

Article

Identifying Malware Packers through Multilayer Feature Engineering in Static Analysis

Ehab Alkhateeb ^{1,*} , Ali Ghorbani ¹  and Arash Habibi Lashkari ² 

¹ Canadian Institute for Cybersecurity (CIC), Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada; ghorbani@unb.ca

² Behaviour-Centric Cybersecurity Center (BCCC), School of Information Technology, York University, Toronto, ON M3J 1P3, Canada; ahabibil@yorku.ca

* Correspondence: ehab.alkhateeb@unb.ca

Abstract: This research addresses a critical need in the ongoing battle against malware, particularly in the form of obfuscated malware, which presents a formidable challenge in the realm of cybersecurity. Developing effective antivirus (AV) solutions capable of combating packed malware remains a crucial endeavor. Packed malicious programs employ encryption and advanced techniques to obfuscate their payloads, rendering them elusive to AV scanners and security analysts. The introduced research presents an innovative malware packer classifier specifically designed to adeptly identify packer families and detect unknown packers in real-world scenarios. To fortify packer identification performance, we have curated a meticulously crafted dataset comprising precisely packed samples, enabling comprehensive training and validation. Our approach employs a sophisticated feature engineering methodology, encompassing multiple layers of analysis to extract salient features used as input to the classifier. The proposed packer identifier demonstrates remarkable accuracy in distinguishing between known and unknown packers, while also ensuring operational efficiency. The results reveal an impressive accuracy rate of 99.60% in identifying known packers and 91% accuracy in detecting unknown packers. This novel research not only significantly advances the field of malware detection but also equips both cybersecurity practitioners and AV engines with a robust tool to effectively counter the persistent threat of packed malware.



Citation: Alkhateeb, E.; Ghorbani, A.; Habibi Lashkari, A. Identifying Malware Packers through Multilayer Feature Engineering in Static Analysis. *Information* **2024**, *15*, 102. <https://doi.org/10.3390/info15020102>

Academic Editor: Libing Wu

Received: 10 January 2024

Revised: 30 January 2024

Accepted: 7 February 2024

Published: 9 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: obfuscated malware; antivirus; packed malware; payloads; malware packer classifier; feature engineering

1. Introduction

The introduction of new technologies like smartphones, tablets, and wearable devices has revolutionized the way we engage with digital information, allowing us to access it instantly from anywhere in the world. Additionally, communication and information sharing have become more accessible than ever with the help of social networks, email, instant messaging, and IP telephony. Unfortunately, the growth in technology has also resulted in the emergence and spread of various types of computer malware, such as viruses, worms, and trojans. Cybercriminals and hackers often use these malicious software programs for harmful purposes, making protection against them more challenging than ever. In fact, malware deployment has become a common strategy in digital warfare, which can include cyberespionage or cybersabotage [1,2].

Currently, the majority of executable files being analyzed, including malicious ones, are known as PE files (Portable Executable) and have the PE format, primarily used for the Windows family of operating systems—under which most malware applications are written. To bypass AV engines and hinder code analysis, malware writers often use specialized applications such as packers to obfuscate their files, demonstrating a keen interest in investing significant funds to leverage this technology [3]. McAfee's report [4] states that over 80% of malware programs are packed.

The main objective of packers is to perform packing, a form of code obfuscation that can involve encrypting or compressing a file. Packer identification is the process of determining the packer application used to pack an original code. As such, the first stage and priority in the AV engine are to identify the packer used to unpack the packed malware, to determine the attacker's intention or simply quarantine the file if it is a malicious packer.

The current gap in packer identification involves various aspects, including dataset composition, packer family classification, the recognition of unknown packers, and the landscape of packer attack vectors is expansive, with numerous emerging antianalysis techniques. Antivirus engines strive for practical and low-overhead approaches to address these challenges, often favoring static analysis methods for their minimal overhead. This research is dedicated to tackling the current packer identification problem by detecting both known and unknown packers, utilizing static analysis for feature engineering. Our proposed approach places significant emphasis on feature engineering to identify the most prevalent features in static analysis, with the ultimate goal of bridging the existing gap in packer identification.

The research contribution in this paper can be summarized as follows:

- We meticulously constructed datasets for benign and malware packers through a multistage process, ensuring well-organized samples for training our classifier.
- Our proposed packer classifier incorporates a multilayer feature engineering approach, selecting engineered features based on their prevalence and performance.
- The classifier achieves a high level of accuracy while maintaining exceptional efficiency, surpassing classical signature-based methods.
- Notably, our classifier excels in detecting both family-based packers and unknown packers in real-world scenarios.

The rest of this article is organized as follows: Section 3 provides a brief background on packed malware, including an overview of packers and their specifications, as well as details about the handling life cycle of malware packers. Section 4 describes the proposed approach using multilayer feature engineering. Section 5 discusses the construction of the dataset, encompassing benign and malware-packed samples, along with the experiments conducted and their corresponding results. Section 6 presents a comprehensive discussion regarding the significance and analysis of the obtained results. Section 7 concludes the study. Finally, Section 8 discusses the limitations of the present study and outlines potential areas for future work.

2. Related Works

2.1. Packers Identification Approaches

In recent years, numerous research papers have been published concerning the detection of malware packers. These studies vary in terms of the attributes (or qualities) utilized to distinguish between packed malware and reputable applications. These attributes can be acquired using either static or dynamic analysis methods.

2.1.1. Dynamic Analysis

Dynamic analysis encompasses the execution of the packed executable within a controlled environment to monitor its actions in real time. The authors in [5] presented a thought-provoking insight into the realm of generic unpackers, highlighting their vulnerability due to reliance on specific packer families. This reliance inadvertently exposes these unpackers to faulty assumptions, which are skillfully exploited by malware authors who craft intricate handmade packers. This realization underscores the need for more adaptable and sophisticated approaches in countering the ever-evolving landscape of malware protection.

Hai et al. [6] proposed a technique utilizing metadata signatures and control flow graph (CFG) analysis for identifying packed code. The method included disassembling and generating CFGs using BE-PUM through concolic testing. Within the CFG, obfuscation techniques were pinpointed using formal criteria, and the packer was determined by applying

the chi-square test to the metadata signature. Assessment with 12 packers and 12,814 malware samples unveiled 608 disparities compared to tools like PEiD, CFF Explorer, and VirusTotal. Nonetheless, the intricacies of the approach contributed to slower processing.

Alkhateeb et al. [7] introduced a dynamic API analysis for packed malware detection, leveraging Naive Bayes and Levenshtein distance for robust differentiation. The primary data, based on API frequency, underwent training using a heuristic approach. Results from experiments on 1000 benign and 4000 malware samples demonstrated superior detection rates for packers such as PECompact and UPX, achieving over 90% accuracy, albeit with a minor computational cost in dynamic analysis.

Men'endez et al. [8] combined entropy analysis and classification algorithms. Their approach imitated behavior and achieved accuracy rates of nearly 98% in the best case and 75% in the worst across 57 antivirus engines, particularly targeting Windows' disk-resident malware.

Munkhbayar et al. [9] proposed dynamic entropy analysis for packer detection, categorizing samples as increasing, decreasing, or static. The method utilized SAX representations of entropy values and several similarity algorithms for classification. However, the limited sample size affected technique efficiency and performance. Graphically visualized patterns, instead of substantial features, were used for classification. The time complexity of the dynamic analysis was a major concern.

Munkhbayar et al. [10] introduced a multilayer executable packer detection approach using entropy values in symbolic representations (SAX). The method aimed to detect repacked and multipacked benign executables. However, the process's time complexity, involving memory extraction, debugger usage, and unpacking process verification, posed challenges. Additionally, relying solely on entropy for detection proved inadequate for the wide array of packers.

Lim et al. [11] introduced a memory analysis technique called Mal-Flux, designed to extract hidden code from packed samples by focusing on the end of unpacking routines. Operating within dynamic analysis, Mal-Flux was computationally intensive, resembling earlier efforts in dynamic and emulation analysis.

Examining the dynamic techniques for detecting malware discussed earlier, we can deduce that dynamic analysis is both time-intensive and resource-demanding, leading to escalated scalability concerns.

2.1.2. Static Analysis

Static analysis examines packed files' attributes and structures without execution. In [12], the authors proposed Learning with Local and Global Consistency (LLGC) for semi-supervised classification of packed binaries. With 10% of the malware dataset from Perdisci et al. [13], LLGC achieved a 9.5% lower accuracy than reported by Perdisci. The highest accuracy, 99.3%, was observed using random forest (supervised) features. One example of how supervised ML algorithms can be applied was discussed in [14], where the authors explored the potential of utilizing supervised machine learning for intrusion detection systems (IDS), achieving a high level of classification performance. Another study [15] adopted an ensemble approach, using 209 features, including structural and raw data from the PE32 file header. Ugarte et al. [16] proposed an anomaly detection method for identifying variations in packed executables. In packing detection [17], XOR-based algorithms with lower entropy values were suggested. Two approaches, ESCAPE [18] and PEAL [19], were employed. Feature vectors, normalized with WEKA, achieved a 97.4% detection accuracy using five machine learning algorithms. Mimura et al. [20] proposed the exploration of large-scale datasets as a potential avenue for research. However, their approach relied solely on a simple signature-based packer detector with an approximate 30 percent false-negative rate. This reliance posed limitations on their ability to comprehensively identify the packer names associated with the samples. As a result, the applicability of their experimental results may be constrained, particularly when dealing with highly sophisticated packers that can effectively evade signature-based packer detectors.

Jin et al. [21] proposed a PE header-based method for packer classification, achieving around 0.99% precision and recall. Similar to [22], the approach used PE file header analysis with nine features and the Euclidean distance for classification. Despite fast header checking, the simplicity of the classification may lead to less accurate packer detection.

Various graph-based methods have been proposed for packer identification. In the work by Saleh et al. [23], a static analysis approach is employed to construct a control flow graph, utilizing a disassembler for extracting instruction flow. However, its susceptibility to evasion by new packers arises from its reliance on large data signatures and the need for frequent updates. Similarly, Li et al. [24] utilize graph representation and IDA Pro to extract assembly instructions, addressing obfuscation through various filtration methods. Despite its efficacy, this approach faces challenges in processing extensive datasets, leading to the abandonment of packer classification for certain types. In a related work, Liu et al. introduce a two-stage packer identification method (2-SPIFF) in [25], based on function call graphs and file attributes. A limitation of this method is that it primarily focuses on distinguishing between packed and nonpacked executable files.

Kancherla et al. [26] suggested static analysis for efficient packer identification. They used digital image processing with byte and Markov plots to visualize packers, extracting texture features for support vector machine training. Markov plots performed better, achieving high accuracy for UPX and Themida. However, they obtained limited effectiveness against unknown packers and multipacking, with accuracies ranging from 83.94% (Armadillo) to 99.05% (Themida).

Jung et al. [27] introduced a technique for packer identification using byte sequences, incorporating two main components: detecting encrypted sections within PE files and analyzing byte frequencies. The method demonstrated an average accuracy of 91.6% in identifying various packing algorithms. Moreover, Dam et al. [28] proposed an association rule mining method for multiclass packer detection based on YARA rules. Despite achieving high accuracy for malicious programs, the study did not address unknown packer detection. Biondi et al. introduced a static analysis packer classifier with 6 features, yielding a total of 119 features [29]. In another approach, Bergenhotlz et al. employed recurrent neural networks for packer identification based on x86 instruction mnemonics [30]. A similar strategy utilizing PE raw features was adopted for malware classification and packers, as demonstrated in [31]. Nouredine et al. proposed a self-evolving packer classifier that leveraged packer clustering in both offline and online phases. However, they faced challenges related to the fragility of the Levenshtein distance on ASM sequences, particularly when dealing with diverse new packer families [32].

Upon analyzing the discussed static packer identification methods, it becomes apparent that static analysis takes precedence over dynamic analysis, primarily due to its efficiency in rapidly assessing code or binaries without execution. The emphasis on static analysis stems from its practicality in scenarios with resource constraints, as it eliminates the need for complex execution environments. Nevertheless, recent research on packer identification faces challenges, particularly in proper family classification and identifying unknown packers.

To address these challenges, we implemented a classifier featuring a multilayer feature engineering approach. This involved systematically generating and refining a diverse set of features from various layers of the data or code being analyzed. By doing so, we aimed to capture a more comprehensive representation of the underlying characteristics and patterns inherent in the data. This strategy enables a more nuanced understanding of the intricacies of unknown packers and enhances the effectiveness of our identification and classification methods.

3. Background

3.1. Packers

Packers are described as computer software or a tool. The original purpose of packers is to protect applications from reverse engineering. While this may be accomplished for

legitimate causes—to save disk space or lower data transmission time—packers are also utilized by hackers as a form of code obfuscation. The packing creates an extra layer of code that envelopes a piece of malware to conceal it. Malware creators can either design their own packer (malicious packer) or use a legitimate one for this purpose.

3.2. The Functioning of Packing

Packing involves employing packers to obfuscate, and encrypt executable files, such as malware. The objective is to avoid detection by security software and analysts. This action introduces complexities that hinder antivirus and similar security tools from recognizing and scrutinizing malicious content effectively. A packer program transforms a designated file into a new form; during runtime execution, the unpacked code is generated and written in memory. Figure 1 illustrates the conversion process and the subsequent unpacking and writing of code in memory.

A stub, often a compact fragment of code, is embedded within a packed executable. This specific code snippet acts as the initial loader when the packed file is executed. Its principal objective is to activate the process of unpacking and decrypting the compressed and obscured contents of the packed file. This, in turn, grants permission for the original executable code to operate within the system.

The duties of the stub code encompass the following functionalities:

- Initialization: as the packed executable is launched, the stub code is loaded into memory.
- Decryption and extraction: containing directives for decrypting and extracting the authentic payload of the packed file, the stub code addresses the typical encryption and obfuscation employed to elude detection.
- Execution: upon successful decryption and unpacking of the content, control is seamlessly handed over from the stub code to the unpacked code, allowing its execution.
- Countermeasures against analysis: certain stubs might incorporate countermeasures designed to complicate the efforts of security researchers attempting to comprehend and scrutinize the concealed content.

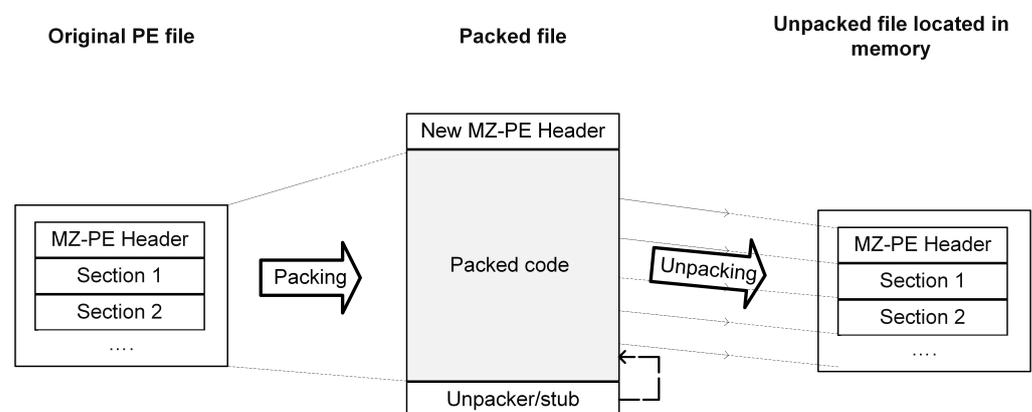


Figure 1. The procedural steps involved in both packing and unpacking processes for a malware sample, illustrating the transformation and manipulation undergone by the code during these essential phases.

The presence of the stub is pivotal, enabling the packed executable to function accurately and carry out its intended malicious operations. All the while, it upholds the vital obfuscation and evasion characteristics intrinsic to packing techniques. Acting as the point of entry, the stub initiates the unfolding and execution of the covert content harbored within the packed file.

3.3. Packer Identification and Unpacking

The recognition of packers assumes a pivotal role in advancing malware analysis by simplifying the unpacking procedure of compacted malicious software. Through precise

identification of the employed packer, antivirus engines or analysts can employ targeted unpacking methods (profile or generic) to undo the alterations introduced during the packing process. One common tool used in profile unpacking is an executed script designed to unpack using a debugger [33], and the Intel PIN tool is an example tool that can be utilized for generic unpacking [34]. As a result, the malware's authentic code is unveiled, enabling the revelation of its genuine intent, encompassing activities like data theft, system manipulation, or other forms of malevolent behavior.

4. Approach

First, we present the overview of our proposed methodology, illustrated in Figure 2, featuring an advanced packer classifier, followed by in-depth details. The classifier is structured through three pivotal stages: dataset construction, feature engineering layers, and feature set selection and packer classification.

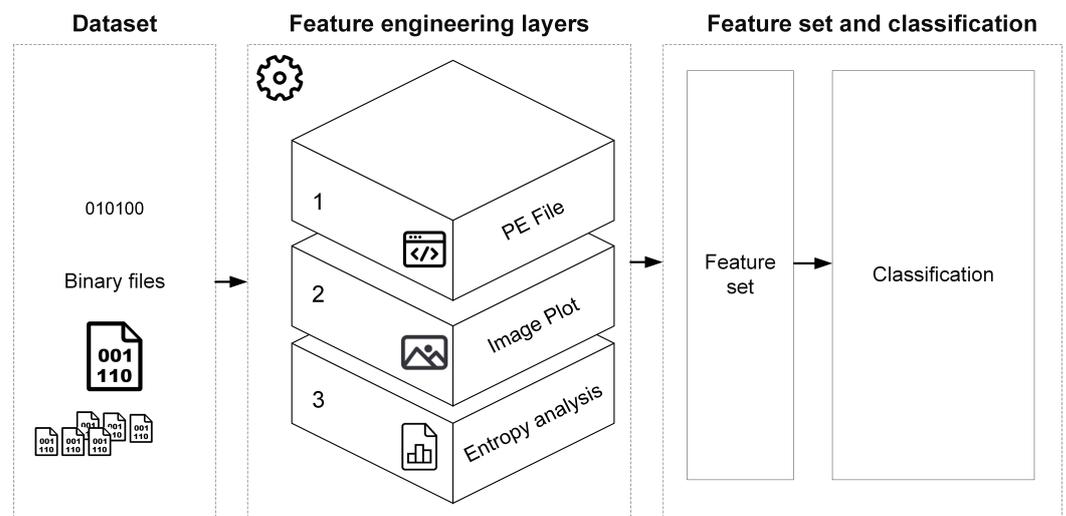


Figure 2. The integrated systemic framework: the proposed approach encompasses key components, including the dataset, feature engineering, feature set, and classification stages.

In the initial phase, we meticulously curate a comprehensive dataset, prioritizing both the precision of collection and the intricacy of processing. This meticulous approach lays a solid foundation for subsequent analyses, ensuring the inclusion of diverse samples and eliminating potential biases. Further details about the data construction mechanism can be found in Section 5, where the dataset construction intricately influences the subsequent feature engineering stage. The quality and nature of the data, along with their domain of origin, are pivotal factors. Maintaining a delicate balance and ensuring integrity during data preprocessing, especially in machine learning and statistical analyses, are paramount, providing a well-balanced dataset that offers a more faithful representation of real-world distributions. This, coupled with data integrity, ensures that insights derived from the data are not only credible but also meaningful.

The essence of our methodology lies in the feature engineering stage, where we strategically choose features that capture both the low-level and high-level characteristics of executable files. This intentional fusion of features provides nuanced insights into the intrinsic attributes of the code, fostering a more holistic understanding.

The apex of our methodology is realized in the feature set selection and classification stage. Here, we empirically choose the best-performing features and integrate them into cutting-edge machine learning techniques. This final step enables us to effectively differentiate between files with specific family names, those employing unknown packers, and benign files. Our meticulous feature selection, combined with robust classification algorithms, significantly enhances the overall proficiency of our approach.

4.1. Feature Engineering

In the preparation phase of applying machine learning algorithms, the transformation of raw data sources into meaningful features holds paramount importance. These features, often manifesting as columns within the data matrix supplied to the machine learning algorithm, encapsulate distinct attributes of the observations. This conversion process is instrumental in enhancing the algorithm's ability to discern patterns, make accurate predictions, and uncover insights from the data. This critical step, commonly called feature engineering, is of the utmost importance in the machine learning process. The purpose of feature engineering can be summarized as follows:

- Engineering or preparing the input data into features that can be comprehended by the machine learning algorithm, thereby meeting its requirements.
- Engineering or transforming variables into features that enhance the performance of machine learning algorithms in terms of predictive accuracy, interoperability, or both.

Feature engineering plays a vital role in machine learning as it can enhance model performance, extracting datasets compatible with algorithms, and extract additional information and insights from the data. The primary objective is to identify and select features that accurately classify packer families and detect unknown packers while minimizing costs. The system design encompasses multiple layers and specific features, leveraging three key components: PE, image plot, and entropy. Each defensive element is designed to safeguard a particular area that is susceptible to being targeted by hackers. By incorporating these components, the approach strives to elevate its overall security stance. Also, including diverse features extracted from these components enables the system to identify complex and unknown packers. This diverse set of features empowers the system to detect and classify different types of packers, including those that utilize advanced or previously unidentified techniques. The flexibility in feature selection contributes to the system's effectiveness in identifying and mitigating potential threats.

4.1.1. The PE File Format

This section provides a concise overview of the feature engineering techniques we utilized for the PE file format, encompassing PE section names, section numbers, and more. For extracting raw features, the use of the open-source tool *pefile* [35] was instrumental.

The PE headers contain information about the file format, which allows the identification of a packed program from a benign program using details kept in the PE file header and the file's structural features [36]. To identify packed samples, a total of 51 carefully selected features were used as shown in Table 1. We extracted numerous features from the PE headers, such as the DOS executable header features, which contain relocation information. The optional header includes features such as the optional header magic number that determines whether an image is a PE32 or PE32+ executable and many others. Additionally, we extracted section details such as section numbers and section names. Section names are important since most packers, especially those of the protectors' packer family (nonmalicious packers), have a fixed section name. Some may not even be compressed or encrypted in most cases. We propose a packer section dictionary, a repository that contains packer algorithms and their unique PE section names. For instance, UPX sometimes uses UPX1, UPX0, or UPX2 to refer to different section names. Thus, we utilized the Levenshtein distance (LD) to categorize all section names from a specific packer under one label. Table 2 below gives details on the different sections of UPX.

Table 1. PE layer features.

AddressOfEP	BaseOfCode	BaseOfData	Checksum	DllCharacteristics	e_cblp	e_cp	e_cparhdr	e_crlc	e_cs	e_csum	e_ip	e_lfanew	e_lfarlc	e_minalloc	e_oemid	e_oeminfo	e_sp	e_ss	ImageBase	LoaderFlags	Magic	MajorLinkerVersion	MajorSubsystemVersion	MinorLinkerVersion	MinorSubsystemVersion	Reserved1	sec_Num	secaddress	SectionAlignment	SizeOfCode	SizeOfHeapCommit	SizeOfHeapReserve	SizeOfImageSizeOfIniData	SizeOfStackReserve	Subsystem	entSize	e_magic	e_maxalloc	e_ovno	FileAlignment	MajorImageVersion	MajorOperatingSystemVersion	MinorImageVersion	MinorOperatingSystemVersion	NumberOfRvaAndSizes	SizeOfHeaders	SizeOfStackCommit	SizeOfUninitData	Tag
-------------	------------	------------	----------	--------------------	--------	------	-----------	--------	------	--------	------	----------	----------	------------	---------	-----------	------	------	-----------	-------------	-------	--------------------	-----------------------	--------------------	-----------------------	-----------	---------	------------	------------------	------------	------------------	-------------------	--------------------------	--------------------	-----------	---------	---------	------------	--------	---------------	-------------------	-----------------------------	-------------------	-----------------------------	---------------------	---------------	-------------------	------------------	-----

Table 2. A sample of UPX dictionary sections.

UPX0	→ Packed.UPX
UPX1	→ Packed.UPX
UPX2	→ Packed.UPX
UPX!	→ Packed.UPX
.UPX0	→ Packed.UPX
.UPX1	→ Packed.UPX

In [37], Vladimir Levenshtein proposed the well-known distance measure that bears his name. The Levenshtein (or edit) distance is based on the minimum number of insertions, deletions, and substitutions required to transform one string into another.

Let LD be the Levenshtein distance function. Then, for example, we have

$$LD(\text{sea}, \text{set}) = 1,$$

since we can simply substitute t for a to transform sea to set. As another example,

$$LD(\text{trail}, \text{fails}) = 3,$$

since we could delete the t, substitute f for r and insert s to transform trail into fails, and no fewer operations will succeed.

The Levenshtein distance can be computed efficiently as follows. Let S and T be two strings, and let $LD(i, j)$ be the Levenshtein distance between the first i characters of S and the first j characters of T . Also, let $|S|$ and $|T|$ be the length of sequences S and T , respectively. Then, the Levenshtein distance between S and T is given by $LD(|S|, |T|)$, where

$$LD(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0 \\ \min \begin{cases} LD(i - 1, j) + 1 \\ LD(i, j - 1) + 1 \\ LD(i - 1, j - 1) + I(i, j) \end{cases} & \text{otherwise.} \end{cases} \quad (1)$$

The work in [38] gives more details about the Levenshtein distance. In this research, we can identify UPX packer under one label, “UPX”, using the Levenshtein distance. Thus, using the LD provides a simple way for dictionary reduction and helps us identify new UPX packers in the wild; this includes new UPX versions and customized packers from malware creators that utilize UPX.

4.1.2. Image Plot

The process of identifying packers using image features involves analyzing the visual representation of a binary file to recognize patterns commonly associated with packing techniques. This requires employing machine learning algorithms or pattern recognition techniques to train a model on a dataset comprising both known packed files per family and nonpacked files. Subsequently, the trained model can be applied to identify the presence of a packer in new, previously unseen files based on their extracted image features. This method, among various techniques in malware analysis and cybersecurity, enhances the detection of potentially malicious software. An effective strategy for gleaning essential insights from PE files involves converting raw PE byte data into a grayscale image with

specific dimensions. Initially, the binary executable undergoes transformation into an 8-bit 1D vector, with each 8-bit value being mapped to an intensity value, forming a pixel. In the plot generation phase, this one-dimensional vector is further converted into a two-dimensional vector and resized to a fixed size, as illustrated in Figure 3. This process navigates through successive phases within the image plot layer, encompassing the initial transformation of packed binary images, pixel transformation, and resizing, culminating in the strategic selection of image bases. This visual journey unravels the nuanced progression within each phase.

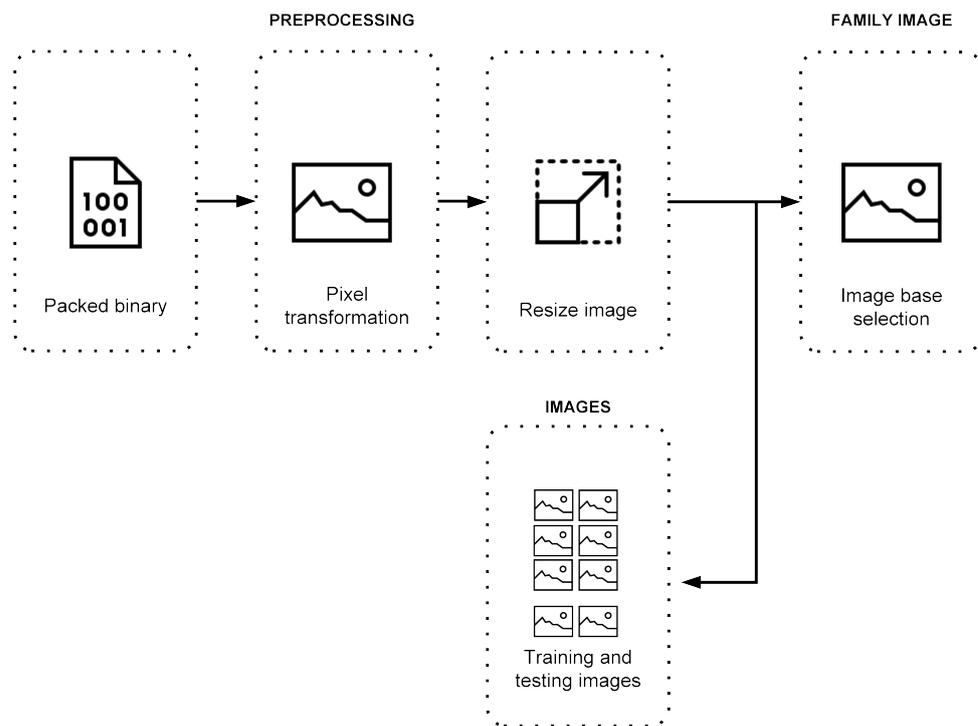


Figure 3. Stages within the image plot layer: preprocessing, family image selection, and testing and training.

The image base serves as a reference image; for instance, in the case of a packed UPX, we select one image from the collected packed samples. This chosen image becomes a base for calculating comparative features, including the Manhattan distance and Gabor jets filter distances. To derive values for these comparative features, each sample is compared with the list of family images.

Image difference is a method used to analyze and measure the variances or alterations between two images. It is widely applied in diverse fields like computer vision, image processing, and computer graphics.

Two images are compared by subtracting the corresponding pixel values during the image difference process. The outcome reveals the variations between the images at each pixel location. The magnitude or absolute difference of the pixel values can serve as a metric for quantifying the dissimilarity between the images.

Manhattan distance:

The Manhattan distance is always a non-negative value, representing the total distance traveled along the grid lines to move from one point to another. It is commonly used in various fields, including image processing, computer vision, clustering, and pathfinding algorithms. In image processing, it can be used to measure the dissimilarity between two images based on pixel intensities or other image features.

The Manhattan distance, also called the taxicab distance or the city block distance, calculates the distance between two real-valued vectors.

The distance between a point P and a line L is defined as the smallest distance between any point $M \in L$ on the line and P :

$$d(P, L) \equiv \min_{M \in L} d(M, P) \quad (2)$$

The Manhattan distance between two points is defined as:

$$d(M, P) \equiv |M_x - P_x| + |M_y - P_y| \quad (3)$$

We calculated a Manhattan normalized distance by calculating the Manhattan distance of each pixel, then summing the total distances, and calculating the sum of the Manhattan distance over the image base size. Also, we added two more features, such as the zero difference (the number of segments not equal to zero) and the normalized zero difference over the image size. For each packer family, we assigned an image base to represent the packer when comparing other images. In the experiments we conducted in Section 5, we used each packer's image base and calculated the normalized Manhattan distance and other features we mentioned. For instance, when testing new data, the model compares the tested file with each packer's family image base. The one with the minimum distance is selected, and its values are stored.

Gabor filters and jets:

The Gabor filter (assumed to be centered at zero) is the product of a sinusoid and a Gaussian:

$$g(x, y; \lambda, \theta, \phi, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \phi\right) \quad (4)$$

where

$$x' = x \cos \theta + y \sin \theta, \quad y' = -x \sin \theta + y \cos \theta. \quad (5)$$

The filter has the following characteristics:

Wavelength: the number of cycles/pixel is λ . Orientation: the angle of the normal to the sinusoid is θ . Phase: the offset of the sinusoid is ϕ . Aspect ratio: ellipticity is produced with $\gamma < 1$. The spatial envelope of the Gaussian, σ , is managed by the bandwidth, which at unity gives $\sigma = 0.56\lambda$. Gabor filters are a good model of simple cells in the visual cortex. The half-response spatial frequency bandwidth b (in octaves) of a Gabor filter is related to the ratio σ/λ :

$$b = \log_2 \frac{(\sigma/\lambda)\pi + \sqrt{\log 2/2} \sigma}{(\sigma/\lambda)\pi - \sqrt{\log 2/2} \lambda} = (1/\pi) \sqrt{\log 2/2} \frac{2^b + 1}{2^b - 1}. \quad (6)$$

The value of σ cannot be specified instantly. It can only be changed via the bandwidth b . The bandwidth value must be specified as a real positive digit. The default is 1, in which case σ and λ are linked as follows: $\sigma = 0.56\lambda$. The smaller the bandwidth, the larger σ , the support of the Gabor function, and the number of visible parallel excitatory and inhibitory stripe zones.

Gabor jets are a set of filters that are used to extract the local frequency details from the images. These filters are normally linear filters with impulse responses described by a harmonic function and a Gaussian function. Gabor jets are widely employed in face images to detect eyes or other regions of the face, as well as in ATR systems as in [39] for target decision. We utilized disparity similarity measures; for instance, one might locate the position of the particular location in the PE file as shown in Figure 4 by scanning over the whole image. At each position in the image, the similarity between the reference Gabor jet and the Gabor jet at that location is computed using the similarity function. For this calculation, both traditional Wiskott et al. [40] and innovative Gunther et al. [41] similarity functions can be used.

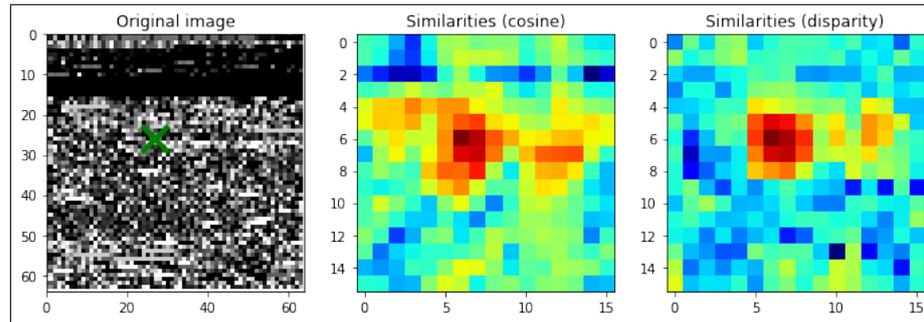


Figure 4. Generation of Gabor jets from an image derived from a packed original sample.

Another benefit of the disparity similarity function is providing a reference Gabor jet extracted at a reference location. The disparity function can compute the spatial offset of another given Gabor jet. It can evaluate the disparity (difference in spatial position) between two Gabor jets as long as they arise from a similar region. Thus, for such cases, we used the disparity distance to find the difference in spatial position between two Gabor jets.

Algorithm 1 illustrates the extraction of feature sets for Gabor jets and Manhattan distance, including other features to compare distances and differences between input images I_n and images of each packer family, represented as G_n .

Algorithm 1 Image plot feature set

Input: (I_i) ▷ Images.
Output: (M_i, G_i) ▷ Feature sets
1: $c \leftarrow \text{len}(F_i)$ ▷ F_i images for packer families; c is # of families
2: $l \leftarrow \text{len}(I_i)$ ▷ l number of the input images
3: **function** CALCULA_MANH_DIST(I_i)
4: **for** $k \leftarrow 1$ to c **do**
5: **for** $j \leftarrow 1$ to l **do**
6: $M_i = \text{Find}(F_k, I_j)$ ▷ Find difference of F_k and I_j
7: **end for**
8: **end for**
9: **return** M_i ▷ Features
10: **end function**
11: **function** CALCULA_GABORJET_DIST(I_n)
12: **for** $v \leftarrow 1$ to c **do**
13: **for** $n \leftarrow 1$ to l **do**
14: $G_i = \text{Find}(F_v, I_n)$ ▷ Gabor jets between F_v and I_n
15: **end for**
16: **end for**
17: **return** G_i ▷ Features
18: **end function**

4.1.3. Entropy Analysis

Entropy analysis, a widely utilized scientific concept, evaluates the disorder, randomness, or uncertainty in measurable entities. Its applications span fields such as cryptanalysis [42] and malware detection [43], proving crucial in identifying packers [44]. Derived from scanning byte sequences or the entire file, the rolling entropy sequence helps grasp the nature of the packed executable during unpacking and early packer identification. Diverse measures, including Shannon's entropy [45], quantify pattern change, as outlined in the subsequent equation:

$$H(x) = - \sum_{i=1}^n P_i \log_2 P_i \quad (7)$$

where $H(x)$ is the measured entropy value and $p(i)$ is the probability of the i th unit of information in the series of n symbols of event x . The base number of the logarithm (b) can

be any real number greater than 1. However, a value of 2, 10, or e is generally used. In this section, we propose an accurate entropy analysis of the packed files. The entropy for all PE file sections can cause false positive rates for packer identification. Thus, to use entropy, we propose a PE section analysis, where we analyze the appropriate PE sections and truncate the unwanted ones that cause worse entropy results. PE files could have different sections; each one could have several pieces of data.

In this layer, we provide entropy analyses for PE sections. The `.text` section in executable files is typically read-only and executable, ensuring that its instructions cannot be modified during runtime and are directly executed by the processor. Other sections like `.data` or `.text` store data or uninitialized variables. Located at a specific memory address, the `.text` section contains a consecutive series of binary instructions executed sequentially. These instructions encompass various operations, such as arithmetic calculations, control flow instructions (e.g., branches and jumps), and function calls. In essence, the `.text` section plays a vital role as it holds the executable instructions of a program, enabling the computer to execute the code and carry out the intended operations. Based on several statistical analyses conducted on the `.text` section of various packed and benign files, we have observed that most benign applications have entropy values below 7, as shown in Figure 5.

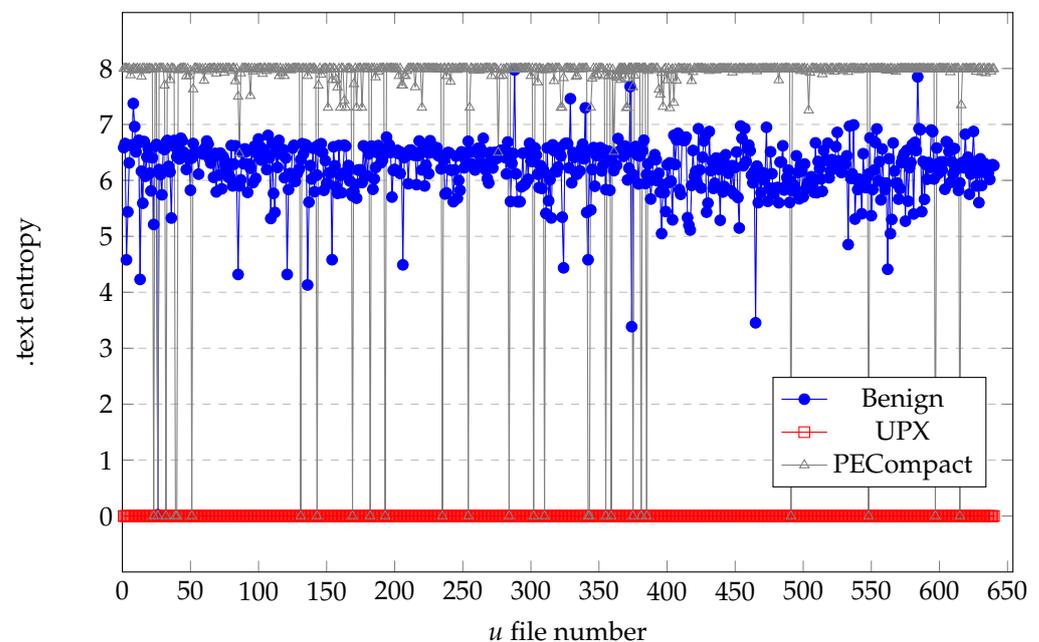


Figure 5. Comparison of the entropy score in the `.text` section between normal files and packed files.

The inclusion of an authorized list of entropy values can be a critical feature in appropriately assigning values to other features. In a previous update to the PE section features, we introduced a novel attribute referred to as a “tag”, corresponding to the name of the packer used. Each string value was associated with a specific numerical value denoted as n . For example, the packer “UPX” was assigned a value 1. The value of n is influenced by the entropy score of the `.text` section only if the results indicate a high entropy value exceeding 7 and the original tag value is 0, indicating the absence of a detected packer name. In such instances, the n value is set to 10, signifying an unidentified packer.

Furthermore, as part of our implementation, we incorporated the calculation of the average entropy across all sections of a PE file. The average entropy of PE sections represents the average level of randomness or uncertainty within each section. By computing the average entropy, we gain insight into the extent to which data are evenly distributed or compressed across the sections of the PE file. A higher average entropy indicates a greater degree of randomness or compression, while a lower average entropy suggests a more structured and predictable arrangement of data.

classification results with those obtained using the PEid application. This comparative examination not only affirms the efficacy of our methodology but also provides valuable insights into its performance compared to an established tool. Building on this foundation, we transition into a subsequent phase of experimentation dedicated to the identification of unknown packers. Through a series of rigorous tests, we showcase the resilience of our approach in effectively discerning and classifying these enigmatic entities. Lastly, our evaluation extends to a broader context, systematically juxtaposing our approach with other pertinent works in the field.

5.1. Dataset Preparation

In Figure 7, we utilized the dataset as described in [46], sourced from two distinct origins. The first source consisted of online/benign samples, totaling 2162 samples, representing applications downloaded from the Softpedia website and then manually packed. Softpedia ensures the safety of these applications through rigorous checks with genuine application authors and multiple antivirus scanners before making them available online. Our second data source was VirusTotal, from which we obtained malware samples and then manually packed them, totaling 5116 samples used in our experiments. These malware samples were categorized based on packer names or families. To ensure accurate classification, we developed Python code for multisignature detection of each packer.

Our dataset included 2000+ signature records of various wild packers and their versions, unified under a single name for malware sample identification. The collection used a Windows 10 system for data download and a MacBook Pro with an M1 processor for image generation and feature results. Incorporating diverse sources aimed to enhance our classifier's robustness, considering variability in PE file structures crucial for an effective identification.

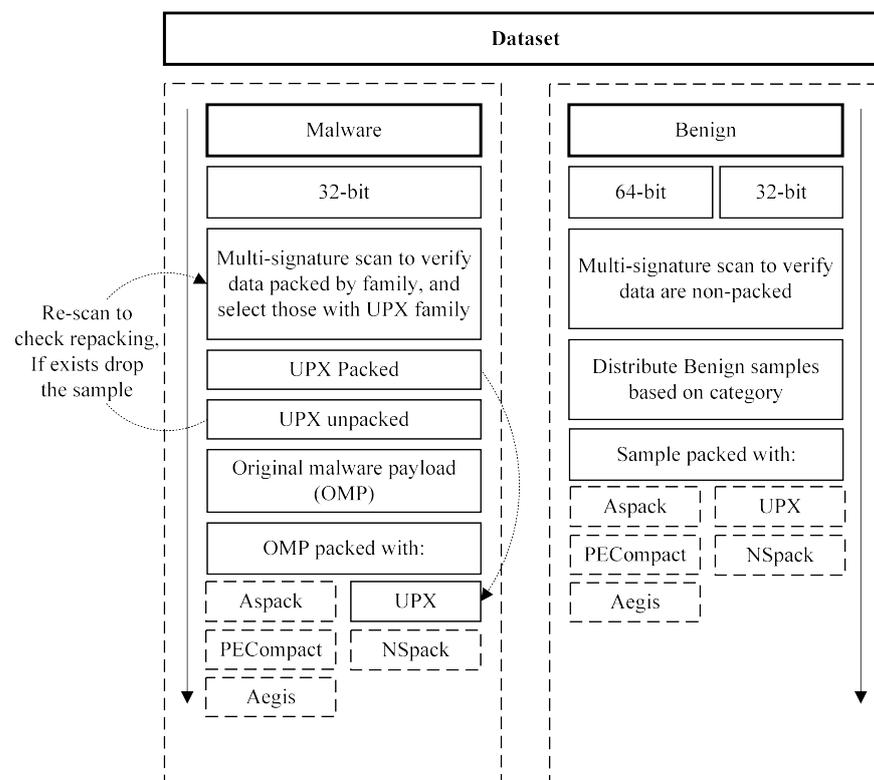


Figure 7. Dataset construction [46]. This intricate process involves meticulous steps such as data gathering, cleaning, and augmentation, ensuring a diverse and representative collection of samples.

Using a multisignature process, we initially selected UPX-packed samples, easily unpacked due to UPX's widespread recognition. After unpacking, we extracted the payload. Some samples, identified with another packer during the scan, were termed repacked

samples. Malicious actors often employ repacking and multilayer packing to resist analysis. Through various processes, we ensured our malware dataset contained only nonpacked payloads, which were then packed with five different packers.

5.1.1. Benign Samples

We systematically categorized the benign samples into 13 specific categories, ensuring a comprehensive representation of various program types. The categorization process aimed to establish a well-organized dataset that spanned a diverse range of benign samples, covering a wide array of software functionalities. This meticulously curated dataset is instrumental in training and evaluating machine learning models tailored to packer identification. Its diversity, capturing various software functionalities and packing strategies, provides a robust foundation for developing models capable of recognizing patterns across a broad spectrum of scenarios. Refer to Table 3 for a detailed breakdown of each benign category and the corresponding sample counts. Additionally, Table 4 showcases the packed samples, focusing on the most popular packers used by malware authors, as reported in [46].

Table 3. Benign file categories.

Bookmark managers	22	Browsers	43
Camera	13	Clipboard	23
Screen capture	11	Misc	29
System information	20	Tweaks	25
IP scanners	19	Desktop enhancement	21
Internet remote utilities	19	Graphic	90
Audio and video	91	Total	426

Table 4. Packed benign files.

Packer Name	Type	Source	Successfully Packed Samples
UPX	Protector	Online/benign	392
PECompact	Protector	Online/benign	384
NSPack	Protector	Online/benign	328
Aegis	Hostile	Online/benign	410
Aspack	Protector	Online/benign	398
		Total	1912

5.1.2. Malware Samples

The malware samples used in this experiment were classed based on the packer's family using two signature-based detection repositories. First, we selected 25K malware samples from VirusTotal and partitioned the samples into five folders, each folder containing 5K samples for ease of use.

We used two packer signature-based repositories to identify packed samples; we listed each packed sample under its main packer category. For those UPX-packed samples detected, we used them as our base for the malware data, unpacked the packed UPX samples detected, and used them to generate new packed samples for the other packers. All samples used were 32-bit files as illustrated in Table 5.

Table 5. Packed Win32 malware files.

Packer Name	Type	Source	Packed Samples
UPX	Protector	VirusTotal/malware	870
Aegis	Hostile	VirusTotal/malware	870
PECompact	Protector	VirusTotal/malware	863
NSPacker	Protector	VirusTotal/malware	869
Aspack	Protector	VirusTotal/malware	856
Total			4327

5.2. Evaluation Metrics

When performing classification predictions, we encounter four types of outcomes:

True positives (TP): these occur when our predictive model correctly identifies an observation as belonging to a specific class, and indeed, it does belong to that class.

True negatives (TN): in contrast, true negatives manifest when our model accurately predicts that an observation does not belong to a particular class, and in reality, it does not.

False positives (FP): false positives arise when our model erroneously predicts that an observation belongs to a certain class, even though it does not.

False negatives (FN): conversely, false negatives materialize when our model mistakenly predicts that an observation does not belong to a class, when in fact, it does.

We employed four essential machine learning evaluation metrics for our multiclass classification:

- **Accuracy:** this metric quantifies the percentage of correct predictions for the test data.
- **Precision:** precision signifies the fraction of relevant examples (true positives) among all the examples that our model predicted to belong to a specific class.
- **Recall:** recall measures the proportion of examples that our model correctly predicted as belonging to a class, in relation to all the examples that truly belong to that class.
- **F1-score:** the F1-score is the harmonic mean of precision and recall, offering a balanced assessment of a classifier's performance in multiclass scenarios.

These metrics collectively provide a comprehensive evaluation of our multiclass classification model's effectiveness in discerning and categorizing data points across multiple classes.

5.3. Experimental Results

This section offers a comprehensive overview of the outcomes obtained from our series of experiments. These experiments encompassed a trio of pivotal investigations, wherein we meticulously assessed a diverse array of feature sets that spanned three integral layers integrated into our pioneering packer identification methodology. These layers, namely Layer 1 (L1) for analyzing PE file formats, Layer 2 (L2) for scrutinizing image plots, and Layer 3 for delving into entropy analysis, were instrumental in our pursuit of robust packer identification.

For each of these experiments, we systematically partitioned our dataset into distinct training and testing subsets. To gauge the effectiveness of our approach, we harnessed the power of four distinct machine learning algorithms, subjecting each classifier to rigorous evaluation. The salient findings, highlighted in Figure 8, showcase the performance of these classifiers concerning the specific feature sets employed.

Notably, a remarkable observation emerged: the amalgamation of all three layers into the L1L2L3 feature set yielded the most impressive results. This consolidation proved especially potent when deployed with our chosen classifiers, random forest (RF) and J48. Of these, the random forest classifier delivered an astonishing accuracy rate of 99.6%.

Figure 8 visually presents the performance of each classifier, emphasizing the effectiveness of different feature sets, with the L1L2L3 feature set particularly standing out. For a detailed examination, Table 6 complements this visual representation by providing an extensive breakdown of results for each machine learning classifier.

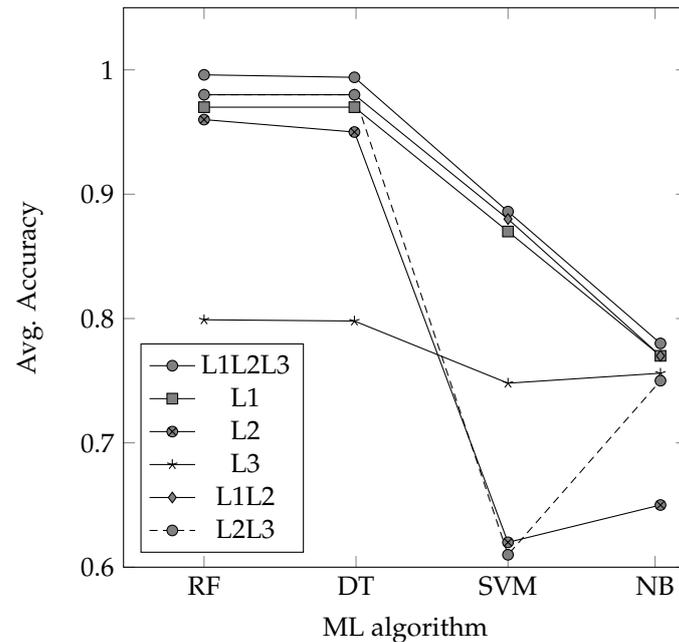


Figure 8. Family-based identification: feature set's average accuracy showcasing the superior performance of L1L2L3.

Tables 6 and 7 go beyond accuracy and include essential metrics like precision, recall, and F1-score for a more nuanced evaluation. The classification encompassed six distinct families, delineating five associated with packers and one category representing nonpacked samples (benign samples), as explained in Table 3. Through visual and tabular formats, this dual representation enhanced the clarity and depth of our performance evaluation and classification outcomes. To calculate each feature's importance, the mean accuracy decrease method works by evaluating the model's performance both before and after randomly permuting the values of a specific feature. The decrease in model accuracy resulting from permuting the feature is then computed, and this decrease indicates the feature's importance. Figure 9 illustrates the top 20 important features out of the 59 features used in previous experiments. It is evident that features derived from the image plot have a significant and impactful influence. We can also observe the significance of engineered features in comparison to raw features. Additionally, other features, such as "tag" and ".text" entropy, rank among the top 10 important features. Utilizing a diverse set of features enhances the approach's resilience against attackers who aim to pinpoint vulnerabilities within the various layers of the engineered system. This diversity in features ensures that potential attack vectors are scattered across multiple dimensions, making it more challenging for malicious actors to identify and exploit specific weaknesses in individual layers. As a result, the overall security and robustness of the system are significantly bolstered, reducing the likelihood of successful targeted attacks.

Our approach stands out for achieving high processing times and demonstrating efficient computational performance during training and inference stages. This efficiency is consistently observed across various operational scenarios, making our method reliable for tasks requiring swift processing. In benchmarks, our approach consistently showcased remarkable speed, highlighting its capability to handle complex tasks efficiently. This efficiency, coupled with our utilization of an 80–20 split for training and testing data (80% for training and 20% for testing), makes our approach a dependable choice for computational tasks demanding rapid and effective processing. Simultaneously, the implementation of image resizing yielded a substantial reduction in storage requirements, optimizing storage space by nearly fourfold in terms of image size. On average, each image now occupies a mere 2.91 KB of storage space, highlighting the efficacy of our resizing strategy. This

reduction is resource-efficient and facilitates more streamlined data processing, contributing to overall system efficiency.

Table 6. Classification results for L1L2L3: precision, recall, and F1-score.

Feature Set	Classifier	Family	Precision	Recall	F1-Score
L1L2L3	RF	UPX	1.00	1.00	1.00
		PECompact	1.00	0.99	0.99
		NSPacker	1.00	1.00	1.00
		Aegis	1.00	1.00	1.00
		ASPack	1.00	0.99	0.99
		Benign	0.96	0.99	0.98
	J48	UPX	0.99	1.00	1.00
		PECompact	0.99	0.98	0.98
		NSPacker	1.00	1.00	1.00
		Aegis	1.00	1.00	1.00
		ASPack	0.98	0.99	0.99
		Benign	0.95	0.92	0.93
	SVM	UPX	0.93	0.92	0.92
		PECompact	0.68	0.91	0.78
		NSPack	0.95	0.77	0.85
		Aegis	0.98	1.00	0.99
		ASPack	0.99	1.00	1.00
		Benign	0.56	0.22	0.31
	NB	UPX	0.99	0.45	0.62
		PECompact	0.52	0.80	0.63
NSPack		0.81	0.63	0.71	
Aegis		1.00	1.00	1.00	
ASPack		1.00	1.00	1.00	
Benign		0.37	0.55	0.44	

Table 7. Classification results for L1L2: precision, recall, and F1-score.

Feature Set	Classifier	Family	Precision	Recall	F1-Score
L1L2	RF	UPX	0.98	1.00	0.99
		PECompact	0.99	0.99	0.99
		NSPacker	0.99	1.00	0.99
		Aegis	1.00	1.00	1.00
		ASPack	1.00	0.98	0.99
		Benign	0.95	0.94	0.94
	J48	UPX	0.98	0.97	0.97
		PECompact	0.98	0.97	0.98
		NSPacker	0.99	0.99	0.99
		Aegis	1.00	1.00	1.00
		ASPack	0.98	0.98	0.98
		Benign	0.96	0.95	0.95
	SVM	UPX	0.93	0.98	0.95
		PECompact	0.68	0.93	0.78
		NSPack	0.95	0.79	0.86
		Aegis	0.97	1.00	0.98
		ASPack	1.00	0.99	0.99
		Benign	0.74	0.17	0.28
	NB	UPX	0.96	0.35	0.51
		PECompact	0.54	0.20	0.29
NSPack		0.77	0.63	0.70	
Aegis		1.00	1.00	1.00	
ASPack		1.00	0.99	1.00	
Benign		0.20	0.95	0.33	

In summation, this section offers an extensive examination of our experimental outcomes, illustrating the prowess of our packer identification approach and shedding light on the formidable performance of the random forest and J48 classifiers in our context.

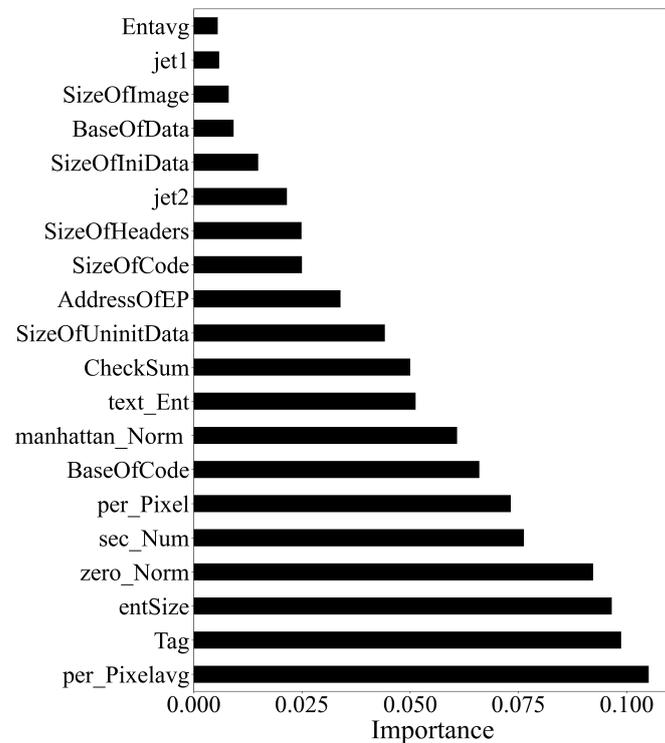


Figure 9. Feature importance: significance visualized.

5.4. Comparing with PEid

We embarked on a comprehensive comparison between our novel packer identification approach and the well-established tool PEid. PEid holds a prominent reputation as software designed to identify packers and compilers employed within executable files. This comparative analysis stands as a pivotal stride in appraising the efficacy and resilience of our approach. Figure 10 demonstrates the accuracy of packer identification results across PEid and our approach. We can notice the significant improvement achieved by our approach, which outperformed PEid.

5.5. Unknown Packers

Unknown packers are those new packers in the wild that were not trained in our machine learning model. To determine the robustness of our classifier, we have selected 1243 samples, 250 benign samples, 204 samples packed with MPress packer, and 789 samples for DarkComet RAT. To make our analysis interesting, we plunged into 216,612 malware samples to extract those under DarkComet RAT; we used the Yara signature detection to extract them. Those samples provided by VirusTotal were spotted between the years 2017 and 2019 as illustrated in Figure 11 and Table 8.

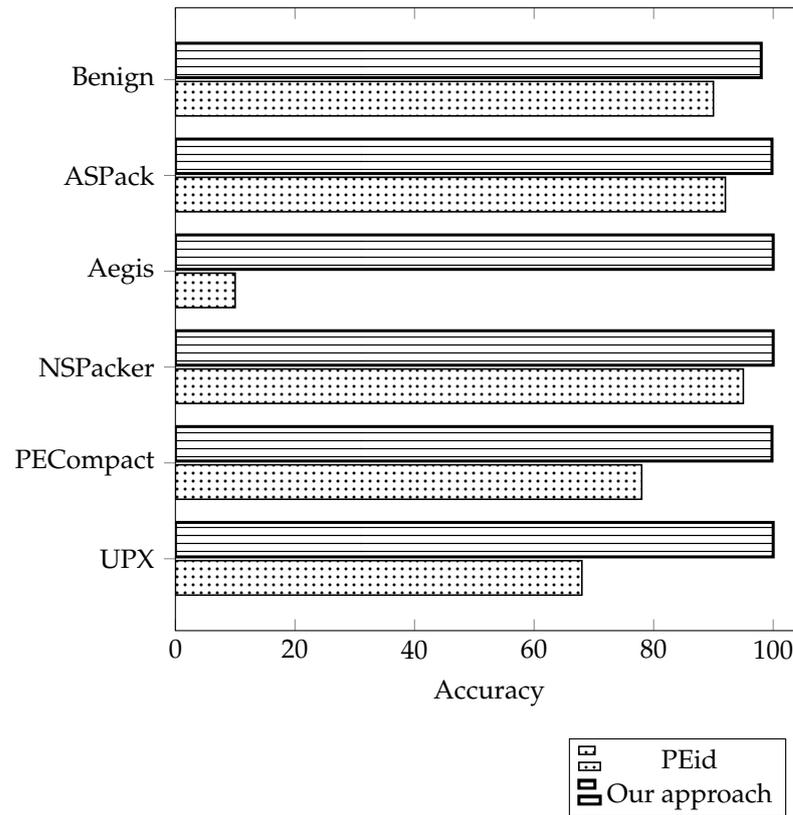


Figure 10. PEid Identification vs. Proposed Approach.

DarkComet is a remote access trojan (RAT) application that operates covertly, collecting system information, user data, and network activity. RAT applications often employ packing techniques to evade detection [47]. DarkComet attempts to steal stored credentials. In our innovative approach, we leverage classification techniques to accurately predict the file packer families used in malicious software. This process is crucial for identifying unknown packers, as beautifully illustrated in Figure 6. Our methodology involved executing a carefully crafted unpacking profile procedure, which aimed to reveal the hidden layers of these packed files. However, in cases where the unpacking procedure failed to expose the content, we categorized the sample as an unknown packer from an antivirus perspective.

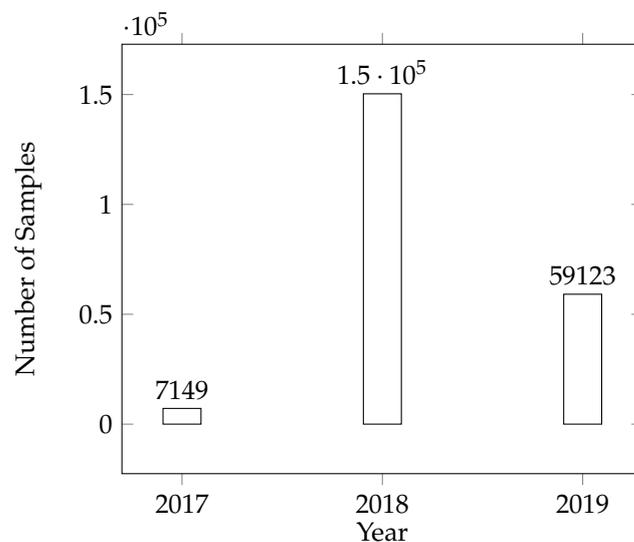


Figure 11. VirusTotal malware samples between the years 2017 and 2019.

Table 8. Unknown packers and benign files.

Packer Name	Type	Source	Samples
MPRESS	Protector	VirusTotal/malware	204
DarkComet	Hostile	VirusTotal/malware	789
N/A	Benign	Online/benign	250
Total			1243

The results for MPRESS, DarComet packers, and benign samples are depicted in Figure 12. The best average accuracy is the RF classifier, achieving 91%. MPRESS in most classifications best matched with Aegis, while DarkComet samples were identified with Aspack packer, and very few were packed with UPX and Nspack. Meanwhile, most benign files were detected as a benign class.

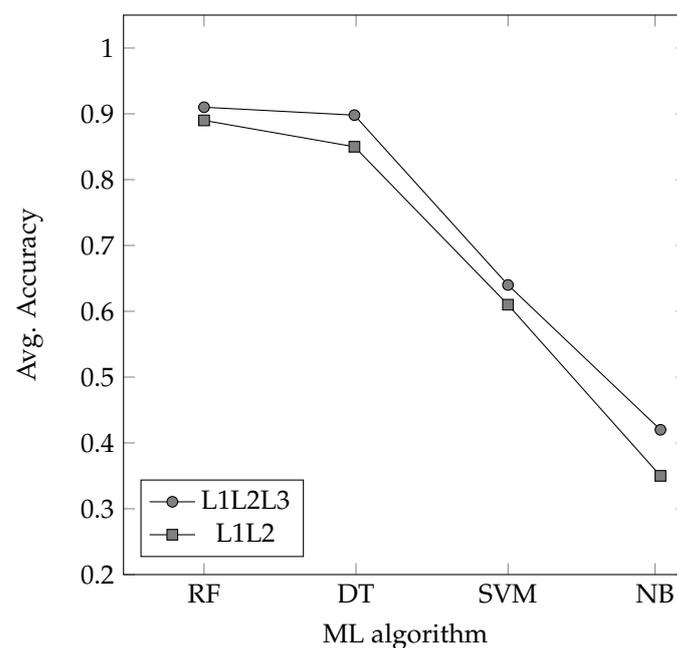


Figure 12. Unknown packers' identification: average accuracy highlighting performance metrics, with feature set L1L2L3 demonstrating the highest performance.

5.6. Comparing with Previous Works

Based on the results of the preceding experiments, it was observed that the suggested combined feature set outperformed the single feature set. To enhance the precision and accuracy of the proposed integrated feature set, it is crucial to compare it with previous works that rely on similar image plot, dataset, and accuracy information. As far as we know, the only previous study that utilized the image plot field for packer classification is [26]. In their work, Kancherla et al. [26] examined the structured analysis of several fields of the PE file and highlighted that including these characteristics improved the identification rate of packers. Meanwhile, Alkhateeb et al. [7] used VirusTotal samples and employed API PE features along with Levenshtein distance. We used data from the same source and compared the results.

Table 9 displays the accuracy scores of the top-performing classifiers for image plot and PE features, indicating that the integrated feature set performed better than the individual feature sets. The combined feature set outperformed the single-layer feature set by 11.1% in terms of the accuracy score, as observed in the results for the random forest and other classifiers. Furthermore, we compared our work with hybrid approaches, as in [48,49], as hybrid approaches combine both static and dynamic analysis for feature extraction;

our approach was superior in terms of the classification accuracy as well as in identifying unknown packers with proper dataset integrity.

Table 9. Comparing the proposed work with earlier works.

Works	Approach	Dataset	Accuracy	Unknown Packers	Dataset Integrity
Kancherla et al. [26]	Img	OffensiveCompu	81.34%	✗	✗
Alkhateeb et al. [7]	PE	VirusTotal	88.50%	✗	✗
Park et al. [48]	PE	VXHeavens	99.49%	✗	✗
Gao et al. [49]	PE	VirusShare	97.80%	✗	✗
Proposed work	PE/Img	VirusTotal	99.60%	✓	✓

6. Discussion

The experimental results underscore the system's high accuracy in discerning both known and unknown packers. This achievement is primarily credited to our meticulous feature engineering process, where the selection of prevalent features played a pivotal role. Notably, our empirical experiments revealed that the first layer of features offers crucial insights, particularly as many unknown packers exhibit behavior akin to known packers. Additionally, the results show that using the engineered PE features, such as tag, supersedes traditional raw features adopted by classical methods, making our approach superior.

The second layer of features is dedicated to extracting artifacts from images that represent an executable's raw bytes. Given the dynamic nature of packers and their impact on PE files during the packing process, employing an image plot representation allows us to visualize the distinctive shape of a packer. This visualization is invaluable, as packer images contain rich information that significantly enhances our detection capabilities. Figure 13 illustrates three samples each of the Aegis and NSpack packers, showcasing the striking similarities within each packer family. Our results indicate that the different features used contribute significantly to the identification capabilities.

The third layer introduces the entropy analysis of PE sections, further augmenting the assessment of features from the first layer and fortifying the detection process. A thoughtful selection and refinement of these features were instrumental in achieving accurate identification and preventing a decline in the identification rate.

These results unequivocally highlight the superior performance of the combined layers within our proposed approach, surpassing alternative methods. This underscores the robustness of our multilayer strategy in packer identification, a testament to its effectiveness in addressing various challenges. Importantly, this achievement is further underscored by the consistently high accuracy achieved in identifying unknown packers, reinforcing the efficacy of our innovative approach in enhancing detection capabilities and advancing the state of the art in the field.

The proposed approach is adaptable to various tasks, including threat detection like malware family classification, albeit requiring adjustments to meet specific requirements. The classifier offers flexibility, allowing continuous updates to incorporate new packer family feeds and optimizations. Moreover, real-world deployment of such approaches, as discussed in [50], encounters challenges such as statistical heterogeneity, systemic bias, and data poisoning attacks. In practice, our proposed approach has been actively applied, notably aiding our initial generic unpacking method, an integral component of our ongoing work.

In summary, our multilayered feature engineering approach, encompassing prevalent features, image analysis, and entropy analysis, demonstrated remarkable effectiveness in identifying both known and unknown packers. This comprehensive strategy resulted in the development of a robust and reliable detection system.

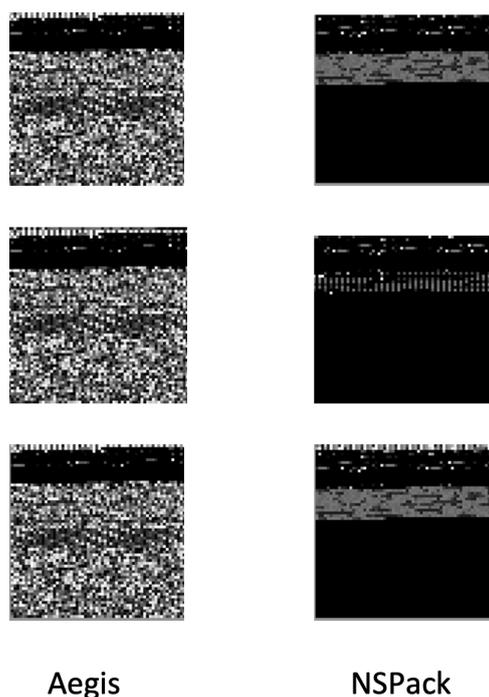


Figure 13. Aegis and NSPack image plot: a visual representation highlighting distinctive characteristics and patterns identified in the image plots generated for Aegis and NSPack.

7. Conclusions

In the face of the escalating threat of malicious attacks, combating malware has become a complex challenge. Packed malware, in particular, poses a significant hurdle for antivirus (AV) detection due to its dynamic and complex nature. Traditional security tools rely on scanning systems to identify suspicious behavior, taking action against files that raise concerns by quarantining or flagging them. However, malware creators continually adapt and devise new techniques to evade these conventional approaches and overcome existing security measures. Among these techniques, packing stands out as malware authors' most widely employed obfuscation method to elude system protection.

Detecting packed PE files is crucial, given the widespread use of the PE file format in Windows OS. This research introduced an innovative feature engineering approach that enhanced the performance of machine learning classifiers in identifying packed PE files. The proposed technique extracted features with minimal time and resource consumption compared to a dynamic analysis by employing static analysis. Our proposed method used the most prevalent features extracted during static analysis for packer identification.

To ensure a robust identification rate, we initially concentrated on balancing the dataset and performed a harmonious analysis and separation of the dataset in multiple stages. During the feature engineering process, we prioritized prevalent features and modified them to enhance identification accuracy for both family-based and unknown packers.

The study investigated the effectiveness of multilayer feature engineering in detecting packed malware. The results revealed that this approach, combined with machine learning algorithms, achieved an impressive accuracy rate of 99.60% for identifying packed malware families. Additionally, it achieved an accuracy of 91% for detecting unknown packed malware. These findings underscore the significance of utilizing this approach as it enables the triggering of appropriate response mechanisms to mitigate potential threats effectively.

8. Limitations and Future Work

This section delineates the constraints of our framework and proposes potential directions for future research, specifically focusing on two critical areas: unpacking techniques and advancing a multilayer packing identification approach. Ongoing efforts aim to

strengthen unpacking capabilities for precisely restoring the original malware code. While profile unpacking is implemented in the current approach, upcoming work will prioritize developing generic unpacking methods for extracting content from unknown packed files. To achieve this, we plan to implement advanced heuristic algorithms for dynamic analysis, adapt to evolving packing techniques, and integrate hardware-level analysis tools such as Intel PIN. This dual approach is vital to ensure a more accurate identification and unpacking of malicious code, optimizing the process for improved speed and efficiency. This is particularly crucial, considering the challenge posed by packers employing diverse techniques to conceal unpacking routines.

In terms of computational efficiency, it is noteworthy that transforming and analyzing binary data, such as large PE files, can be computationally intensive, potentially leading to increased processing time and resource demands. Optimizing algorithms and employing parallel processing techniques can enhance efficiency in these tasks.

Additionally, we plan to delve into the classification of packers used in repacking and multipacking scenarios. Our research will contribute to understanding and classifying packers in these contexts, considering the nuances of repacking (using one or two packer algorithms) and multipacking (employing three or more packer algorithms). This exploration is crucial, as not all packing algorithms successfully achieve the repacked or multipacked state. Addressing these challenges will further advance the effectiveness of our packing identification system.

Author Contributions: Conceptualization, E.A., A.G. and A.H.L.; data curation, E.A.; investigation, E.A.; methodology, E.A.; software, E.A.; supervision, A.G. and A.H.L.; writing—original draft, E.A.; writing—review and editing, A.G. and A.H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by funding from the National Science and Engineering Research Council of Canada (NSERC) and the Canadian Institute for Cybersecurity (CIC).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The authors thank VirusTotal for providing the malware dataset used in this research. The malware dataset can be requested at <https://www.virustotal.com> (accessed on 21 December 2019). Additionally, the authors thank Softpedia for providing various benign downloadable software applications, which can be found at <https://www.softpedia.com> (accessed on 9 December 2019).

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could appear to influence the work reported in this paper.

Abbreviations

The following abbreviations are used in this manuscript:

AV	Antivirus
CFG	Control flow graph
PE	Portable Executable
API	Application programming interface
VX	Virus eXchange
GUI	Graphical user interface
SVM	Support vector machine
FCG	Function call graph
ASM	Assembly
LD	Levenshtein distance
UPX	Ultimate Packer for eXecutables
ATR	Automatic target recognition
RAT	Remote access trojan

References

1. Jajodia, S.; Shakarian, P.; Subrahmanian, V.; Swarup, V.; Wang, C. *Cyber Warfare: Building the Scientific Foundation*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 56.
2. Herrmann, D. Cyber Espionage and Cyber Defence. In *Information Technology for Peace and Security: IT Applications and Infrastructures in Conflicts, Crises, War, and Peace*; Reuter, C., Ed.; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2019; pp. 83–106. [CrossRef]
3. Liță, C.V.; Cosovan, D.; Gavriliuț, D. Anti-emulation trends in modern packers: A survey on the evolution of anti-emulation techniques in UPA packers. *J. Comput. Virol. Hacking Tech.* **2018**, *14*, 107–126. [CrossRef]
4. McAfee. The Good, the Bad, and the Unknown. 2017. Available online: http://www.techdata.com/mcafee/files/MCAFEE_wp_appcontrol-good-bad-unknown.pdf (accessed on 12 January 2021).
5. Ugarte-Pedrero, X.; Balzarotti, D.; Santos, I.; Bringas, P.G. SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 659–673.
6. Hai, N.M.; Ogawa, M.; Tho, Q.T. Packer identification based on metadata signature. In Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop, Orlando, FL, USA, 5–6 December 2017; pp. 1–11.
7. Alkhateeb, E.M.; Stamp, M. United Arab Emirates A Dynamic Heuristic Method for Detecting Packed Malware Using Naive Bayes. In Proceedings of the 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 19–21 November 2019; pp. 1–6.
8. Menéndez, H.D.; Llorente, J.L. Mimicking anti-viruses with machine learning and entropy profiles. *Entropy* **2019**, *21*, 513. [CrossRef] [PubMed]
9. Bat-Erdene, M.; Park, H.; Li, H.; Lee, H.; Choi, M.S. Entropy analysis to classify unknown packing algorithms for malware detection. *Int. J. Inf. Secur.* **2017**, *16*, 227–248. [CrossRef]
10. Bat-Erdene, M.; Kim, T.; Park, H.; Lee, H. Packer detection for multi-layer executables using entropy analysis. *Entropy* **2017**, *19*, 125. [CrossRef]
11. Lim, C.; Ramli, K.; Kotualubun, Y.S.; Syailendra, Y. Mal-flux: Rendering hidden code of packed binary executable. *Digit. Investig.* **2019**, *28*, 83–95. [CrossRef]
12. Ugarte-Pedrero, X.; Santos, I.; Bringas, P.G.; Gastesi, M.; Esparza, J.M. Semi-supervised learning for packed executable detection. In Proceedings of the 2011 5th International Conference on Network and System Security, Milan, Italy, 6–8 September 2011; pp. 342–346.
13. Perdisci, R.; Lanzi, A.; Lee, W. Classification of packed executables for accurate computer virus detection. *Pattern Recognit. Lett.* **2008**, *29*, 1941–1946. [CrossRef]
14. Dini, P.; Elhanashi, A.; Begni, A.; Saponara, S.; Zheng, Q.; Gasmi, K. Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity. *Appl. Sci.* **2023**, *13*, 7507. [CrossRef]
15. Santos, I.; Ugarte-Pedrero, X.; Sanz, B.; Laorden, C.; Bringas, P.G. Collective classification for packed executable identification. In Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference, Perth, Australia, 1–2 September 2011; pp. 23–30.
16. Ugarte-Pedrero, X.; Santos, I.; García-Ferreira, I.; Huerta, S.; Sanz, B.; Bringas, P.G. On the adoption of anomaly detection for packed executable filtering. *Comput. Secur.* **2014**, *43*, 126–144. [CrossRef]
17. Naval, S.; Laxmi, V.; Gaur, M.S.; P, V. An efficient block-discriminant identification of packed malware. *Sadhana* **2015**, *40*, 1435–1456. [CrossRef]
18. Naval, S.; Laxmi, V.; Gaur, M.S.; Vinod, P. ESCAPE: Entropy score analysis of packed executable. In Proceedings of the Fifth International Conference on Security of Information and Networks, Jaipur, India, 25–27 October 2012; pp. 197–200.
19. Laxmi, V.; Gaur, M.S.; Faruki, P.; Naval, S. PEAL—Packed executable analysis. In Proceedings of the International Conference on Advanced Computing, Networking and Security, Surathkal, India, 16–18 December 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 237–243.
20. Mimura, M.; Ito, R. Applying NLP techniques to malware detection in a practical environment. *Int. J. Inf. Secur.* **2022**, *21*, 279–291. [CrossRef]
21. Jin, Q.; Duan, J.; Vasudevan, S.; Bailey, M. Packer classifier based on PE header information. In Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, Urbana IL, USA, 21–22 April 2015; pp. 1–2.
22. Choi, Y.S.; Kim, I.K.; Oh, J.T.; Ryou, J.C. Pe file header analysis-based packed pe file detection technique (phad). In Proceedings of the International Symposium on Computer Science and its Applications, Hobart, TAS, Australia, 13–15 October 2008; pp. 28–31.
23. Saleh, M.; Ratazzi, E.P.; Xu, S. A control flow graph-based signature for packer identification. In Proceedings of the MILCOM 2017—2017 IEEE Military Communications Conference (MILCOM), Baltimore, MD, USA, 23–25 October 2017; pp. 683–688.
24. Li, X.; Shan, Z.; Liu, F.; Chen, Y.; Hou, Y. A consistently-executing graph-based approach for malware packer identification. *IEEE Access* **2019**, *7*, 51620–51629. [CrossRef]
25. Liu, H.; Guo, C.; Cui, Y.; Shen, G.; Ping, Y. 2-SPIFF: A 2-stage packer identification method based on function call graph and file attributes. *Appl. Intell.* **2021**, *51*, 9038–9053. [CrossRef]

26. Kancherla, K.; Donahue, J.; Mukkamala, S. Packer identification using Byte plot and Markov plot. *J. Comput. Virol. Hacking Tech.* **2016**, *12*, 101–111. [[CrossRef](#)]
27. Jung, B.; Bae, S.I.; Choi, C.; Im, E.G. Packer identification method based on byte sequences. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5082. [[CrossRef](#)]
28. Dam, K.H.T.; Given-Wilson, T.; Legay, A.; Veroneze, R. Packer classification based on association rule mining. *Appl. Soft Comput.* **2022**, *127*, 109373. [[CrossRef](#)]
29. Biondi, F.; Enescu, M.A.; Given-Wilson, T.; Legay, A.; Noureddine, L.; Verma, V. Effective, efficient, and robust packing detection and classification. *Comput. Secur.* **2019**, *85*, 436–451. [[CrossRef](#)]
30. Bergenholtz, E.; Casalicchio, E.; Ilie, D.; Moss, A. Detection of metamorphic malware packers using multilayered LSTM networks. In *Proceedings of the International Conference on Information and Communications Security*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 36–53.
31. Damaševičius, R.; Venčkauskas, A.; Toldinas, J.; Grigaliūnas, Š. Ensemble-based classification using neural networks and machine learning models for windows pe malware detection. *Electronics* **2021**, *10*, 485. [[CrossRef](#)]
32. Noureddine, L.; Heuser, A.; Puodzius, C.; Zendra, O. SE-PAC: A Self-Evolving Packer Classifier against rapid packers evolution. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy, Virtual Event, USA, 26–28 April 2021*; pp. 281–292.
33. Cheng, B.; Leal, E.A.; Zhang, H.; Ming, J. On the feasibility of malware unpacking via hardware-assisted loop profiling. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, 9–11 August 2023; pp. 7481–7498.
34. D’Alessio, S.; Mariani, S. PinDemonium: A DBI-based generic unpacker for Windows executables. In *Proceedings of the Black Hat 2016*, Las Vegas, NV, USA, July 2016. Available online: <https://www.politesi.polimi.it/handle/10589/120861> (accessed on 12 June 2021).
35. Carrera, E. PEFile. 2023. Available online: <https://github.com/erocarrera/pefile> (accessed on 12 June 2021).
36. Rezaei, T.; Hamze, A. An efficient approach for malware detection using PE header specifications. In *Proceedings of the 2020 6th International Conference on Web Research (ICWR)*, Tehran, Iran, 22–23 April 2020; pp. 234–239.
37. Levenshtein, V.I. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **1966**, *10*, 707–710.
38. Ristad, E.S.; Yianilos, P.N. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.* **1998**, *20*, 522–532. [[CrossRef](#)]
39. Wellman, M.; Nasrabadi, N. Gabor Jets for Clutter Rejection in Infrared Imagery. Defense Technical Information Center. 2004. Available online: <https://apps.dtic.mil/sti/pdfs/ADA487612.pdf> (accessed on 12 June 2021).
40. Wiskott, L.; Krüger, N.; Kuiger, N.; Von Der Malsburg, C. Face recognition by elastic bunch graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.* **1997**, *19*, 775–779. [[CrossRef](#)]
41. Günther, M.; Haufe, D.; Würtz, R.P. Face recognition with disparity corrected Gabor phase differences. In *Proceedings of the International Conference on Artificial Neural Networks*, Lausanne, Switzerland, 11–14 September 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 411–418.
42. Biryukov, A.; Nakahara, J., Jr.; Yıldırım, H.M. Differential entropy analysis of the IDEA block cipher. *J. Comput. Appl. Math.* **2014**, *259*, 561–570. [[CrossRef](#)]
43. Donabelle, B.; Richard, M.L.; Mark, S. Structural entropy and metamorphic malware. *J. Comput. Virol. Hacking Tech.* **2013**, *9*, 79–192.
44. Cozzi, E.; Graziano, M.; Fratantonio, Y.; Balzarotti, D. Understanding linux malware. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 20–24 May 2018; pp. 161–175.
45. Shannon, C.E. A mathematical theory of communication. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **2001**, *5*, 3–55. [[CrossRef](#)]
46. Alkhateeb, E.; Ghorbani, A.; Habibi Lashkari, A. A survey on run-time packers and mitigation techniques. *Int. J. Inf. Secur.* **2023**, *1*–27. [[CrossRef](#)]
47. Kazoleas, I.; Karampelas, P. A novel malicious remote administration tool using stealth and self-defense techniques. *Int. J. Inf. Secur.* **2022**, *21*, 357–378. [[CrossRef](#)]
48. Park, L.H.; Yu, J.; Kang, H.K.; Lee, T.; Kwon, T. Birds of a Feature: Intrafamily Clustering for Version Identification of Packed Malware. *IEEE Syst. J.* **2020**, *14*, 4545–4556. [[CrossRef](#)]
49. Gao, X.; Hu, C.; Shan, C.; Han, W. MaliCage: A packed malware family classification framework based on DNN and GAN. *J. Inf. Secur. Appl.* **2022**, *68*, 103267. [[CrossRef](#)]
50. Thantharate, P.; Anurag, T. CYBRIA-Pioneering Federated Learning for Privacy-Aware Cybersecurity with Brilliance. In *Proceedings of the 2023 IEEE 20th International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, Boca Raton, FL, USA, 4–6 December 2023; pp. 56–61.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.