate code section including unpacker.Here we extract it and transform it into a Hex-string as the input features of the string-kernel-based SVM classifier.

Although the packer can be very tricky by adding padding, indirect jumps, and other obfuscation methods to the PE file, it can frustrate our approach easily. In practice, we try to change the value of the N from 150 to 50, the accuracy of the classification seldom fluctuates. According to results of the experiments, we can conclude that the unpacking features can be represented by just a few bytes located by the AddressOfEntryPoint in the PE Header.

5 Evaluation

AS we know, the effectiveness of a classification system based on string subsequence kernel can be controlled by the free parameters, "length of a subsequence" and "weight decay parameter". In the experiments,we set decay parameter for subsequence kernel 0.9 and substring length for subsequence kernel 4.

5.1 Data Sets

We performed experiments on 2180 executables in PE format, including 1280 packed viruses collected from the Malfease Project dataset [11], and 900 non-packed benign executables obtained from a clean installation of Windows XP Home. Table 1 summarizes the proportion of normal executable and packed malware.

5.2 Experimental Procedures

SVM classifier is a supervised methods,therefore,we first applied Peid tools to some of the executables in our collection and obtained a labeled dataset, which will be used to train and test the performance of our proposed system. Next, we begin to train and test our system. The training and testing samples were

Table 1. Proportion of Normal Executable and Packed Malware

Packer Name	Sample size
Unpacking Executables	900
Armadillo	600
Aspack	180
Bobsoft	180
Nspack	40
PECompact	20
Petite	60
UPX	160
Upack	20
WinUpack	20

randomly drawn from the original data sets, and the number of instances in all the training data subsets were restricted with 3% percent to 15% percent for the unpacked and packed instances.In order to estimate the generalized accuracy, a 10-fold cross-validation procedure was repeated 5 times. In each of the cross-validation iteration, the training set was randomly partitioned into 10 disjoint instance subsets. Each subset was utilized once in a test set and nine times in a training set.

5.3 Evaluation Metrics

In the context of classification tasks, the terms true positives, true negatives, false positives and false negatives are used to compare the given classification of an item (the class label assigned to the item by a classifier) with the desired correct classification (the class the item actually belongs to). And for a class X.Based on these, we calculate four metrics that are commonly used in machine learning literature to measure per class performance,which is the Accuracy,Precision,TPR(True Positive Rate) and FPR(False Positive Rate)[12].

5.4 Performance

Firstly, we evaluate wether the proposed system can effectively identify the single packed executables and un-packed executables. In this expirement, each time we trained and test the proposed system with a dataset including just one type of packed malware and un-packed files. Here, we run SVM nine times on each kind of packed malware and the same un-packed files as training and test dataset with the 5% and 10% overall dataset as training samples, respectively. Then the average results are calculated. Fig.3 and Fig.4 report the performance of Jumps methods averaged in terms of TPR, FPR, Precision, and Accuracy. As represented in Fig.3 and Fig.4 , the accuracy of the proposed system is not less than 98%, and false positive is under 0.1% even with the 5% overall dataset as training samples. Therefore,it is considered that the proposed system perform well on the identification of single type of packer.

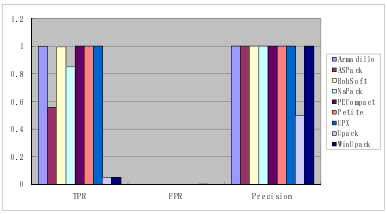

Fig. 3. Performance of JUMPS over 5% overall dataset as training samples

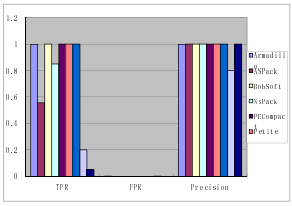

Fig. 4. Performance of JUMPS over 10% overall dataset as training samples

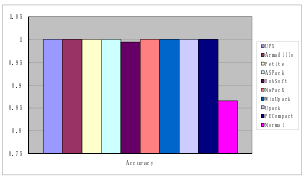

Fig. 5. Performance of JUMPS in detecting multi-packer

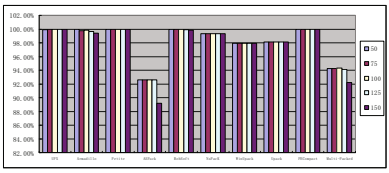

Fig. 6. Performance of JUMPS with different lengths of the features

In the above experiments, we take N=100 as default to test the performance of our approach. In order to test its robustness, we also evaluate its accuracy across a range from N=50 to N=150. its results were shown in Fig N.From Fig. 6, we can find that the accuracy of the classification seldom fluctuates.

6 Conclusions

In this paper we do not focus on the improvements in malicious code detection accuracy achieved after unpacking, Instead, we focus on the accuracy and computational cost related to the classification of packed executables into the packed and non-packed,inlcuding the identification issues of different packers . we first analysis its feature of the typical packed malicious code and described how to extract discriminant features from executable files in Portable Executable format. Then we developed the packer recognition system based on string-kernel-based SVM. The experimental results manifest that Jumps's effectiveness at detecting packed malware is excellent.

In future, we would like to evaluate our scheme on a larger dataset of packed and unpacked malicious executables for further improving the detection accuracy,and also attempt to explore the inflection point of lengths of the extracted code.Moreover,we also plan to improve our approach and applied it into the malicious code with multi-packer in the same samples.

Acknowledgement. Supported by Specialized Research Fund for the Doctoral Program of Higher Education (20090131120009), Outstanding Young Scientists

Foundation Grant of Shandong Province (BS2009DX018), the Key Science-Technology Project of Shandong Province(2010GGX10117), Open Research Fund from Key Laboratory of Computer Network and Information Integration In Southeast University, Ministry of Education,China (K93-9-2010-05), DNSLAB(K201206007).

References

1. http://www.peid.info/
2. Royal, P., Halpin, M., Dagon, D., Edmonds, R., Lee, W.: Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: Proceedings of the 22nd Annual Computer Security Applications Conference, pp. 289–300 (2006)
3. Kang, M.G., Poosankam, P., Yin, H.: Renovo: a hidden code extractor for packed executables. In: Proceedings of the 2007 ACM Workshop on Recurring Malcode, WORM 2007, pp. 46–53 (2007)
4. Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Security and Privacy 5, 40–45 (2007)
5. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. 7, 2721–2744 (2006)
6. Perdisci, R., Gu, G., Lee, W.: Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In: Proceedings of the Sixth International Conference on Data Mining, ICDM 2006, pp. 488–498 (2006)
7. Martignoni, L., Christodorescu, M., Jha, S.: OmniUnpack: Fast, Generic, and Safe Unpacking of Malware. In: Proceedings of the 21st Annual Computer Security Applications Conference, ACSAC 2007, Miami Beach, Florida, USA (December 2007)
8. Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Security and Privacy 5, 40–45 (2007)
9. Perdisci, R., Lanzi, A., Lee, W.: Classification of packed executables for accurate computer virus detection. Pattern Recogn. Lett. 29, 1941–1946 (2008)
10. Charras, C., Lecroq, T.: Sequence comparison. Laboratoire d'Informatique de Rouen et Atelier Biologie Informatique Statistique Socio-Linguistique. Universit'e de Rouen, France (1998)
11. http://malfease.oarci.net/
12. http://en.wikipedia.org/wiki/Receiver_operating_characteristic/