Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

On the adoption of anomaly detection for packed executable filtering

*Xabier Ugarte-Pedrero**, *Igor Santos*, *Iván García-Ferreira*, *Sergio Huerta*, *Borja Sanz*, *Pablo G. Bringas*

S³Lab, DeustoTech — Computing, Deusto Institute of Technology, University of Deusto, Avenida de las Universidades 24, 48007, Bilbao, Spain

ARTICLE INFO

Article history:
Received 23 January 2014
Received in revised form
12 March 2014
Accepted 28 March 2014

Keywords:
Malware
Packer
Anomaly detection
Machine-learning
Computer security

ABSTRACT

Malware packing is a common technique employed to hide malicious code and to avoid static analysis. In order to fully inspect the contents of the executable, unpacking techniques must be applied. Unfortunately, generic unpacking is computationally expensive. For this reason, it is important to filter binaries in order to correctly handle them. In previous work, we proposed the adoption of anomaly detection for the classification of packed and not packed binaries using features based on the Portable Executable structure. In this paper, we extend this work and thoroughly evaluate the method with a different dataset and two different feature sets, rendering new conclusions. While anomaly detection is reaffirmed as a sound method for the discrimination of packed and not packed binaries, Portable Executable structure based features present limitations to distinguish custom packed files from not packed files.

Introduction

Malware is the term used to designate software that was coded with malicious intentions, such as damaging computers or networks or even obtaining economic benefit in an illegitimate way. Security products such as Anti-Virus solutions and operating systems have evolved in order to detect and prevent the infection and execution of this kind of software. Consequently, malware writers have developed new techniques to evade detection. A very common technique is software packing, which consists of compressing or encrypting the malicious code, impeding the disassembly of the protected code. This content is then decrypted at runtime, prior to its execution. Some reports claim that up to the 80% of the malware analysed is packed (McAfee, 2009). Packed malware can be analysed with traditional automated dynamic execution techniques that explore the real functionality of the binary. Nevertheless, these techniques usually do not cover every possible execution path. In fact, many malware samples present complex functionality that cannot be easily triggered in an automated execution. In these cases, the code must be statically analysed in order to discover all its functionality, making necessary to unpack the sample. When the packer used to protect the sample is known, specific unpacking routines can be applied to extract the original code. On the contrary, for unknown packers it is necessary to generically unpack the code (according to Morgenstern and Pilz, 2010, 35% of malware is packed by a custom packer). A

* *Corresponding author*. Tel.: +34 944139000.
E-mail addresses: xabier.ugarte@deusto.es, xabiugarte@gmail.com (X. Ugarte-Pedrero), isantos@deusto.es (I. Santos), ivan.garcia.ferreira@deusto.es (I. García-Ferreira), shuerta@deusto.es (S. Huerta), borja.sanz@deusto.es (B. Sanz), pablo.garcia.bringas@deusto.es (P. G. Bringas).
http://dx.doi.org/10.1016/j.cose.2014.03.012
0167-4048/

correct classification of samples can help the analyst to correctly handle binaries.

Previous approaches have applied supervised machine-learning techniques for the classification of packed and not packed binaries using different heuristics (Perdisci et al., 2008b). Nevertheless, supervised approaches learn from both classes: packed and not packed files. Alternatively, anomaly detection methods can be applied in cases in which it is not reliable to model one of the classes. In this context, we consider that it is more realistic to collect a representative dataset of not packed samples, considering that packed binaries can present a higher variability. On the one hand, new packers are developed continuously. Malware creators can also employ modified versions of existing packers, or even custom made protection engines. On the other hand, common compilers normally follow standard conventions to form the resulting binaries. Following this intuition, we propose the application of a distance-based anomaly detection approach to classify packed and not packed binaries. More concretely, we evaluate two different feature sets, based on the Portable Executable structure and operational code frequency, and we apply a data reduction approach and evaluate different distance measures and distance selection rules.

In order to conduct this study, we define the following research questions:

- Which is the feature set that best discriminates packed from not packed files?
- What is the impact of the data reduction approach over the results obtained?
- What is the impact on the results of the different distance measures evaluated?
- What is the impact on the results of the different distance selection rules?
- Does our anomaly detection approach present sound results for the classification of packed and not packed files?

Finally, we discuss how these findings can be useful for the deployment of a binary filtering system in different contexts.

In previous work (Ugarte-Pedrero et al., 2011, 2012) we proposed a similar method for the classification of packed binaries. Nevertheless, this paper extends this work in several manners.

- We measure the appropriateness of different groups of features based on the Portable Executable structure for the classification of packed and not packed binaries. To this end, we test the performance for several common supervised machine-learning algorithms.
- We present a new threshold selection approach and a new normalization process for the anomaly detection method proposed in previous work, in order to avoid considering any data regarding packed samples for the classification.
- We extend the experiments by considering two different approaches for the data reduction approach (discarding or including outliers).
- We evaluate our approach over two different feature sets. In previous work, we tested a Portable Executable structure based feature set. In these experiments, we have considered a new feature set based on operational code frequency.
- We evaluate the method over a new dataset that (i) has been sanitized and (ii) includes custom packed binaries.

The remainder of this paper is structured as follows. Section Dataset selection describes the process followed to select the dataset. Section Feature selection details the feature sets employed for classification. Section Distance-based anomaly detection describes the anomaly detection method proposed. Section Evaluation presents the results obtained for the different experiments conducted. Section Conclusions and discussion describes and discusses the conclusions obtained from the experiments, and outlines avenues for future work. Finally, Section Related work compares this work with most related publications.

Dataset selection

In order to evaluate the adoption of anomaly detection for the classification of packed binaries, we configured a set of 4000 binaries.

The possible biases and limitations of the dataset were thoroughly studied and discussed. Nevertheless, the intrinsic nature of packers, the efforts of malware creators to evade detection, and the limitations of already existing tools make difficult to discriminate packed and not packed files. Actually, Royal et al. (2006) formulated the task of determining the existence of an unpack-execute process as an undecidable problem.

The possible risks to the validity of the experiment were reduced to the extent possible by defining a methodology for binary selection and labelling.

In this way, the dataset must fulfil several requirements. First, it must contain both goodware and malware for both packed and not packed classes, in order to ensure that the model discriminates packed files from not packed ones avoiding possible biases. In addition, the variability must be ensured to guarantee the inclusion of samples from different origins (e.g., system files, common tools ...), generated by different compilers. Secondly, different types of packers must be considered. On the one hand, off-the-shelf packers use very different techniques to protect samples. While some of them are simple compressors, others employ encryption and anti-analysis techniques or even instruction-set virtualization. On the other hand, current malware also employs custom packers (i.e., custom made protection), using a legitimate file as a carrier, making detection more difficult.

In order to ensure that all kinds of packers are represented in the dataset, different kinds of commercial packers and custom packers must be included. Finally, all the samples included in the dataset should be correctly labelled. This is usually the most difficult task when creating a dataset, and sometimes it is necessary to assume the existence of noise in it. In order to minimise possible errors in the labelling process, it is important to consider the actual limitations of the tools employed for the analysis.

Several tools, such as PEiD, identify known packer signatures by searching for common fingerprints in the headers and the unpacking stub of the packer. Nevertheless, malware writers sometimes modify their samples to evade signature-

based detection. These tools also provide heuristic detection capabilities, looking for certain suspicious properties. Similarly, these heuristics can also be evaded applying different techniques. Some generic unpackers try to detect an unpack-execute process in order to determine the correct moment to dump the memory process which contains already unpacked code (e.g., identification of the execution of previously written memory addresses at different granularity levels Kang et al., 2007; Stewart, 2006). This characteristic can be considered a dynamic detection mechanism. Nevertheless, many generic unpackers can be easily evaded employing a wide variety of anti-debugging, anti-vm, anti-emulation or anti-dumping mechanisms. Moreover, the unpacked sample resulting from the unpacking step cannot be used in our experiments, considering that these tools sometimes reconstruct or modify the header of the file.

Considering these aspects, we first obtained binary files from different sources in order to conform the dataset.

- **Malware samples.** We first obtained a set of 13,173 malware samples from VxHeavens. 6216 of these samples were reported by PEiD as packed with well-known packers, while the other 6957 samples were not detected by PEiD. In addition, we obtained 1548 samples from the Zeus malware family provided by the Spanish security company S21Sec. These samples were divided into 8 different groups according to their versions, and were gathered between 2009 and 2011.
- **Goodware samples.** Secondly, we obtained unique legitimate executable files from a Microsoft Windows XP installation with several tools installed such as text editors, internet browsers or ofimatic tools. In addition, in order to select these samples, we analysed them with PEiD to finally select a set of 1645 samples not reported by PEiD as packed. In order to ensure that all the samples were unique, their MD5 hash was computed and compared.

Secondly, we selected a set of packers to manually protect a group of samples. More concretely, we selected 10 common packers representing different kinds of packing techniques: *Armadillo*, *Asprotect*, *FSG*, *MEW*, *Packman*, *RLPack*, *UPX*, *Themida*, *TELock*, *SLV*.

Finally, we selected the samples for each of the groups that conform the dataset.

- **Manually packed malware samples.** We randomly selected a set of 500 malware samples that were reported by PEiD as not packed and manually packed them with 10 different packers.
- **Custom packed malware samples.** In order to include in the dataset custom packed malware samples, we considered the Zeus family. According to Wyke (2011), Zeus uses custom packers for the protection of the binaries as a first layer of protection. Additionally, these binaries are sometimes protected by a second layer of protection with a known packer. Given this background, we first selected 1000 Zeus samples for which no known packer was detected by PEiD. Given the low probability of these modern samples being not packed, we considered them as packed by custom packers or modified versions of known packers.
- **Manually packed goodware samples.** Additionally, we manually packed 500 not packed goodware samples with the same off-the-shelf packers used to protect malware samples.
- **Not packed malware samples.** First, we randomly selected 1000 unique malware samples that were reported by PEiD as not packed. In order to reduce the risk of including packed samples (not detected by PEiD) in the not packed set, we performed an additional step in order to discard possibly packed files. First, we filtered suspicious files by applying heuristic rules: file entropy >7 (following the confidence intervals proposed by Lyda and Hamrock, 2007), number of import address table entries < 5, number of imported dlls $<= 2$, and ratio of standard sections $<= 0.5$. Secondly, we manually analysed the filtered samples with PEiD and StudPE making use of the heuristic analysis capabilities of PEiD, and the overview of the file structure provided by StudPE, in order to label them as packed or not, discarding the most suspicious samples.

In this way, 183 of the 1000 samples were considered suspicious and 74 were labelled as packed after manual analysis. The majority of these samples were modified versions of known packers such as UPX, NSpack or Mew, among others. The 74 packed files were discarded, and another set of 150 unique samples was randomly selected. The same process was followed, identifying 38 suspicious samples, from which 26 were actually packed. Finally, 74 samples were randomly selected from this last set in order to substitute the initially discarded samples.

- **Not packed goodware samples.** Finally, the remaining 1145 goodware samples were inspected to form not packed goodware dataset. Following the same process applied to not packed malware samples, we proceeded to select the samples suspicious of being packed. In this case, 104 samples were suspicious of being packed. We manually inspected the samples with PEiD and StudPE, but finally did not discard any of the samples. Moreover, some of the samples were installer applications, which could eventually be considered as packed. Nevertheless, we maintained them in the dataset considering that these samples do not belong to the kind of runtime packers that we seek to discriminate, and thus provide a more realistic sample distribution.

Feature selection

In order to test the adoption of anomaly detection for the classification of packed and not packed binaries, we consider a set of features based on the structure of the executable. Some of these features are based on heuristics and have been employed in previous work. In addition, we also consider a feature set based on the statistical frequency of bytes in the executable file.

Executable structure based features

The features included in this study can be divided into two categories: heuristics and structural features. On the one

hand, we consider the heuristics proposed in previous work by Perdisci et al. (2008a,b) and propose a set of complementary heuristics that can be efficiently extracted from the file structure. On the other hand, we propose the adoption of PE header features. In order to test the capacity of the different feature sets to discriminate packed and not packed binaries, we evaluated the performance of different machine-learning classifiers and apply the Friedman test in order to determine if there is a statistically significant difference in the results when different feature sets are employed.

Heuristics

First, we adopt a baseline set of heuristics employed in previous work. In this way we include entropy-related features (file entropy, header entropy, average entropy of code sections, and average entropy of data sections), number of standard and not standard sections (i.e., sections that follow the conventions defined by Microsoft), number of executable sections, number of readable, writeable and executable sections, and number of external functions imported.

Secondly, we consider several complementary heuristic values. These heuristics are divided in 3 groups: general heuristics, heuristics related to the section of entry point, and heuristics related to entropy.

General complementary heuristics.

- *Maximum raw data per virtual size ratio.* Maximum ratio among all the sections.
- *Minimum raw data per virtual size ratio.* Minimum ratio among all the sections.
- *Ratio of sections with virtual size greater than raw data.*
- *Number of imported DLLs.*
- *Ratio of readable and executable sections.*
- *Ratio of readable and writeable sections.*

In addition, in order to make the initial set of heuristics proposed by Perdisci et al. (2008a, b) conformant with the rest of heuristics, we normalise the values considering the number of sections present in the executable file. Regarding the heuristic corresponding to the ratio of sections with execute permissions, we consider it a structural feature. In order to avoid the repetition of redundant features, we include this feature in the structural feature group.

- *Ratio of standard sections.*
- *Ratio of not standard sections.*
- *Ratio of readable, writeable and executable sections.*

Complementary heuristics related to the section of entry point

- *Entry point outside any section.* Indicates that the entry point of the PE file does not point to any section in particular. In these cases, all the features related to the section of entry point are considered missing values.
- *Section of entry point is standard.* Indicates whether the section of entry point is a standard section.
- *Section of entry point is an Import Address Table section.* Indicates whether the section of entry point is an Import Address Table section.
- *Raw data per virtual size ratio of the section of entry point.* Ratio between raw data and virtual size for the section of entry point.
- Entropy of the section of entry point.

Heuristics related to file entropy

- *Minimum section entropy.*
- *Maximum section entropy.*
- *Average section entropy.*
- *Ratio of sections with entropy in a certain range.* More concretely, the entropy ranges considered where 0 to 0.5, 0.5 to 1, 1 to 1.5, 1.5 to 2, 2 to 2.5, 2.5 to 3, 3 to 3.5, 3.5 to 4, 4 to 4.5, 4.5 to 5, 5 to 5.5, 5.5 to 6, 6 to 6.5, 6.5 to 7, 7 to 7.5 and 7.5 to 8.

PE Header features

- **DOS Header.** All Portable Executable files (PE) start by the DOS header, which is maintained for compatibility reasons. This header ensures that the binary can be executed in a DOS system. The header contains 17 defined fields of 2 bytes, together with 28 bytes of reserved space. As some packers insert data into the reserved space, we have included 14 possible fields of 2 bytes, making a total of 31 fields.
- **COFF File Header.** The COFF File Header is located at the file offset indicated by the field *l_fanew* and is preceded by a signature of 4 bytes corresponding to the ASCII characters 'P' 'E' '\0' '\0'. This header is composed of 7 fields with a size between 2 and 4 bytes. The last field of the header (named *Characteristics*) has a size of 2 bytes and defines 15 valid flags and 1 reserved flag.
- **Optional Header.** If the PE file is an image file, it will contain an optional header, not required in object files. The size of this header is delimited by the corresponding field in the COFF File header. In addition, the *Number of RVA and sizes* field in this header indicates the number of data directory entries contained in the header. The number and size of fields of this header depends on whether it corresponds to a 32 bit or 64 bit image file. For 32 bit files, 30 fields are present in this header. Similarly to the Optional Header, the *Characteristics* field defines 16 possible flags, where 5 are reserved and 11 represent valid values.
- **Data Directories.** The data directory entries are located after the optional header. These entries are pointers to tables in the payload of the file (usually a section of the PE), that are used by the operating system to load the PE and execute it. The number of directory entries is indicated by the field *Number of RVA and sizes* in the optional header, and should not exceed 16. In most cases, this field contains the value 16, but in some cases certain packers like SLV present unusual values, presumably to confuse analysis tools when parsing the header.
- **Sections.** Every PE file can have a variable number of sections. PE sections are used to organize code and data, resources, import and export information, and any other data necessary for the execution of the application. Section headers are located after the Optional Header, at the offset indicated by the field *Size of Optional Header* in the COFF File

Header. Each section is formed by 10 fields. The *Characteristics* field has a size of 4 bytes and defines 11 reserved flags, 16 valid flags, and a 4 bit field that indicates the alignment of the data.

In order to test the capacity of the described header fields to discriminate packed and not packed files, we selected a set of features representing both reserved and valid fields. Accordingly, we defined a separate feature for each flag declared in the *Characteristics* field of each header. A PE file can contain any number of sections. For this reason, we cannot include all the header fields for all the sections. Alternatively, we included in our feature set a group of 36 values that *summarise* all the sections in the binary. In this way, we included 28 features for the 28 different flags that a section can have. Each feature represents the ratio of sections that have the corresponding flag set. Additionally, we included the total and average values of the *raw data size*, the *virtual data size*, *number of relocations*, and *number of line numbers*. The rest of the section header fields are pointers or values that cannot be easily summarised for all sections and were thus discarded.

Finally, we discarded the magic values that are common for every executable file, and the *TimeStamp* field in order to avoid possible biases, specially when manually generated files are included in the dataset. As a result, the set of features extracted was comprised of 9 heuristics used in previous work, 33 additional heuristic values, and 199 structural features (i.e., header fields).

In order to evaluate the capacity of the features proposed for the classification of packed binaries, we tested the performance of different machine learning algorithms for different groups of features. To this aim, we measured the performance of bayesian networks using different learning algorithms (Naïve Bayes, K2, Hill Climber and TAN), support vector machines with different kernels (polynomial, normalised polynomial, PUK and RBF), K nearest neighbours with different k configurations (1, 3 and 5), the C4.5 algorithm, random forest with different number of random trees (10, 30 and 50), rules based classifiers (JRip, and PART), Multilayered Perceptron with one hidden layer of 243 nodes, and Bagging with the 100% of the training set with the C4.5 algorithm. Totally, we tested 19 different configurations.

In order to test the performance of these classifiers, we employed the well-known tool Weka (Garner (1995)), applying K-fold cross validation with a 10X10CV configuration.

Considering that Weka only provides the paired-t test for pairwise comparisons, we applied the statistic proposed by Iman and Davenport (1980) based on the non-parametric Friedman test for the comparison of multiple classifiers over several datasets (further discussion on these approaches was addressed by Demšar (2006)).

The statistic proposed is applied over the average results for the 10 runs and 10 folds for each configuration. Although there are variations of these kind of tests that consider multiple observations per cell, they require the independence of the observations, a property that cannot be assumed when k-fold cross validation is applied.

Finally, we performed a post-hoc analysis based on the Holm's step down procedure in order to contrast the rank of each feature set against its baseline, in order to test the statistical significance of the improvement in each of the performance measures obtained.

Table 1 – Friedman test over accuracy, True Positive Rate (TPR), False Positive Rate and area under the curve (AUC). The ranks for each algorithm and feature set are omitted. The average ranks and the F_F score are shown.

	B	S	BS	BA	BSA	F_F
Acc.	4.5263	3.0526	1.6315	3.7894	2	25.1474
TPR	4.7368	2.8947	2	3.7368	1.6315	32.6074
FPR	4	3.4736	1.4736	4	2.0526	21.5736
AUC	4.3421	3.3684	1.7894	3.6842	1.8157	20.0779

The results obtained (see Tables 1–5) indicate that there is a statistically significant difference on the results when structural features are employed in conjunction with baseline heuristics for the detection of packed binaries.

Additionally, we propose the inclusion of an additional set of heuristics. Results show that, although these heuristics improve the results of several classifiers when considered together with the baseline heuristics, there is not a statistically significant difference for any of the performance measures tested.

When these additional heuristics are added to the baseline heuristics and the structural features, we can draw similar conclusions. Despite in the case of TPR the results obtained by the complete feature set outperform the rest of combinations, in the case of accuracy, FPR and AUC, the best results are obtained when these heuristics are not included.

Furthermore, as there is not a clear difference between the feature set comprising the baseline heuristics and structural features, and the feature set that includes also the additional features, we have also applied the Holm procedure to compare these feature sets, considering as baseline, in each case, the one with the lowest rank. In this way, for the accuracy, false positive rate, and AUC, the results indicate an improvement of baseline heuristics with structural features over the complete feature set with p values of 0.4726, 0.2591 and 0.9591 respectively. This improvement is not statistically significant in any of the three cases at a suitable α level. In the case of true positive rate, the complete feature set outperforms the baseline heuristics with structural features with a p value of 0.4726. Again, this difference is not statistically significant.

As a conclusion, we can affirm that there is a significant improvement in classification performance when structural features are added to the initial set of heuristics proposed in previous work. On the contrary, there is not a statistically

Table 2 – Holm step-down procedure for the Bonferroni–Dunn test over the accuracy of the different classifiers for each feature set.

i	Feature set	$z=(R_0 - R_i)/SE$	p	$\alpha/(k-i)$
1	BS	5.6428	0.0000000167	0.0125
2	BSA	4.9246	0.0000008449	0.0167
3	S	2.8727	0.0040692965	0.0250
4	BA	1.4363	0.1508971723	0.05

(B) baseline heuristics, (S) structural features, (A) additional heuristics.

Table 3 – Holm step-down procedure for the Bonferroni–Dunn test over the True Positive Rate of the different classifiers for each feature set.

i	Feature set	$z=(R_0-R_i)/SE$	p	$\alpha/(k-i)$
1	BSA	5.6428	0.0000000167	0.0125
2	BS	4.9246	0.0000008449	0.0167
3	S	3.1805	0.0014700445	0.0250
4	BA	1.5389	0.1238122238	0.05

(B) baseline heuristics, (S) structural features, (A) additional heuristics.

Table 5 – Holm step-down procedure for the Bonferroni–Dunn test over the AUC of the different classifiers for each feature set.

i	Feature set	$z=(R_0-R_i)/SE$	p	$\alpha/(k-i)$
1	BS	5.3350	0.0000000955	0.0125
2	BSA	5.2837	0.0000001265	0.0167
3	S	2.2571	0.0239985552	0.0250
4	BA	1.6415	0.1006801107	0.05

(B) baseline heuristics, (S) structural features, (A) additional heuristics.

significant difference in the results when the additional heuristics are included. When the additional heuristics are added to the original heuristics, the results obtained outperform the baseline, but not at a statistically significant level. Considering these results, we do not discard the set of additional heuristics due to the lack of strong evidence against this feature set.

Operational code frequency based analysis

Several approaches have addressed malware detection applying machine-learning methods, and *n*-gram frequency based feature sets (Santos et al., 2009; Schultz et al., 2001; Kolter and Maloof, 2004).

Perdisci et al. (2008b) employed this technique not only for the task of discriminating malware from goodware, but also to discriminate between packed and not packed binaries. In fact, when compression or encryption algorithms are applied, it is common to observe high entropy values and flat byte histograms (Sun, 2012). Not packed files, in contrast, present a different byte frequency. One of the reasons for these differences is the prevalence of certain operational codes. Following this idea, Perdisci et al. (2008b) proposed an approach based on the presence of certain *n*-grams for the classification of packed and not packed binaries. In order to select the most relevant *n*-grams or byte combinations, they employed Information Gain (IG) over a set of packed and not packed instances. In this way, they selected the byte sequences that best discriminate both classes.

Unfortunately, IG cannot be calculated if only one of the classes is taken into consideration. Our anomaly detection method avoids taking any assumption about the packed executable class in order to construct a packer-agnostic classification system.

For this reason, we adapt the approaches proposed in previous work for anomaly detection. To this aim, we elaborated a list of all possible operational codes according to the Intel specification for the x86 architecture Intel (2013), and measure the frequency for each possible byte sequence.

The operational codes defined by Intel have a variable length. In fact, the instruction size for this architecture varies from 1 byte to 15 bytes.

For this study, some fields were excluded given that they do not adapt correctly to a simple n-gram model, or because they introduce too much complexity deriving an exponential number of possible different byte sequences. For this reason, floating point instructions were not included due to the limited use of this kind of instructions and the complexity they present. Some instructions use bits 3–5 of the ModR/M byte to encode the instructions. Some byte sequences represent several instructions that are discriminated considering the above-mentioned bits. As the representation of these semantic differences through an *n*-gram model is not straightforward, we omitted the ModR/M byte, assuming that some byte sequences represent, in fact, a group of similar instructions. Accordingly, we defined a set of 823 byte sequences formed by the 2 first fields in the instruction format shown in Fig. 1, with length from 1 to 5 bytes. This list of operational codes was gathered from Tables A-2 to A-5 of the Appendix A of the Volume 2C of the Intel 64 and IA-32 Architectures Software Developer's Manual Intel (2013). This approach, in contrast to a basic *n*-gram model, only considers the subset of *n*-grams for $n=1$, $n=2$, $n=3$, $n=4$ and $n=5$ that represent an Intel instruction. Consequently, the set of possible values is reduced from 1,103,823,438,080 to 823.

Given this background, we define our feature set as a set of tuples such that, for a given binary b, $SF_b=\{(s_{1,b},f_{1,b}),(s_{2,b},f_{2,b}),(s_{3,b},f_{3,b}),\ldots(s_{n,b},f_{n,b})\}$, where $s_{i,b}$ represents every possible sequence, and $f_{i,b}$ is the frequency of that sequence in the file. More concretely, $f_{i,b}$ is the total number of occurrences of the sequence divided by the total number of sequences found for that binary.

Table 4 – Holm step-down procedure for the Bonferroni–Dunn test over the False Positive Rate of the different classifiers for each feature set.

i	Feature set	$z=(R_0-R_i)/SE$	p	$\alpha/(k-i)$
1	BS	5.9506	0.0000000027	0.0125
2	BSA	4.8220	0.0000014206	0.0167
3	S	2.0519	0.0401738703	0.0250
4	BA	1.0259	0.3049017882	0.05

(B) baseline heuristics, (S) structural features, (A) additional heuristics.

Distance-based anomaly detection

In this section, we describe the distance-based anomaly detection method proposed for the classification of packed binaries.

One of the motivations for the adoption of anomaly detection is to avoid the overfitting problems that supervised machine-learning algorithms may present. In this sense, Breiman (1996) pointed out the instability presented by several machine-learning algorithms except K Nearest Neighbours. Distance based approaches present several advantages:

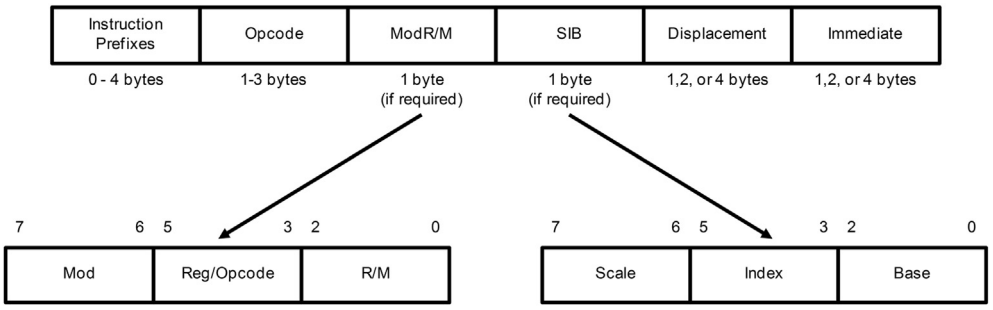


Fig. 1 – Intel 64 and IA 32 instruction format, adapted from the Intel 64 and IA 32 Architectures Software Developer's Manual.

- **No assumptions about the distribution of the data.** Nearest neighbours based approaches do not assume any particular distribution of the data, on the contrary to statistical approaches such as Gaussian models. Instead, they just consider the distance from test instances to the available data.
- **Computational complexity of the distance measures.** The nearest neighbours method allows the adoption of efficient distance measures. In this way, the computational complexity of the approach is $O(N^2)$, depending on the number N of samples that conform the instances representing normality.
- **Data-reduction capabilities.** In order to reduce the computational complexity, we can reduce the number N of instances for comparison applying clustering techniques.
- **Straightforward adoption of anomaly score.** The anomaly score adopted is directly related to the distances measured from the test instance to the rest of instances. The process of adjusting thresholds for different use-cases is straightforward.

Given this background, we propose a distance-based anomaly detection method, combined with clustering techniques for the reduction of data in order to provide an efficient detection of packed samples. In addition, we propose 3 different neighbour selection rules and test them under different conditions.

Representation of normality

In order to represent normality, we use a set of not packed executables. As opposed to other supervised learning approaches, instance-based classification does not need a model training phase. In this way, any sample that deviates sufficiently from a representation of normality (not packed executables) is classified as packed. This method, on the contrary to K Nearest Neighbours, does not measure the distance from the testing instance to packed samples. Despite the fact that the evaluation process of this method requires the use of a previously labelled dataset, its deployment would only require not-packed samples, reducing the efforts needed to find and label a set of representative packed binaries.

In this way, we define our normality model as a set of i points $\mathcal{NP} = \{np_0, np_1, \ldots, np_i\}$ defined in an n-dimensional feature space such that $np_i = \{f_{i,0}, f_{i,1}, \ldots f_{i,n}\}$.

Afterwards, we normalise the instances in our normality model by calculating the well-known standard score or z-score (see Equation (1)), for each instance and feature.

$$z_{i,n} = \frac{x_{i,n} - \mu_n}{\sigma_n} \tag{1}$$

In this way, we obtain a representation of normality $\mathcal{N} = \{\mathcal{NP}', \mathcal{M}, \mathcal{D}\}$ where $\mathcal{NP}'$ is the set of normalised points $\mathcal{NP}' = \{np'_0, np'_1, \ldots, np'_i\}$, each point is defined as a set of normalised feature values such that $np'_i = \{z_{i,0}, z_{i,1}, \ldots, z_{i,n}\}$, $\mathcal{M}$ is a set formed by the means for each feature $\mathcal{M} = \{\mu_0, \mu_1, \ldots, \mu_n\}$ and $\mathcal{D}$ is a set formed by the standard deviations for each feature $\mathcal{D} = \{\sigma_0, \sigma_1, \ldots, \sigma_n\}$.

Distance between files

The classification of an executable consists of 3 different phases: (i) feature extraction, (ii) normalisation with respect to μ and σ of the reference model, and (iii) calculation of the distance from the resulting point to the model.

As a result, the distance measured provides an anomaly-score with respect to the model, allowing to stablish a threshold to discriminate files as packed. In this study, we have considered 2 different distance measures: Manhattan distance and Euclidean distance.

Since we have to compute this measure with respect to a variable number of points representing not packed executables, a combination metric is required in order to obtain a final distance value which considers every measure performed.

Our approach consists of applying 3 different distance selection rules: (i) Mean distance rule, (i.e., average distance to all the points in the model), (ii) Maximum distance rule, (i.e., distance to the least similar point), and (iii) Minimum distance rule, (i.e., distance to the most similar point).

In this way, when an executable is analysed, the final distance value calculated depends on the distance measure and the combination rule selected.

Dataset reduction

The proposed approach requires us to compute as many distance values as executables in the not packed set. For this reason, we improve the efficiency of our system by designing a data reduction phase, consisting in the application of the Quality Threshold (QT) clustering algorithm proposed Heyer

et al. (1999). This clustering algorithm is applied to the original dataset to obtain a reduced version that maintains the original features of the dataset. In this way, the number of comparisons performed, and thus, the comparison time required for the analysis of each sample are much lower.

The second objective is to measure the precision of our system when the training set is incrementally reduced, in order to evaluate the trade-off between efficiency and accuracy. In addition, this data reduction approach enables us to test the performance of the system when a unique representation of a 'normal' executable is used, and to determine if it can correctly classify packed and not packed executables.

This method requires to configure a similarity threshold value to determine the maximum radial distance of any cluster. On the contrary to other approaches such as *K*-means, the number of clusters generated for a dataset depends on this threshold and it must not be specified. For each cluster, we calculate its centroid and add it to the reduced version of the dataset. Finally, the instances not belonging to any cluster can either be included in the final dataset or eliminated. In our experiments, we evaluate both approaches. The main disadvantage of this method is the high number of distance calculations needed. Nevertheless, in our case, this computational overhead is admissible because we only have to reduce the dataset once.

Selection of a threshold

As we have mentioned, the distance from a given binary to the representation of normality can be considered an anomaly score. Nevertheless, the result of this classification process should be a binary attribute in most situations: packed or not-packed (e.g., we may want to decide which samples should be processed with a generic unpacker).

Normally, supervised classification methods adjust the parameters of the model considering all the instances present in the training set, trying to maximise a score function such as classification accuracy. Afterwards, the model is evaluated by classifying the test-set. In our case, the training-set is comprised of only not-packed binaries. Accordingly, several approaches can be considered for the selection of a threshold:

- **Maximum tolerable false positive rate.** This approach tries to approximate the false positive rate of the testing set, using the training set. In order to select this threshold, we randomly extract from the training set a 10% of binaries (i.e., validation set). In this way, we measure the distance from these instances to the rest of the training set, and select the thresholds that produce a fixed number of false positives. These thresholds, in an ideally distributed dataset, will approximate the false positive rate for the testing set. Finally, we can test the associated false negative rate and accuracy in order to evaluate their reliability for the detection of packed binaries.
- **Number of desired positives.** Another possible approach is the selection of the most anomalous instances for inspection, considering that they might present a higher probability of being packed. In this way, depending on the processing capabilities of a hypothetical automatic unpacking system, it would be possible to apply a threshold according to the number of positives it produces. Nevertheless, in order to deploy such approach it is necessary to make some assumptions about the distribution of binaries to analyse. For instance, an Anti-Virus company might want to process a malware database. In such scenario, the majority of samples analysed would be packed, and thus, it would not make sense to accept a low number of binaries as packed. Alternatively, an on-line binary analysis system (e.g., Anubis[1]) can receive binaries from many different sources, and the distribution of packed/not packed binaries might vary.

While the first approach can be easily evaluated, the second approach entirely depends on the deployment scenario. For this reason, we adopt the *maximum tolerable false positive rate* as an evaluation criteria for our anomaly detection method.

Evaluation

In order to evaluate the performance of our anomaly detection system, we applied a variation of k-fold cross validation, with $k = 10$. First, we divided the data set (comprised of 2000 not-packed samples) into 10 groups of 200 binaries. For each fold, 1800 samples were selected as training instances, while 200 were selected for testing. In our approach, packed binaries are not considered for the training phase and thus, for each fold, all the available samples are used for testing.

Afterwards, for each training-set selected, a 10% of the instances (180) were reserved as validation set in order to adjust the possible thresholds. In this way, each fold was configured in the following way:

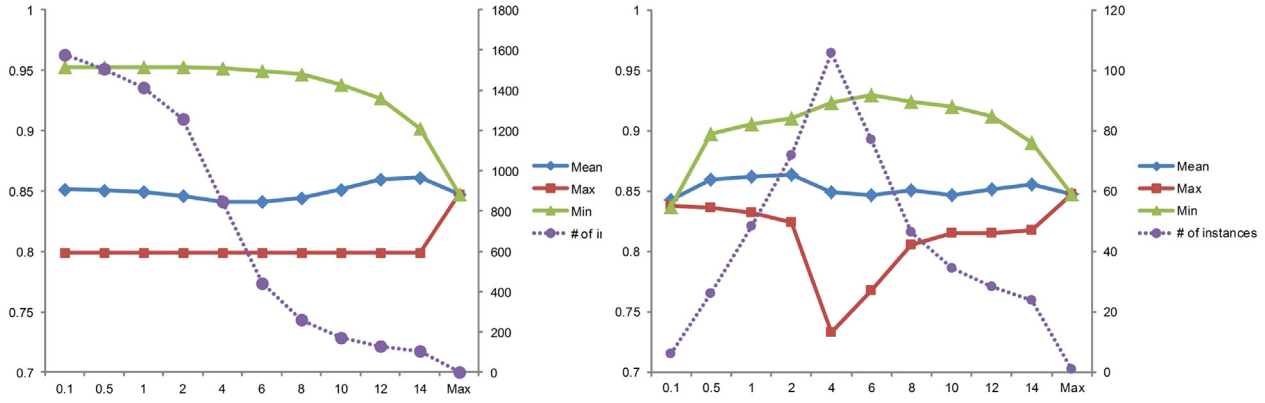
- 1620 not packed samples as training set.
- 180 not packed samples as validation set.
- 2000 packed samples for testing.
- 200 not-packed samples for testing.

We can observe that the testing set is highly unbalanced. For this reason, we evaluate the performance of the approach considering the false positive rate (FPR), false negative rate (FNR), and the area under the curve (AUC), avoiding the accuracy. In this way, the FPR measures the not-packed samples incorrectly classified as packed, the FNR measures the packed samples considered not-packed, and the AUC measures the overall performance of the approach regardless of the threshold selected.

Regarding the data reduction process applied to each configuration, in all cases, the minimum number of instances for each cluster was set to 2. The distance thresholds for each configuration are dependant on the distance measure and feature set employed. In order to correctly represent different reduction rates for each configuration, the thresholds were adjusted considering the number of instances resulting from the reduction process.

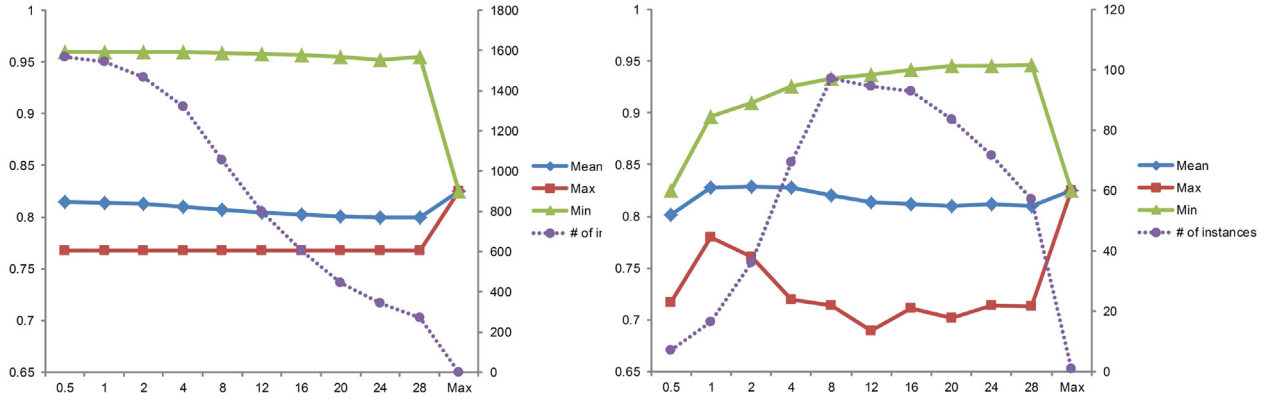
Considering that it is not reliable to represent all the results in terms of the TPR and the FPR obtained for all the different

[1] Anubis: a service for analysing malware. http://anubis.iseclab.org/.



(a) Outlier points not discarded. (b) Outlier points discarded.

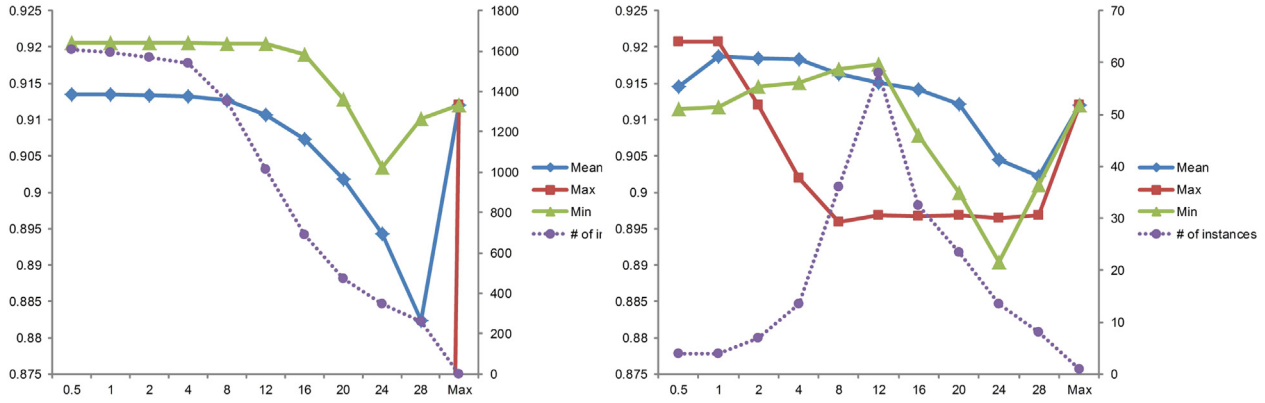
Fig. 2 – AUC obtained for PE header features, Euclidean distance and the 3 selection rules, when different data reduction thresholds are applied.



(a) Outlier points not discarded. (b) Outlier points discarded.

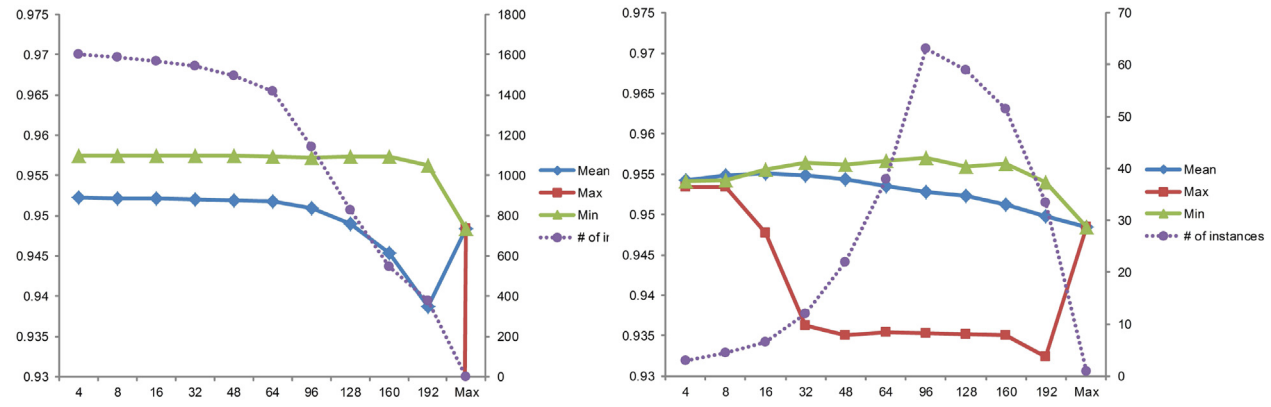
Fig. 3 – AUC obtained for PE based features, Manhattan distance and the 3 selection rules, when different data reduction thresholds are applied.

reduction rates, we plot the AUC together with the number of instances obtained as a result of the reduction process (see Figs. 2–5). Additionally, we show and describe the results obtained for the most interesting reduction rates for each configuration (see Tables 8–11 and 14–17). In each case, the reduction threshold was selected based on the trade-off between the AUC and the reduction rate, selecting the combination which presented the highest possible reduction rate while still maintaining a sound



(a) Outlier points not discarded. (b) Outlier points discarded.

Fig. 4 – AUC obtained for operational code frequency, Euclidean distance and the 3 selection rules, when different data reduction thresholds are applied.



(a) Outlier points not discarded.

(b) Outlier points discarded.

Fig. 5 – AUC obtained for operational code frequency, Manhattan distance and the 3 selection rules, when different data reduction thresholds are applied.

AUC. The selection of the appropriate configuration should fall on the expert's sound judgement, as it depends on the specific deployment scenario and its requirements.

Evaluation of the method for portable executable structure based feature set

Results when no dataset reduction is applied

First, we list the results obtained for Euclidean distance (see Table 6) and Manhattan distance (see Table 7), applying the different selection rules, when no reduction process was applied. In both cases, we measure separately the FNR for the subset of manually packed binaries and the subset of custom packed binaries (Zeus samples). Similarly, we show the AUC when custom packers were not considered and the overall AUC, when all packed binaries were included in the study. As we can observe, there is, in both cases, a clear difference between the detection rate for off-the-shelf and custom packers.

In the case of Euclidean distance we can observe that the best results are achieved for Mean and Min distance selection rules, specially for files protected by off-the-shelf packers. In this case, a good trade-off between FPR and FNR is obtained for FPRs around 0.05. Nevertheless, custom packed files can only be correctly discriminated when the minimum distance is considered, assuming a significantly high false positive rate (0.14).

In the case of Manhattan distance we can observe similar results. In this case, the AUC and the trade-off between FPR

Table 6 – Results obtained for PE based features and Euclidean distance.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC(P)	AUC
Mean	0.0100	0.0056	0.0040	0.8013	0.9371	0.9723	0.8511
	0.0200	0.0167	0.0120	0.5650	0.9238		
	0.0500	0.0500	0.0290	0.3850	0.9133		
	0.1000	0.1000	0.0640	0.0586	0.8751		
	0.1500	0.1500	0.1160	0.0010	0.7921		
	0.2000	0.2000	0.1535	0.0010	0.6829		
	0.2500	0.2500	0.2135	0.0010	0.5920		
	0.3000	0.3000	0.2770	0.0010	0.4437		
Max	0.0100	0.0056	0.0040	0.7997	0.9371	0.9040	0.7991
	0.0200	0.0167	0.0125	0.6579	0.9233		
	0.0500	0.0500	0.0400	0.4081	0.9013		
	0.1000	0.1000	0.0720	0.1840	0.8671		
	0.1500	0.1500	0.1115	0.1165	0.8261		
	0.2000	0.2000	0.1590	0.0932	0.7415		
	0.2500	0.2500	0.2380	0.0780	0.5822		
	0.3000	0.3000	0.3040	0.0657	0.5373		
Min	0.0100	0.0056	0.0045	0.8030	0.9371	0.9824	0.9526
	0.0200	0.0167	0.0110	0.5764	0.9193		
	0.0500	0.0500	0.0505	0.0289	0.7584		
	0.1000	0.1000	0.0945	0.0010	0.1432		
	0.1500	0.1500	0.1400	0.0010	0.0528		
	0.2000	0.2000	0.2185	0.0010	0.0302		
	0.2500	0.2500	0.2660	0.0010	0.0266		
	0.3000	0.3000	0.3280	0.0010	0.0237		

Table 7 – Results obtained for PE based features and Manhattan distance.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC(P)	AUC
Mean	0.0100	0.0056	0.0040	0.8182	0.9373	0.9782	0.8147
	0.0200	0.0167	0.0110	0.4837	0.9357		
	0.0500	0.0500	0.0305	0.2810	0.9220		
	0.1000	0.1000	0.0655	0.0398	0.8911		
	0.1500	0.1500	0.0940	0.0031	0.8623		
	0.2000	0.2000	0.1445	0.0010	0.8335		
	0.2500	0.2500	0.2170	0.0010	0.7930		
	0.3000	0.3000	0.2770	0.0010	0.7522		
Max	0.0100	0.0056	0.0040	0.7881	0.9371	0.9335	0.7674
	0.0200	0.0167	0.0100	0.6710	0.9367		
	0.0500	0.0500	0.0395	0.3728	0.9186		
	0.1000	0.1000	0.0655	0.2795	0.9035		
	0.1500	0.1500	0.1035	0.2024	0.8785		
	0.2000	0.2000	0.1540	0.1310	0.8580		
	0.2500	0.2500	0.2130	0.0910	0.8385		
	0.3000	0.3000	0.2830	0.0674	0.7930		
Min	0.0100	0.0056	0.0045	0.8008	0.9373	0.9846	0.9596
	0.0200	0.0167	0.0100	0.5351	0.9225		
	0.0500	0.0500	0.0550	0.0158	0.4081		
	0.1000	0.1000	0.0935	0.0010	0.1019		
	0.1500	0.1500	0.1400	0.0010	0.0397		
	0.2000	0.2000	0.2085	0.0010	0.0276		
	0.2500	0.2500	0.2690	0.0010	0.0238		
	0.3000	0.3000	0.3530	0.0010	0.0164		

and FNR presents slightly sounder results. As in the previous case, Mean and Minimum distance selection rules achieve sound results for off-the-shelf packers, while custom packers can only be correctly discriminated when the minimum distance is considered, again, at a high cost (0.14 FPR).

Despite the results obtained for the minimum distance, these observations indicate that the custom packed binaries employed for testing are not as anomalous as off-the-shelf packers when PE based features are considered for classification.

Results obtained for the reduced dataset

Figs. 2 and 3 plot the number of instances obtained from the reduction process together with the AUC obtained for each configuration.

More concretely, Fig. 2a shows the evolution of the results when outliers are not discarded. We can observe that, while the results for Max and Mean distance selection rules remain stable, the results for Min distance degrade when the number of samples that represent normality decreases. In this way, the AUCs obtained present values over 0.95 even when the number of instances used for comparison is reduced to 847 (threshold 4.00), and values over 0.94 when the number of instances is over 261 (threshold 8.00). Table 8 shows the result for the threshold 4.00, the configuration which presents the highest reduction rate while maintaining an AUC over 0.95.

Fig. 2b plots the evolution of the AUC when outlier points were discarded during the clustering process, showing a different trend. In this case, we can observe that the number of instances for comparison grows when a higher threshold is applied (i.e., a higher number of clusters are formed). Nevertheless, when the threshold surpasses a certain value (in this case, 4.00), the number of instances decreases, given that the clusters formed include a higher number of instances. While the results for the Mean distance selection rule are stable, the Max and Min distance selection rules are affected by the number of samples in the model. The highest AUC is obtained when the minimum distance is considered and the 6.00 threshold produced 77 instances, a number below the highest number of instances used for comparison (106). Table 9 shows

Table 8 – Results obtained for PE based features, minimum distance selection rule and Euclidean distance, with the training set reduced with 4.00 threshold and outliers not discarded.

Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
0.0100	0.0056	0.0045	0.8030	0.9371	0.9517
0.0200	0.0167	0.0110	0.5764	0.9193	
0.0500	0.0500	0.0510	0.0289	0.7504	
0.1000	0.1000	0.0945	0.0010	0.1444	
0.1500	0.1500	0.1285	0.0010	0.0680	
0.2000	0.2000	0.2100	0.0010	0.0332	
0.2500	0.2500	0.2645	0.0010	0.0258	
0.3000	0.3000	0.3210	0.0010	0.0219	

Table 9 – Results obtained for PE based features, minimum distance selection rule and Euclidean distance, with the training set reduced with 6.00 threshold and outliers discarded.

Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
0.0100	0.0056	0.0040	0.8012	0.9371	0.9298
0.0200	0.0167	0.0110	0.5663	0.9240	
0.0500	0.0500	0.0340	0.3684	0.9073	
0.1000	0.1000	0.0670	0.0115	0.8435	
0.1500	0.1500	0.1090	0.0010	0.5117	
0.2000	0.2000	0.1765	0.0010	0.0514	
0.2500	0.2500	0.2355	0.0010	0.0320	
0.3000	0.3000	0.3225	0.0010	0.0211	

Table 10 – Results obtained for PE based features, minimum distance selection rule and Manhattan distance, with the training set reduced with 8.00 threshold and outliers not discarded.

Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
0.0100	0.0056	0.0040	0.8010	0.9373	0.9591
0.0200	0.0167	0.0100	0.5351	0.9225	
0.0500	0.0500	0.0550	0.0158	0.4038	
0.1000	0.1000	0.0940	0.0010	0.1017	
0.1500	0.1500	0.1390	0.0010	0.0404	
0.2000	0.2000	0.2110	0.0010	0.0277	
0.2500	0.2500	0.2810	0.0010	0.0234	
0.3000	0.3000	0.3375	0.0010	0.0199	

the results for the 6.00 threshold. We can appreciate that the results present a slight improvement for higher thresholds, obtaining higher AUCs for an equivalent number of samples employed to build the model. This tendency might be produced by the noise reduction capabilities of the data reduction approach employed. As the outlier samples not included in any cluster are discarded, the possible negative effects of these samples are smoothed. For higher thresholds, some of the instances considered outliers for lower thresholds are included in the clusters. First, the effects of possible noise are smoothed. Second, the number of instances discarded is reduced, ensuring that a higher number of samples are represented in the final model.

Fig. 3 shows the evolution for Manhattan distance when different reduction thresholds were applied. When outlier points were not discarded, the 3 configurations show stable results, even for the minimum distance. Table 10 shows the results for the minimum distance when a 8.00 threshold was applied for reduction. In this case, although the number of instances is 1054, the AUC obtained is over 0.959. Nevertheless, for environments in which efficiency is an important aspect to consider, the expert might select a higher threshold, given that the results are not notably affected even when the number of instances is decreased to 273 (0.955).

In Fig. 3 we can observe that, when the outliers are discarded, the results for the mean distance are considerably stable. For the maximum distance the results degrade for a higher number of instances, and for the minimum distance the results are better for higher thresholds. In this case we can see a tendency similar to the effect observed for Euclidean distance. When higher thresholds are applied, the results slightly improve. Again, this tendency might be caused by the noise reduction capabilities of the algorithm. Table 11 shows the results for the minimum distance and a 28.00 threshold. This configuration produces the highest AUC with 57 instances during comparison, a number which considerably below the maximum number of instances tested (97).

Table 11 – Results obtained for PE based features, minimum distance selection rule and Manhattan distance, with the training set reduced with 28.00 threshold and outliers discarded.

Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
0.0100	0.0056	0.0040	0.8033	0.9373	0.9468
0.0200	0.0167	0.0130	0.4268	0.9297	
0.0500	0.0500	0.0395	0.0355	0.8590	
0.1000	0.1000	0.0685	0.0010	0.7334	
0.1500	0.1500	0.1205	0.0010	0.1126	
0.2000	0.2000	0.1845	0.0010	0.0309	
0.2500	0.2500	0.2525	0.0010	0.0229	
0.3000	0.3000	0.3105	0.0010	0.0189	

Table 12 – Results obtained for operational code frequencies and Euclidean distance.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
Mean	0.0100	0.0056	0.0220	0.7890	0.8010	0.9136
	0.0200	0.0167	0.0300	0.6945	0.7311	
	0.0500	0.0500	0.0580	0.5245	0.6014	
	0.1000	0.1000	0.1010	0.2010	0.4296	
	0.1500	0.1500	0.1400	0.0557	0.2685	
	0.2000	0.2000	0.1635	0.0402	0.2117	
	0.2500	0.2500	0.2210	0.0263	0.1105	
	0.3000	0.3000	0.2935	0.0152	0.0148	
Max	0.0100	0.0056	0.0220	0.8008	0.8045	0.5318
	0.0200	0.0167	0.0295	0.7196	0.7435	
	0.0500	0.0500	0.0555	0.6128	0.6352	
	0.1000	0.1000	0.0980	0.5290	0.5589	
	0.1500	0.1500	0.1555	0.4956	0.5354	
	0.2000	0.2000	0.2215	0.4835	0.5226	
	0.2500	0.2500	0.2650	0.4789	0.5150	
	0.3000	0.3000	0.3115	0.4716	0.5071	
Min	0.0100	0.0056	0.0215	0.8099	0.8085	0.9206
	0.0200	0.0167	0.0295	0.6943	0.7273	
	0.0500	0.0500	0.0525	0.5318	0.6075	
	0.1000	0.1000	0.0945	0.1716	0.4007	
	0.1500	0.1500	0.1315	0.0671	0.2627	
	0.2000	0.2000	0.1890	0.0228	0.1028	
	0.2500	0.2500	0.2675	0.0142	0.0118	
	0.3000	0.3000	0.3505	0.0110	0.0071	

Table 13 – Results obtained for operational code frequencies and Manhattan distance.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
Mean	0.0100	0.0056	0.0145	0.8542	0.8801	0.9522
	0.0200	0.0167	0.0285	0.6299	0.7975	
	0.0500	0.0500	0.0530	0.1237	0.3735	
	0.1000	0.1000	0.0885	0.0487	0.0518	
	0.1500	0.1500	0.1535	0.0237	0.0110	
	0.2000	0.2000	0.1975	0.0200	0.0108	
	0.2500	0.2500	0.2190	0.0162	0.0097	
	0.3000	0.3000	0.2810	0.0105	0.0073	
Max	0.0100	0.0056	0.0065	0.9547	0.9279	0.2788
	0.0200	0.0167	0.0180	0.9171	0.8672	
	0.0500	0.0500	0.0405	0.8749	0.7916	
	0.1000	0.1000	0.1185	0.8415	0.7519	
	0.1500	0.1500	0.1665	0.8310	0.7348	
	0.2000	0.2000	0.2115	0.8219	0.7201	
	0.2500	0.2500	0.2665	0.8104	0.7104	
	0.3000	0.3000	0.3275	0.8047	0.7037	
Min	0.0100	0.0056	0.0120	0.8677	0.8305	0.9574
	0.0200	0.0167	0.0250	0.6231	0.6759	
	0.0500	0.0500	0.0480	0.2076	0.3710	
	0.1000	0.1000	0.1165	0.0247	0.0099	
	0.1500	0.1500	0.1790	0.0151	0.0063	
	0.2000	0.2000	0.2410	0.0117	0.0056	
	0.2500	0.2500	0.2950	0.0090	0.0048	
	0.3000	0.3000	0.3530	0.0075	0.0006	

Evaluation of the method for operational code frequencies

Results when no dataset reduction is applied

Table 12 shows the results for Euclidean distance when no reduction was applied. In this case, while the Mean and Min distance selection rules show sound results regarding AUC, the maximum distance cannot be considered for classification. Regarding the trade-off between FPR and FNR, we can notice that both configurations must assume a considerably high FPR to achieve a sound FNR (over 0.15).

Regarding Manhattan distance (see Table 13), the results obtained are sounder than the Euclidean distance based configurations. In this case, when mean and minimum distances are applied, the AUC obtained reaches 0.9522 and 0.9574 respectively. For the maximum distance, on the contrary, the results are not acceptable (0.2788 AUC), and thus we must discard the configuration. Regarding the trade-off between FPR and FNR for both mean and the minimum distances, the tolerance of a 0.10 FPR is sufficient to achieve a sound FNR for both off-the-shelf and custom made packers.

On the one hand, in both cases we can observe that the configurations based on the maximum distance are notably affected by this feature set, probably because of its instability in the presence of outliers. On the other hand, the results do not present a notable difference regarding the FNR achieved for off-the-shelf and custom packed binaries, like in the case of the PE based feature set.

Results obtained for the reduced dataset

In this section, we describe the results obtained when various reduction configurations were tested. The number of instances resultant from the reduction process are plotted in Figs. 4 and 5, together with the obtained AUC values for each configuration.

When outlier points are not discarded, we can observe that the maximum distance presents results below the acceptable limits. Besides, both mean and minimum distances produce results that deteriorate when the number of instances decreases. Again, Table 14 shows the results for the selected thresholds that maintain sound AUCs for a minimum number of instances used for comparison. In the case of mean distance, sound AUC values over 0.913 are achieved for reduction thresholds below 8.00. For the minimum distance, a reduction threshold of 12.00 can be applied without observing a degradation in the results, obtaining an AUC over 0.92. In both cases, we can observe that the results tend to deteriorate when the threshold applied is increased, showing a tendency similar to the PE structure based feature set.

In the case when outliers were discarded (see Fig. 4b), the results for mean distance present a soft degradation as the reduction threshold is increased. The minimum distance presents an improvement until the maximum number of instances for comparison is achieved (12.00 threshold). For higher thresholds, it presents a considerable degradation. The maximum distance, despite of the unacceptable results when outliers are not discarded (see Fig. 4a), presents sound results for the lowest thresholds, and a degradation when a higher number of clusters is generated (showing a considerable sensitiveness to the instances used for comparison).

Accordingly, Table 15 shows the detailed results for the selected configurations. For mean distance, a 4.00 threshold presents a sound AUC (0.9184). For this selection rule, higher thresholds present a slight degradation in the results. In the case of Max distance, only the lowest thresholds (e.g., 1.00) that discard most of the samples produce sound results. Nevertheless, as we noticed in Table 13, this selection rule is very sensitive to outliers. Finally, in the case of minimum

Table 14 – Results obtained for operational code frequencies and Euclidean distance, outliers not discarded, with the training set reduced with a 4.00 threshold for Mean distance selection rule, and a 12.00 threshold for Min distance selection rule.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
Mean	0.0100	0.0056	0.0220	0.7890	0.8010	0.9132
	0.0200	0.0167	0.0300	0.6945	0.7311	
	0.0500	0.0500	0.0580	0.5246	0.6014	
	0.1000	0.1000	0.1010	0.2027	0.4298	
	0.1500	0.1500	0.1395	0.0563	0.2702	
	0.2000	0.2000	0.1635	0.0401	0.2129	
	0.2500	0.2500	0.2215	0.0264	0.1139	
	0.3000	0.3000	0.2935	0.0152	0.0151	
Min	0.0100	0.0056	0.0215	0.8099	0.8085	0.9205
	0.0200	0.0167	0.0295	0.6948	0.7274	
	0.0500	0.0500	0.0525	0.5318	0.6074	
	0.1000	0.1000	0.0955	0.1714	0.4011	
	0.1500	0.1500	0.1330	0.0667	0.2628	
	0.2000	0.2000	0.1895	0.0241	0.1078	
	0.2500	0.2500	0.2735	0.0139	0.0122	
	0.3000	0.3000	0.3450	0.0110	0.0076	

Table 15 – Results obtained for operational code frequencies and Euclidean distance, outliers discarded, with the training set reduced with a 4.00 threshold for Mean distance selection rule, a 1.00 threshold for Max distance selection rule, and a 12.00 threshold for Min distance selection rule.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
Mean	0.0100	0.0056	0.0220	0.7882	0.8008	0.9184
	0.0200	0.0167	0.0305	0.6914	0.7293	
	0.0500	0.0500	0.0570	0.5209	0.6017	
	0.1000	0.1000	0.1040	0.1385	0.3944	
	0.1500	0.1500	0.1400	0.0487	0.2398	
	0.2000	0.2000	0.1625	0.0374	0.1988	
	0.2500	0.2500	0.2190	0.0243	0.0508	
	0.3000	0.3000	0.2845	0.0156	0.0123	
Max	0.0100	0.0056	0.0220	0.7877	0.8006	0.9208
	0.0200	0.0167	0.0300	0.6909	0.7290	
	0.0500	0.0500	0.0570	0.5183	0.6028	
	0.1000	0.1000	0.1000	0.1332	0.3924	
	0.1500	0.1500	0.1420	0.0447	0.2171	
	0.2000	0.2000	0.1605	0.0369	0.1867	
	0.2500	0.2500	0.2270	0.0230	0.0219	
	0.3000	0.3000	0.2965	0.0140	0.0108	
Min	0.0100	0.0056	0.0220	0.7872	0.8003	0.9176
	0.0200	0.0167	0.0315	0.6862	0.7257	
	0.0500	0.0500	0.0585	0.5285	0.6039	
	0.1000	0.1000	0.1040	0.1582	0.4038	
	0.1500	0.1500	0.1375	0.0523	0.2560	
	0.2000	0.2000	0.1710	0.0353	0.1919	
	0.2500	0.2500	0.2550	0.0181	0.0164	
	0.3000	0.3000	0.3180	0.0134	0.0099	

distance we can appreciate that the results depend on the number of instances resulting of the clustering process. In this case, the configuration that produces the highest number of samples obtained the best AUC (12.00 threshold).

Fig. 5a shows the results obtained for different reduction thresholds with Manhattan distance, when outlier points were not discarded in the clustering process. Like in the previous case, the Max distance does not present sound results, while mean and minimum distance present considerably stable results, degrading for higher thresholds. Considering the tendency of the results, we select the highest possible threshold (maximum reduction ratio) that does not degrade considerably the AUC: 96.00 (1140 instances) for mean distance and 160.00 (540 instances) for minimum distance. The results for the best threshold configurations are detailed in Table 16.

Finally, when outliers were discarded (see Fig. 5b), the results show a slight performance improvement for the minimum distance as the number of generated clusters increases with higher thresholds, and then deteriorates when the highest thresholds are applied, reaching the best results for a 96.00 threshold, when the maximum number of instances is reached. In the case of the mean distance, the results show a stable deterioration when the applied threshold is increased, obtaining sound results for a 16.00 threshold. For higher thresholds, we can observe a deterioration of the AUC. The results for Maximum distance, like in the case of Euclidean distance, are sensitive to the variability of the instances generated after the clustering process, but show interesting values for the lowest reduction thresholds (8.00). The detailed results for these configurations are listed in Table 17.

Evaluation of the efficiency of both feature sets

In this section, we evaluate the efficiency of the method proposed considering the dataset reduction method and the 2 feature sets employed for the evaluation.

Table 16 – Results obtained for operational code frequencies and Manhattan distance, outliers not discarded, with the training set reduced with a 96.00 threshold for Mean distance selection rule, and a 160.00 threshold for Min distance selection rule.

Sel. Rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
Mean	0.0100	0.0056	0.0145	0.8535	0.8770	0.9509
	0.0200	0.0167	0.0270	0.6708	0.8084	
	0.0500	0.0500	0.0535	0.1313	0.3934	
	0.1000	0.1000	0.0870	0.0517	0.0858	
	0.1500	0.1500	0.1495	0.0251	0.0110	
	0.2000	0.2000	0.1950	0.0210	0.0108	
	0.2500	0.2500	0.2200	0.0186	0.0106	
	0.3000	0.3000	0.2860	0.0113	0.0075	
Min	0.0100	0.0056	0.0115	0.8803	0.8427	0.9574
	0.0200	0.0167	0.0245	0.6533	0.6866	
	0.0500	0.0500	0.0450	0.2282	0.3997	
	0.1000	0.1000	0.1025	0.0314	0.0131	
	0.1500	0.1500	0.1775	0.0158	0.0076	
	0.2000	0.2000	0.2455	0.0100	0.0063	
	0.2500	0.2500	0.3125	0.0077	0.0044	
	0.3000	0.3000	0.3575	0.0058	0.0020	

Table 17 – Results obtained for operational code frequencies and Manhattan distance, outliers discarded, with the training set reduced with a 16.00 threshold for Mean distance selection rule, a 8.00 threshold for Max distance selection rule, and a 96.00 threshold for Min distance selection rule.

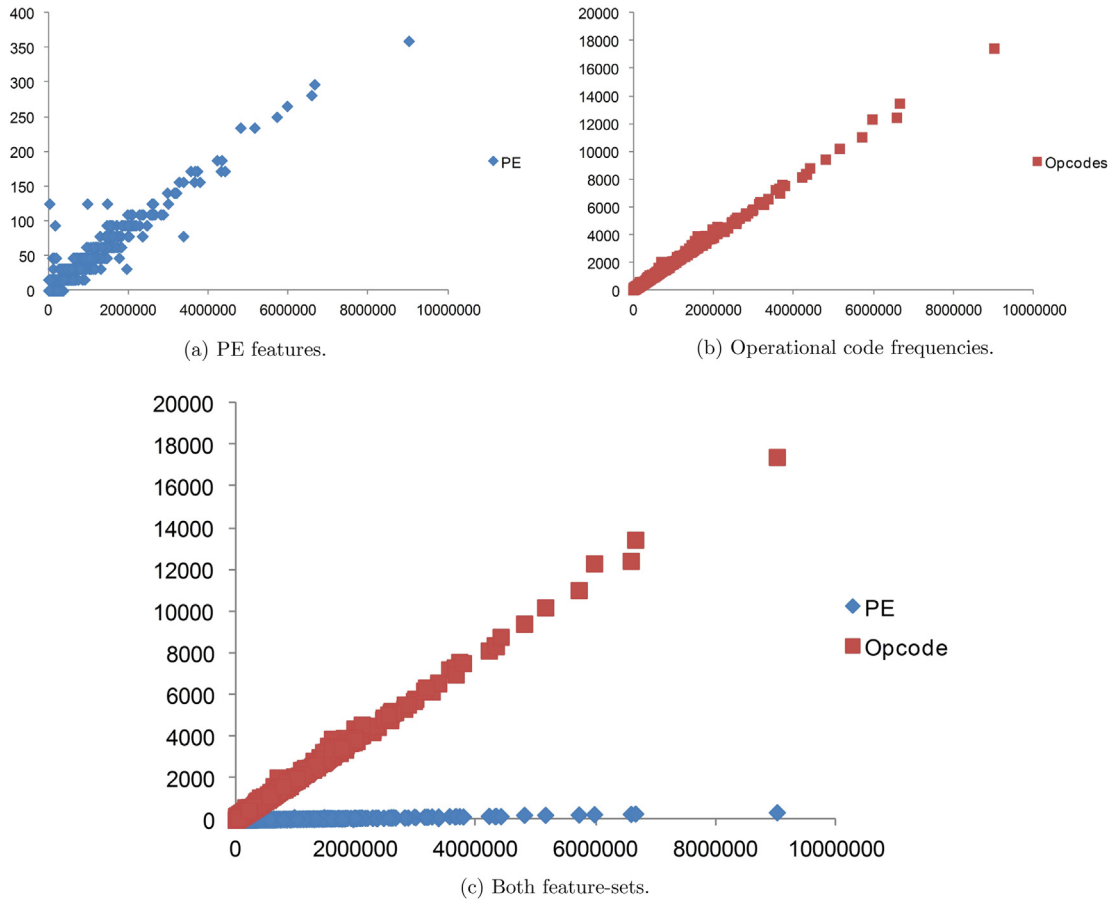
Sel. rule	Max FPR	Tr. FPR	FPR	FNR(P)	FNR(CP)	AUC
Mean	0.0100	0.0056	0.0135	0.8563	0.8874	0.9550
	0.0200	0.0167	0.0325	0.4802	0.7422	
	0.0500	0.0500	0.0570	0.0985	0.3021	
	0.1000	0.1000	0.0945	0.0405	0.0124	
	0.1500	0.1500	0.1530	0.0228	0.0109	
	0.2000	0.2000	0.1990	0.0173	0.0091	
	0.2500	0.2500	0.2300	0.0128	0.0075	
	0.3000	0.3000	0.2770	0.0097	0.0060	
Max	0.0100	0.0056	0.0135	0.8579	0.8917	0.9534
	0.0200	0.0167	0.0305	0.5343	0.7787	
	0.0500	0.0500	0.0560	0.1090	0.3323	
	0.1000	0.1000	0.0895	0.0498	0.0350	
	0.1500	0.1500	0.1540	0.0236	0.0110	
	0.2000	0.2000	0.1940	0.0221	0.0097	
	0.2500	0.2500	0.2395	0.0145	0.0072	
	0.3000	0.3000	0.2765	0.0113	0.0070	
Min	0.0100	0.0056	0.0140	0.8550	0.8802	0.9570
	0.0200	0.0167	0.0325	0.4936	0.7374	
	0.0500	0.0500	0.0575	0.0856	0.2373	
	0.1000	0.1000	0.1180	0.0267	0.0110	
	0.1500	0.1500	0.1630	0.0155	0.0084	
	0.2000	0.2000	0.2160	0.0115	0.0076	
	0.2500	0.2500	0.2575	0.0070	0.0060	
	0.3000	0.3000	0.3035	0.0042	0.0060	

First, the feature extraction was performed in an isolated virtual machine using VMWare. The host machine was an Intel Core i5 650 clocked at 3.20 GHz and 16 GB of RAM memory, while the guest machine was configured with 2 processors, 4 GB of RAM memory and Windows XP SP3 as operating system.

Fig. 6 summarises the feature extraction process. We can observe that, on the one hand, the feature extraction time is directly proportional to the file size in both feature sets. In the case of the PE based feature set, this effect is caused by the values that depend on the full content of the file, like entropy. On the other hand, we can observe in Fig. 6c that the extraction of operational code frequencies is more time-consuming than the extraction of PE based features. In this way, the average comparison time is 0.0576 ms/KB for PE based features and 2.0188 ms/KB for operational code frequency extraction. Although the extraction time in both cases depends on the implementation of our feature extraction algorithm, we can safely conclude that the extraction of operational code frequencies requires more computational resources than the PE based feature set.

The normalisation process of each fold consumes on average 148.4375 ms with a standard deviation of 62.1736 for the PE based features, and 735.9375 ms with a standard deviation of 121.1315 for operational code frequencies.

Fig. 7 shows the time required by the dataset reduction algorithm for each configuration. First, we can notice that Euclidean distance, in both cases, presents higher reduction times. This difference is caused by the complexity of



(a) PE features.

(b) Operational code frequencies.

(c) Both feature-sets.

Fig. 6 – Feature extraction times for the different feature-sets. The X axis represents the file size, while the Y axis represents the time required to extract the features, expressed in milliseconds.

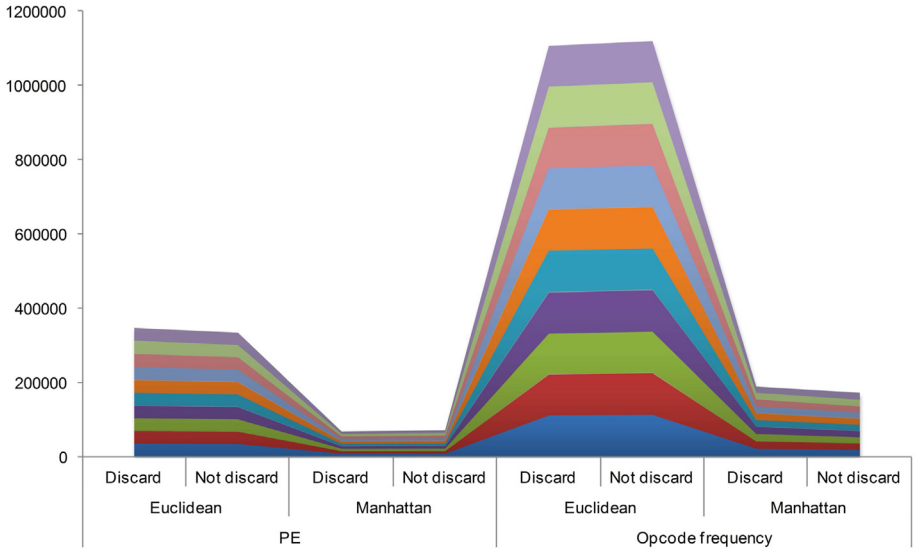


Fig. 7 – Reduction times for the different configurations. The X axis represents the different experiment configurations, and the Y axis represents the reduction time, expressed in milliseconds. Each of the 10 folds evaluated in the experiment is represented in a different colour (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

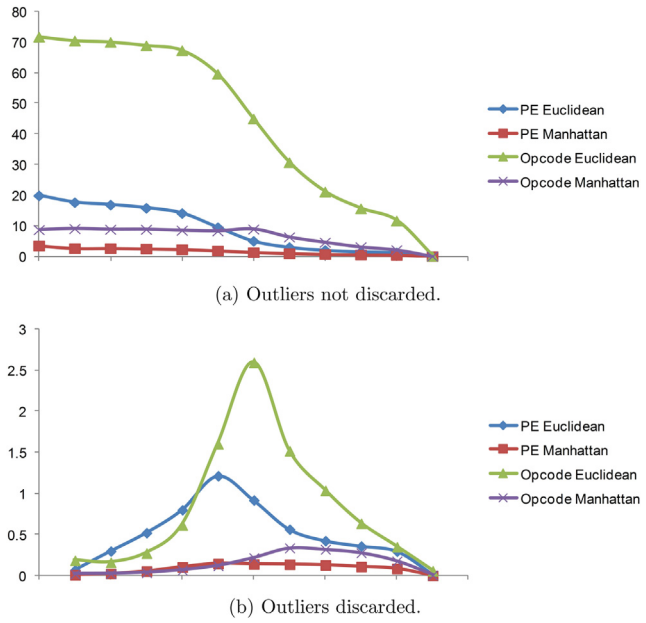


Fig. 8 – Distance calculation times averaged for each possible configuration. The X axis represents the different thresholds applied. The threshold applied for the reduction varies for each configuration. Nevertheless, the thresholds are plotted, in each case, in ascending order. The Y axis shows the average time (in milliseconds) to measure the distance from a testing instance to the model.

each distance measure. PE based feature set requires, again, a lower processing time. This is caused by the number of possible features present in each feature set. While the PE based features is composed of 241 features, the feature set composed of operational codes is represented by 823.

Finally, Fig. 8 shows the average time required to measure the distance from a testing instance to the not-packed model. This time is directly proportional to the number of instances used for comparison in each case. In Fig. 8a we can observe the results when outliers were not discarded. First, the PE based features require lower processing times than operational code frequencies. Second, Manhattan distance is more efficient than Euclidean distance. Fig. 8b shows the average distance calculation time when outliers are discarded. In this case, as the number of instances that represent the model is lower, the time required for comparison decreases with respect to the previous approach. In these cases, the number of instances representing the model reaches its maximum value for the central thresholds. Once the threshold surpasses a certain value, the clusters generated contain a higher number of instances and the number of clusters decreases. This effect has a direct effect on comparison time, that can be observed in Fig. 8b.

Conclusions and discussion

The results presented in Section Evaluation show the performance of the anomaly detection approach for 2 different feature sets. In this section we answer to several research questions established for this study.

Which is the feature set that best discriminates packed from not packed files?

The first observation is that the only distance selection rule that produces sound results for both off-the-shelf packers and custom packers with PE based features is the minimum distance, while in the case of operational code frequencies, both mean and minimum distance are valid approaches.

Secondly, we can observe that PE based features present a higher FNR for custom packers than for off-the-shelf packers. This effect is not noticeable for operational code frequencies.

The third observation is that, while both feature sets present unsound results for the Max distance selection rule, operational code frequencies based approach is extremely sensitive to outliers in the model.

From these observations, we can conclude that the PE based feature set presents limitations to capture the difference between not packed binaries and custom packed files. Nevertheless, when the minimum distance is selected, our approach can provide sound detection rates if it assumes higher FPRs. This effect is caused by the different nature of the packing approach used to protect the samples.

Finally, if we observe the efficiency of both feature sets, we can observe that operational code frequencies are not as efficient as PE based features. Nevertheless, the results indicate that the operational code frequency based approach should not be discarded. Another interesting option would be to combine both approaches in order to provide a final decision.

What is the impact of the data-reduction approach over the results obtained?

First, we can notice that in the case of PE based features, the results present a degradation when the number of instances is decreased and outliers are not discarded. When outliers are discarded, we can observe that the results show a slight improvement for the highest thresholds, if compared to the lowest ones at equivalent reduction rates. In this case, the number of instances decreases due to the presence of a lower number of clusters comprising a higher number of instances. The improvement observed might be caused by the noise reduction capabilities of the data reduction algorithm.

In the case of operational code frequencies the results tend to degrade as the reduction threshold increases, regardless of the number of instances used for comparison for both data reduction approaches.

Another observation is that, for PE based features and Euclidean distance, sounder results are achieved when outliers are not discarded. In the case of operational code frequencies, there is not a noticeable difference among both approaches.

Regarding the efficiency, the only difference among both approaches is the number of instances used for comparison.

To answer the research question established, although all configurations are not affected equally by the data reduction process, we can conclude that, in all cases, it is possible to find a

trade-off between efficiency and effectiveness, trying to minimise the processing time while maintaining sound results.

What is the impact on the results of the different distance measures evaluated?

In this case, we can observe that for PE based features, the Manhattan distance does not produce sounder results than Euclidean distance. More concretely, while for Mean and Max distance selection rules the results are improved, when the minimum distance is selected (the only distance measure with acceptable performance), the results are nearly equal. However, for operational code frequencies, the results observed for Manhattan distance actually differ, presenting sounder results for both AUC, and better trade-offs between FPR and FNR.

What is the impact on the results of the different distance selection rules?

Regarding the distance selection rules, we can observe that, with the exception of certain configurations (operational code frequencies with dataset reduction and low reduction thresholds), the Max distance selection rule does not present a sound performance, probably for its sensitiveness to outliers.

In the case of PE based features, the only distance measure with a good performance is the Min selection rule.

This difference might be caused by the impact of the results for the classification of custom packers. The distance between a not packed instance and a custom packed instance is low, a fact that affects the final value when the average distance is calculated.

Does our anomaly detection approach present sound results for the classification of packed and not packed files?

With the exception of certain configurations, the anomaly detection method proposed is capable of classifying packed and not packed binaries efficiently.

While the results obtained are not as sound as those obtained for supervised approaches, our model only considers the properties of not packed binaries, making it independent of the packer used.

The trade-offs between FPR and FNR selected for each configuration tend to produce high false positives rates (near 0.10 FPR). Nevertheless, these thresholds have been selected to maximise the packer detection rate at an assumable false positive rate. The simplicity of the distance-based approach allows to adjust the threshold according to the requirements of the deployment scenario.

Given the results obtained, the proposed approach stands as a valid method to discriminate packed samples from not packed binaries. Traditional signature scanning methods tend to produce poor detection rates. In fact, malware writers often try to modify the packed binaries in order to resemble unprotected files or other different packers, in an effort to avoid detection or to confuse the analyst. Although many security products employ heuristic detection mechanisms to complement signature based scanning, classic heuristics can be easily evaded (). On the contrary, the approach presented in this paper models not packed samples. Typically, these kind of software is compiled and linked by common software development tools that follow well-known standards and conventions. In this way, any deviation to this model is considered suspicious.

This detection approach can be applied to different contexts. First, given the amount of malware samples that are collected every day, it is not reliable to analyse them one by one. Sample triage not only improves the efficiency of automatic analysis systems (e.g., in scenarios in which resource consuming tasks must be applied to recover the original code of the binary), but also can improve overall detection rates by allowing the application of the correct treatment to each sample. In this sense, automatic unpackers are generally time-consuming and require the execution of the sample. A sound filtering of samples would help the analyst to selectively apply such processes to new and previously unknown samples.

Additionally, this approach allows to adjust the detection threshold depending on the requirements of the deployment scenario. Lower thresholds will tend to produce higher false positives, while obtaining good detection rates. On the contrary, higher thresholds will tend to produce lower false positive rates, as well as a lower detection rate. The number of samples labelled as packed will directly depend on the threshold selected. In some cases, the capacity to process the samples under time-consuming unpacking engines is limited. In other cases, a higher detection rate is preferable, even at a high computational cost derived from processing samples that are actually not packed.

Another aspect to consider in order to select an adequate threshold is the distribution of the samples to be analysed. The proportion of packed instances may be different for a set of binaries already labelled as malicious, than for a set of binaries submitted for analysis in an on-line submission system, in which any user could submit both packed, not packed, benign, or malicious samples.

In addition to this, modern anti-virus products include sandbox solutions in order to execute suspicious binaries in an isolated environment. These solutions allow to execute the sample in the host machine, but limit their functionality in order to avoid infection. During the execution, the behaviour of the sample is analysed in order to determine if it contains any malicious payload. The proposed approach could be used to improve the classification of the sample prior to its execution in such environment, labelling as suspicious any sample that may contain protected code or data.

Finally, the approach proposed could be affected by the possible noise in the not-packed model employed for classification. For these experiments, we have applied a sanitisation process to the dataset to reduce the noise to the extent possible. Nevertheless, for a real-world deployment of such system, it would be interesting to study new approaches to filter packed samples using other manual methods.

Future work can be oriented in different ways. On the one hand, other anomaly detection approaches can be tested such as probabilistic models or one-class support vector machines.

On the other hand, we could study different feature selection and weighing techniques that consider only one of the classes. Unfortunately, the majority of such approaches make use of information from both classes in order to measure the relevance of each attribute.

Related work

Several approaches have addressed the extraction of static features from binary files for the classification of packed and not packed binaries.

Some approaches rely on heuristics to detect packed samples: Lyda and Hamrock (2007) proposed the use of entropy to discriminate packed and not packed files. Perdisci et al. (2008a) proposed a method based on the extraction of several heuristic values from PE files in order to discriminate packed binaries from unprotected binaries using supervised machine-learning based classification models.

Shafiq et al. (2009a,b), similarly, applied the same heuristics for packed binary detection. Afterwards, instead of unpacking the samples, they applied machine-learning based techniques considering different feature sets for packed and not-packed binaries. More concretely, they evaluated the capacity of PE based features to discriminate goodware and malware. Nevertheless, they did not apply this feature set for the detection of packed files.

Other approaches focus on the contents of the file. In this way, Perdisci et al. (2008b) combined their previously proposed method based on heuristics with n-gram analysis in order to filter executables and selectively apply a generic unpacker to those samples previously classified as packed, prior to the application of malware detection techniques to the samples.

Sharif et al. (2008) proposed the use of n-gram based classification to distinguish packed from unpacked memory regions as a method to determine the correct moment to dump the memory content as part of a generic unpacking method.

Similarly, most generic unpacking methods rely on different heuristics for the detection of the correct moment to dump the unpacked memory content (i.e., when the original entry point is reached). These heuristics can be considered as dynamic packer detection methods. Nevertheless, the application of these methods to filter executables is nonsense, considering that they require the actual unpacking of the sample.

Other approaches have focused on entropy analysis, and explored in depth the statistical properties of packed binaries in order to distinguish the packers used for protection (Sun, 2012), or the kind of packing techniques used to protect a given sample (Jacob et al., 2013) (i.e., compression, encryption).

Finally, Caballero et al. (2009) proposed a mixed dynamic and static approach consisting on hybrid disassembly and data-flow analysis to extract self-contained *transformation functions*, identifying code and data dependencies, and extracting the function interface (input and output parameters). In particular, Caballero et al. applied cryptographic function extraction to identify the unpacking routine of the ZBot malware, and employed it to statically unpack a different sample of the same family.

Acknowledgement

We would like to acknowledge S21Sec for the malware samples described in this paper.

This research was partially supported by the Basque Government under a pre-doctoral grant given to Xabier Ugarte-Pedrero (PRE_2013_2_65) and by the OTRI/Deiker at the University of Deusto under a predoctoral grant given to Iván García-Ferreira.

REFERENCES

Breiman L. Bagging predictors. Mach Learn 1996;24:123–40.

Caballero J, Johnson N, McCamant S, Song D. Binary code extraction and interface identification for security applications. In: Proceedings of the 17th Annual Network and Distributed System Security Symposium, ISOC; 2009. pp. 391–408.

Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 2006;7:1–30.

Garner SR. Weka: The Waikato environment for knowledge analysis. In: Proceedings of the New Zealand Computer Science Research Students Conference; 1995. pp. 57–64.

Heyer LJ, Kruglyak S, Yooseph S. Exploring expression data: identification and analysis of coexpressed genes. Genome Res 1999;9:1106–15.

Iman RL, Davenport JM. Approximations of the critical region of the friedman statistic. Commun Statistics-Theory Methods 1980;9:571–95.

Intel. Intel 64 and ia-32 architectures software developer's manual; 2013.

Jacob G, Comparetti PM, Neugschwandtner M, Kruegel C, Vigna G. A static, packer-agnostic filter to detect similar malware samples. In: Detection of intrusions and Malware, and vulnerability assessment. Springer; 2013. pp. 102–22.

Kang M, Poosankam P, Yin H. Renovo: a hidden code extractor for packed executables. In: Proceedings of the 2007 ACM workshop on Recurring malcode; 2007. pp. 46–53.

Kolter JZ, Maloof MA. Learning to detect malicious executables in the wild. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining (KDD); 2004. pp. 470–8.

Lyda R, Hamrock J. Using entropy analysis to find encrypted and packed malware. IEEE Secur Priv 2007;5:40–5.

McAfee. The good, the bad and the unknown. Available at, http://www.mcafee.com/us/resources/white-papers/wp-good-bad-the-unknown.pdf; 2009.

Morgenstern M, Pilz H. Useful and useless statistics about viruses and anti-virus programs. In: Proceedings of the CARO Workshop; 2010.

Perdisci R, Lanzi A, Lee W. Classification of packed executables for accurate computer virus detection. Pattern Recognit Lett 2008a;29:1941–6.

Perdisci R, Lanzi A, Lee W. Mcboost: boosting scalability in malware collection and analysis using statistical classification of executables. In: Computer Security Applications Conference, 2008. ACSAC 2008. Annual, IEEE; 2008b. pp. 301–10.

Royal P, Halpin M, Dagon D, Edmonds R, Lee W. Polyunpack: automating the hidden-code extraction of unpack-executing malware. In: Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC); 2006. pp. 289–300.

Santos I, Penya Y, Devesa J, Bringas PG. N-Grams-based file signatures for malware detectionIn Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS; 2009. pp. 317–20.

Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings of the 22th IEEE Symposium on Security and Privacy; 2001. pp. 38–49.

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); 2009. pp. 29–33.
Shafiq M, Tabish S, Mirza F, Farooq M. Pe-miner: mining structural information to detect malicious executables in realtime. In: Recent advances in intrusion detection. Springer; 2009b. pp. 121–41.
Sharif M, Yegneswaran V, Saidi H, Porras P, Lee W. Eureka: a framework for enabling static malware analysis. In: Proceedings of the European Symposium on Research in Computer Security (ESORICS); 2008. pp. 481–500.
Stewart J. Ollybone: semi-automatic unpacking on ia-32. In: Proceedings of the 14th DEF CON Hacking Conference; 2006.
Sun L. REFORM: a framework for malware packer analysis using information theory and statistical methods. Ph.D. thesis; 2012.
Ugarte-Pedrero X, Santos I, Bringas PG. Structural feature based anomaly detection for packed executable identification. In: Computational intelligence in security for information systems. Springer; 2011. pp. 230–7.
Ugarte-Pedrero X, Santos I, Bringas PG. Boosting scalability in anomaly-based packed executable filtering. In: Information security and cryptology. Berlin Heidelberg: Springer; 2012. pp. 24–43.
Wyke J. What is zeus?; 2011.

Xabier Ugarte-Pedrero finished his studies in Computer engineering in 2010 at Deusto University. During 2010–2011, he coursed a MSc in Information Security, at the University of Deusto. He joined DeustoTech in september 2010. His research interests are: malware unpacking and analysis, natural language processing, software engineering and micro-bot development and digital electronics. He is currently a PhD student working in a dissertation about malware unpacking.

Igor Santos finished his PhD in 2011 with a dissertation about malware detection. He is a researcher in Deustotech, where he conducts research focused mainly in the areas of information security, malware detection, content filtering, natural language processing, information retrieval methods, opinion mining, and applied machine learning. He is also lecturer in the Faculty of Engineering of the University of Deusto, where he has taught in undergraduate and postgraduate courses.

Iván García-Ferreira finished his studies in Computer Science in 2008 in Nottingham Trent University. In 2009 he obtained a CompTIA Security+ certification and joined an antivirus company. In 2010–2011 he coursed a MSc in Information Security at the University of Deusto. As a result of the research for his master thesis about security in windows applications, he found and published several vulnerabilities in software. Nowadays, he is working on his PhD in software verification and vulnerability analysis in DeustoTech Computing, at the University of Deusto.

Sergio Huerta completed his Ph.D. in mathematics in February 2013 with a dissertation on rational homotopy. He joined DesutoTech in March 2013. His research interests include artificial intelligence, machine learning, digital security, malware detection, malware diffusion and complex networks. He also teaches at the Faculty of Engineering of the University of Deusto.

Borja Sanz completed his PhD in Computer Science in 2012. His research area focuses on malware detection in mobile devices, mainly on the Android platform, machine learning algorithms and Natural Language Processing techniques. He also works on the identification and modelling of new security threats.

Pablo G. Bringas, PhD in Computer Science and Artificial Intelligence, MSc in Telecommunications, MSc in Technical Informatics and Software Engineer. He is currently Head Researcher at DeustoTech – Deusto Technology Foundation, in the S3Lab (Laboratory for Smartness, Semantics and Security), and Assistant Professor at the University of Deusto. He is a member of the Executive Committee of the Spanish National Consultation Council on Cyber Security.