HAL
open science

Effective, Efficient, and Robust Packing Detection and Classification

Fabrizio Biondi, Michael A Enescu, Thomas Given-Wilson, Axel Legay, Lamine Noureddine, Vivek Verma

To cite this version:

Fabrizio Biondi, Michael A Enescu, Thomas Given-Wilson, Axel Legay, Lamine Noureddine, et al.. Effective, Efficient, and Robust Packing Detection and Classification. Computers & Security, In press, pp.1-15. hal-01967597

HAL Id: hal-01967597

https://inria.hal.science/hal-01967597v1

Submitted on 31 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Effective, Efficient, and Robust Packing Detection and Classification

Fabrizio Biondi*, Michael A. Enescu†, Thomas Given-Wilson†,
Axel Legay‡§, Lamine Noureddine†, and Vivek Verma†
*CéntraleSupelec, France
†Inria, France
‡Aalborg University, Denmark
§UC Louvain, Belgium

***Abstract*—Packing is a widespread tool to prevent static malware detection and analysis. Detecting and classifying the packer used by a given malware sample is fundamental to being able to unpack and study the malware, whether manually or automatically. Existing literature on packing detection and classification has focused on effectiveness, but does not consider the efficiency required to be part of a practical malware-analysis workflow. This paper studies how to train packing detection and classification algorithms based on machine learning to be both highly effective and efficient. Initially, we create ground truths by labeling more than 280,000 samples with three different techniques. Then we perform feature selection considering the contribution and computation cost of features. Then we iterate over more than 1,500 combinations of features, scenarios, and algorithms to determine which algorithms are the most effective and efficient, finding that a reduction of 1-2% effectiveness can increase efficiency by 17-44 times. Then, we test how the best algorithms perform against malware collected after the training data to assess them against new packing techniques and versions, finding a large impact of the ground truth used on algorithm robustness. Finally, we perform an economic analysis and find simple algorithms with small feature sets to be more economical than complex algorithms with large feature sets based on uptime/training time ratio.**

Keywords: Packer detection, packer classification, entropy, machine learning, feature selection, Portable Executable file, obfuscation, malware.

I. Introduction

Context. Malware is a large and growing problem for cybersecurity. Ransomware damage alone in 2017 has been estimated costing more than 5 billion dollars, up 15x from 2015 [21]. According to Cisco, malware volume increased by 12 times from 2015 to 2017, and the Nyetya (or NotPetya) attack alone wiped 1 million computers [14].

Increasingly large numbers of malware are being created and released online: the Cisco Talos team reports capturing more than 1 million new unique malware samples per day [14]. With such prolific malware growth, effective malware detection and analysis has become fundamental for user and system security.

Malware analysis techniques can be divided in *static* and *dynamic*. Static analysis examines the malware code directly, via disassembly or measuring syntactic properties. Dynamic analysis requires execution of the malware samples in a sandbox virtual environment. Due to the computational cost of starting a sandbox being in the order of seconds or more before the malware analysis can even begin, dynamic analysis is much more expensive than static analysis.

One highly effective technique to hinder static analysis is to *pack* malware. Packing encompasses a variety of techniques that compress and encrypt the content of the malware, preventing static analysis. The malware's code is unpacked and executed only at runtime.

Challenges and Difficulties. Packing makes malware analysis significantly harder since the sample must be either unpacked or analyzed dynamically to detect malicious behavior. A first challenge (*packing detection*) then is to detect whether or not a potential malware sample has been packed. If a sample has been determined to be packed, a second challenge (*packing classification*) then is to detect which packer was used to pack the sample. If either of these challenges fails or provides incorrect results, then static analysis will fail. Further, dynamic analysis will be required (or applied erroneously). Due to these concerns packing detection and packing classification are significant to building effective malware analysis workflows. However, both *packing detection and packing classification must be computationally significantly cheaper than dynamic analysis*, since in case of multiple layers of packing the detection-classification-unpacking loop may have to be executed multiple times to prepare the sample for static analysis.

Creating malware detection techniques that perform well on real malware (not just in laboratory tests) is known to be a hard problem [7]. One main reason for this is that the malware ecosystem evolves, changing the frequency and composition of malware, including the packing used. This forces malware detection and analysis tools to be constantly updated to keep pace with the evolving malware ecosystem. Such constant updating is often expensive, hence *understanding which techniques to use to optimize effectiveness while minimizing cost* is vital to robust solutions.

State of the Art. Both packing detection and packing classification have been studied in literature, and approaches to solving them are available in antivirus like ClamAV [1] and tools like PEiD [3]. Most techniques are based on *syntactic signatures*, on *entropy metrics*, or on *machine learning* (ML).

Syntactic signatures are sequences of bytes characterizing a

Corresponding author: Fabrizio Biondi: biondif@gmail.com

```
[UPX 0.50 - 0.70]
signature = 60 E8 00 00 00 00 58 83 E8 3D
ep_only = true

[UPX 0.72]
signature = 60 E8 00 00 00 00 83 CD FF 31 DB 5E
ep_only = true
```

Fig. 1. A syntactic signature used by PEiD to identify files packed by UPX from version 0.50 to version 0.72. The `signature` field represents the sequence of bytes characterizing the packer and version, while the `ep_only` flag determines whether the sequence is found only at the entry point of the binary or anywhere in it.

specific (version of a specific) packer. An example is shown in Figure 1 and used by PEiD to recognize versions from 0.5 to 0.70 and 0.72 of UPX. Such static syntactic techniques are unable to detect packers and versions for which they do not possess a signature. The problem is compounded by the signatures needing to be updated manually when analysts confirm a new version or packer.

The first published academic work to consider the problem of packing detection was [19]. Their approach was to use entropy as a proxy for packing, since packed files tend to have much higher entropy. Their approach was calculated to be able to achieve a false positive rate of 0.038 and false negative of 0.005, however this was not verified experimentally. Multiple successive works study using entropy metrics to detect a sample as packed [13], [16], [25], [30], [31] or to classify which packer was used for a sample [15]. This paper uses entropy as one of the features in a feature vector, leaving to the ML algorithm to decide how relevant it is to classify each specific packer family.

ML techniques use supervised learning on features extracted from training sets of samples to build packer detectors and packer classifiers. For both detectors and classifiers, the ML algorithms learn the features that identify packing from a given ground truth, making them able to detect or classify based on the features found in new samples.

The challenge of packer detection was also considered in [23] that tested a variety of machine learning approaches on a variety of features. Their best results were using Multi Layer Perceptron that achieved 10-fold cross-validation accuracy of 0.9995. Machine learning on multiple features was also used by [32]. In [29] the problem of packer detection was generalized to packer classification. Again the approach was to test multiple algorithms, but used different features to those above. Other works have since also considered classification [10], [17]. In [9] the classification is to determine not only the packer, but also if multiple packers have been used on the same program. Their best results where accuracy of 0.985 for re-packer, and 0.975 for multi-layer packer. The sample set used was 6 benign programs each packed with every combination of 2 out of 19 different packers. However, the results cited above only focus on increasing the efficacy of the detection and classification, and do not consider the computational costs of optimizing only for efficacy when running a system that scales to millions of samples per day.

The best detection results achieved are accuracy of 0.9996 using Fuzzy Unordered Rule Induction Algorithm with Information Gain on a sample set 63,000 samples of which 21,000 are packed [32]. The best classification results achieved are: true positive 0.994, false positive 0.002, precision 0.995, and recall 0.994 on a set of 17,919 packed samples. This paper instead achieves an F-score up to 0.9999 on more than 280,000 samples, but more importantly studies how these results depend on the construction of the ground truth, choice of metrics, and type of validation, and how these results hold in scenarios more realistic than cross-validation.

Computational cost has largely not been considered as an aspect of packing detection and classification. In [6] they consider cost of feature extraction for use in malware classification, with packing detection playing a minor rôle and the independent cost for packing detection not being considered. In [29] the timing of building the model for classification was considered, but not the time to classify a sample. In this paper we show that feature extraction may dominate the classification time if features are not also selected taking into account their extraction time. Hence feature selection impacts efficiency in ways that were not considered in [29].

To the best of our knowledge, the economy of regularly re-training ML algorithms for packing detection and classification has not been considered in the literature.

We note that both techniques based on syntactic signatures and techniques based on ML rely on statically-extracted syntactic properties of the malware samples analyzed, such as entropy. It is commonly accepted that such properties are severely limited for detecting and classifying malware, and behavioral properties have been shown to be more appropriate for the malware detection and classification problems [22]. However, the limitations of syntactic properties for malware detection stem precisely from the existence of packing techniques: the need for behavioral properties comes from the fact that syntactic properties would detect and classify the packer instead of the malware. Since here detecting and classifying the packer is exactly our goal, syntactic properties are fitting for such goals, as shown by the literature on packing detection and classification. Additionally, since packing detection and classification typically play a rôle in a larger malware analysis workflow, we here consider the cost of packing detection and classification to be critical to the evaluation. Since behavioral properties are vastly more expensive to extract than syntactic properties, this work relies only upon syntactic properties. Moreover, we are not even concerned on whether the analyzed samples contain malicious code or not: packed goodware is detected and classified in the same way as packed malware, and we have no interest distinguishing between the two.

Our Solution This paper focuses on proposing solutions for packing detection and packing classification. To be a practical part of a malware analysis workflow, a solution must exhibit the following properties.

Effective The solution has a high true positive and a low false positive rate. This is measured by testing the technique

against packed and unpacked samples and measuring the solution's F-score.

Efficient The solution has a low computational cost, ideally no more than hundredths of a second for packing detection and classification on a single sample. Since classification includes feature extraction, features have to be chosen to both maximize classification effectiveness and minimize extraction cost.

Robust The solution maintains its effectiveness when used on samples different from the ones it has been built with. This is measured by temporal assessment, i.e. testing the solution on samples captured after the solution has been built, representing the evolving ecosystem of packers used by malware in the wild.

In this paper we want to produce solutions to packing detection and classification that are effective, efficient, and robust. Our approach is to study ML classifiers and construct one that is fast enough to be integrated in a malware analysis workflow, while preserving its effectiveness and robustness. To obtain this, we consider a large number of features and study how using such features impacts not only the effectiveness of the technique, but also its computational cost. Additionally, we perform a temporal assessment demonstrate how algorithms trained on samples from a given time period perform against samples from later periods. Finally, we present an economical analysis to understand which algorithms to use to maximize the uptime of the algorithms compared to their training time.

Contributions The contributions of the paper can be summarized as follows:

- We study ways to produce ground truth of different quality for training ML classification algorithms. We apply 3 different methods of packing detection and classification on a database of 281,344 samples, then produce two different ground truth: a 3CONS ground truth with higher-quality labeling but less samples following Sun et al. [29], and a 1CONS ground truth with more samples but lower-quality labeling. We pose a research question on the relative effectiveness of algorithms trained on the two ground truths. While algorithms trained on 3CONS seem to perform $\sim 1-2\%$ better than algorithms trained on 1CONS, we show that this is an artifact of cross-validation due to the smaller amount of data in 3CONS.
- We extract a large number of features for machine learning classification, and perform a careful feature selection based on both the features' contribution to the algorithms' effectiveness, and the cost of extracting the feature from a sample. We find that some features require $\sim 10^{-1}s$ to extract while others require $\sim 10^{-5}s$ or even $\sim 10^{-7}s$ to extract, hence feature selection strongly impacts training and classification costs.
- We perform a large scale optimization of the parameters of the machine learning classifiers, each depending on the choice of: ground truth, features, and scenario. We pose a research question on how much effectiveness is decreased by also optimizing for cost, finding that decreasing the effectiveness by 1-2% can reduce the classification cost per sample by 17 to 44 times.
- We perform temporal assessment and address a research question by testing the best algorithms against malware samples gathered after the ground truth data to understand how effectiveness decreases with the evolution of the ecosystem. We find that algorithms trained on the 3CONS lose $\sim 6-30\%$ of their effectiveness against the $\sim 1-3\%$ of the algorithms trained on 1CONS.
- Based on the experiments above, we perform an economical analysis evaluating which algorithms have the best ratio of uptime to training cost. We pose a research question on whether simple algorithms with less features or complex algorithms with more features are more convenient to use, finding that the latter cost up to 60 times more to train for an uptime only 3-4 times higher than the former.

The rest of the paper is structured as follows. Section II recalls background material useful for understanding the paper. Section III presents the research methodology and questions posed. Section IV presents the feature used for the ML classification and comments on their extraction costs. Section V details the ground truth generation methods used. Section VI presents the experiments performed to answer the research questions. Section VII presents and discusses the experimental results. Section VIII discusses the longer term economical analysis of the best performing algorithms. Section X concludes.

II. Background

This section recalls background on malware analysis workflows, packing and unpacking, and the PE file format.

A. Analysis of Packed Malware

Malware analysis has to account for packing techniques used to obfuscate malware and hinder reverse engineering.

Packing makes malware analysis significantly harder since the binary must be either unpacked or analyzed dynamically to detect malicious behavior. The packing detection and packing classification workflow fragment of a typical malware analysis workflow is shown in Fig. 2. Observe that a failure of packing detection leads to static analysis of packed samples (that tends to fail [22]), or attempted unpacking of already unpacked samples likely to conclude in expensive dynamic analysis. Similarly, classification failures typically rely on generalized unpacking [11], [20], [26] that is more expensive and more likely to end in expensive dynamic analysis. Specialized unpackers may exist for common packers and are cheaper and less error-prone than generalized unpackers, hence knowing which exact packer has been used to pack a binary may allow the application of a specialized unpacker, improving the cost and success rate of the unpacking process. Thus an efficient and effective malware analysis workflow requires efficient and effective packing detection and packing classification solutions.

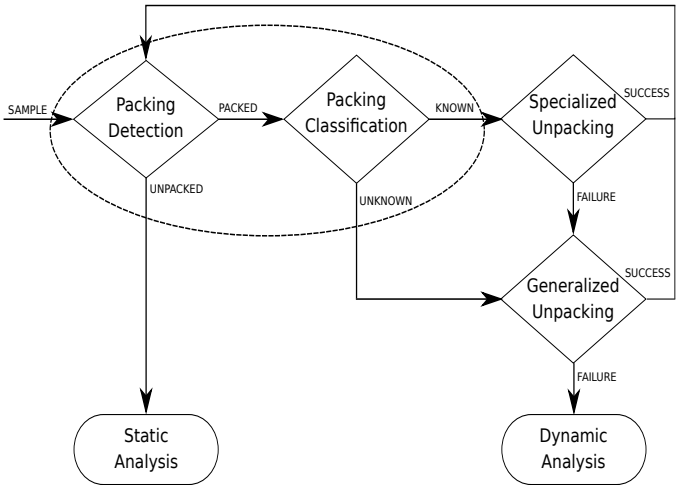


Fig. 2. A flow chart of the packing detection and packing classification workflow fragment of a typical malware analysis workflow. If the malware is detected as unpacked, it is analyzed statically. Otherwise, the packing technique is classified. In case of known classification, the payload is retrieved with a specialized unpacker and delivered back to packing detection (to account for multi-layered packing). If the classification or the specialized unpacking fail, generalized unpacking is attempted, with the payload delivered back to packing detection in case of successful extraction, and the whole sample analyzed dynamically otherwise. The dotted circle highlights the parts explored by this paper: packing detection and packing classification.

B. The PE File Format

This section summarizes the Windows Portable Executable (PE) binary format that is the focus of this paper.

The PE format includes a PE header with the magic number (in ASCII) `PE00`, a file header, and an optional header. The file header includes metadata such as the number of sections in the file; the creation time; a pointer to the debug information (if any); the size of the optional header; and other characteristics. The optional header includes the major and minor OS version; the size of the code section, of all initialized and uninitialized data, of the whole image, and of the headers; the address of the image, code, and data sections, and entry point; the initial (commit) and maximal (reserve) stack sizes; a file checksum; and an array with pointers to other important parts of the file.

After the PE header the sections of the binary start. Standard sections (and their common name) include code (`.text`), initialized data (`.data`, `.crt`), uninitialized variables (`.bss`), resource (`.rsrc`), import table and address table (`.idata`), export table (`.edata`), relocations table (`.reloc`), thread-safe variables (`.tls`), and debug resources (`.rdata`). Each section includes information on its address, virtual and raw data sizes, and whether it is readable, writable, and/or executable. A summary of a PE file structure is shown on the left column of Figure 3.

C. Packing and Unpacking

An overview of the life cycle of program being packed and loaded into memory is shown in Fig 3. The packing software takes the binary file as input and compresses (and/or encrypts) the file to create a new binary file. The new binary file is also equipped with an unpacking stub to decompress (and/or decrypt) and execute the packed content at runtime.

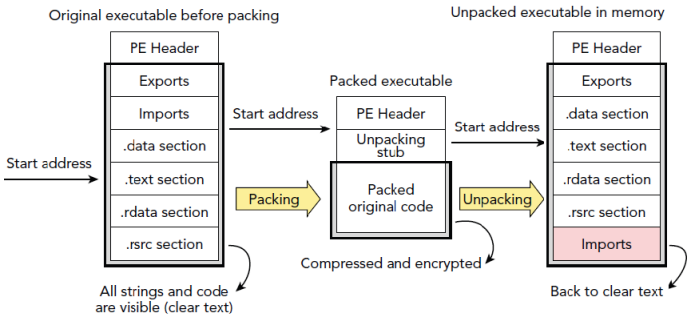


Fig. 3. Life cycle of a program being packed and loaded into memory [18]. The packer takes all the content of the PE file in the highlighted rectangle (not the header) and compressed and encrypts this into a new payload. An unpacking stub is added that is placed at the updated start address of the PE file's execution. The PE header is updated accordingly. The unpacking is performed when the program is executed, by the unpacking stub decrypting and decompressing the original content and recreating the parts of the file shown in the black rectangle. Finally the unpacking stub returns execution to the reinstated start address and the program executes as normal.

Packers are typically used for two main reasons: to reduce binary size, and to thwart detection or analysis [18], [28]. If the original binary contains malware, the packing process will make such malware harder to recognize and analyze. Indeed, the malicious code is changed after packing due to the compression and/or encryption. Therefore, the packed file will thwart a classical signature-based AV scanner, as no signature match with the unpacked malware will be found. This is why the analysis and detection of a packed malware can only take effect after the executable file is completely unpacked.

III. Methodology

This sections overviews the research questions (**RQ1**-**RQ4**) considered here and the experimental methodology used to address them.

A. Supervised ML Classification Algorithms

The approach used here to perform packing detection and classification is built by evaluating and choosing between multiple supervised ML algorithms as is common [23], [29], [32]. The rest of this section recalls key information on how supervised ML operates, and how our implementation of these algorithms is done.

Supervised ML classification refers to classification problems where classification algorithms are trained on a *ground truth*, i.e. a set of fully-labeled data. Each *element* in the ground truth is composed of a *label* describing the class of the element (packed or unpacked for detection, and packer name for classification) and a vector of *features* describing the element's relevant characteristics. Each algorithm is *trained* on the ground truth to understand how the labels are connected to the respective features. Each algorithm is *tested* on a new feature vector and having the algorithm guess the label for the element corresponding to the new feature vector.

In our case each element is a sample binary and its features describe properties that can be efficiently extracted from the binary. For the feature extraction we rely on the PeLib C++ PE file manipulation library published by Avast [4] and based on the original PeLib library created by Sebastian Porst [24]. The PeLib library has been developed by Avast specifically to handle malware, including corrupted or exotic PE files produced by malware obfuscation and packing techniques. Our implementation of feature extraction in C++ allows has very low feature extraction times. The features we use are described in Section IV together with their average extraction times.

The generation of a ground truth is a problem in itself, since large labeled databases of malware or packed samples are largely unavailable. Sun et al. [29] annotate a malware database by using three sources of packer labels (Kaspersky, Microsoft, and the Computer Associates (CA) Threat Management Team) and using the PEiD tool to settle disagreements between the sources. They then keep only the samples for which all tools agree, creating a ground truth with only the packer families for which there are at least 100 labeled samples. We follow a similar principle of annotating malware data with three different sources, but construct two different ground truths based on whether the sources agree or not, and study how these two different ground truths impact the effectiveness and robustness of the classification algorithms. Details on ground truth generation are given in Section V.

The ML classification algorithms here are: Gaussian Naive Bayes, Decision Tree, Random Forest, and Extra Trees implementations from the numpy Python library [2] version 1.14.2.

The *effectiveness* of a trained algorithm is measured by its *F-score*. For binary classification, i.e. packed/unpacked, it is computed as usual [12]. For multiclass classification, the F-score is computed by considering all samples as equivalently important (independently from their class), and is defined as follows[1]. Consider a data set A of "real" $(sample, label)$ pairs. Classify the samples of A with the algorithm, obtaining the set B of "predicted" $(sample, labels)$ pairs. Then the F-score is just the ratio of correctly labeled samples, i.e. $F(A, B) = \frac{|A \cap B|}{|B|}$. Note that with this averaging method F-score, precision, and recall coincide. The choice of this averaging method depends on the assumption that the distribution of the samples in the classes in the ground truth is representative of the real case. This is discussed in more detail in Section V. Also note that a different choice of F-score averaging function for multiclass classification can be used to model, for instance, that the analyst is more interested in correctly classifying rare packing techniques than common ones, or vice versa.

Ideally, the samples in the set A should be separate from the samples used to train the algorithm. We do this in the temporal assessment, described below. However, to reduce the data usage, 5-round 80-20 cross validation is used in this paper for the feature selection and parameter optimization. This is, we randomly select 80% of the ground truth for training and 20% for F-score computation, we repeat this operation 5 times, and average the five F-scores to obtain the reported F-score.

The *efficiency* of a trained classifier algorithm is measured as the inverse of the total time it takes to label a sample. This time includes the time required to extract the features from a sample (*extraction time*) and the time for the algorithm to predict a label for the features extracted (*classification time*).

B. Experimental Feature Selection

Using more features does not necessarily increase the effectiveness of the algorithms, as some features can be misleading or insufficiently related to the class labels. Hence it is common in machine learning to experimentally determine which features to keep and which to discard, in a process known as *feature selection*. This is important here since we want to optimize for speed and more features require higher extraction costs and longer ML algorithm training and classification times. Since most of the algorithms haveparameters that can modify their effectiveness, we need to find the optimal parameters for each feature combination to guarantee a fair comparison.

For each of the scenarios of interest (packing detection, packing classification, or both at the same time) and each of the two ground truths, we test each algorithm on each combination of feature categories and parameters. Then we keep the top 3 algorithms by F-score, and the top 3 algorithms by ratio between F-score and total classification cost per sample (i.e. feature extraction cost plus algorithm classification cost). This provides insight as to how much F-score is lost when selecting algorithms and features that are optimized for both F-score and cost instead of only for F-score.

RQ1 How much F-score is lost and how much cost is reduced by optimizing for both F-score and cost instead of just F-score?

C. Temporal Assessment

As the malware ecosystem evolves, new packing techniques and new version of known packer techniques appear, impairing the effectiveness of packing detection and classification algorithms. We test the robustness of our algorithms by training them on data collected from February to June 2017 and testing them on data collected from July to October 2017, to understand how their effectiveness degrades with time. This provides insight on how often the algorithms have to be retrained.

RQ2 How do algorithms trained on the 3CONS ground truth perform compared to algorithms trained on the 1CONS ground truth?

RQ3 How robust are the best algorithms found by feature selection and parameter optimization against data collected after their training data?

D. Economical Analysis

Based upon the robustness results we evaluate the situation of a malware analyst choosing which algorithm and features to use for packing detection and classification as part of their malware analysis workflow. For illustration we fix required F-score of at least 0.96 as required, implying that when the

[1]This averaging method is called `F1-micro` in the `scikit` Python package; see Section 3.3.2.8.2 of http://scikit-learn.org/stable/modules/model_evaluation.html

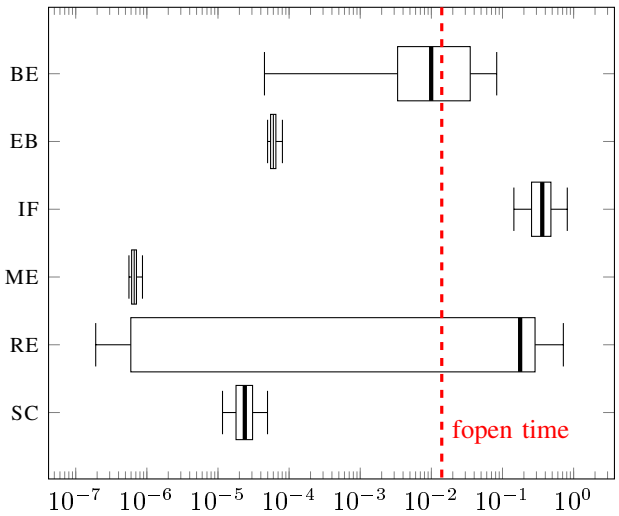


Fig. 4. Feature extraction costs for the feature categories. The dotted red *fopen time* line represents the empirically-calculated average time for opening the binary before extracting any feature, i.e. 14ms. Extraction cost of the EB, ME, and SC categories is negligible. Extraction cost of the BE and RE categories varies widely according to the size of the file, which sections are present, and whether it is stripped of its debug information and resources. Extraction cost of the IF category is consistently high.

F-score drops below this point the ML algorithm must be retrained. Considering the uptime and cost required to train the given ML algorithm, we determine how many seconds of uptime per second of training each algorithm can yield.

RQ4 Which algorithms yield the best ratio between uptime and training time?

IV. Feature Description

This section overviews the features used in the experiments here, including citing other works that have used them where relevant. A wide variety of features have been used in prior work on packer detection and classification, most of which detect possible artifacts of the packing process such as increased entropy and removal of metadata. Here the focus is on static features that can be extracted without dynamic execution, as the time-cost of extraction is a major consideration of the experiments. As remarked in Section I, the dynamic analysis required to extract dynamic features suffers from the high cost of starting a sandbox for each sample, which static analysis and this paper avoid.

Due to the lack of extensive feature extraction cost analysis in prior work, the approach here is to extract a large number of features (119) and use feature selection to find the ones with high contribution to detection and classification while having low extraction cost. When a feature value cannot be extracted (e.g. entropy of a missing section), a constant outside the value range is used instead (e.g. -1 for entropy). For clarity these 119 features are here divided into 6 feature categories. Fig. 4 summarizes the extraction costs for each feature category.

A. Byte Entropy – BE

The *byte entropy* (BE) feature category includes features that measure the Shannon entropy of different parts of the binary file. There are 6 features in this category corresponding to the Shannon entropy of the: `.text` section; `.data` section; `.rsrc` section; PE header; section containing the entry point; and entire file.

Shannon entropy has been used as a proxy for packed or encrypted data in packer detection and classification [19], [31], [32]. To compute byte entropy of n bytes, the frequency $F(v)$ of each of the 256 byte values is computed and represented as a probability distribution where $P(v) = F(v)/n$ is the probability that a byte has value v. Then entropy is computed as $-\sum_{v=0}^{255} P(v) \log_2 P(v)$.

The intuition here is that since Shannon entropy corresponds with compression and encryption, higher Shannon entropy will correspond to packing. The various byte entropy features attempt to exploit which sections are most significant. Computing Shannon entropy of the whole file and its section requires at least a full file scan to compute the byte frequencies. The extraction costs varies widely according to file size and which sections are present, as shown in Fig. 4, and is usually comparable to the time required to open the file.

B. Entry Bytes – EB

The *entry bytes* (EB) features are the first 64 bytes of the binary from the entry point. Each of these 64 features is the value of the byte converted to a natural number.

The intuition is that the bytes at the entry point often characterize the unpacking stub. Syntactic signatures are commonly based on the bytes after the entry point, as shown in Fig. 1. The bytes can be read directly from the file with a direct memory access. The average extraction cost for this feature category is consistently negligible compared to the time required to open the file, as shown in Fig. 4.

C. Import Functions – IF

The *import* (IF) feature category includes 5 features related to the import table and imported DLLs and functions. Related works have identified significant features here [6], [31], [32]. Zakeri et al. [32] give a list of 16 API functions that are more common in malware than in cleanware. The features in this category are: number of DLLs imported; number of imported functions in the import table directory; number of addresses found in the import address table; number of imported functions that appear in the Zakeri et al. list; ratio between the number of imported functions that appear in the Zakeri et al. list and the total number of functions imported.

The intuition here is that packers often remove the import table to hide the packed program's functionalities, and keep only the functions necessary to reconstruct it. Extracting these features requires scanning the import table and import address table when present. The average extraction cost for this feature category is in the tenth of seconds, as shown in Fig. 4.

D. Metadata – ME

Metadata (ME) features are those extracted from the PE header that provide information about the program. The included 21 metadata features here are: the 8 PE characteristics; PE checksum; base address (converted to decimal) of the image and of the `.text` section; OS major/minor version; byte size of the image, `.text` section, headers, (un)initialized data, stack reserve, and stack commit; and section alignment (i.e. size of a memory page). All of these have been used in prior works [6], [31], [32].

The intuition here is that this metadata provides information that may characterize a specific packer, or the information a packer provides to attempt to hide itself. The information for these features is available in the PE file header at specific addresses, so the features can be extracted very efficiently with direct file access. The average extraction cost for this feature category is consistently negligible compared to the time required to open the file, as shown in Fig. 4.

E. Resource – RE

The *resource* (RE) feature category includes 2 features relating to the resource (`.rsrc` and `.rdata`) sections of the file, following [6], [8]. The first feature is a boolean for the presence of the debug directory in `.rdata`. The second feature is the number of resources defined in `.rsrc`.

The intuition for the presence of the debug directory is that packers can remove debug information to hinder reverse-engineering. The number of resources defined is a proxy to detect anomalous programs, such as when packers store their packed information in the resources section. Extracting these features requires checking the presence of the debug directory and parsing the `.rsrc` section to count the number of defined resources. The average extraction cost for this feature category is negligible if no resources exist, but can vastly increase if parsing a large resource section, as shown in Fig. 4.

F. Section – SC

The *section* (SC) feature category gathers 21 features that relate to the sections information. The majority of the features here have been used in prior works [6], [31], [32]. 10 integer features represent the number of sections of the PE file that: are standard; are non-standard; are executable; are writeable; are executable and readable; are executable and writeable; are readable and writeable; are executable, readable, and writeable; have raw data size zero; have different virtual and raw data size. 7 boolean features relate to the existence of the following packing artifacts: `.text` section not executable; a non-`.text` section is executable; `.text` section is not present; entry point is not in `.text` or `.tls` section; entry point is not in one of the standard sections; entry point not in an executable section; and address not matching file alignment. The remaining 4 features are: the ratio of standard sections to all sections; the ratio between the raw data and virtual size of the section with the entry point; the maximum ratio of raw data to virtual size; and the minimum ratio of raw data to virtual size.

The intuition here is that packers can use non-standard sections to change the program structure (e.g. UPX insert sections named `.upx0`, `.upx1`, etc.). Also, packing introduces many artifacts into the sections of the program. All of the features in this category can be computed from the PE header. The average extraction cost for this feature category is consistently negligible compared to the time required to open the file, as shown in Fig. 4.

V. Ground Truth Generation

One challenge in testing, training, and classification, is the lack of high-quality packed datasets – a lack of a *ground truth* – that can be used to reliably train classifiers. While it is possible to generate packed binaries using existing packers on binaries [9], this is largely not representative of packed binaries in the wild. Malware authors often use custom packers, to which we do not have access. In addition, many packing tools are proprietary or commercial, with few open options available to us for testing. To handle this challenge, we bootstrap two different ground truths by using three existing packing detection tools on our datasets: a 3CONS ground truth with the files that all three detection tools agree on (mimicking Sun et al. [29]), and a 1CONS ground truth with the files that are detected as packed by at least one of the detection tools (a superset of all files in the 3CONS ground truth).

We have built a dataset of 281,344 binaries kindly provided by Cisco and prof. Sandeep K. Shukla. The dataset has been created by capturing binary file streams available to our partners in the period from February to October 2017, and is thus representative of the stream of binary files that has to be analyzed by a medium-sized security company protecting a number of customers. We ran three heterogeneous packing detection tools based on different techniques on our dataset: Packerid [27], the Yara rules for packing detection curated by VirusTotal [5], and a proprietary tool by Cisco that checks the binary's hash against Cisco's database. If one or two tools detect a binary as being packed by a given technique, the binary is added to the 1CONS ground truth. When all three tools agree on a binary being packed by a given technique, we consider this to be strong evidence that the binary is indeed packed by the technique, and add it to both the 3CONS and the 1CONS ground truths. If a file is detected as more than one technique it is discarded, to avoid polluting the labeling in the ground truths. Note that the detection tools either report the packing technique detected, or `unpacked` if no packing technique is detected (or the file hash is not known, i.e. never report "unknown"). The ground truth selection is summarized in Table I. Table II reports the number of samples of each packer family in each ground truth. It is evident that the 3CONS ground truth has significantly less samples that the 1CONS ground truth, due to the rarity of an exact consensus between the three tools. In the experiments we use only packer families with at least 10 samples, meaning that a classifier trained on the 3CONS ground truth will not be able to correctly classify Armadillo, AutoIt, etc. and no classifier will be able to correctly classify

TABLE I. Generation of consensus (3CONS) and non-consensus (1CONS) ground truth based on three tools (Packerid, Yara, and a Hash-based proprietary tool). File1 is detected by two out of three as TheMida, hence it is added to the non-consensus ground truth as a TheMida sample. File2 is detected by all tools as UPX, hece it is added to the consensus ground truth as a UPX sample. File3 is not detected as packed by any tool, hence it is added to both ground truths as an unpacked sample. File4 is detected as more than one packing technique, hence it is not added to any ground truth.

File	Detection results		Ground truth	Label
File1	Packerid Yara Hash-based	TheMida TheMida unpacked	1CONS	TheMida
File2	Packerid Yara Hash-based	UPX UPX UPX	3CONS 1CONS	UPX
File3	Packerid Yara Hash-based	unpacked unpacked unpacked	3CONS 1CONS	unpacked
File4	Packerid Yara Hash-based	UPX Armadillo unpacked	discarded	-

TABLE II. Number of samples for each packer family in the two ground truths. Only families with ≥ 10 samples are used, so algorithms trained on 3CONS will be able to identify less families than algorithms trained on 1CONS.

Packer	3CONS	1CONS	Packer	3CONS	1CONS
Armadillo	0	6,849	NsPacK	17	60
ASPack	6,037	6,172	NeoLite	2	104
ASProtect	179	206	PackMan	24	78
AutoIt	0	1,048	PEArmor	0	793
CAB	0	75	PECloak	0	21
cpEllie	0	119	PECompact	1,240	1,327
cpFlush	0	19	PENinja	0	18
cpGlyph	0	15	PEPacK	6	7
CreateInstall	0	12	PEtite	9	13
D1S1G	0	360	ProtectSW	0	629
DarkComet	0	27	RARSFX	0	65
DevCv	0	36	RLPack	1	123
dlThunder	0	1,192	SafeDisc	0	13
dlUpatre	0	628	StealthPE	0	321
dUP2XPatcher	0	15	TASM	0	101
eXPressor	12	14	TheMida	13	150
EXEStealth	1	91	UPack	62	115
FSG	13	17	UPX	4,857	63,402
InnoSetup	0	975	WinRAR	0	48
InstallShield	4	22	WinZip	69	141
MEW	217	234	WiseInstaller	14	72
MoleBox	7	49	YodaProtect	1	95
mPress	0	847			
NSIS	7	2,039	unpacked	188,729	188,729

PEPacK. This can be solved by adding samples of the required families to the database.

We note that since the three techniques used (Packerid, Yara, and hash-based detection) are used to build the ground truth with the method above, it is not possible for us to evaluate the accuracy of such techniques against an independently-generated ground truth. Evaluating the techniques on a ground truth produced by the techniques themselves would be unsound. However, these techniques do represent the state of the art of static-signature-based packing detection tools used in practice by security researchers, so we expect them to be quite precise in practice.

Note that the ground truths are very unbalanced towards unpacked and UPX-packed files. This represents the distribution of the classes in the data sources we used, and generally means that algorithms that are better at detecting unpacked and UPX-packed samples will have a higher F-score than algorithms that are better at detecting other packer families. This reflects the intention of being able to correctly classify as many of the samples as possible. Here the averaging technique for multiclass F-score (as detailed in Section III) treats all samples as equally important. Note that this means classification on unpacked samples as unpacked is strongly favoured here as this is by far the most dominant family of samples.

VI. Experimental Setup

This section presents the procedure and experiments we have followed to implement and evaluate our approach. All the experiments were performed on a server with 14-core processors running at 2GHz, allowing up to 56 parallel threads, and with 128 GB of RAM, kindly provided by Cisco.

A. Definition of Classification Scenarios

Packing detection and classification can be considered as two separate challenges, as shown in Fig. 2. However, by manipulating the ground truth, we can construct classifiers that address either of the two challenges or both at the same time. We consider these as different *scenarios*.

DET Packing detection. An algorithms trained for the DET scenario only determines whether a sample is packed or not. An algorithm can be trained for the DET scenario by relabeling as `packed` every sample that is not labeled as `unpacked` in the ground truth, thus discarding the information about the specific packer family.

CLAS Packing classification. An algorithm trained for the CLAS scenario assumes that a sample is packed, and determines by which family. An algorithm can be trained for the CLAS scenario by removing every sample labeled as `unpacked` from the ground truth.

BOTH Packing detection and classification. An algorithm trained for the BOTH scenario determines both whether a sample is packed or not, and by which packer family if it is packed. An algorithm can be trained for the BOTH scenario by using the whole ground truth with samples labeled as `unpacked` or by family, like shown in Table II.

Each of the algorithms for the three scenarios above can be trained on the 3CONS or on the 1CONS, creating six possible scenario–ground truth combinations. In the rest of the paper we will refer to them by their scenario name and ground truth name, e.g. DET–3CONS refers to the case in which the classifiers are trained for packing detection on the 3CONS ground truth.

B. Feature Selection and Parameter Optimization

Section IV details the 119 features we use, divided in six categories: byte entropy (BE), entry bytes (EB), import functions (IF), metadata (ME), resource (RE), and section (SC). Testing the contribution of each category to the classification would be insufficient, since they are most likely not independent. Hence, to determine which feature categories fit our effectiveness and efficiency requirements, we to test every combination of

TABLE III. Parameters used for algorithm optimization. All parameter combinations for each algorithm have been tested.

Algorithm	Parameter	Values
Naive Bayes	prior	Gaussian
Decision Tree	splitter	best,random
	maximum depth	10,15,...,40
	maximum features	0.1,0.3,...,0.9
	minimum sample split	2,3,10
	minimum sample leaf	1,3,5
	criterion	gini,entropy
Random Forest Extra Trees	number of estimators	5,10,...,50
	maximum depth	10,15,...,40
	maximum features	0.1,0.3,...,0.9
	criterion	gini,entropy
	bootstrap	true,false

feature categories. For six categories this correspond to all the 63 non-empty subsets of the powerset of six elements.

Six scenario–ground truth combinations are defined at the end of Section VI-A. To perform a fair comparison, for each of the 378 combinations of scenario–ground truth–feature subset we perform a parameter optimization on each of the classification algorithms, i.e., we create a set of possible parameters for each algorithm (shown)in Table III and we find the best combination of parameters by testing them all. We measure F-score by cross-validation, average extraction and classification time per sample, and we rate the algorithms by the ratio of their F-score to extraction plus classification time.

C. Temporal Assessment

Since the malware and packing ecosystems evolves, new packing techniques and families as well as new versions of know families appear. To test the robustness of the packing detection and classification systems we are proposing, we perform a temporal assessment test. That is, we select the best algorithms for each scenario–ground truth combination, we train them with data from February to June 2017, and we test how they perform against data from July, August, September, and October 2017 in two-weeks intervals.

VII. Experimental Results

This section presents and discusses the results of the experiments from the previous section.

A. Feature Selection and Parameter Optimization

Table IVa presents for each scenario and ground truth, the three algorithm–feature categories with the highest F-score. This provides insight for each scenario into which features contribute to the algorithms and which do not. We observe the following facts.

- The F-score of the best algorithms for the 3CONS ground truth is always higher than the results for the best algorithms for the 1CONS ground truth on the same scenario. This is expected because the 3CONS ground truth has higher quality than the 1CONS ground truth, meaning that less files are labeled incorrectly in 3CONS than in 1CONS. However, recall from Table II that 3CONS contains significantly less families than 1CONS. This means that algorithms trained on 3CONS will be less effective that algorithms trained on 1CONS when used against malware in the wild. This is explored more in detail in Section VII-B, and in general represents a caveat against relying uniquely on cross-validation for fields with evolving environments like malware and packing detection.
- The Random Forest algorithm and its Extra Trees variant completely dominate: no Naive Bayes or Decision Tree reached the top 3 by F-score in any scenario–ground truth combination. This validates the intuition that higher complexity algorithms are required to achieve very high classification scores, due to the complexity of the packing problem.
- All the top algorithms require many different features to achieve the highest scores. The EB feature category appears in all the top 18 algorithms, the BE and IF categories in 14 of them, the ME and SC in 13, and the RE in 9. This provides insight on the contribution of the various feature categories to the algorithms. As a consequence, the feature extraction times are relatively high, often in the order of half a second per file.

Table IVb presents for each combination of scenario and ground truth, the three algorithm– feature categories with the highest ration between F-score and extraction plus classification cost. This provides insight as to which features and algorithms achieve a high F-score while having a low cost. We observe the following facts.

- The Decision Tree algorithm completely dominates, appearing as the algorithm with the highest F-score to cost ratio in every scenario and ground truth. The relative simplicity of Decision Tree compared to Random Forest allows it to achieve a slightly lower F-score with a significantly lower classification time, i.e. 0.3ms per sample.
- None of the feature categories used require any significant extraction time over the 14ms required on average to read the file itself (as explained in Section IV. This restricts the feature categories used to SC (used in 14 of the top algorithms), ME (12 algorithms), and EB (8 algorithms).
- Comparing the F-scores with the best algorithms overall for each scenario, in Table IVa, we note that the F-score of the best algorithms by F-score/cost ratio in Table IVb usually does not decrease by more than 1-2%. On the other hand, the sum of extraction and classification costs per sample are from 17 to 44 times lower. Hence, with accurate algorithm and feature selection it is possible to decrease the classification times significantly while keeping a very high F-score (**RQ1**).

B. Temporal Assessment

Figure 5 presents the results of the temporal assessment for the six combinations of scenarios and ground truths. The algorithms were trained on data from February to June as their ground truth, with the label of each graph indicating which consensus was used for the training. All algorithms were tested against data from July to October that was determined to be

TABLE IV. Best algorithms and feature categories (Byte Entropy BE, Entry Bytes EB, Import Function IF, Metadata ME, Section SC, Resource RE) for each configuration of scenario (detection DET, classification CLAS, or both BOTH) and ground truth (3CONS or 1CONS).

a Top 3 algorithms by F-score

Scen-GT	Name	Algorithm	Features	Ext (s)	Clas (s)	F-score
DET-3CONS	D3F3	Extra Trees	BE EB IF RE SC	0.6188	0.0015	0.9998
	D3F2	Extra Trees	EB ME IF SC	0.3922	0.0021	0.9998
	D3F1	Extra Trees	BE EB IF SC	0.4248	0.0021	0.9998
DET-1CONS	D1F3	Random Forest	BE EB ME IF RE	0.6187	0.0024	0.9895
	D1F2	Random Forest	BE EB ME IF	0.4284	0.0029	0.9897
	D1F1	Random Forest	BE EB ME IF RE SC	0.6188	0.0024	0.9897
CLAS-3CONS	C3F3	Random Forest	EB ME IF RE SC	0.5826	0.0050	0.9997
	C3F2	Random Forest	EB ME IF SC	0.3922	0.0038	0.9998
	C3F1	Random Forest	BE EB IF	0.4284	0.0019	0.9998
CLAS-1CONS	C1F3	Extra Trees	BE EB ME SC	0.0502	0.0049	0.9978
	C1F2	Extra Trees	BE EB ME IF RE SC	0.6188	0.0061	0.9979
	C1F1	Random Forest	BE EB ME IF RE SC	0.6188	0.0048	0.9997
BOTH-3CONS	B3F3	Extra Trees	BE EB ME SC	0.0502	0.0033	0.9999
	B3F2	Extra Trees	EB IF RE	0.5826	0.0058	0.9999
	B3F1	Extra Trees	BE EB IF	0.4284	0.0055	0.9999
BOTH-1CONS	B1F3	Random Forest	BE EB ME SC	0.0502	0.0037	0.9873
	B1F2	Random Forest	BE EB ME IF RE SC	0.6188	0.0053	0.9874
	B1F1	Random Forest	BE EB ME RE SC	0.2406	0.0046	0.9875

b Top 3 algorithms by F-score / (extraction cost + classification cost)

Scen-GT	Name	Algorithm	Features	Ext (s)	Clas (s)	F-score
DET-3CONS	D3R3	Decision Tree	ME	0.0140	0.0003	0.9942
	D3R2	Decision Tree	SC	0.0140	0.0003	0.9979
	D3R1	Decision Tree	ME SC	0.0140	0.0003	0.9984
DET-1CONS	D1R3	Decision Tree	EB SC	0.0140	0.0003	0.9749
	D1R2	Decision Tree	EB ME	0.0140	0.0003	0.9748
	D1R1	Decision Tree	EB ME SC	0.0140	0.0003	0.9765
CLAS-3CONS	C3R3	Decision Tree	EB ME	0.0140	0.0003	0.9989
	C3R2	Decision Tree	SC	0.0140	0.0003	0.9977
	C3R1	Decision Tree	ME SC	0.0140	0.0003	0.9979
CLAS-1CONS	C1R3	Decision Tree	EB SC	0.0140	0.0003	0.9938
	C1R2	Decision Tree	EB ME SC	0.0140	0.0003	0.9948
	C1R1	Decision Tree	ME SC	0.0140	0.0003	0.9935
BOTH-3CONS	B3R3	Decision Tree	ME	0.0140	0.0003	0.9950
	B3R2	Decision Tree	ME SC	0.0140	0.0003	0.9990
	B3R1	Decision Tree	SC	0.0140	0.0003	0.9989
BOTH-1CONS	B1R3	Decision Tree	ME SC	0.0140	0.0003	0.9630
	B1R2	Decision Tree	EB SC	0.0140	0.0003	0.9683
	B1R1	Decision Tree	EB ME SC	0.0140	0.0003	0.9747

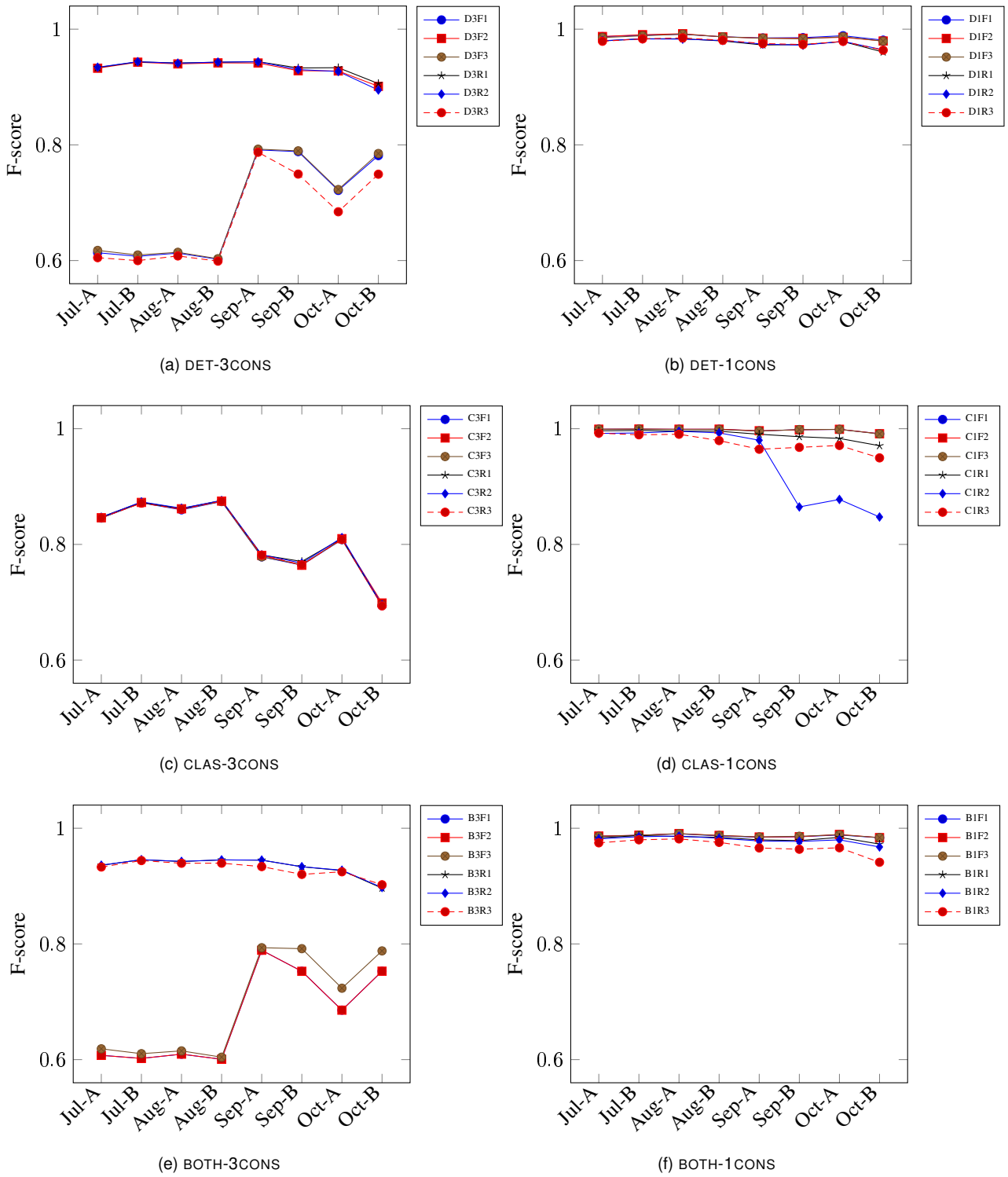
packed or not according to 1CONS, since 1CONS is more similar to the malware ecosystem in the wild than 3CONS. Hence, cross-validation was not used here.

We observe the following facts:

- The algorithms trained on the 3CONS ground truth perform much worse than the algorithms trained on the 1CONS ground truth, losing 6-30% of their F-score (**RQ2, RQ3**). The algorithms trained on 1CONS lose only 1-3% of their F-score, except for the outlier C1R2 (**RQ3**).
  The reason is that, as shown in Table II, the 3CONS ground truth contains less data and packer families than the 1CONS ground truth, hence such algorithms are not able to recognize a large amount of packer families. This contrasts with the results of Tables IVa and IVb in Section VII-A, where the algorithms trained on 3CONS had a higher F-score that the ones trained on 1CONS. The reason is that the values in Section VII-A are obtained by cross-evaluation, i.e. by testing and training on the same data sets. The important insight is that cross-evaluation favors the construction of smaller ground truths with more precise labels, but at the cost of reducing the representativeness of the ground truth and training algorithms that will be ineffective against real data.

TABLE V. Economical analysis for the best algorithms for the BOTH–1CONS scenario–ground truth combination, assuming that an algorithm has to be retrained when its F-score drops to 0.96.

Algorithm	Uptime before F-score=0.96 (s)	Training cost (s)	Uptime to training ratio
B1F1	25,728,120	328,921	80
B1F2	22,048,920	362,979	61
B1F3	20,656,080	356,457	58
B1R1	13,560,480	19,704	688
B1R2	11,274,120	18,990	593
B1R3	7,410,960	5,750	1,289



(a) DET-3CONS

(b) DET-1CONS

(c) CLAS-3CONS

(d) CLAS-1CONS

(e) BOTH-3CONS

(f) BOTH-1CONS

Fig. 5. Temporal assessment results. Each graph evaluates the F-score of the 3 best algorithms by F-score (Table IVa) and the 3 best algorithms by F-score/cost ratio (Table IVb) on a scenario-ground truth combination. Each algorithm is trained on data collected from February to June 2017 and tested on data collected on the first two weeks (A) and second two weeks (B) of July to October 2017. Algorithms trained on the 3CONS ground truth perform significatively worse, since there are many packer families that they are not able to recognize, as shown in Table II. For algorithms trained on the 1CONS ground truth, the robustness difference between the best algorithms by F-score and the best algorithms by F-score/cost ratio is small but perceivable.

VIII. Economical Analysis

We perform an economical analysis using the algorithms for the BOTH–1CONS scenario–ground truth combination. Assume that an algorithm is useful while its F-score is at least 0.96, after which time the algorithm then has to be retrained. Consider the 6 algorithms whose robustness is plotted in Figure 5(f). For each algorithm, we use the temporal assessment values to do a quadratic least squares regression and extrapolate the time at which the algorithm will reach an F-score of 0.96, requiring retraining. Such projection are presented in the second column of Table V. The third column of Table V presents the training costs for the parameter optimization for the algorithms from the experiments in Section VII-A. Finally, the fourth column of Table V presents the ratio between the time before retraining and the training time, i.e. how many seconds of uptime are obtained for each second spent training the algorithm. As the table shows, the simple Decision Tree algorithm with just the ME and SC feature categories has the highest ratio, since its very low training time means that it is more economically convenient to use it even if it has to be retrained more frequently than the other algorithms (**RQ4**).

IX. Discussion

This section discusses the results in more details and summarizes the lessons learned.

On Feature Selection: This paper performs an evaluation of the cost and effectiveness of using different sets of features for supervised classification. Not all features have the same extraction cost, and training classifiers on large feature sets decreases their performance compared to training them on smaller feature sets. Hence, the efficiency of the classification is highly impacted by the features used.

Table IV shows that using all available features does *not* yield the highest F-score. This confirms that using less features can improve the extraction, training and classification times. To understand which features to exclude, we have examined not only their contribution to the F-score of the classification but also their cost. Similarly, we have evaluated complex algorithms like Random Forest with simpler algorithms like Decision Tree. The economical analysis in Table V shows that when both effectiveness and efficiency are considered then accepting *a small decrease in effectiveness can increase efficiency by many orders of magnitude*. This addresses research questions **RQ1** and **RQ4**.

This insight is of paramount importance to analysis and companies running large-scale ML-based malware detection and classification systems. Due to the evolution of the malware ecosystem, such systems require regular retraining on recent data representative of the current ecosystem. While carefully identifying efficient algorithms and selecting features that are cheap to extract and evaluate, the system uptime to training time ratio can be sharply increased, resulting in lower costs for the analyst and better protection for the users.

On Ground Truth: Many academic papers manually build a small-size ground truth of manually-verified samples. This is of course not possible for large-scale ground truths like the one used in this paper. Due to the difficulty and sometime inherent ambiguity of classifying samples, producing a highly reliable malware ground truth is extremely time-expensive and sometimes impossible. Hence, analysts and companies have to often rely on low-confidence ground truth, where the classification labels cannot be inherently trusted.

This paper constructs two ground truths of different sizes and reliability to understand the impact of the size and reliability of the ground truth on supervised classification: a 3CONS ground truth that is smaller but more reliable, and a 1CONS ground truth that is larger but less reliable. Importantly, some packer families that are present in the 1CONS ground truth had to be excluded from the 3CONS ground truth due to the lack of samples from such families that were classifier with high enough confidence.

Table IV shows that classifiers trained on the 3CONS ground truth achieve a higher F-score compared to classifiers trained on the 1CONS ground truth. However, the F-score of the classifiers are evaluated by cross-validation on the same ground truth used to train them, meaning that the classifiers trained on 3CONS are not evaluated on the packer families that appear in 1CONS but not in 3CONS. This causes the classifiers trained on 3CONS to perform significantly worse than the classifiers trained on 1CONS in the temporal assessment in Figure 5, where all the packer families are considered. Research questions **RQ2** and **RQ3** are closely related to this result.

The temporal assessment is more representative of a real packing detection and classification scenario than cross-validation, so we conclude that *training classifiers on a larger, lower-reliability ground truth like* 1CONS *is preferable than training classifiers on a smaller, higher-reliability ground truth like* 3CONS. Due to the evolution of the malware and packer ecosystem, wide coverage is necessary to properly train supervised classifiers, even at the cost of some label quality. This can be considered good news, since increasing a database's size with samples with low-confidence labels is easier than increasing the confidence in the labels of the database.

A. Threats to Validity

This section details the possible threats to the validity of this paper's methodology and how they were addressed.

- As remarked in Section V, we chose the three signature-based techniques used to annotate the database according to their widespread usage among security researchers and analysts. However, frequently the techniques were in disagreement on the labeling of a sample, not just whether a technique considers a sample packed and another considers it unpacked, but also cases where the three techniques labeled the same sample with up to six different packing families. Samples with such disagreements have been removed from the database, but this shows the fragility of these techniques against advanced packers. While on one hand this motivates the necessity of ML-based techniques like the ones proposed in this paper, on the other hand it can be considered a warning on the

unreliability of malware and packing classification ground truth.

- As remarked in Section III and Section V, the creation of a database to use as ground truth requires some care to be aware of the representativeness and bias of the data. Additional bias is introduced by the choice of the effectiveness metric used to evaluate the classifiers (in this paper, the F-score) and the averaging method for multiclass classification (in this paper, all samples are considered equally important). The choice in this paper reflects the fact that we give the same importance to false positive and false negatives in packing detection, and that we aim at correctly classifying the highest possible number of samples. Different choices may have led to different results. Analysts should always consider the origin of their data and their classification goals when choosing analysis parameters like the effectiveness metric and its averaging method.
- The features used in this paper have been divided in thematic-based groups (entropies, metadata, etc.) in Section IV. The grouping is necessary since features are not independent, so their effectiveness has to be evaluated on all possible combinations, and testing all combinations of 119 features is computationally unfeasible. However, dividing the groups by theme means testing together features with possibly very different extraction costs. For instance, Figure 4 shows that the cost of analyzing the resource section of the PE file varies from negligible to seconds. This is because if the resource section has been removed or is empty this is verified in negligible time, otherwise parsing it can require several seconds. Unfortunately when not using the resource features we lose the very significant information on whether or not the resources section is empty. Future work will consider alternative feature groups.
- The economical analysis in Section VIII assumes that the analyst retrains a supervised classifier when its F-score drops to 0.96. The section shows how in this case it is convenient to use simple algorithms and small feature sets to allow cheap and frequent retraining, increasing the uptime to training time ratio by orders of magnitude. However, it may be the case that the analyst requires a much higher F-score, e.g. 0.99, and this is attainable only by complex algorithm and large feature sets. While this does not invalidate our analysis, in this case obviously the requirements would not allow the cheap and frequent retraining we recommend.

B. Limitations and Future Work

- A limit of supervised learning is to not be able to recognize classes that were not present in the ground truth. In this paper's case, this means that packer families for which a classifier has not been trained will not be recognized, and as shown in Section VII-B this is the main cause of the ineffectiveness of classifiers trained on the 3CONS ground truth. More generally, unsupervised learning techniques like clustering should be used to provide information about malware packed with previously unknown packing techniques. We consider this as future work.
- The construction of the ground truth has been shown to be fundamental in determining the effectiveness of the packing classification process. Additional packing detection techniques apart from the ones used in Section V could be used to improve the labeling of the ground truth. Additionally, it could be possible to rate the packing detection techniques with different confidence values based on their reliability to better understand how to solve conflicting labelings in a more advanced manner than just the consensus/non-consensus paradigm used in this paper.
- Repacked malware, i.e. malware packed using multiple packers in sequence, has not been considered here (and removed from the ground truth when found, as explained in Section V). We refer to Bat-Erdene et al. [9] for further discussion on this topic.
- Encoders that just encode part of the binary have also not been considered here. For instance, XORing part of the code in a malware is a common technique to hide easily-identifiable malicious code. We consider encoding detection and decoding as future work.

X. Conclusions

This paper has the goal of understanding the impact of ground truth generation, algorithm selection, and feature selection on ML-based packing detection and classification, following the example of works on empirical testing of ML malware analysis including Allix et al. [7].

We find that the size of the ground truth is more relevant than its quality. Supervised ML algorithms can classify correctly only for classes that they have been trained on, so having a significant amount of samples for each class is more relevant than the samples' labeling being highly reliable. In particular, cross-validation testing is deceiving on fields like malware and packing detection where new types of samples and packing techniques appear in the wild. This contributes to explaining why algorithms can perform well in the lab and badly in the wild, as found report [7].

We find that the number and extraction costs of the features used has a dramatic impact on the training and classification times, and consequently on the economical viability of using ML algorithms. In fact, selecting features and algorithm for both effectiveness and efficiency can decrease training and classification costs by orders of magnitude against a small decrease in effectiveness. In practice this implies that often using a simple algorithm (e.g. Decision Tree) on a reduced feature set and retraining it often results in an optimal expenditure of time compared to using a more complex algorithm (e.g. Random Forest) over many features and retraining it sparingly. This of course assumes that the minimum F-score requirements are not too high, in which case complex algorithms on large feature sets may be necessary. Our economical analysis can be adapted as required considering cost and robustness.

We note that our results depend on some basic engineering choices, in particular the implementations of PE feature extraction and ML algorithm. More efficient or optimized implementations will possibly give different results. However, our methodology remains sound. We encourage other researchers and institutions to evaluate their packing detection and classification algorithms with the methodology presented here and to publish their findings.

Acknowledgments. The authors would like to thank Steve Rich, Adam Weller, Alain Zidouemba, and the Talos team from Cisco for providing packed samples, stimulating discussion, and assessing the pertinence of the research questions; and Sandeep K. Shukla for providing packed samples.

References

[1] ClamAV. https://www.clamav.net/. September, 2018.
[2] NumPy. http://www.numpy.org. September, 2018.
[3] PEiD, version 0.95. http://appnee.com/peid/. September, 2018.
[4] PeLib, version 1.0. https://github.com/avast-tl/pelib. September, 2018.
[5] VirusTotal Packer YARA Ruleset. https://github.com/Yara-Rules/rules/blob/master/Packers/packer.yar. September, 2018.
[6] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*, pages 183–194, 2016.
[7] Kevin Allix, Tegawendé F. Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. Empirical assessment of machine learning-based malware detectors for android. *Empirical Software Engineering*, 21(1):183–211, Feb 2016.
[8] Randall Balestriero, Hervé Glotin, and Richard G. Baraniuk. Semi-supervised learning enabled by multiscale deep neural network inversion. *CoRR*, abs/1802.10172, 2018.
[9] Munkhbayar Bat-Erdene, Taebeom Kim, Hyundo Park, and Heejo Lee. Packer detection for multi-layer executables using entropy analysis. *Entropy*, 19(3):125, 2017.
[10] Munkhbayar Bat-Erdene, Hyundo Park, Hongzhe Li, Heejo Lee, and Mahn-Soo Choi. Entropy analysis to classify unknown packing algorithms for malware detection. *International Journal of Information Security*, 16(3):227–248, 2017.
[11] Guillaume Bonfante, Jose Fernandez, Jean-Yves Marion, Benjamin Rouxel, Fabrice Sabatier, and Aurélien Thierry. CoDisasm: Medium Scale Concatic Disassembly of Self-Modifying Binaries with Overlapping Instructions. In *22nd ACM Conference on Computer and Communications Security*, Denver, United States, October 2015.
[12] Nancy Chinchor. Muc-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding*, MUC4 '92, pages 22–29, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.
[13] Yang-Seo Choi, Ik kyun Kim, Jin-Tae Oh, and Jae cheol Ryou. PE File Header Analysis-Based Packed PE File Detection Technique (PHAD). *International Symposium on Computer Science and its Applications*, pages 28–31, 2008.
[14] Cisco. Annual Cybersecurity Report, 2018. https://www.cisco.com/c/m/en_au/products/security/offers/cybersecurity-reports.html. September, 2018.
[15] Neminath Hubballi and Himanshu Dogra. Detecting packed executable file: Supervised or anomaly detection method? *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 638–643, 2016.
[16] Guhyeon Jeong, Euijin Choo, Joosuk Lee, Munkhbayar Bat-Erdene, and Heejo Lee. Generic unpacking using entropy analysis. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 98–105. IEEE, 2010.
[17] Kesav Kancherla, J Kevin Donahue, and Srinivas Mukkamala. Packer identification using byte plot and Markov plot. *Journal of Computer Virology and Hacking Techniques*, 12:101–111, 2015.
[18] Michael Hale Ligh, Andrew Case, Jamie Levy, and Aaron Walters. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley Publishing, 1st edition, 2014.
[19] Robert Lyda and James Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, 5(2):40–45, 2007.
[20] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *23rd Annual Computer Security Applications Conference (ACSAC 2007), December 10-14, 2007, Miami Beach, Florida, USA*, pages 431–441, 2007.
[21] Steve Morgan. *Global Ransomware Damage Costs Predicted To Exceed $5 Billion In 2017*. Cybersecurity Ventures, May 2017. https://cybersecurityventures.com/ransomware-damage-report-2017-5-billion/.
[22] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *23rd Annual Computer Security Applications Conference (ACSAC 2007), December 10-14, 2007, Miami Beach, Florida, USA*, pages 421–430, 2007.
[23] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters*, 29(14):1941 – 1946, 2008.
[24] Sebastian Porst. PeLib. http://www.pelib.com. September, 2018.
[25] Jithu Raphel and P. Vinod. Information theoretic method for classification of packed and encoded files. In *Proceedings of the 8th International Conference on Security of Information and Networks*, SIN '15, pages 296–303, New York, NY, USA, 2015. ACM.
[26] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *Proceedings of the 22Nd Annual Computer Security Applications Conference*, ACSAC '06, pages 289–300, Washington, DC, USA, 2006. IEEE Computer Society.
[27] Mike Sconzo. Packerid. https://github.com/sooshie/packerid, July 2016.
[28] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition, 2012.
[29] Li Sun, Steven Versteeg, Serdar Boztaş, and Trevor Yann. Pattern recognition techniques for the classification of malware packers. In *Proceedings of the 15th Australasian Conference on Information Security and Privacy*, ACISP'10, pages 370–390, Berlin, Heidelberg, 2010. Springer-Verlag.
[30] Xabier Ugarte-Pedrero, Igor Santos, and Pablo G. Bringas. Structural feature based anomaly detection for packed executable identification. In Álvaro Herrero and Emilio Corchado, editors, *Computational Intelligence in Security for Information Systems*, pages 230–237, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
[31] Xabier Ugarte-Pedrero, Igor Santos, Iván García-Ferreira, Sergio Huerta, Borja Sanz, and Pablo G Bringas. On the adoption of anomaly detection for packed executable filtering. *Computers & Security*, 43:126–144, 2014.
[32] Mohaddeseh Zakeri, Fatemeh Faraji Daneshgar, and Maghsoud Abbaspour. A static heuristic approach to detecting malware targets. *Security and Communication Networks*, 8(17):3015–3027, 2015.

Fabrizio Biondi obtained his BSc and MSc in Computer Science from the University of Bologna, Italy. He then undertook a PhD with Andrzej Wąsowski at the IT University of Copenhagen, developing techniques for the computation of information leakage of systems modeled as Markovian processes. He is a Maître des Conférences at CentraleSupélec and Chair of Cybersecurity in Threat Analysis for Région Bretagne. Fabrizio's research is focused on machine-learning-based malware detection and fingerprinting, vulnerability detection, and unconditional cryptography.

Michael Enescu obtained his BSc and MSc from The University of British Columbia in Vancouver, Canada in 2012 and 2016, respectively. During his MSc, Michael focused on the intersection of security and formal methods, conducting research in quantitative information flow in software using formal techniques. In addition, Michael spent 2017 at Inria Rennes as a PhD student, where his research shifted focus to malware analysis, with continued research in approximate quantitative information flow. Michael currently works as a software engineer in the fin-tech industry.

Thomas Given-Wilson holds a BCST from the University of Sydney, and both BS(Hons)IT and PhD from the University of Technology, Sydney. He worked on static analysis tools for NICTA before moving to France to join Inria Saclay as a post-doctoral researcher. At Inria Saclay Thomas' research focused upon privacy, concurrency, and quantified information flow. Since moving to inria Rennes Thomas' research has expanded to also include cryptography, malware analysis, cybersecurity, and human motion models.

Axel Legay held positions at University of Liège and CMU (under the supervision of Ed Clarke). He is a full-time researcher at INRIA where he leads research on the Internet of Things and systems of systems. His main research interests are in developing formal specification and verification techniques for Software Engineering. Axel is a referee for top journals and conferences in formal verification and simulation, and program co-chair of INFINITY'09, FIT'10, Runtime Verification 2013, Splat 2014, FORMATS 2014, ATVA 2016, and TACAS 2017. He has been the Inria representative for more than 10 Inria projects over the six last years.

Lamine Noureddine got his BSc in Computer Science and his MSc in Computer Security from the University of Science and Technology Houari Boumediene (USTHB), Algiers, Algeria. In his two university degree projects, he worked on shellcode detection as well as on malware analysis and classification. He is now pursuing a PhD degree under the supervision of Axel Legay and Fabrizio Biondi, at INRIA Rennes, in France. His research topic is about developing new packing detection techniques to stop malware propagation.

Vivek Verma has graduated in computer science and engineering from IIT Kanpur, India. He is presently working as a software engineer in Visa, India. He has research interned in University of Texas at Dallas, USA as well as Inria at Rennes, France. His interest lies in and has worked in collaborative efforts towards application of machine learning in system security.