Pattern Recognition Techniques for the Classification of Malware Packers

Li Sun[1], Steven Versteeg[2], Serdar Boztaş[1], and Trevor Yann[3]

[1] School of Mathematical and Geospatial Sciences, RMIT University,
GPO Box 2476V, Melbourne, Australia
li.sun@ca.com, serdar.boztas@ems.rmit.edu.au
[2] CA Labs, Melbourne, Australia
steven.versteeg@ca.com
[3] HCL Australia, Melbourne, Australia
tyann@hcl.in

Abstract. Packing is the most common obfuscation method used by malware writers to hinder malware detection and analysis. There has been a dramatic increase in the number of new packers and variants of existing ones combined with packers employing increasingly sophisticated anti-unpacker tricks and obfuscation methods. This makes it difficult, costly and time-consuming for antivirus (AV) researchers to carry out the traditional static packer identification and classification methods which are mainly based on the packer's byte signature.

In this paper[1], we present a simple, yet fast and effective packer classification framework that applies pattern recognition techniques on automatically extracted randomness profiles of packers. This system can be run without AV researcher's manual input. We test various statistical classification algorithms, including $k-$Nearest Neighbor, Best-first Decision Tree, Sequential Minimal Optimization and Naive Bayes. We test these algorithms on a large data set that consists of clean packed files and 17,336 real malware samples. Experimental results demonstrate that our packer classification system achieves extremely high effectiveness ($> 99\%$). The experiments also confirm that the randomness profile used in the system is a very strong feature for packer classification. It can be applied with high accuracy on real malware samples.

1 Introduction

The Internet has become an essential part of human life in the modern information society. While the Internet brings enormous convenience to users, *Mal*icious soft*ware* (Malware) which includes computer viruses, worms, Trojan horses, other malicious or unwanted software, produce a tremendous impact on users' security, reliability and privacy. This is a serious threat to the security of computer networks. In recent years, the number of malware infections has risen sharply.

Many efforts have been made to combat malware. For every malware binary, antivirus (AV) researchers need to reverse the malware and update the AV scanner to detect

[1] This work was supported by CA Labs and an ARC linkage grant. Ms. Sun is currently a Ph.D. candidate sponsored by CA Labs.

R. Steinfeld and P. Hawkes (Eds.): ACISP 2010, LNCS 6168, pp. 370–390, 2010.

and counter it. Unfortunately, malware authors are aware of this. Most new malware implement various obfuscation techniques in order to disguise themselves, therefore preventing successful analysis and thwarting the detection by AV scanners.

Packing is a favorite obfuscation technology used by malware. It has been reported that the majority of malware in the Wildlist, (the list of current in-the-wild viruses [1]), is runtime packed [2, 3, 4]. The packing technique which is informally discussed in this section will be formally defned later on in this paper.

Packing malware has several benefits for the attacker. Firstly, it usually reduces its file size, thus allowing easy transfer on the net. Secondly, it increases the AV scanner's scanning time as runtime packers require additional work from the scanner, such as checking the file format and the code, unpacking, etc. Thirdly, it is more resistant to AV scanners due to the compression and encryption it employs. Lastly, the packer's complexity can be enhanced almost without limit by applying various new armoring techniques as described in Section 2.2. *The quick analysis and identification of a complex packer is a very challenging task to the AV researchers.*

Packer classification is an important step in malware analysis, if it can be designed so that it gives a reliable approach to detect packed files. The objective of packer classification is to quickly detect and identify the packer, allowing AV researchers to easily and correctly unpack the file and retrieve original payload for further malware detection and analysis. An efficient and effective packer classification system can yield benefits to both back-end AV researchers and real time anti-malware engines, running live on the client machine.

However, in recent years, the traditional packer identification technique based on signature scanning has been confounded by counter-counter attacks by packer and malware developers. An increasing number of anti-unpacking tricks and obfuscation methods are now being applied to packers. The targets include both existing common packers and the newly emerging ones, with the aim of providing a hard shell for malware.

As the number of new packers keeps growing, *a reliable computer-based packer classification scheme would facilitate the identification and characterizing process and reduce cost significantly*. In this paper, an automatically generated randomness profile based scheme is proposed as a replacement for the inefficient manually created signatures, which is the current approach, to the packer classification problem.

The main contributions of this paper are

- We develop a complete packer classification system that runs automatically, without the need to manually reverse engineer every packer variation. To the best of our knowledge, this is the first published system that achieves high accuracy on real malware.
- We refine Ebringer et al. [5]'s sliding window randomness test algorithm to produce randomness feature set of the packer and conduct experiments to give guidance for optimal parameter settings.
- We evaluate the effectiveness of four pattern recognition algorithms for classifying packers using the randomness profile information.
- We carry out large scale testing on real malware. The tests cover various packers with different levels of sophistication.

The remainder of the paper is organized as follows. Section 2 provides research background on packers, existing classification methods and pattern recognition techniques. It also discusses related work. Section 3 gives the implementation of our system. Section 4 describes experimental setting and reports evaluation results of different classification algorithms. Section 5 discusses future directions and concludes the paper.

2 Background

2.1 Packing

A *packer* is an executable program that takes an executable file or dynamic link library (DLL), *compresses* and/or *encrypts* its contents and then packs it into a new executable file. The packed files discussed in this paper are in Portable Executable (PE) format [6].

Assuming the original executable file contains already known malicious code, the signature based AV scanner should be able to detect it. However, the appearance of the malicious code is changed after packing due to the compression and/or encryption. Therefore, the packed file will thwart malware detection as no signature match will be found. The analysis and detection of malware can only be undertaken after the file is unpacked, i.e. the compressed/encrypted original file has been decompressed/decrypted completely.

A packed file contains two basic components. The first part is a number of data blocks which form the compressed and/or encrypted original executable file. The second part is an unpacking stub which can dynamically recover the original executable file on the fly. When the packed file is running, the unpacking stub is executed firstly to unpack the original executable code and then transfers the control to the original file. The execution of the original file is mostly unchanged and starts from its original entry point (OEP) with no runtime performance penalties (after the unpacking has been completed.)

2.2 Packer Evolution

As the variety of packing programs grows, their level of sophistication also increases. This is because that anti-unpacker tricks continually evolve and are implemented quickly by malware writers in a range of packers, from long-existing packers to modern new packers, with the goal of protecting malware. The main types of anti-unpacker tricks are listed below and most tricks have been described in Ferrie's papers [7, 8, 9, 10, 11]. Basically the harder/more the tricks that the packer employ, the higher the level of sophistication of the packer.

- *Anti-dumping* mainly alters the process memory of the running process to hinder further analysis on the dumped memory. The alteration is mainly applied on some useful information, such as PE header, imports, entry point codes, etc.
- *Anti-debugging* prevents AV researchers from using a debugger easily. The debugger is the most common tool used to trace the execution of malware in action.
- *Anti-emulating* attacks the software-based environment such as an emulator or a virtual machine, e.g., VMware [12]. Such an environment is essential to safely execute/monitor malicious behavior.
- *Anti-intercepting* thwarts page-level interception which stops the packer execution of newly written pages.

A lot of existing packers can be easily modified as their source codes are freely available. Moreover, most newly emerging packers can be customized. For example, as shown in Figure 1, Themida, one of most well-known sophisticated packers, provides various protection options and the user is allowed to configure the protection level of the packer. Consequently, the number and the complexity of new packer strains and variants are dramatically increased.

Fig. 1. Sample packer protection options of Themida

2.3 Traditional Signature Based Packer Classification

The traditional packer classification approach is mainly based on matching the packer's byte signature. Byte-signature based packer scanners, such as PEiD [13] and pefile [14], use a signature database to determine if a binary contains packed-code. A packer signature is typically a distinctive set of bytes which occurs at the entry point or in sections in a PE file. In this approach, the incoming packer is checked against the database of the signatures for known packers. If there is an exact match, the packer is considered being used and the name of the packer is also identified.

The byte-signature based packer classification method is effective at detecting known packers, however, the large diversity of packers, and the number of different variants of each packer, severely undermines the effectiveness of classical signature-based detection. Besides, this approach is expensive as the signature detection and updating need to be performed accurately by AV experts.

2.4 Pattern Recognition

Pattern recognition techniques have recently been used in anti-malware community, mainly for the purpose of identifying new or unknown malware [15, 16, 17, 18, 19, 20, 21, 22, 23]. Pattern recognition aims to recognize a particular class from a measurement vector. Different pattern classes with different measurement vectors correspond to different points in measurement space and patterns with similar appearance tend to cluster together. Therefore, a mapping relationship can be established from the measurement space into the decision space. There are two essential technologies involved in a pattern recognition system, namely feature extraction and classification.

Feature extraction retrieves the common features (patterns) among a set of objects. A feature is the measurement of a property of an object. A feature set describes the essential properties of an object using a greatly reduced number of parameters. However, only the features that properly represent the original object can lead to a satisfying pattern recognition result. In other words, the extracted features of objects in each class should be represented in a distinctive way. This permits a set of objects to be classified into different classes.

Classification is the process that first analyzes the training set, develops a classification model for each class and then applies the model to classify the testing set based on their features. A *training set* is a collection of records for which the class label have been provided by a trusted source. A *testing set* is used to verify the accuracy of the model and consists of records with class labels that you want to predict. A classification model (also called a *classifier*) can be built as follows: given a set of N training data in which each record consists of a pair: a feature vector of n features $\mathtt{x} = \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ and the associated "truth" class y_j, produce a relationship $f : \mathtt{x} \rightarrow y_j$ that maps any feature vector $\mathtt{x} \in X$ to its true class $y_j \in y$. Four classifiers used in this paper are detailed described in Appendix A. They are Naive Bayes (NB), Sequential Minimal Optimization (SMO), $k-$Nearest Neighbor (kNN) and Best-first Decision Tree (BFTree).

2.5 Related Work

The closest work to this research was conducted by Perdisci et al. [24]. They applied various pattern recognition techniques to classify executables into two categories, packed and non-packed. Nine features are combined together for classification, namely number of standard and non-standard sections, number of executable sections, number of readable/writable/executable sections, number of entries in the PE file's Import Address Table (IAT), PE header entropy, code section entropy, data section entropy and file entropy. The system achieved very high accuracy (above 95%) using NB, J48 decision tree, bagged, kNN or Multi Layer Perceptron (MLP) classifiers. However, this approach is not able to detect what family of packers a packed file belongs to.

Other than packer classification, there have been a few recent attempts to use pattern recognition techniques for automated malware detection [15, 16, 17, 18, 19]. Most of them only classified files between malicious and benign, but not by family of malware.

In early attempts, Tesauro et al. [15,16] developed a neural network for virus detection. The system is specially designed for the detection of boot sector viruses using feature trigrams. The *trigrams* are three byte strings. They are selected in the feature set if they appear frequently in viral boot sectors but infrequently in uninfected software and definitely do not appear in legitimate ones. In the experiments, 200 viral boot sectors and 100 legitimate boot sectors were used as the data set in which half of them were the training set and the other half were the test set. Using a classification threshold of 0.5, performance on the test set was typically 80-85% for the viral boot sectors and 100% for the legitimate boot sectors. The classifier has also been incorporated into the IBM anti-virus product and has caught approximately 75% of new boot sector viruses.

Schultz et al. [19] used data mining methods to detect malware. They used three types of features, binary profiles of DLLs, strings and sequences of n adjacent bytes (also called n-grams), and paired each feature with a single learning algorithm. That is, a rule-base classifier was applied to the binary profiling; string data was used to fit a naive Bayes; and an ensemble of multi- NB classifiers is used on the n-grams data. In the latter, the n-grams data were partitioned into six parts firstly, then each classifier was trained on each partition of the data. The experiments were carried on a data set which contained 3301 malicious programs and 1001 clean programs. Among them, 38 of the malicious programs and 206 of the benign programs were in the Windows Portable Executable (PE) format. The results showed that naive Bayes with strings achieved the best accuracy than others. However, their experiments did not provide a fair comparison among the classifiers as different features were used for different classifiers. Moreover, different training sets were used to training different classifiers.

Using the same ideas, MECS [18] extracted byte sequences from the executables, converted these into n-grams, and constructed several classifiers: kNN, NB, support vector machines, decision trees J48, boosted NB, boosted J48 and boosted SVM. Muazzam tried other features [17]. Instead of using fixed length instructions or *n-gram* features, the author used Vector Space Model [25, 26, 27] to extract variable length instruction sequence as the primary classification feature and applied an array of classification models, including logistic regression, neural network, decision tree, SVM, Bagging and random forest.

Several researchers addressed the issue of email classification with the aid of machine learning techniques [20, 21, 22, 23]. Cohen [20] used a rule-based algorithm to classify email into folders based on the text of messages. Sahami et al. [21] employed a naive Bayes technique to the problem of junk E-mail filtering. Androutsopoulos et al. [22, 23] also used this approach to classify spam emails and legitimate ones. They compared two classifiers, naive Bayes and k-nearest neighbor. Both algorithms achieved very high classification accuracy but k-nearest neighbor with $k = 2$ slightly outperformed others.

3 Methodology

In this section, we present our approach to the packer classification problem by analysing the performance of various statistical classifiers, as a replacement for signature matching approaches. Firstly, it extracts a unique feature set, randomness profile, from each

packed file. Then it maps the randomness feature vectors into an $n-$dimensional vector space in which various learning algorithms can be applied.

3.1 Feature Extraction

Feature extraction retrieves the 'characteristic' of the packer which represents the packer in a distinctive way. This permits the packed file to be compared with the candidate packers.

Ebringer et al. developed a randomness test [5] that preserves local detail of the packer. The randomness test measures the amount of "randomness" in different parts of a sample executable program. It was noted by authors that the randomness distribution of each packer family exhibits a distinctive pattern which suggests a kind of signal of each packer. In this paper, we investigate whether packer's randomness profile contains sufficient information to classify packers with high accuracy.

In order to extract the best features from packed files, we employ a refined version of the sliding window randomness test with trunk pruning method [5]. Compare with the original sliding window randomness test (see Appendix B), the window size and skip size used in this research are set to the same value. That is, there is no overlapping windows. Therefore, no repeat information is used in the feature set. All parameters are determined empirically (see Appendix C and D). Both window size and skip size are set to 32 and the pruning size is 50.

3.2 Classification

Classification refers to the way the packed file is examined and assigned to a predefined class. In this research, we first evaluate Ebringer et al.'s randomness signature scanning technique [5] on a large malware data set and then apply pattern recognition techniques to classify packers.

In a randomness signature based packer classification system, a packer signature is simply calculated as the average value of a set of randomness profiles of training files pre-classified as this packer. During the identification process, the distance between the test file and each packer signature is measured. The shorter the distance, the more likely the file is packed with this packer.

Instead of manual identification of signatures, a randomness signature can be automatically created from packed files. However, as stated in [5], although the preliminary packer classification results on a small clean data set are good, the results on real malware samples were of insufficient accuracy. This might be due to the fact that they used a small data set in the experiments. Besides, each tested packer contains only one version of this specific packer. They suggested that using larger data set might improve the performance.

Therefore, we firstly repeated their experiment using a large malware data set described in Table 4. As shown in Table 1 and Table 2, the performance of the system using large data set is still unsatisfactory. the average true positive rate in three tests ($n = 30, 40$ and 50) are all below 90%, while in the $n = 50$ test, NSPACK and PETITE only achieve 40.63% and 65.13% respectively.

Table 1. Performance of the sliding window algorithm using malware sample data set, where the window size $w = 32$ and the skip size is 32, the distance measure is *Cosine* measure and the pruning method is *Trunk*. The pruning size are in the range $30 - 50$.

Pruning size	Total files	TP rate
30	17336	85.68%
40	17336	85.27%
50	17336	85.17%

Table 2. Detailed performance of the sliding window algorithm using malware sample data set, where the window size $w = 32$ and the skip size is 32, the distance measure is *Cosine* measure and the pruning method is *Trunk*. The pruning size is 50.

Packer	Total files	TP rate
FSG	5105	99.53%
NSPACK	256	40.63%
PECOMPACT	1058	75.80%
PETITE	152	65.13%
UPACK	834	98.68%
UPX	9931	79.12%

The low accuracy in the results obtained by the randomness signature approach motivated us to investigate the pattern recognition techniques described in Section 2.4 for packer classification using the extracted randomness profile. In the remainder of this paper, we evaluate four statistical classifiers, namely Naive Bayes, Sequential Minimal Optimization, $k-$Nearest Neighbor and Best-first Decision Tree. These classifiers are selected since they are relatively fast. This is very important for a client-side AV scanner which needs to scan millions of files in a short user-tolerable time frame.

4 Experiments and Results

4.1 Data Sets

Experiments have been carried out on a large set of real malware samples. There are two types of data sets, namely a malware sample data set and a mixed data set. The malware sample data set only contains real malware samples which have reliable predefined class. The mixed data set has both packed clean files and real malware samples. The packers used in this data set are mixed with low complex packers and sophisticated packers. Below are the detailed descriptions of these two data sets.

Malware Samples Preparation. All malware samples have been prepared by an independent third party, Computer Associates (CA) Threat Management Team in Melbourne, Australia. To construct the data set, real malware downloaded over January and February 2009 by CA are collected. Each file is scanned by three AV scanners, Microsoft, Kaspersky and CA, for packer labeling. In addition, CA's VET engine and anti-virus Arclib Archive Library are used to determine whether the file can be unpacked. Files that reported by Microsoft, Kaspersky or CA, or that can be unpacked by either the VET engine or Arclib are identified as packed file. As a result of this method, we got a total of 103,392 packed files. The top five packers detected are UPX, ASPACK, FSG, UPACK and NSIS installer. They comprise a total of 91.35% of packed files.

For the collection of 103,392 packed files, each file is assigned as having been packed by one packer. This is done by combining the packer scanning results of all three scanners. The packer name is set if it is identified by any scanner and is 100% confirmed if it is identified by all three scanners. If there is conflicting information, the result taken is the one given by the two scanners which agree. If all three scanners disagree, PEiD is further applied to identify packer. PEiD is a byte-signature based packer scanner [13] which is supported by a large number of packer signatures. However, it is so popular that many packers start to use fake signatures to hide from PEiD detection. Therefore, PEiD's scanning results are not reliable and are only used by us for confirming information.

All scanners contain a different packer signature schema. For some packers, some scanners might only provide the packer family name without the version information. When collecting the version information, if any scanner obtains the version detail, this information is used. If there is no version information, PEiD is used to retrieve the version detail.

Table 3 lists four packer detection results extracted from our database. The first file is detected as Aspack by all three scanners. PEiD also confirms the packer. Therefore, it belongs to the Aspack family. Though Kaspersky doesn't provide the version information, all other scanners indicate it is Aspack 2.12. The second file is detected as Aspack by CA and Kaspersky, and CA provides its version number,namely version 2.0, while Kaspersky doesn't. So PEiD is further used to confirm that it is Aspack 2.0. CA identifies the third file as PC Shrinker while the other two do not have any scanning result. In this case, PEiD is used. It not only confirms the packer family, but also gives out the version 0.71. In the last example, there is information conflict between the scanning results. CA says Petite 2.1 and Microsoft gives Petite 2.3. Again, PEiD is used. Its result is Petite 2.1 or 2.2. All results are adjusted and Petite 2.1 is finally assigned to this file.

Table 3. Determination of packer name for malware samples

No	CA	Microsoft	Kaspersky	PEiD	Family	Detail
1	ASPack 2.12	ASPack v2.12	ASPack	ASPack v2.12	ASPack	ASPack 2.12
2	ASPack 2.0	NULL	ASPack	ASPack v2.001	ASPack	ASPack 2.0
3	PC Shrinker	NULL	NULL	PC Shrinker v0.71	PC Shrinker	PC Shrinker 0.71
4	Petite 2.1	Petite 2.3	NULL	Petite v2.1 (2)	Petite	Petite 2.1

Malware Sample Data Set. As discussed in the previous section, the packer names assigned to the samples are not 100% precise, especially when there is conflicting information between the scanning results from different scanners. In order to get a reliable training data set for the packer classification experiment, two criteria are used to select the packers and files in the malware sample set. These two constraints are:

- *Only confirmed cases are used*. In other words, the file's packer name has been identified as same by all three scanners, i.e., the first file in Table 3.
- *Only packers with a sufficient number of confirmed cases are chosen*. In this paper, each packer should have more than 100 confirmed packed files to be chosen.

According to the selection conditions above, 6 packers of 17,336 files, for file sizes ranging from $2 - 6880$ KB are chosen from the sample collection described in Section 4.1. The details of the set is presented in Table 4. As the top packer in the collection, UPX has 39,799 confirmed packed files. However, to balance the distribution of the sample set, only 9,931 randomly selected samples from these files are used. Note that each packer contains samples with different versions. For example, packer NsPack has cross versions of 2.x, 2.9, 3.4, 3.5, 3.6 and 3.7.

Table 4. Data set one: malware sample set

Packer	Versions	Total Number
FSG	1.33 and 2.0	5,105
NSPACK	2.x, 2.9, 3.4, 3.5, 3.6 and 3.7	256
PECOMPACT	2.xx	1,058
PETITE	2.1 and 2.2	152
UPACK	0.2x-0.3x	834
UPX	UPX, UPX(LZMA), UPX(Delphi), 2.90, 2.92(LZMA), 2.93 and 3.00	9,931
		17,336

Mixed Sample Data Set. The above malware sample data set consists of six popular packers. Though each packer contains cross version samples, these packer's complexity is relatively low. In order to assess the robustness of our classification system, the system capability of classifying a wide range of packers that have not only different variants but also different levels of sophistication, we also create the mixed sample data set.

One problem is that our sample collection does not have a sufficient number of reliable samples of the sophisticated packers. To address this problem, the selection conditions used for malware sample data set are relaxed. The new selection criteria are:

- The file's packer name is identified by two scanners, or is identified by one scanner and confirmed by PEiD.
- Packers with a sufficient number of classified cases (more than 100) are chosen.

466 files of two packers, Asprotect and Mew, that match the above criteria have been added into the data set. In addition, a popular and sophisticated packer, Themida, is chosen for this data set. As most samples of Themida in the database contain conflicting packer information, Themida packed clean files are used instead. 117 clean files in the UnxUtils binaries [28] are packed with Themida v1.8.0.0 demo version. Figure 1 shows that Themida provides various protection options and the user is allowed to configure the protection level of the packer. Consequently, the number and the

complexity of Themida variants are dramatically increased. In this paper, six different combinations of packing options are evenly applied on these files.

The details of the set is presented in Table 5. Though this data set is not as reliable as the previous malware sample set, the experimental results on this data set will still provide an overall score of the system effectiveness.

Table 5. Data set two: mixed sample set

Packer	Versions	Total Number
ASPROTECT	unknown, 1.2 and 1.23	205
FSG	1.33 and 2.0	5,105
MEW	11 and 11 SE 1.2	261
NSPACK	2.x, 2.9, 3.4, 3.5, 3.6 and 3.7	256
PECOMPACT	2.xx	1,058
PETITE	2.1 and 2.2	152
THEMIDA	v1.8.0.0 with 6 option sets	117
UPACK	0.2x-0.3x	834
UPX	UPX, UPX(LZMA), UPX(Delphi), 2.90, 2.92(LZMA), 2.93 and 3.00	9,931
		17,919

4.2 Evaluation Metrics

When comparing the performance of different classification techniques, it is important to assess how well a classification model is able to correctly predict records to the actual classes. Several metrics are conventionally in use to numerically quantify classification effectiveness performance.

To introduce the metrics, let us define that for a class y_j, a record is *positive* if it is predicted to belong this specific class and is *negative* if it is predicted to belong other classes. Suppose that for a test set with n records, the set of positive records and negative records for the class are known (for example, as the result of human judgment), and P and N are the number of positive records and negative records respectively, $n = P+N$. Using four important counts defined below, $P = TP + FN$ and $N = FP + TN$.

- *TP* represents the true positives which is the number of positive records correctly identified as specific class.
- *FP* represents the false positives, the number of negative records which do not belong to the class but were incorrectly identified as it.
- *TN* represents the true negatives which refers the number of negative records correctly identified as other classes.
- *FN* represents the false negatives, that is the number of positive records which belong to the class but were incorrectly identified as other classes.

The *accuracy* is the percentage of test set records that are correctly identified by the classifier. That is,

$$Accuracy = \frac{TP + TN}{n} = \frac{TP + TN}{P + N} \tag{1}$$

Accuracy provides an overall performance of the effectiveness. However, this measure has one limitation. Suppose that a test set contains a large number of negative records and very small number of positive records, and we use a classifier which labels every class as negative (no matter what the input data). That is, TN is very high and TP is very low. Despite the classifier being very primitive, it will achieve a very high classification accuracy on this data set.

The *true positive rate (TPrate)* and *false positive rate (FPrate)* are introduced to measure the proportion of the positive records that are correctly identified and the proportion of the negative records that are incorrectly identified, respectively. For each class, they are calculated as

$$TPrate = \frac{TP}{P} = \frac{TP}{TP + FN} \quad \text{and} \quad FPrate = \frac{FP}{N} = \frac{FP}{FP + TN}. \tag{2}$$

Two other fundamental ways to measure classification effectiveness are *precision* and *recall*. Precision is the proportion of records classified as positive which are classified correctly, and recall is the proportion of positive records that have been correctly identified. So, for each class, the precision is defined as

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

and the recall is

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

4.3 Pattern Recognition Results

As described before, in the feature extraction process, each packed file is passed to the sliding window randomness test with a window size 32 bytes (256 bits). There is no overlap between windows, i.e., the skip size a is 32. Then the feature vector is constructed by extracting low randomness values in the range of 30 to 50 from the output using the Trunk pruning method.

Classification was carried out using the Weka 3.6.0 (Waikato Environment for Knowledge Analysis) machine learning package, developed by University of Waikato [29, 30, 31]. All selected statistical classifiers, NB, SMO, kNN (called IBk in WEKA) and BFTree are implemented in Weka. In the experiments, all classifiers used the default settings defined by Weka.

In the tests, 10-fold cross validation [32] is used. For each data set, the whole set is randomly partitioned into ten equal-size subsets. There are a total of 10 runs. During each run, one subset is used for testing and the other nine subsets are used for training. Therefore, each vector is used as a test sample exactly once.

Results of the Malware Sample Data Set. Experiments were firstly carried on the malware sample data set with feature vector size (pruning size) 50. The performance, in terms of effectiveness and efficiency, of various statistical classifiers are listed in

Table 6. All these classifiers work very well with high positive rate ($> 93\%$) and low false positive rate. This provides very strong evidence that the randomness profile plays a significant role in a packer classification system.

Among all classifiers, the kNN classifier achieves the best overall performance. Its TP rate is 99.6% and FP rate is only 0.1%. Moreover, it takes least time to build a model on training data (*Model building time* in Table 6).

Table 6. Comparison of statistical classifiers. The feature vector contains 50 points.

Classifier	TP rate	FP rate	Model building time (s)
Bayes.NaiveBayes	93.9%	1.1%	0.91
Functions.SMO	98.9%	0.8%	72.11
Lazy.kNN(k=1)	99.6%	0.1%	0.02
Trees.BFTree	99.3%	0.5%	16.45

Two other sets of experiments are used to determine the k values used in the kNN classifier and the size of the extracted feature vector. Table 7 shows that $k = 1$ outperforms other two k values, 3 and 5. Table 8 shows feature vectors of 30-50 points all achieve high effectiveness (TP $> 99\%$) while feature vector of 50 points yields the best result.

Table 7. Comparison of kNN with different k values. The feature vector contains 50 points.

k	TP rate	FP rate	Model building time (s)
1	99.6%	0.1%	0.02
3	99.5%	0.2%	0.02
5	99.4%	0.3%	0.02

Table 8. Comparison of kNN (k = 1) with different feature vector size

Vector size	TP rate	FP rate	Precision	Recall
30	99.4%	0.3%	99.4%	99.4%
40	99.6%	0.1%	99.6%	99.6%
50	99.6%	0.1%	99.8%	99.7%

Results of the Mixed Sample Data Set. The above results of malware sample data set shows that our packer classification system can achieve extremely effective performance using the pattern recognition techniques. Through the experiments, it is proved that this novel packer classification technique works well with different packer variants. However, it was unknown whether the conclusions drawn in previous sections can be applied to packers with different level of sophistication. To address this, an experiment was run on the mixed sample data set. This data set contains not only packed clean files and malware samples, but also lowly complex packers and highly complex packers.

50 randomness values are extracted from the file to construct the feature vector. In the classification process, the kNN (k=1) classifier is used. The results in Table 9 show that sophisticated packers, such as Asprotect and Themida, can also be effectively classified by applying the pattern recognition techniques on packer's randomness profile. As shown, the average TP rate is 99.4%. Among 9 packers, the TP rates of Upack and FSG obtain nearly perfect while all other packers achieve more than 90%.

Table 9. Detailed accuracy by class using the mixed sample data set. The classifier is kNN with $k = 1$ and the feature vector contains 50 points.

Packer	TP rate	FP rate	Precision	Recall
ASPROTECT	92.7%	0.1%	95.5%	92.7%
FSG	99.9%	0.0%	99.9%	99.9%
MEW	99.6%	0.0%	100.%	99.6%
NSPACK	91.0%	0.1%	90.7%	91.0%
PECOMPACT	98.5%	0.1%	98.2%	98.5%
PETITE	98.0%	0.0%	98.7%	98.0%
THEMIDA	92.3%	0.1%	86.4%	92.3%
UPACK	100.%	0.0%	99.4%	100.%
UPX	99.7%	0.3%	99.8%	99.7%
Weighted Avg	99.4%	0.2%	99.5%	99.4%

5 Conclusions and Future Work

This paper has discussed packers and presented a *fast yet effective* packer classification system which applies pattern recognition techniques. In this approach, the low randomness profile of the packer is extracted and then passed to a statistical classifier.

Our work demonstrates that the randomness profile combined with strong pattern recognition algorithms can be used to produce a highly accurate packer classification system on real life data. Such a system identifies the packer automatically and therefore is essential to keeping up with the accelerating growth in packer varieties.

The system has been tested on a large data set, including clean packed files and more than $17,000$ malware samples from the wild. The data set has a wide coverage of packers since that files are packed by not only different versions of packers, but also packers of different complexity. We evaluated four popular fast statistical classifiers, namely Naive Bayes, Sequential Minimal Optimization, $k-$Nearest Neighbor and Best-first Tree. All four classifiers were extremely effective, three of the four algorithms achieved an average true positive rate of around 99% or above, Naive Bayes was the lowest, with a true positive rate of around 94%. The $k-$Nearest Neighbor classifier with $k = 1$ obtains the best overall performance. Its true positive rate is 99.6% while false positive rate is 0.1%. Moreover, it is the fastest classifier.

The system also reveals that the low randomness profile of the packed file, normally produced by the PE header and unpacking stub, contains important packer's information. Thus it is very useful in distinguishing between families of packers.

Following the very encouraging preliminary results described here, there are several other promising steps which can be undertaken. Our future work will focus on the exploration of different statistical classifiers' performance as the main tool in a packer classification system. Several other classifiers can be added to the list, such as Random Forest, other Bayesian methods, Bagging, etc. Furthermore, we can apply various multi-classifier algorithms [33] or rank the output of different classifiers instead of choosing the best one among them.

It is still unknown whether there are a set of attributes more important than others in the extracted profile. We need to make enhancements to the feature extraction algorithm to select most important features from the randomness profile. Useful packer features other than the randomness profile, such as PE header information, string information, can also be explored and incorporated into the feature vector to improve the system's effectiveness. Moreover,new pruning methods that consider not only low randomness values but also certain sections can be developed. For example, we can only retain the lowest values in the code section.

Acknowledgements

The authors wish to thank Dr. Tim Ebringer for his important contributions during discussions related to this work.

References

1. The WildList Organization International: WildList, http://www.wildlist.org/
2. Brosch, T., Morgenstern, M.: Runtime Packers: The hidden problem? Black Hat USA (2006), http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf
3. Bustamante, P.: Mal(ware)formation Statistics (2007), http://research.pandasecurity.com/malwareformation-statistics/
4. Morgenstern, M., Marx, A.: Runtime Packer Testing Experiences. In: 2nd International CARO Workshop (2008), www.datasecurity-event.com/uploads/runtimepacker.ppt
5. Ebringer, T., Sun, L., Boztaş, S.: A Fast Randomness Test that Preserves Local Detail. In: Proceedings of 18th Virus Bulletin International Conference, pp. 34–42 (2008)
6. Pietrek, M.: An In-depth Look into the Win32 Portable Executable File Format (2002), http://msdn.microsoft.com/msdnmag/issue/02/02/PE/print.asp
7. Ferrie, P.: Anti-unpacker Tricks Current. In: 2nd International CARO Workshop (2008), http://www.datasecurity-event.com/uploads/unpackers.pdf
8. Ferrie, P.: Anti-unpacker Tricks 2 Part One. Virus Bulletin, 4–8 (December 2008)
9. Ferrie, P.: Anti-unpacker Tricks 2 Part Two. Virus Bulletin, 4–9 (January 2009)
10. Ferrie, P.: Anti-unpacker Tricks 2 Part Three. Virus Bulletin, 4–9 (Febuary 2009)
11. Ferrie, P.: Anti-unpacker Tricks 2 Part Tour. Virus Bulletin, 4–7 (March 2009)
12. VMware workstation, http://www.vmware.com/products/ws/
13. PEiD, http://www.peid.info/
14. Carrera, E.: pefile, http://code.google.com/p/pefile/
15. Kephart, J.O., Sorkin, G.B., Arnold, W.C., Chess, D.M., Tesauro, G.J., White, S.R.: Biologically Inspired Defenses against Computer Viruses. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 985–996 (1995)
16. Tesauro, G.J., Kephart, J.O., Sorkin, G.B.: Neural Networks for Computer Virus Recognition. IEEE Expert 11(4), 5–6 (1996)
17. Siddiqui, M.A.: Data Mining Methods for Malware Detection. Master's thesis, University of Central Florida, Orlando (2008)
18. Kolter, J.Z., Maloof, M.A.: Learning to Detect and Classify Malicious Executables in the Wild. JMLR 7, 2699–2720 (2006)

19. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data Mining Methods for Detection of New Malicious Executables. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 38–49 (2001)
20. Cohen, W.W.: Learning Rules that Classify E-mail. In: Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access, pp. 18–25 (1996)
21. Sahami, M., Dumais, S., Heckerman, D., Horvitz, E.: A Bayesian Approach to Filtering Junk E-mail. AAAI Technical Report WS-98-05, pp. 55–62 (1998)
22. Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C.D., Stamatopoulos, P.: Learning to Filter Spam E-mail: A Comparison of a Naive Bayesian and a Memory-based Approach. In: Proceedings of Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), pp. 1–13 (2000)
23. Androutsopoulos, I., Koutsias, J., Chandrinos, K.V., Spyropoulos, C.D.: An Experimental Comparison of Naive Bayesian and Keyword-based Anti-spam Filtering with Encrypted Personal Messages. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 160–167 (2000)
24. Perdisci, R., Lanzi, A., Lee, W.: Classification of Packed Executables for Accurate Computer Virus Detection. Pattern Recognition Letters 29(14), 1941–1946 (2008)
25. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill Book Co., New York (1983)
26. Frakes, W.B., Baeza-Yates, R.: Information Retrieval: Data Structures and Algorithms. Prentice Hall, Englewood Cliffs (1992)
27. van Rijsbergen, C.J.: Information Retrieval, Butterworths (1979)
28. Syring, K.M.: GNU Utilities for Win32 (2004), `http://unxutils.sourceforge.net/`
29. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
30. Holmes, G., Donkin, A., Witten, I.H.: Weka: A Machine Learning Workbench. In: Proceedings of 2nd Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia (1994)
31. Weka, `http://www.cs.waikato.ac.nz/~ml/weka/`
32. Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: IJCAI, pp. 1137–1145 (1995)
33. Chou, Y.Y., Shapiro, L.G.: A Hierarchical Multiple Classifier Learning Algorithm. In: Proceedings of 15th International Conference on Pattern Recognition (ICPR 2000), vol. 2, pp. 2152–2155 (2000)
34. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson Education, Inc., London (2006)
35. Zhang, H.: The Optimality of Naive Bayes. In: FLAIRS Conf. (2004)
36. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based Learning Algorithms. Machine Learning 6(1), 37–66 (1991)
37. Burges, C.J.C.: A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2, 121–167 (1998)
38. Platt, J.C.: Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Microsoft Research (1998)
39. Quinlan, J.R.: Induction of Decision Trees. Machine Learning 1(1), 81–106 (1986)
40. Shi, H.J.: Best-first Decision Tree Learning. Master's thesis, The University of Waikato (2007)
41. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
42. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. Wadsworth, Monterey (1984)

A Pattern Recognition Algorithms

A.1 The Naive Bayes (NB) Algorithm

The NB classifier [34] uses a statistical approach to the problem of pattern recognition. The Bayes rule is the fundamental idea behind a NB classifier. For a feature vector $\mathtt{x}$ with n attributes $\mathtt{x} = \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ and a class variable y_j, let $P(\mathtt{x}|y_j)$ be the class-conditional probability for the feature vector $\mathtt{x}$ whose distribution depends on the class y_j. Then $P(y_j|\mathtt{x})$, the *posteriori* probability that feature vector $\mathtt{x}$ belongs to class y_j can be computed from $P(\mathtt{x}|y_j)$ by Bayes rule:

$$P(y_j|\mathtt{x}) = \frac{P(\mathtt{x}|y_j) \times P(y_j)}{P(\mathtt{x})} \tag{5}$$

The NB algorithm applies "naive" conditional independence assumptions which states that all n features $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ of the feature vector $\mathtt{x}$ are all conditionally independent of one another, given y_j. The value of this assumption is that it dramatically simplifies the representation of $P(\mathtt{x}|y_j)$, and the problem of estimating it from the training data. In this case,

$$\begin{aligned} P(\mathtt{x}|y_j) &= P(\mathtt{x}_1 \ldots \mathtt{x}_n|y_j) \\ &= \prod_{i=1}^{n} P(\mathtt{x}_i|y_j) \end{aligned} \tag{6}$$

and equation (5) becomes

$$P(y_j|\mathtt{x}) = \frac{P(y_j)\prod_{i=1}^{n} P(\mathtt{x}_i|y_j)}{P(\mathtt{x})} \tag{7}$$

In a classification system, the feature vector $\mathtt{x}$ belongs to the class y_j with the highest probability $P(y_j|\mathtt{x})$. Since $P(\mathtt{x})$ is always constant for every class y_j, it is sufficient to choose the class that maximizes the numerator in (7), $P(y_j)\prod_{i=1}^{n} P(\mathtt{x}_i|y_j)$. In other words,

$$y_{max} = \arg\max_{y_m} P(y_j) \prod_{i=1}^{n} P(\mathtt{x}_i|y_j) \tag{8}$$

The NB classifier is easy to implement and can be trained very efficiently in a supervised learning setting, computation time varies approximately linearly with the number of training samples. Despite the apparent over-simplified assumptions of independence, the NB classifier often competes well with more sophisticated classifiers [35].

A.2 The $k-$Nearest Neighbor (kNN) Algorithm

k-Nearest Neighbor [36] is amongst the simplest of all machine learning algorithms. The idea behind it is quite straightforward. To classify the test packed file, the system firstly finds the k training files which are the most similar to the attributes of the test file.

These training files are called *Nearest Neighbors* as they have the shortest distance to the test file. Then the test file is categorized based on the category of its nearest neighbors. In the case where neighbors belong to more than one class, the test is assigned to the majority class of its nearest neighbors.

As shown in Figure 2, the test file (green cross) can be classified as the first class of the red plus sign or the second class of the blue minus sign. If set k to 1, the test file will be classified as the class of red plus sign since it is inside the inner circle. Similarly, if set k to 3, the test file will be classified as the class of the blue minus sign as there are two blue minus signs but only one red plus sign inside the outer circle.

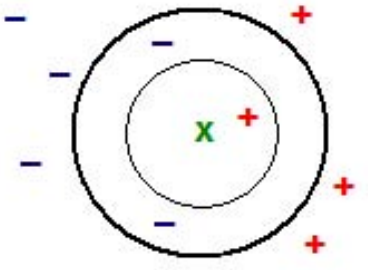

Fig. 2. Examples of kNN algorithm

A.3 The Sequential Minimal Optimization (SMO) Algorithm

SMO is a fast implementation of Support Vector Machines (SVM) [37]. Given data of two classes as two sets of feature vectors in an $n-$dimensional space, a SVM constructs an optimal hyperplane that separates a set of one class instances from a set of other class instances and maximizes the margin between the two data sets. That is to say if two parallel hyperplanes are constructed, one on each side of the hyperplane and passing through the nearest data point in each data class, the distance between the parallel hyperplanes needs to be as far apart as possible while still separating the data into two classes.

Training a SVM is slow due to solving a very large quadratic programming (QP) optimization problem. SMO [38] decomposes the large QP problem of SVM into QP sub-problems, which can be solved analytically and thus avoids using an entire numerical QP as an inner loop. In addition, SMO requires no extra matrix storage at all therefore the amount of memory required is linear in training set size. It allows SMO to handle very large training sets.

A.4 The Best-First Decision Tree (BFTree) Algorithm

The BFTree algorithm is a decision tree [39] that maps from attributes of an item to conclusions about its target class. In a tree that describes a set of packed files, each internal node represents a test on a file feature, each branch from a node corresponds to a possible outcome of the test, and each terminal node contains a packer class prediction. In each step of tree expansion, the best-first top-down strategy is applied [40]. i.e. the "best"

node which maximally reduces the impurity (e.g. information [41] and Gini index [42]) among all nodes available for splitting, is added to the tree first. This partitioning of the feature space is recursively executed until all nodes are non-overlapping or a specific number of expansions is reached. For the latter, pruning methods are used to decrease the noise and variability in the data and therefore to achieve better performance.

B Sliding Window Randomness Test

As described in [5], there are four steps involved in a sliding window version of local randomness test (see Algorithm 1). Firstly, for each file, all bytes are counted and a byte-frequency histogram is built. Secondly, using the global bytes information, the Huffman tree is constructed for the entire file by inserting bytes into the tree in the order of increasing frequency. Thirdly, a "length-encoding" array $e_{B_0}, \ldots, e_{B_{255}}$ is constructed where B_i is the corresponding byte. The entries of the array give the distance to the root of the tree for each element. Thus e_{B_0} gives the distance to the root for the byte B_0, e_{B_1} gives the distance to the root of the byte B_1, and so on. This distance is also the number of bits needed to encode this byte, in the prefix-free Huffman code. At the end, we set a window size w and a skip size a, so that there are a total $\lceil \frac{n-w}{a} \rceil$ windows (indexed by $1, 2, \ldots, \lceil \frac{n-w}{a} \rceil$) for the whole file. The randomness value r_i in each window is calculated as $\sum_{j=a(i-1)+1}^{a(i-1)+w+1} e_{b_j}$, i.e., as the total code length of the corresponding data. At the end, the r_i are scaled so that the minimum value is zero and the maximum value is one.

Data : The packed file in the form of bytes $b_1, \ldots, b_j, \ldots, b_n$ where $b_j \in \{B_0, B_1, \ldots, B_{255}\}$, a window size w and a skip size a

Result : An array of $\lceil \frac{n-w}{a} \rceil$ samples of the randomness, ranging from 0.0 to 1.0

begin

1. Build a byte-frequency histogram for all bytes $B_0, B_1, \ldots, B_{255}$ in the entire file ;
2. Construct the Huffman tree by inserting bytes into the tree in the order of their frequency ;
3. Construct an array $e_{B_0}, \ldots, e_{B_{255}}$ containing the encoding length for each of the input bytes.

for i from 1 to $\lceil \frac{n-w}{a} \rceil$ **do**

Set the randomness value $r_i \longleftarrow \sum_{j=a(i-1)+1}^{a(i-1)+w+1} e_{b_j}$;

endfor

4. **for** i from 1 to s **do**

Rescale r_i between 0.0 and 1.0, where $\min(r_i) = 0.0$ and $\max(r_i) = 1.0$;

endfor

end

Algorithm 1. The sliding window algorithm: generate randomness measurements for a file, output proportional to file length. This is a revised version of Algorithm 2 in [5].

C Determination of Window Size w and Skip Size a

Two sets of experiments have been initially carried out to determine the window size w and the skip size a for the sliding window algorithm. The data set used in the experiments comprises of a total of 708 packed clean files of six packers. For each packer, each file in the UnxUtils binaries [28] is packed with this packer. The selected collection contains 118 executable files whose file size ranged from 3 to 1058 KB, though most files (116 out of 118) are in the range $3 - 191KB$. Six packers used are FSG 2.0, Mew 11, Morphine 2.7, RLPack 1.19, Upack 0.399 and UPX 2.03w.

As shown in [5], the Cosine measure combined with Trunk pruning gives the best performance. We therefore use this combination in the following experiments to find the best parameter settings. Five files are removed from the data set as they produce more than n same smallest randomness values, which are all rescaled to 0. As the result, there are total 703 files in each experiment.

In each set of experiments, the total number of original bytes used for the feature vector remains roughly the same. In other words, similar features of the file are used for packer classification. For example, when test the window size w, the skip size is set to $\frac{w}{2}$ and the pruning size n will vary as w changes. Consider a run with $w = 32, a = 16$ and $n = 100$, the number of bytes from the file used is are around $100 \times 16 + 32 = 1632$. As another example, if w is set to 16, then $a = 8$ and n will be set to 200 ($200 \times 8 + 16 = 1616$).

The results are illustrated in Table 10 and 11. For w, the true positive rate of window size of 8 and 16 are slightly higher than 32. However, windows of small size carries less flexible information than big size does. Besides, feature extraction efficiency is also a factor when selecting the window size. The smaller the window size, the more randomness outputs are generated and the more time it takes. Therefore, 32 is a suitable window size used in the algorithm for packer classification purposes. If the experimental results are examined with respect to the skip size a, it is noted that when using a similar amount of information for comparison, the performance of different systems are close. In Table 11, if windows do not overlap, that is, $a = w = 32$, the system achieves slightly high TP rate than others. In this case, the total number of outputs of the randomness test and the extracted feature vector size are the smallest, so the system performs most efficiently as both of the feature extraction process, including the scaling and pruning, and the classification process is fast.

Table 10. Determination of the window size w, where the skip size $a = w/2$, the distance measure is *cosine* measure and the pruning method is *Trunk*. The pruning size varies with different window size so that that the extracted features used for comparison are roughly same.

Window size	Skip size	Pruning size	Total files	TP rate
8	4	400	703	98.29%
16	8	200	703	98.29%
32	16	100	703	98.15%
64	32	50	703	97.29%

Table 11. Determination of the skip size a, where the window size $w = 32$, the distance measure is *cosine* measure and the pruning method is *Trunk*. The pruning size varies with different skip size so that the extracted features used for comparison are roughly same.

Skip size	Pruning size	Total files	TP rate
2	800	703	98.00%
4	400	703	98.15%
8	200	703	98.15%
12	134	703	98.15%
16	100	703	98.15%
20	80	703	98.00%
32	50	703	98.29%

D Determination of Pruning Size n

This set of experiments is used to determine how detailed the information used for classification should be. This depends on the feature vector size (pruning size n). If n is small, the classification takes less time building the model and classifying the file. However if n is too small, there may not be sufficient information to distinguish between packers. If n is too large, the classification process is slow and also information noise generated by compressed/encrypted data will effect system performance. Therefore, experiments are run with various pruning size n in the range of $30 - 70$ and other settings given in Table 12. Thus, the information used are around $1 - 2$ KB. Table 12 suggests that n should be set between $30 - 50$.

Table 12. Determination of the pruning size n, where the window size $w = 32$, the skip size $a = 32$, the distance measure is *cosine* measure and the pruning method is *Trunk*

Pruning size	Total files	TP rate
30	703	98.57%
40	703	98.57%
50	703	98.29%
60	703	95.87%
70	703	96.01%