Highlighting the Impact of Packed Executable Alterations with Unsupervised Learning

Alexandre D'Hondt[1][0000−0001−6160−8624], Charles Henry Bertrand Van Ouytsel[1][0000−0001−5720−6569], and Axel Legay[1][0000−0003−2287−8925]

Université Catholique de Louvain, Rue Archimede 1, Louvain-la-Neuve, Belgium
alexandre.dhondt,charles-henry.bertrand,axel.legay}@uclouvain.be

Abstract. For many years, executable packing has been used for a variety of applications, including software protection but also malware obfuscation. Even today, this evasion technique remains an open issue, particularly in malware analysis. Numerous studies have proposed static detection techniques based on various algorithms and features, taking advantage of machine learning to build increasingly powerful models. These studies have focused in particular on supervised learning, but unsupervised learning remains relatively unexploited yet. Furthermore, most studies related to adversarial learning focused on attacks in the feature space while those targeting features identified as significant in supervised models are still rather limited. Such features may be still manipulated from the problem space for causing misclassification. The objective of this study is to apply alterations on packed samples based on realistic modifications and visualize their effect using unsupervised learning. To this end, the Packing Box experimental toolkit is used to build a dataset, train models, apply alterations, retrain models and then highlight the consequences of these alterations on the trained models.

Keywords: executable packing · packer detection · machine learning · static analysis · unsupervised learning · adversarial examples

1 Introduction

Executable packing consists of applying a series of transformations, including – but not limited to – compression, encryption, code mutation or anti-reversing protection measures, which modify the syntax of an executable without changing its semantics. Such transformations are widely used to hide parts of binary files so that they are no longer detected by conventional security systems. Detection is carried out according to the level of interaction with the binary, ranging from static techniques, i.e. without execution, to dynamic techniques, which involve executing the binary and collecting information on its behavior. In the context of this detection, even though these techniques can be combined, the main concern is to maintain a certain level of performance, combining accuracy and processing speed. For this reason, although static techniques are limited in the amount of information they can collect in comparison to dynamic techniques, *static detection approaches* are still constantly evolving because they are faster to apply.

Unsupervised learning is aimed to handle unlabeled data and to find hidden patterns. It is divided into two families of algorithms depending on their purpose ; *clustering*, where data is grouped based on similarity, and *dimensionality reduction*, which allows to simplify data while preserving most essential information. Key clustering algorithms include K-means and dimensionality reduction includes Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE). As of our knowledge, there are very few studies in the current literature exploiting unsupervised learning. In 2021, Noureddine *et al.* [10] proposed a self-evolving packer classifier operating in two phases: the offline one used available packed samples and packer labels to generate clusters while the online one updated clusters incrementally as new samples were processed. This approach relied on state-of-the-art features from the work of Biondi *et al.* [2] whose most significant features were later identified by Bertrand Van Ouytsel *et al.* [1] in his extended analysis of supervised approaches. Unsupervised learning remains relatively unexplored and therefore presents a range of opportunities for classification and visualization.

Adversarial learning is aimed to improve robustness and performance by confronting models with adversarial examples which are inputs designed to deceive them. Adversarial attacks are classified by their threat model defining the attack surface and the characteristics of the attacker as of the comprehensive framework of Carlini *et al.* [3]. Regarding attacker's capabilities, two problem spaces are considered ; the feature space relates to perturbations that can directly be applied to features in the model while the problem space concerns alterations of input data. While adversarial learning was already extensively covered in the literature, especially in the field of malware analysis, most studies addressed attacks from the feature space perspective. However, in the scope of binary executables, it is necessary to design alterations in the *problem space* as, given the nature of features, it is impossible to map them back to a working executable [9]. Such alterations shall be based on parts of binaries that do not break their working, including headers, sections [5] or padding bytes [8]. This approach of altering samples in the problem space was used by D'Hondt *et al.* [6] for targeting static packing detectors found in the wild and for showing weaknesses in their heuristics.

In this paper, we focus our attention on K-Means – a very common clustering algorithm – using a dataset of Portable Executable (PE) samples, some of which are packed with UPX – a very popular packer – leveraging the set of significant features from Bertrand Van Ouytsel *et al.* [1] and we use two carefully chosen alterations from the work of D'Hondt *et al.* [6] – the other three alterations offering no significant results – with visualization based on dimensionality reduction algorithms with the objective of highlighting the effects of such alterations.

The remainder of this paper first presents, in Section 2, the methodology we used to build our dataset, train models and visualize the results before and after alterations of packed samples. In Section 3, we then show visualizations obtained and highlight the effects of alterations on classification from a visual perspective. We finally conclude in Section 4 on what we could observe and the opportunities it provides for further study.

2 Methodology

This section discusses the methodology we followed to obtain visualizations and to evaluate the effects of alterations as discussed in Section 3.

Our methodology relies on the experimental toolkit called *Packing Box* introduced in 2023 by D'Hondt *et al.* [7], an easy-to-deploy Ubuntu-based Docker image, that provides a comprehensive framework for fine-tuning each stage of the machine learning pipeline, including algorithm parameter tuning, feature and alteration definition, visualization and more.

For our experiment, we set up our *reference dataset* **BL** (or baseline) with 432 not-packed samples and 123 UPX-packed samples coming from *dataset-packed-pe*, an alternative version of PackingData from Choi *et al.* [4], and we enriched it with 300 samples mass-packed with the Packing Box. We got a dataset of 855 samples, 432 not-packed and 423 UPX-packed. We use the *features set* of Biondi *et al.* [2] reduced to its most significant features according to the study of Bertrand Van Ouytsel *et al.* [1]. This way, we kept 75 features including the 64 bytes after the Entry Point (EP), the entropy of the whole file and the code section, whether a code section is present or the EP is in a code section or in a standard section, the number of addresses in the Import Address Table (IAT), the number of functions imported in the Import Directory Table (IDT), the number of sections having read-write-execute or write-execute flags, the number of standard section names and the ratio of malicious API imported.

Packing Box integrates a range of machine learning algorithms, including K-Means, DBSCAN and many others. We limited our scope to K-Means *clustering algorithm* as it is straightforward to use and provides good classification results. We parameterized K-Means with 2 clusters, a maximum of 300 iterations and the Lloyd algorithm. Packing Box also includes *dimensionality reduction* algorithms, including PCA and t-SNE. For our experiment, we reduced our data to visualize it in two dimensions by first applying a PCA with 20 components then applying a t-SNE with 2 components and a perplexity of 100 – determined by successive trials until we got clear clusters.

For our *alterations*, we selected A01 and A02 from the work of D'Hondt *et al.* [6] defined as follows. **A01** *rename packer sections* changes common packer or empty section names to standard section names, starting with the 6 most common, including *.text*, *.data* or *.bss*, then choosing random standard names among a list based on the PE format specification. This alteration typically targets the *number of standard sections*. **A02** *move EP to new low-entropy code section* moves the EP to a new code section with a standard name that holds a *trampoline* code jumping to the Original EP and it pads the section up to a given length with a low-entropy pattern. It targets features such as whether the *EP is not in a standard section* and the *bytes after the EP*.

For the evaluation, because we made our dataset with labels, we could inspect classification metrics such as accuracy, precision and recall but also homogeneity and completeness, two clustering metrics that require true labels and respectively represent to what extent clusters contain only data points from the same label and to what extent data points with a given label are part of the same cluster.

3 Results

This section discusses the results obtained with the methodology from Section 2, relying on visualizations and evaluation metrics.

First, we trained K-Means on the reference dataset **BL** and obtained the metrics shown in Table 1. We generated a first visualization using PCA and t-SNE. Predictions are depicted on Figure 1 on the left side with the resulting clusters. On the right side, we show the true labels. Precision is maximal which means that there was no false positive. However, 2 false negatives (out of 855 samples) cause the other metrics to be non-maximal. On the visualization, we see clear separate clusters, hence the clustering metrics are almost at their maximum.

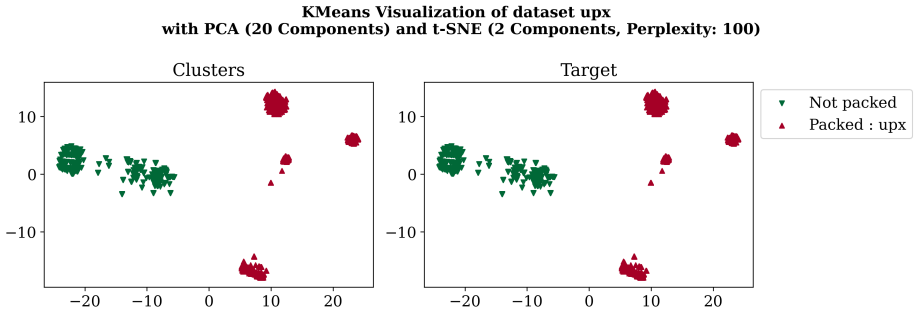


Fig. 1. Visualization of clusters of not-packed and UPX-packed samples

Then, we applied alterations on the 423 UPX-packed samples of **BL** to make datasets **A01** and **A02**, with the goal of influencing their features so that they look like not-packed samples. Afterwards, we retrained K-Means on both resulting datasets and regenerated visualizations.

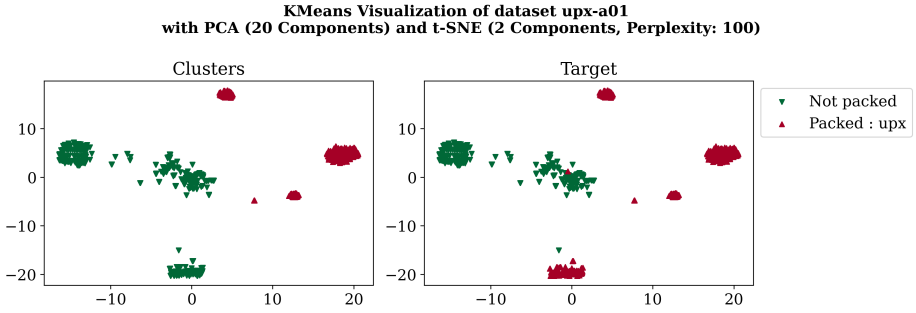


Fig. 2. Visualization of clusters of not-packed and UPX-packed samples after applying alteration A01 (renaming common packer section names to standard ones)

On Figure 2, we can see on the bottom that, between predictions and true labels, the alteration causes a group of packed samples to now be classified as not packed, resulting in *evasion*. Metrics in Table 1 confirm this result as the precision is maximal, meaning no false positive, and the recall drops to 41%, meaning many false negatives. Moreover, clustering metrics drop to very low values as now many data points from the packed label are not matching the right cluster anymore.

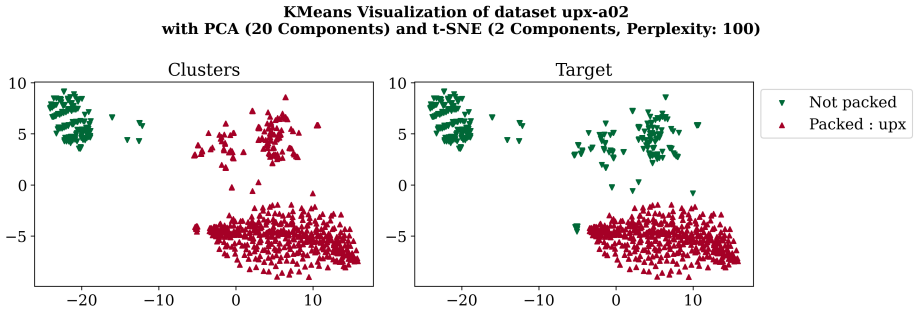


Fig. 3. Visualization of clusters of not-packed and UPX-packed samples after applying alteration A02 (moving EP to new section)

On Figure 3, we can see on the top-right that, between predictions and true labels, the alteration makes a bunch of not packed samples be detected as packed, then resulting in *alarm flooding*. Metrics in Table 1 confirm this result as the precision is now significantly impacted because of false positives while the recall remains maximal as there is no false negative. Clustering metrics drop to similar values as for A01, showing a significant degradation in the quality of clusters.

Dataset	Accuracy	Precision	Recall	Homogeneity	Completeness
BL	99.77%	100.00%	99.54%	97.90%	97.80%
A01	70.29%	100.00%	41.20%	24.40%	33.10%
A02	70.76%	62.85%	100.00%	25.70%	34.10%
A02-A01	71.23%	100.00%	43.06%	25.70%	34.10%

Table 1: Classification and clustering metrics for the K-Means model trained on datasets BL, A01 and A02

Finally, we applied both alterations A02 then A01 – as moving the EP to a new section first is more relevant for yielding a code section with a standard name – and we retrained K-Means once again. This gave us results very similar to those of A02, meaning that A02 provides a greater contribution to features perturbation than A01, as it can be seen in the work of D'Hondt *et al.* [6].

4 Conclusion

In this paper, we briefly stated the problematic of executable packing and its detection. We addressed unsupervised learning and its relatively poor exploitation in the current literature. We also shortly introduced adversarial learning and alterations in the problem space. We then explained our methodology taking advantage of previous studies of alterations and applying clustering and dimensionality reduction algorithms to obtain visualizations of the effects of these alterations. With the results of our experiments, we highlighted decreases in the accuracy of unsupervised-based classification when applying alterations to packed samples with visualizations, showing that an alteration could cause evasion while another could lead to alarm flooding. In the future, we aim to study the effects of alterations on clustering metrics to determine a novel method to evaluate if a sample was altered and to assess the robustness of features.

Acknowledgements. The authors are funded by the CyberExcellence project (RW, Convention 2110186). They want to express their gratitude to Sébastien Martinez who made a significant contribution through his master's thesis.

References

1. Bertrand Van Ouytsel, C.H., Dam, K.H.T., Legay, A.: Analysis of machine learning approaches to packing detection. Computers & Security (2023)
2. Biondi, F., Enescu, M.A., Given-Wilson, T., Legay, A., Noureddine, L., Verma, V.: Effective, efficient, and robust packing detection and classification. Computers & Security (2019)
3. Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., Kurakin, A.: On Evaluating Adversarial Robustness (2019)
4. Choi, M.J., Bang, J., Kim, J., Kim, H., Moon, Y.S., Díaz-Verdejo, J.: All-in-one framework for detection, unpacking, and verification for malware analysis. Security and Communication Networks (2019)
5. Demetrio, L., Biggio, B., Roli, F.: Practical attacks on machine learning: A case study on adversarial windows malware. IEEE Security & Privacy (2022)
6. D'Hondt, A., Bertrand Van Ouytsel, C.H., Legay, A.: Evading packing detection: Breaking heuristic-based static detectors. In: 21st Conference on Detection of Intrusions and Malware & Vulnerability Assessment (2024)
7. D'Hondt, A., Bertrand Van Ouytsel, C.H., Legay, A.: Experimental toolkit for manipulating executable packing. In: Risks and Security of Internet and Systems. Springer Nature Switzerland (2024)
8. Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., Roli, F.: Adversarial malware binaries: Evading deep learning for malware detection in executables. In: 2018 26th European Signal Processing Conference. IEEE (2018)
9. Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Wu, C., Ji, S., Luo, T., Wu, J., Wu, Y.: Adversarial attacks against windows PE malware detection: A survey of the state-of-the-art. Computers & Security (2021)
10. Noureddine, L., Heuser, A., Puodzius, C., Zendra, O.: SE-PAC: A self-evolving packer classifier against rapid packers evolution. In: Proceedings of the 11th ACM Conference on Data and Application Security and Privacy (2021)