J Comput Virol Hack Tech (2016) 12:101–111
DOI 10.1007/s11416-015-0249-8

ORIGINAL PAPER

Packer identification using Byte plot and Markov plot

Kesav Kancherla[1] · **John Donahue**[2] · **Srinivas Mukkamala**[1]

Received: 9 September 2014 / Accepted: 10 July 2015 / Published online: 14 September 2015

Abstract Malware is one of the major concerns in computer security. The availability of easy to use malware toolkits and internet popularity has led to the increase in number of malware attacks. Currently signature based malware detection techniques are widely used. However, malware authors use packing techniques to create new variants of existing malwares which defeat signature based malware detection. So, it is very important to identify packed malware and unpack it before analysis. Dynamic unpacking runs the packed executable and provides an unpacked version based on the system. This technique requires dedicated hardware and is computationally expensive. As each individual packer uses its own unpacking algorithm it is important to have a prior knowledge about the packer used, in order to assist in reverse engineering. In this paper, we propose an efficient framework for packer identification problem using Byte plot and Markov plot. First packed malware is converted to Byte plot and Markov plot. Later Gabor and wavelet based features are extracted from Byte plot and Markov plot. We used SVMs (Support Vector Machine) in our analysis. We performed our experiments on nine different packers and we obtained about 95 % accuracy for nine of the packers. Our results show features extracted from Markov plot outperformed features extracted from Byte plot by about 3 %. We compare the performance of Markov plot with PEID (Signature based PE identification tool). Our results show Markov plot produced better accuracy when compared to PEID. We also performed multi class classification using Random Forest and achieved 81 % accuracy using Markov plot based features.

✉ Kesav Kancherla
kancherla@cs.nmt.edu

John Donahue
jdonahue@nmt.edu

Srinivas Mukkamala
srinivas@cs.nmt.edu

[1] Computer Science Institute for Complex Additive and System Analysis, New Mexico Institute of Mining and Technology, Socorro, NM, USA

[2] Computer Science, New Mexico Institute of Mining and Technology, Socorro, NM, USA

1 Introduction

Malware is unwanted software designed to gain unauthorized access, steal information, and disrupt normal operation. Malware includes computer viruses, worms, trojan horses, spyware, adware and rootkits. According to Symantec [1] there is an increase of 93 % in web based attacks due to the proliferation of toolkits. In 2011 Symantec identified about 403 million unique variants of malware which is an increase of 41 % when compared to last year. Symantec blocked a total of over 5.5 billion malware attacks in 2011.

The steps involved [2] in a cyber attack are reconnaissance, intrusion, malware delivery and clean up. The goal of reconnaissance is to identify vulnerabilities in a system or a network by finding credentials, settings, software and its version. This is done usually by using social engineering attacks like phishing and fraudulent websites. Once information is obtained, the next step is to find way to deliver payload. This is done by enumeration techniques like service scanning. The attacker tries to penetrate the network by exploiting these vulnerabilities. Based on the purpose different types of malware payloads are inserted. Malware can be a nuisance (like Adware, Spyware) or controlling (like trojan, rootkits) or destructive (like seek-destroy viruses). The final

step is a clean-up, which is done by deleting logs, deactivating alarms etc.

There are two different approaches for anomaly based detection: static code analysis and dynamic code analysis. In static analysis first the code is sent through a disassembler and control flow of the executable is analyzed to detect malicious patterns. Static analysis is computationally efficient but performance is affected by packed or encrypted malware. In dynamic analysis the code is executed in a virtual environment and behavior of the executable is analyzed. Even though dynamic analysis produces better results its performance is affected by packed or encrypted malware. However, this approach is computationally expensive and malware behavior depends on the environment it is being executed.

Most of the current anti-virus solutions use signature based approaches for malware detection. The benefits of signature based approaches include low computational complexity and low false positive rate. However these techniques can easily be evaded by simple code obfuscation techniques and use of different packing techniques.

Instead of directly obfuscating malware code, current attackers rely heavily on packers. A packer is a software program that transforms one binary executable by applying various compression techniques that has a different appearance than the original one. Malware author recursively applies different combination of multiple packers to create a new version each time. As each of these variants will have a different signature than the actual copy, Signature based antivirus solutions have to create a new signature to each of these. This will lead to an exponential increase in the signature size, thus making signature based methods ineffective.

Every packer uses its own unpacking algorithm, so it is important to identify packer used while packing the malware. In this work, we propose a solution to packer identification problem using Byte plot and Markov plots. First each executable was converted to Byte plot or Markov plot and extracts texture based features like Wavelet and Gabor features are extracted. We also compared the performance of features extracted from Byte plot and Markov plot using Support Vector Machines (SVM). We have also performed multi class classification using Random Forest and SVMs. Our results show Markov plot based features performed better when compared to Byte plot based features.

The paper is organized as follows: in Sect. 2 we provide background regarding packers, related work is discussed in Sect. 3, in Sect. 4 we explain our methodology and plot generation, Feature extraction is explained in Sect. 5, in Sect. 6 results obtained using SVMs are provided and finally we provide conclusions in Sect. 7.

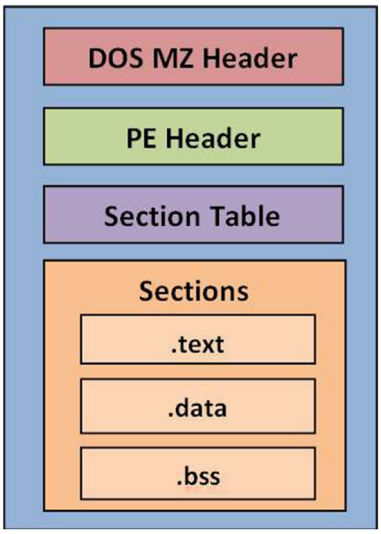

Fig. 1 PE File structure

2 Background

Portable Executable (PE) file format is developed by Microsoft as a common file format which supports all windows versions and on all supported systems. PE consists of MZ header, PE header, section table and sections. The PE Header includes information about machine, number of sections, time date stamp, pointer to symbol table, number of symbols, size of optional header and characteristics.

Section table provides reference to various sections in the PE file and also maintains section permissions for the file. Each of the sections that are maintained in the section table contains information related to how the program runs. Each section is a collection of data in the execution of the program. Section includes .text, .data, .rsrc and .reloc (Fig. 1).

A packer is a software program that compresses an executable file and combines this compressed version with a decompression routine to create a new executable. There are a wide variety of techniques that are used to pack a file. A popular and free packer is the UPX packer (Ultimate Packer for executables). While many packers have proprietary solutions UPX is unique in that the code is completely open source. The UPX packer works by compressing an executable. The compressed executable and the stub code are packed together, so that when the stub code is executed it automatically decompresses and runs the original executable.

2.1 Compression

An executable packed with UPX is an executable itself [3]. The format of the resulting PE file is broken down into three sections. UPX0 is an empty section where the decompressed

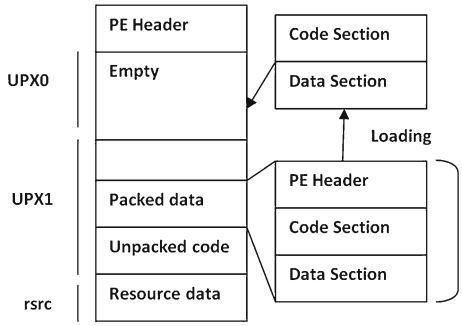

Fig. 2 UPX packer

code will be written. UPX1 is the entry point into the stub portion of the code where the decompression algorithm lives. UPX1 also houses the compressed executable. UPX2 contains all of the imports including Load Library that is needed to restore the imports of the original unpacked executable (Fig. 2).

When the UPX packer is run, the original executable is compressed using an open source version of the NRV [3] (Not Really Vanished) compression library called UCL. The codecs from these libraries include NRV2B, NRV2D and NRV2E. These compression algorithms are chosen for their small footprint, small memory requirements and their fast decompression times. The compressed executable is written to UPX1 along with the decompression code. The entry point of the code is adjusted to point the decompression stub.

2.2 Decompression

When the compressed executable is run, execution starts in the decompression stub. The decompression stub decompresses the original executable into the UPX0 section. Since this packed code was not available to the Windows loader at the time the program was started, the decompression stub must first perform the tasks of the Windows loader before passing execution to the now unpacked code. This mainly involves importing libraries. Now that the original unpacked executable exists in memory, the stub code is free to jump to the UPX0 section where the entry point to the unpacked code lives.

Other packing software employs different compression algorithms and other techniques to optimize speed, obfuscation or usability. One drastically different method is used by the Themida packer. The Themida packer creates a virtual processor within the packed executable. Byte-code can then be created for the virtual processor that can map to the original executable. The makers of Themida, Oreans Technology, call this virtual processor SecureEngine.

The packers used in our test can be broken down into three categories: Compressors, protectors and bundlers.

Table 1 Packer distribution [4]

Packer	Percentage
UPX All_Versions	68.49
NullSoft_PiMP _SFX vna	6.24
NullSoft_NSIS Generic	4.32
FSG V1.0-1.2	3.14
ASPack v2.12	2.94
ASPack vna	2.35
Inno_Setup v2.0.1	2.01
Unknown_33	1.65
UPX V2.9-3.X	1.50
MEW v11SEv1.2	1.12

Compressors aim to reduce the size of a file without attention to protecting the contents of the executable. UPX, Aspack, Pecompact and Winupack are some examples of compressors. UPX is an open source compressor that is the most used of the packers. Aspack is a commercial compressor developed by Aspack Software. Winupack is a free compressor that utilizes UPack and is developed by Dwing. Pecompact is a commercial compressor developed by Bitsum.

Protectors combine encryption schemes along with compression in order to reduce size and prevent the files from being unpacked. Armadillo, Telock and Themida are examples of protectors. Armadillo is a commercial protector distributed by Silicon Realm. Themida is a commercial protector developed by Oreans Technologies. Telock is a free protector developed by TGM. Table 1 gives the distribution of packers with percentage of executable using a particular packer. UPX is widely used packer with 68 %.

Bundlers can package multiple executables and files into a single executable. Installshield is a commercial bundler developed by Flexera. Nearly 75 % of packed executables [4] are reported to be the UPX packer. The Null soft packer is around 11 % [5]. More information about the packers used in this work is given in [6–10].

3 Related work

A common solution [11] to packer problem is unpacking the code before analysis using appropriate unpacker routine. Unpacking is usually performed by running the program and capturing the memory snapshot. Running the program will expose the system to potential attack, thus it is important to run the program in an isolated sandbox environment. However it is very difficult to test malware database with all the available unpackers, given the huge number of packers availability. Thus a prior knowledge about the packer used

will help reduce the computation overhead induced by the unpacking process.

PEiD [12] is a widely used signature based packer identification tool. PEid uses a set of predefined packer signatures to identify best match. One of the signatures for UPX version 2.90 is '60 BE ?? ?? ?? ?? 8D BE 11 C0 01 DB', where each byte is presented as a bi-digitate hexadecimal number and '??' indicates a wild card. PEiD supports three different scanning methods, normal mode scans the specified PE file at its entry point for all designated signatures, deep mode scans the file's entry point containing section and hardcore mode scans the entire file for all the documented signatures. Even though PEiD is simple and explicit; it does not provide any automated tools for signature extraction. It also uses exact string matching, thus its effectiveness depends heavily on signature sensitivity. To solve this issue machine learning based solutions were proposed [13–16].

Dynamic unpacking [17] approaches monitor execution of packed code to extract its actual code. These methods run the executable in an isolated sandbox environment and monitor the memory dump to find actual code. These techniques use different heuristics to determine the exact point where the execution jumps from unpacking routine to actual code. Once it identifies the entry point, memory dump is used to obtain unpacked version of the packed code.

Ollybone [18] is a semi-automatic unpacking technique from IA-32 packed binaries. It is designed to track paged that is re-written and executed by overloading the use/supervisor bit. Thus exploiting the separation of data and instruction translation look aside buffer (TLB) in x86 architecture. Saffron [19] combines OllyBone with Intels PIN to detect control transfers to dynamically created or modified pages and dump memory images at that time.

Omniunpack [20] relies on OllyBone to identify executed pages and invoke Anti Virus (AV) scanning before every dangerous system call. For efficiency it invokes AV scan only when there is control transfer to dynamically modified page and AV scans only pages that are modified since last dangerous system call. However this only works for run time and is not suitable for at-rest file scanning. The use of structural information of the PE executable to determine whether the given sample is packed or if its malicious code is examined in PE-Miner [21] and PE-Probe [22].

Early Machine Learning (ML) related work focused mainly on packer detection [13–15] rather than packer identification. In [16] the authors use easy to extract packer discriminating features that are given input to classifiers like K Nearest Neighbor (KNN), SVM and Naive Bayes. In this work based on the byte frequency histogram, a Huffman tree is built. Later the binary file is divided into equal lengths and each fragment is coded based on Huffman tree. Then encoding lengths of leading N and trailing N are used as features.

In [23] the authors propose the use of Levenshtein Distance (LD) as proximity metric to improve the performance of exact string matching approach. Levenshtein Distance, also known as edit distance [24], is the most widely known string metric that operates between two input strings, returning a score equivalent to the number of edit operations (e.g., insertion, substitution, and deletion) needed in order to transform one input string into another. They used a LD-kernel to make it embeddable with current kernel based methods like SVM.

Visualization is a useful technique widely used in computer security like computer forensics, and network security. However visualization is not widely used for malware analysis. Some work is done in the identification of file types. In [25] Conti et al. introduced visualization of raw binary data of different primitive binary fragments like text etc as images. In [26] Conti et al. extended their previous work to automatically classify different binary fragments using statistical features from Byte plot. Self organizing maps (SOM) are used to visualize and detect malicious code inside an executable in [27]. In [28] Quist and Liebrock developed a visualization framework for reverse engineering. They can identify functional areas and de-obfuscate through a node-link visualization, where nodes represent the address and links represent the state transitions between addresses.

4 Methodology

Digital Image Processing is the application of signal processing techniques to digital images. A digital image is considered as a 2-dimensional signal with intensity as amplitude of the signal. Color images are represented by three 2-dimensional signals with each signal representing one channel. RGB color model is a additive model in which the 'Red', 'Green' and 'Blue' are added to obtain a broad array of colors. Typical image processing operations include image segmentation, image transformation, image restoration etc. These operations are performed by the use of different filters and transformations.

Figure 3 gives the basic outline of our approach. First the binary executable was converted to an 8-bit 1 Dimension vector. Then the 8 bit value was converted to intensity value of the pixel. Finally we converted the single dimensional vector into a 2-dimensional vector during plot generation phase. In our experiments we used Byte plot and Markov plot to generate two dimensional image. A detailed explanation of plot generation is given below. Later we extracted image based features like intensity based features, Wavelet based features and Gabor based features.

4.1 Byte plot

Byte plot visualization generation is given in Fig. 4. In order to convert executable in to Byte plot each pixel value was

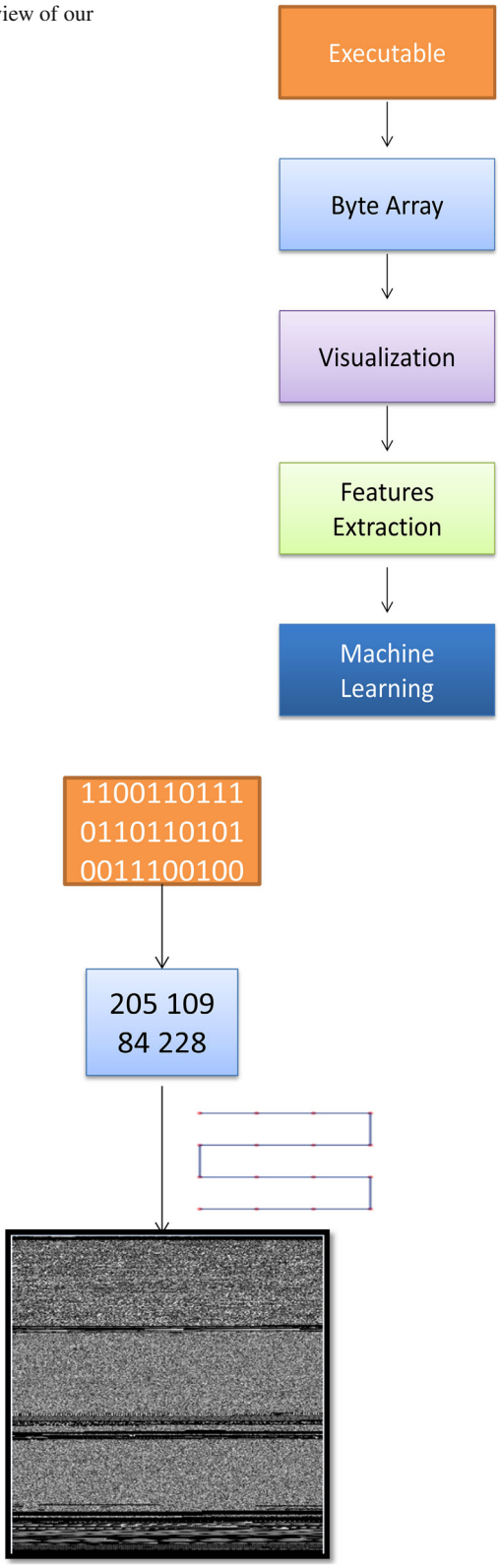

Fig. 3 Overview of our approach

Fig. 4 Byte plot visualization generation

represented by byte value. Byte plot is a grayscale image where a byte value of 0 is black and byte value of 255 is white. Based on the size of the file we converted the one dimensional array in two dimensional arrays where each element is represented by its byte value (0–255). The top left pixel of the Byte plot is the first value of the executable. Second byte value is used to represent second column in the top row. Subsequent values are plotted in a sequence from left to right until the end of the row. Next value will be plotted at the left most column of the row below it. A zigzag pattern is used to plot the entire Byte plot. Extra zeros will be added at the end to fill the missing values at the end of the file.

4.2 Markov plot

Markov plot is based on Markov model. It uses byte level transitions to create a signature. These transitions capture the encoding schemes used by a packer. Even in files with high entropy (above 7.6) we can still identify features in the Markov plot. The image below is of an executable file packed using the UPX packer. The horizontal red line represents transitions to the 0x00 byte. For each malware we created a 256 by 256 image using the method described below.

To create the Markov Byte Plot we started with a complete, weighted, digraph G = <V, E> on 256 nodes. Each vertex, v, represents a unique byte value and each directional edge, $e_{i,j}$, represents a transition from byte i to byte j. Each edge weight, $w(e_{i,j})$ is initialized to zero.

The entire binary file is read. For each transition from b_t(byte t) to b_{t+1} (byte t+1) the corresponding edge $e_{t,t+1}$ is incremented and $w(e_{t,t+1})$ is total number of edges from t to t+1. To calculate the transition probability $Pr(b_t|b_{t-1})$ from the raw counts we divided each edge weight by the sum of all outbound edges from the source node.

$$pr(b_i|b_{i-1}) = \frac{w(e_{i-1,i})}{\sum_{j=0}^{255} w(e_{i-1,j})} \quad (1)$$

Each of the calculated transition probabilities is used to populate the 256 × 256 transition matrix P where p(i, j) = pr(bj|bi). To create a visualization of the transition matrix, each entry p(i, j) is assigned to a pixel. Each pixel is assigned a color by first normalizing the transition probabilities and then mapping them to a red-green-blue spectrum. Figures 5 and 6 provides an example of Byte plot and Markov plot respectively.

5 Feature extraction

We extract three different sets of features [29]: Intensity based, Wavelet based and Gabor based features. Intensity based features are extracted directly from plot. We extract average intensity, variance, mode, skewness, kurtosis, and

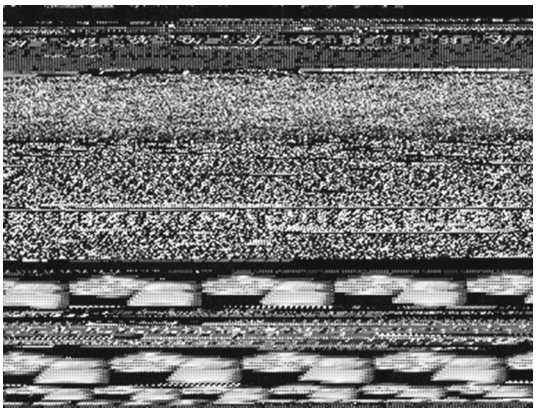

Fig. 5 An example of Byte plot visualization

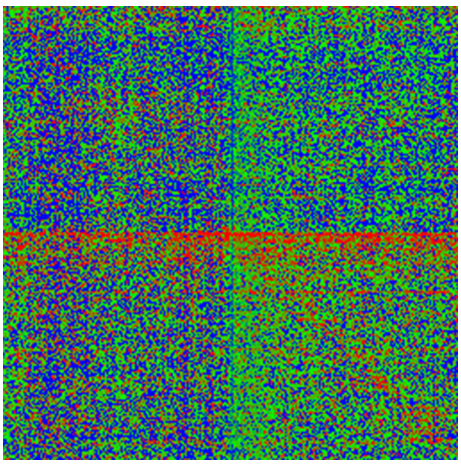

Fig. 6 Markov plot visualization example

number of pixels with intensity value 0 and 255 from the image. Formula for skewness and kurtosis is given below.

$$\text{Skewness} = \frac{\frac{1}{N}\sum_{k=1}^{N}(X_k - X)^3}{\left(\frac{1}{N}\sum_{k=1}^{N}(X_k - X)^2\right)^{3/2}}$$

$$\text{Kurtosis} = \frac{\frac{1}{N}\sum_{k=1}^{N}(X_k - X)^4}{\left(\frac{1}{N}\sum_{k=1}^{N}(X_k - X)^2\right)^{2}} - 3$$

5.1 Wavelet based features

Wavelet transform [30] is a powerful signal processing tool for analyzing signals. It overcomes the problems of Short Time Fourier Transform (STFT) relating to time and space resolution. Discrete Wavelet Transform (DWT) provides high time resolution and low frequency resolution for high frequencies and high frequency resolution and low time resolution for low frequencies. The wavelet transform has excellent energy compaction and de-correlation properties, which can be used to effectively generate compact representations that capture the structure of data. Wavelets can capture texture information efficiently. The wavelet representation consists of coarse overall approximation together with detail coefficients which influence the function at various scales.

Continuous wavelet transformation is given by formula 2

$$X(\tau, a) = \frac{1}{\sqrt{a}}\int x(t)\psi^*\left(\frac{t-\tau}{a}\right)dt \tag{2}$$

Where ψ is the wavelet function given by formula 3, t is time and a is called binary dilation.

$$\Psi_{\tau,a} = \frac{1}{\sqrt{a}}\psi\left(\frac{t-\tau}{a}\right) \tag{3}$$

For a two dimensional signal, wavelet transform is obtained by using multiple filters. DWT can be obtained by decomposing the signal simultaneously with high-pass filter 'h' and low-pass filter 'g'. The outputs will be detailed coefficients from high-pass filter and approximation coefficients (A) from low-pass filter. Detailed coefficients consist of Horizontal (H), Vertical (V) and Diagonal (D) coefficients. This decomposition can be repeated further to increase the frequency resolution. Based on the level of decomposition approximate coefficient will be further decomposed.

In our experiments we applied level 3 wavelet decomposition using Daubechies wavelet 'db4'. After applying wavelet transform we get one set of approximate coefficients and three sets of detailed coefficients. We extract mean, variance, maximum and minimum values from each of these coefficients.

5.2 Gabor based features

Gabor filter [31] is an excellent band pass filters that can be defined as convolution of an image with Gabor function. If I is an input image then Gabor transformed image r is given by formula below

$$r(x, y) = \iint_{\Omega} I(m, n)g(x - m, y - n)dmdn$$

where Ω ontains all pixels in image, g(x, y) is a Gabor filter. λ is the wavelength of the sinusoidal factor, θ is the orientation σ and is the sigma/standard deviation.

$$g\lambda, \theta, \partial(x, y) = e^{-((x'^2+\gamma^2 y'^2)/2\sigma^2)}\cos\left(2\pi\frac{x'}{\lambda} + \partial\right)$$

Gabor filter [31] acts as a local band pass filter, which captures localization properties both in spatial and frequency domains. Gabor filters are used in applications like iris

recognition, fingerprint recognition, gait recognition for segmentation, analysis and classification. Gabor based features are useful in a wide variety of applications. However Gabor filters are computationally intensive thus reducing the efficiency of the system.

6 Results

We collected our dataset from Offensive computing repository [32]. From a repository of over two million samples, the top nine packer families were selected. From each family of packers we randomly sampled 5000 malware for use in our training and test sets. Unpacked malware is also collected from Offensive computing database. The dataset consists of total 534 features, of which 512 features are Gabor based features, 16 features are Wavelet based features and 6 intensity based features. Results obtained using Markov plot and Byte plot are given in Tables 2 and 3 respectively. The best accuracy between Markov and Byte plot are highlighted with bold.

Table 2 Results obtained using Markov plot based features

Packer	Accuracy	False positive rate (FPR)	False negative rate (FNR)
UPX	**95.02**	5.85	4.35
Aspack	**95.9**	3.6	4.5
Armadillo	83.94	16.93	15.1
BobSoft	**92.79**	7.08	7.33
PECompact	**91.48**	10.23	6.73
TELock	96.56	3.60	3.26
WinUpack	**95.97**	4.18	3.86
InstallShield	**91.54**	8.37	8.53
Themida	**99.05**	1.54	0.2

Table 3 Results obtained using Byte plot based features

Packer	Accuracy	False positive rate (FPR)	False negative rate (FNR)
UPX	81.69	19.46	17.09
Aspack	94.99	3.62	6.38
Armadillo	**94.77**	2.78	7.01
BobSoft	89.81	9.76	10.62
PECompact	90.01	13.90	5.93
TELock	**97.30**	1.87	3.53
WinUpack	95.46	3.88	5.20
InstallShield	88.76	13.19	9.20
Themida	97.43	2.78	2.30

6.1 Support vector machines (SVM)

The SVM approach transforms data into a feature space F that usually has a huge dimension. It is interesting to note that SVM generalization depends on the geometrical characteristics of the training data, not on the dimensions of the input space [25,26]. Training a support vector machine (SVM) leads to a quadratic optimization problem with bound constraints and one linear equality constraint. Vapnik shows how training a SVM for the pattern recognition problem leads to the following quadratic optimization problem [33].

$$W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_i k(x_i, x_j) \tag{6}$$

$$\text{Minimize} \sum_{i=1}^{l} y_i \alpha_i \quad \forall i : 0 \leq \alpha_i \leq C \tag{7}$$

where l is the number of training examples, α is a vector of l variables and each component α_i corresponds to a training example (x_i, y_i).

6.2 Model selection using SVMs

In any predictive learning task, such as classification, both a model and a parameter estimation method should be selected in order to achieve a high level of performance of the learning machine. Recent approaches allow a wide class of models of varying complexity to be chosen. Then the task of learning amounts to selecting the sought-after model of optimal complexity and estimating parameters from training data [34,35].

Within the SVMs approach, usually parameters to be chosen are:

(i) The penalty term C which determines the trade-off between the complexity of the decision function and the number of training examples misclassified;
(ii) The mapping function φ.
(iii) The kernel function such that $K\left(x_i, x_j\right) = \varphi\left(x_i\right) . \varphi\left(x_j\right)$.

In the case of RBF kernel, the width, which implicitly defines the high dimensional feature space, is the other parameter to be selected. Grid search was done on each training dataset with 10-fold cross validation to select the C and γ values. Grid search verifies the accuracies of all the combinations of C and γ values within the given range and returns the values of C and γ where the best accuracy was observed.

Figure 7 provides the model curve obtained while performing grid search. Search is performed between values −10 to 10 for both Cost parameter and Gamma parameter. Initial values were set to 0, 0 and with an increment of 1 the values were modified. In the figure below the curves corre-

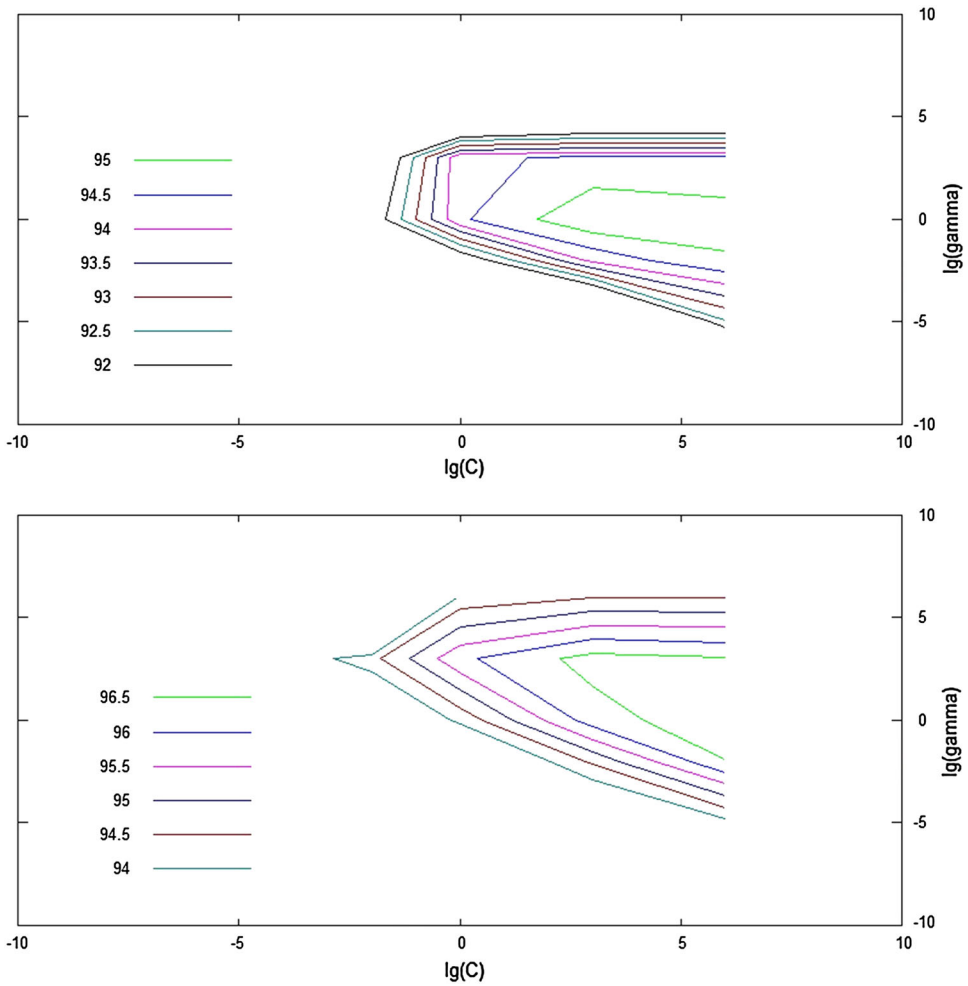

Fig. 7 Model plot for the classification of UPX datasets

sponding to green produced a Cross validation accuracy of 94.5 %. Cross validation is applied during phase to detect over fitting in Model. This plot is used to identify optimal Cost and Gamma values.

6.3 ROC curves

Receiver Operating Characteristic (ROC) Curve is a graphical plot between the sensitivity and specificity. The ROC is used to represent the plotting of the fraction of true positives (TP) versus the fraction of false positives (FP). The point (0, 1) is the perfect classifier, since it classifies all positive cases and negative cases correctly. Thus an ideal system will be initiated by identifying all the positive examples and so the curve will rise to (0, 1) immediately, having a zero rate of false positives, and then continue along to (1, 1).

Detection rates and false alarms are evaluated for the data set and the obtained results were used to form the ROC curves. In each of these ROC plots, the x-axis is the false alarm rate, calculated as the percentage of normal executables considered as malicious; the y-axis is the detection rate, calculated as the percentage of malware detected. A data point in the upper left corner corresponds to optimal high performance, i.e., high detection rate with low false alarm rate [36]. The accuracy of the test depends on how well the test classifies the group being tested into 0 or 1. Accuracy was measured by the area under the ROC curve (AUC). An area of 1 represents a perfect test and an area of .5 represents a worthless test. In our experiment, we got an AUC of 0.9764 on Themida dataset using Markov plot based features as shown below in Fig. 8.

The Area Under Curve (AUC) on each plot represents the overall accuracy of the classifier. The AUC under the ROC curves for the models generated by training 50, 60 and 70 % of the samples in the collected dataset was measured to be between 0.9987 and 0.9991 and observed only minor variations in the AUC with the size of the training datasets.

6.4 Comparison with PEID

We compared the performance of our method with PEID. PEID is a widely used signature based detection tool used to

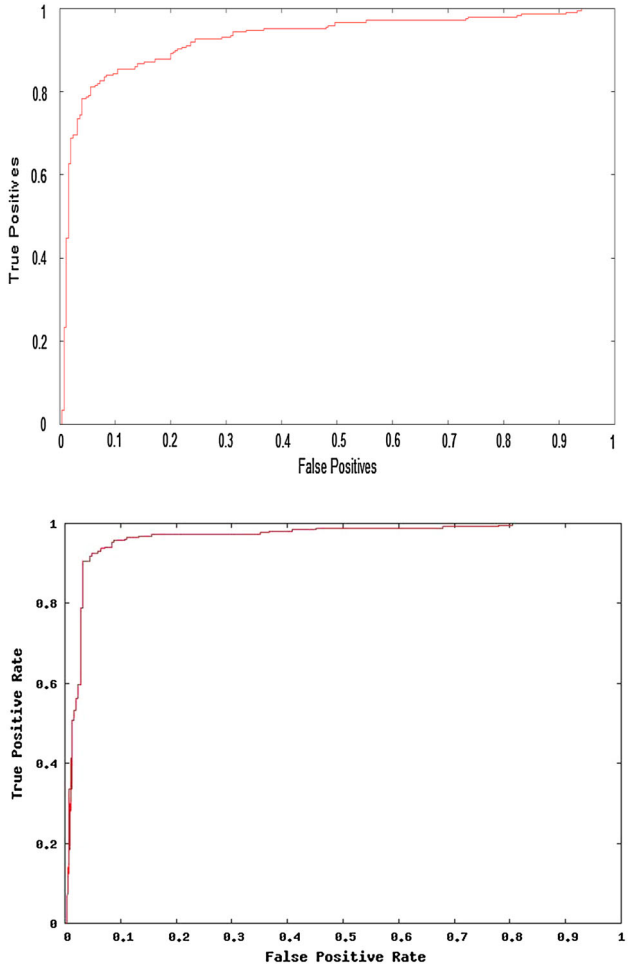

Fig. 8 ROC curves for the classification of UPX datasets

detect most of the common packers, cryptors and compilers for PE files. We compare the performance of our approach both in the context packer detection and packer identification problem. Table 4 provides comparison between performance of PEID and Markov Plot in the context of packer detection problem.

Markov plot performed better than all the packers. Except for TELock where, there was an average difference of 7 % in accuracy between Markov and PEID. For Themida packer Markov plot performed better by showing about 20 % accuracy.

We also compared the performance of Markov plot with PEiD in the context of Packer Identification problem. We used Multiclass SVM for this experiment. The total dataset consists of 9 classes with each class belonging to a packer. Results obtained from Markov plot is used for comparison. Table 4 gives the performance of Markov plot and PEiD for packer identification. Except for Armadillo, Bobsoft and TELock we obtained comparable performance with PEiD. For Themida packer our method performed better than PEiD.

6.5 Multi class

6.5.1 Multi class SVM

Support vector machines (SVM) is originally designed to binary classification. There are two different methods to perform multi class using SVMs; one is by constructing multiple classifiers for each class and combining them and other is construct it to one optimization formulation. One against [33] all is one of the earliest implementations for multi class SVMs. In this method for K-class problem k binary SVM models are constructed. The i_{th} SVM is constructed with i_{th} class considered as positive label and remaining other classes as negative labels.

Another method is one against one method proposed in [37,38]. In this method k $(k-1)/2$ classifiers are built where each classifier Cij is built between two classes ith and jth. A voting strategy is used to decide the final class. If classifier Cij gives a positive output then the count of ' ' is increased by one or else the count of 'j' is increased by one. Another method Directed Acyclic Graph SVM (DAGSVM) proposed in [39] uses a similar approach as one against one. In testing

Table 4 Results obtained using Byte plot based features

Packer	Accuracy Markov Plot (Packer Detection)	Accuracy PEID	Accuracy Markov Plot (Packer Identification)
UPX	95.02	90.2	89.9
Aspack	95.9	88.7	84.8
Armadillo	83.94	76.8	59.8
BobSoft	92.79	88.8	62
PECompact	91.48	85.1	80.2
TELock	96.56	95.9	83.7
WinUpack	95.97	88.5	90.1
InstallShield	91.54	84.8	89.3
Themida	99.05	79.85	93.6

Table 5 Results obtained using multiclass classification

	Byte plot (%)	Markov plot (%)
Random Forest	76.8	78.25
SVM	79.34	**81.34**

phase it builds a rooted binary directed acyclic graph with $k(k-1)/2$ internal nodes and k leaves. Each node represents a binary SVM and final decision is reached by following the path from root node.

6.5.2 Random forests

A random forest is a classifier consisting of a collection of tree structured classifiers $\{h(x, \Theta_k), k = 1, \ldots\}$ where $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class of input X.

The common element in random trees is that for the k^{th} tree, a random vector Θ_k is generated, independent of the past random vectors $\Theta_1 \ldots \Theta_{k-1}$ but with the same distribution; and a tree is grown using the training set and Θ_k, resulting in a classifier $h(x, \Theta_k)$ where x is an input vector. For instance, in bagging the random vector Θ is generated as the counts in N boxes resulting from N darts thrown at random at the boxes, where N is the number of examples in the training set. In random split selection Θ consists of a number of independent random integers between 1 and K. The nature and dimensionality of Θ depends on its use in tree construction. After a large number of trees are generated, they vote for the most popular class [40–42].

The random forest error rate depends on two things:

- The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

The results obtained by performing multiclass classification are given in Table 5. We obtained an accuracy of 81 % in classifying 10 classes (9 different packed malware classes and 1 unpacked malware class). SVM produced best accuracy 81 % using Markov plot which is highlighted bold in Table 5.

7 Conclusions

In this work we proposed a new technique for packer identification using Byte plot and Markov plot visualization. We extracted texture based features like Gabor and Wavelet based features from these plots. The performance of our system is evaluated by using SVM. We performed our experiments on nine well known packed malware. We obtained a best accuracy of 99 % for Themida dataset. For UPX (which is widely used packer) we obtained accuracy of 95 %. The performance of Markov plot is compared with PEiD. In the context of packer detection Markov plot performed better than PEiD by about 7 % for most packers. Except for Armadillo, Bobsoft and TELock we obtained comparable performance with PEiD in the context of packer identification. For Themida packer Markov plot performed better by 20 % in accuracy for packer detection problem and 14 % for packer detection problem.

We also performed multiclass classification on entire dataset using multiclass SVM and Random Forest. We obtained an accuracy of 80 % using Random forest. Even though both the feature sets perform well, Markov based features performed best. On average there is a difference of 3 % between features extracted from Byte plot and Markov plot. Markov plot captures the co-occurrence of byte values in an executable where as Byte plot is Byte representation of executable. Markov plot performed better than Byte plot as it captures more information. We had performed grid search for selecting the optimal values for cost (C) and gamma (γ) parameters. We had plotted the ROC curves using the selected cost (C) and gamma (γ) values to demonstrate the performance of the SVMs.

Our results show the potential of our methodology for packer identification problem. In the future we would like perform experiments on samples packed with multiple layers of packers. We would also like to identify important sections in executable and apply our methodology on these sections.

References

1. Symantec, Internet Security Threat Report, 2011 trends, Vol. 17 Main Report, April 2012. Retrieved on 8/10/2012 http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf
2. Dell Sonic wall Inc, Anatomy of a cyber-attack, Retrieved on 8/10/2012. http://www.sonicwall.com/downloads/EB_Anatomy_of_a_CyberAttack_Final.pdf
3. Yan, W., Zhang, Z., Ansari, N.: Revealing packed malware. IEEE Secur. Priv. **7**(5), 65–69 (2008). doi:10.1109/MSP.2008.126
4. Shadowserver, Packer Statistics, The Shadowserver Foundation, http://www.shadowserver.org/wiki/pmwiki.php/Stats/PackerStatistics
5. Armadillo, Digital River Accessed on June 2014 http://www.digitalriver.com/Armadillo
6. Themida Advanced windows software protection system. Accessed on June 2014 http://www.oreans.com/themida.php
7. PECompact Powerful executable compression for software developers and vendors. Accessed on June 2014 http://bitsum.com/pecompact/
8. Installshield Software Installation Solution Accessed on June 2014 http://www.installshield.com/

9. Ultimate Packer for Executables UPX Accessed on June 2014 http://upx.sourceforge.net/
10. Telock - free PE-File Encryptor/-Compressor Accessed on June 2014 http://www.telock.com-about.com
11. Ban, T., Isawa, R., Guo, S., Inoue, D., Nakao, K.: Efficient malware packer identification using support vector machines with spectrum kernel. In: AsiaJCIS pp. 69–76 (2013)
12. PEiD, most recent release (PEiD 0.95) could be downloaded from http://www.softpedia.com/
13. Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Secur. Priv. **5**, 40–45 (2007)
14. Perdisci, R., Lanzi, A., Lee, W.: Classification of packed executables for accurate computer virus detection. Pattern Recogn. Lett. **29**, 1941–1946 (2008)
15. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**, 2721–2744 (2006)
16. Sun, L., Versteeg, S., Boztaş, S., Yann, T.: Pattern recognition techniques for the classification of malware packers. In: Proceedings of the 15th Australasian Conference on Information Security and Privacy, ser. ACISP'10. Springer, Berlin, pp. 370–390 (2010)
17. Ugarte-Pedrero, X., Santos, I., Bringas, P.G.: Structural feature based anomaly detection for packed executable identification. In: Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems (CISIS), pp. 50–57 (2011)
18. Stewart, J.: OllyBonE v0.1, Break-on-Execute for OllyDbg (2006)
19. Valsmith, Q.D.: Covert debugging: circumventing software armoring techniques, Black Hat USA (2007)
20. Martignoni, L., Christodorescu, M., Jha, S.: OmniUnpack: fast, generic, and safe unpacking of malware. In: 23rd Annual Computer Security Applications Conference (ACSAC) (2007)
21. Shafiq, M., Tabish, S., Farooq, M.: PE-Probe: leveraging packer detection and structural information to detect malicious portable executables. In: Proceedings of the Virus Bulletin Conference (VB), pp. 29–33 (2009)
22. Farooq, M.: PE-Miner: mining structural information to detect malicious ex-ecutables in realtime. In: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID). Springer, Heidelberg, pp. 121–141 (2009)
23. Ban, T., Nakao, K.: Application of string kernel based support vector machine for malware packer identification. In: Proceedings of the 2013 International Joint Conference on Neural Networks, ser. IJCNN'13. San Francisco, CA: Morgan Kaufmann Publishers Inc. (2013)
24. Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Dokl. **10**, 707 (1966)
25. Conti, G., Bratus, S., Shubina, A., Lichtenberg, A., Ragsdale, R., Perez-Alemany, R., Sangster, B., Supan, M.: A visual study of binary fragment types Black Hat USA (2010)
26. Conti, G., Bratus, S., Sangster, B., Ragsdale, S., Supan, M., Lichtenberg, A., Perez, R., Shubina, A.: Automated mapping of large binary objects using primitive fragment type classification digital forensics research conference (DFRWS) (2010)
27. Yoo, I.: Visualizing windows executable viruses using self-organizing maps. In: 2004 International Workshop on Visualization for Cyber Security (VizSec)
28. Quist, D.A., Liebrock, L.M.: Visualizing compiled executables for malware analysis. In: International Workshop on Visualization for Cyber Security (VizSec), pp. 27–32 (2009)
29. Kancherla, K., Mukkamala, S.: Image visualization based malware detection. In: CICS, pp. 40–44 (2013)
30. Akansu, A.N., Serdijn, W.A., Selesnick, I.W.: Wavelet transforms in signal processing: a review of emerging applications. Phys. Commun. **3**(1), 1–18 (2010)
31. Grigorescu, S.E., Petkov, N., Kruizinga, P.: Comparison of texture features based on gabor filters. IEEE Trans. Image Process. **1**(10), 1160–1167 (2002)
32. Offensive Computing Open Malware Accessed on May 2013 http://www.offensivecomputing.net/
33. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. Comparison of classifier methods: a case study in handwritten digit recognition. In: Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks, Jerusalem, pp. 77–87 (1992)
34. Lee, J.H., Lin, C.J.: Automatic model selection for support vector machines, Technical Report, Department of Computer Science and Information Engineering, National Taiwan University (2000)
35. Chang, C.C., Lin, C.J.: LIBSVM: A Library for Support Vector Machines. National Taiwan University, Taipei, Department of Computer Science and Information Engineering (2001)
36. Egan, J.P.: Signal Detection Theory and ROC Analysis. Academic Press, New York (1975)
37. KRebel, U.: Pairwise classification and support vector machines. In: Scholkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods-Support Vector Learning. MIT Press, Cambridge (1998)
38. Friedman, J.H.: Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University (1996)
39. Platt, J.C., Cristianini, N., Shawe-Taylor, J.: Large margin DAGs for multiclass classification. In: Solla, S.A., Leen, T.K., Müller, K.-R. (eds.) Advances in Neural Information Processing Systems 12 (NIPS-99), pp. 547–553. MIT Press, Cambridge (2000)
40. Breiman, L.: Random forests. J. Mach. Learn. **45**, 5–32 (2001)
41. Salford Systems, TreeNet, CART, Random Forests Manual
42. Crammer, K., Singer, Y.: On the Learnability and Design of Output Codes for Multiclass Problems. In: Computational learning theory, pp. 35–46 (2000)