MLxPack: Investigating the Effects of Packers on ML-based Malware Detection Systems Using Static and Dynamic Traits

Qirui Sun
Sungkyunkwan University
Suwon, South Korea

Mohammed Abuhamad†
Loyola University Chicago
Chicago, USA

Eldor Abdukhamidov
Sungkyunkwan University
Suwon, South Korea

Eric Chan-Tin
Loyola University Chicago
Chicago, USA

Tamer Abuhmed†
Sungkyunkwan University
Suwon, South Korea

ABSTRACT

Malware is one of the serious computer security threats. To protect computers from infection, accurate detection of malware is essential. At the same time, malware detection faces two main practical challenges: the speed of malware development and their distribution continues to increase with complex methods to evade detection (such as a metamorphic or polymorphic malware). This research utilizes various characterizing features extracted from each malware using static and dynamic analysis to build seven machine learning models to detect and analyze packed windows malware. We use a large-scale dataset of over 107,000 samples covering unpacked and packed malware using ten different packers. We examined the performance of seven machine learning techniques using 50 dynamic and static features. Our results show that packed malware can circumvent detection when a single analysis is performed while applying both static and dynamic methods can help improve the detection accuracy around 2% to 3%.

CCS CONCEPTS

• **Security and privacy → Software and application security**;

KEYWORDS

Malware Detection, Machine Learning, Software Packing

ACM Reference Format:
Qirui Sun, Mohammed Abuhamad†, Eldor Abdukhamidov, Eric Chan-Tin, and Tamer Abuhmed†. 2022. MLxPack: Investigating the Effects of Packers on ML-based Malware Detection Systems Using Static and Dynamic Traits. In *Proceedings of the 1st Workshop on Cybersecurity and Social Sciences (CySSS '22), May 30, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3494108.3522768

† corresponding author.

CySSS '22, May 30, 2022, Nagasaki, Japan

ACM ISBN 978-1-4503-9177-1/22/05…$15.00
https://doi.org/10.1145/3494108.3522768

1 INTRODUCTION

In recent years, there has been a continuously increasing trend of malware infections posing a significant threat to information security of Internet users. Malware shows a rapid development trend with an increasing number of variants and a wide range of spread and influence. These rising challenges make traditional malicious code detection methods fall back on meeting the requirements for malware detection [3, 5].

Generally, typical malware detection methods can be divided into three main categories: signature-based, behavioral-based, and heuristic-based methods [2, 10]. Signature-based detection methods are based on the idea of pattern matching, generating a unique signature for each known malicious code to create a malicious code library. These signatures include many different attributes such as filenames, content strings, or bytes, and are also explored from the perspective of excluding security vulnerabilities created by these malicious codes to protect system security. When encountering an unknown software, the signature of a tested sample is compared against the malicious code library. If the signature triggers a match, it is flagged as malicious software. The advantages include fast detection time and accurate detection with a low false-positive rate. The disadvantage, however, is dealing with unseen samples and variants, which requires a constant update of the signature database. This updating and maintaining of the signature database requires enormous efforts and resources. Malicious code programmers can bypass this detection mechanism only through simple variants such as obfuscation, compression, and packing.

The behavioral detection methods explore unique malicious behavioral characteristics that can trigger a malicious activity. This is done through observation and research on malware to identify common behaviors that are unique to malicious activities. These methods predict most unknown malware accurately, but they cannot identify the names and families of detected malware. Moreover, such methods are difficult to implement. On the other hand, heuristic malware detection methods adopt machine learning techniques to learn the behavior of malware through in-depth analysis of traits that are extracted using static or/and dynamic analyses. These methods provide efficient and accurate detection of malware [2, 5]. However, considering the current trend of continuous growth of malware and variants, and the large availability of tools that enable circumventing these methods, it becomes increasingly challenging to provide reliable and adaptable malware detection engines [3, 6].

Machine learning (ML)-based malware detection methods have attracted the research community to explore solutions and analyze their effectiveness against evolving challenges. These methods rely on static analysis or/and dynamic artifacts. In this research, we explore the effects of different packing tools on the performance of ML-based methods using both static and dynamic analyses. We use a large-scale dataset of 107,599 windows executable programs, including 70,062 packed samples using ten packers, namely Obsidium, Themida, PECompact, Petite, UPX, kkrunchy, MPRESS, tElock, PELock, and dolphin-dropper-3.

Contributions. The contributions of this work are as follows:

- We investigate the performance of seven machine learning-based malware detection methods using a large-scale dataset of 107,599 executable samples. Our investigation includes studying the effects of packing techniques on the performance of such methods.
- We examine the effect of different malware analyses, *i.e.,* static, dynamic, and combination of both, on the machine learning detection methods when packing is involved. We extracted 36 static features and 14 dynamic features from each malware across the entire dataset.
- Adopting various analyses and machine learning methods, we evaluated the importance of features for accurate detection.

2 MLxPack: DATASET AND METHODS

2.1 Dataset

In this work, we use the malware dataset provided by Aghakhani *et al.* [4]. The experiments are conducted to analyze the machine learning-based malware detection methods on windows packed and unpacked malware. The dataset consists of the dataset from the EMBER, and a commercial vendor [7], as well as the dataset that is made by labeling the executable as packed and unpacked using commercial and publicly available packers: Obsidium, Themida, PECompact, Petite, UPX, kkrunchy, MPRESS, tElock, PELock, and dolphin-dropper-3. The detailed information about the packed dataset is shown in Table 1. Overall, the dataset includes 107,599 executable programs that are divided into 22,544 packed benign, 47,518 packed malicious, 12,472 unpacked benign, and 25,065 unpacked malicious.

2.2 Static and Dynamic Feature Extraction

Static features. To gain a deeper understanding of malware, it is essential to examine its static properties. Conducting such analysis is relatively easy since it does not require running the malware. The static analysis aims to study artifacts such as hashes, embedded strings, embedded resources, header information, among others. Our static analysis is conducted to extract information from the Control Flow Graphs (CFGs) of software samples. Using radare2's [16] Python API – r2pipe, we extracted the CFGs and applied various feature extraction methods to obtain graph-level features such as nodes, edges, density, betweenness, shortest path, *etc.* Moreover, we extended the static features with the top used ten libraries based on their frequency and relocations(relocs) from imports (*i.e.,* symbols imported from libraries). Table 2 shows the detailed information of the 36 extracted features. We categorized these 36 features into eleven groups, including ten libraries from the imports section (*kernel32.dll, user32.dll, advapi32.dll, shell32.dll, ole32.dll, gdi32.dll, comctl32.dll, ntdll.dll, msvcrt.dll, oleaut32.dll*), centrality, shortest path, network density, connected components(CC), average degree connectivity, relocs, and the number of edges and nodes. As for centrality and shortest path, we extracted the minimum, maximum, median, mean, and standard deviation values.

Dynamic features. Dynamic malware analysis allows malware analysts to monitor the execution of malware at every step. The malware is usually executed in a sandbox or VM, and then the process behavior, network behavior, file behavior, and other dynamic behaviors during its operation are extracted. In terms of behavioral analysis, we extracted the number of APIs, DLLs, calls, and modules. In our experiments, we utilized the cuckoo reports that are generated using cuckoo sandbox [18]. These reports contain different information about the target software, such as various static, behavioral, and network analyses, the dropped files and buffers, process memory, VM memory dump, *etc.* We used the dynamic features that are based on behavioral, static, and network analyses for our experiment. For static features, we extracted the number of PE imports, keys, and PE resources used by executable programs. As for the network features, we used the network usages in terms of UDP, DNS, ICMP, TCP, Hosts, and domains. Table 3 shows the adopted 14 dynamic features.

2.3 Machine Learning Methods

Based on the static and dynamic analysis, we applied several machine learning models for windows malware detection. The following seven machine learning algorithms are applied: ❶ Support Vector Machine (SVM), ❷ Decision Tree (DT), ❸ Logistic Regression (LR), ❹ Random Forest (RF), ❺ K-Nearest Neighbors (KNN), ❻ artificial neural network (ANN), and ❼ Adaptive Boosting (AdaBoost) in our experiments. All the hyperparameters used for those machine learning algorithms are provided in Table 4.

Support Vector Machine. Support vector machine is a very powerful machine learning technique with multiple functions, capable of dealing with linear or non-linear classification problems with a low generalization error rate, which means that it has good learning and generalization ability. To select the margin hyperplane to classify training points correctly, 10 is used for the regularization parameter and 0.01 for the kernel coefficient.

Decision Tree. Decision tree is a machine learning method that adopts a tree structure in which each internal node represents a judgment on an attribute, each branch represents the output of a judgment result, and finally, each leaf node represents a classification result. The decision tree is easy to understand and explain, can be visualized and analyzed, and can make feasible and effective results for large data sources in a relatively short time. Generally, all default hyperparameters are used for the algorithm except for max_depth, which was set to 20 to cover more depth.

Logistic Regression. Logistic Regression is a classification technique for binomial outcomes. This technique is fast, simple, easy to understand, suitable for binomial classification problems, and memory-efficient. 10 is used as the value for the inverse of regularization strength to give a high weight to the training data and a lower weight to the complexity penalty, while default values are used for the other hyperparameters.

Random Forest. Random Forests is an ensemble learning method based on Bagging, which can handle classification and regression

Table 1: Distribution of benign and malicious samples using various packers.

Packer name	Benign	Malicious
PELock	3,350	4,157
tElock	1,175	6,816
MPRESS	1,767	2,950
UPX	2,558	5,377
dolphin-dropper-3	2,637	5,245
PECompact	1,202	5,947
Kkrunchy	2,057	4,916
Petite	2,357	4,396
Themida	2,611	4,252
Obsidium	1,830	3,462
Total	**21,544**	**47,518**

Table 2: Distribution of static features from CFGs and radare2 imports.

Feature category	# of features
Imports libraries	10
Betweenness centrality	5
Closeness centrality	5
Degree centrality	5
Shortest path	5
Network Density	1
Connected components	1
Average degree connectivity	1
Number of Edges	1
Number of Nodes	1
Relocs	1
Total	**36**

Table 3: Dynamic features based on their category analysis.

Category	Feature names
Behavioral analysis	apistats
	dll_loaded
	calls
	modules
Static analysis	pe_imports
	pe_resources
Network analysis	networks
	UDP
	DNS_ servers
	ICMP
	TCP
	Hosts
	DNS
	Domains

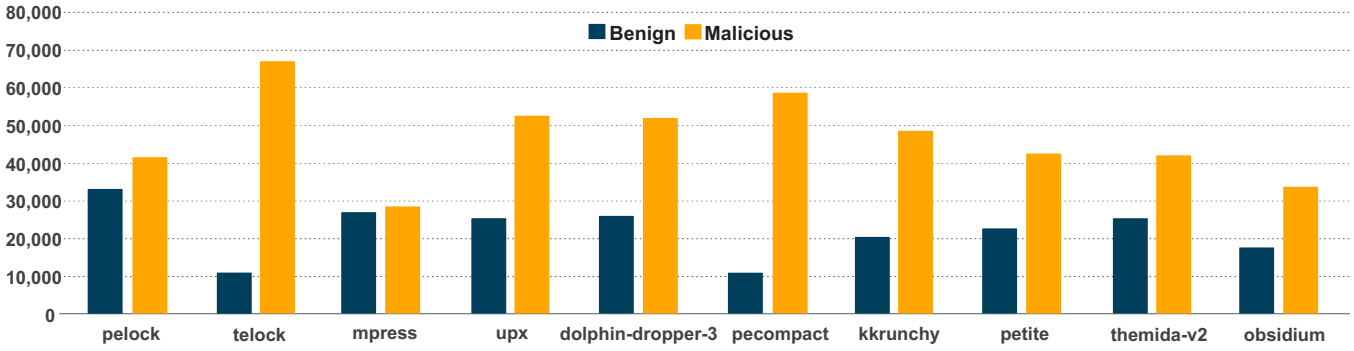

Figure 1: Distribution of benign and malicious samples in packed software dataset. The packed software dataset include samples from 10 packers.

problems well. It comprises many decision trees, and there is no correlation between different decision trees. It can run efficiently on large data sets with high accuracy. We used default hyperparameters for this algorithm.

K-Nearest Neighbors. The core idea of the KNN algorithm is that in a space containing an unknown sample, the data type of the sample can be determined according to the data type of the k samples closest to the sample. The KNN algorithm can be used not only for classification but also for regression. It is easy to implement, has no need to estimate parameters, and supports incremental learning. For this algorithm, leaf size and number of neighbors are set to 40 and 4, respectively, to reduce the number of neighbors and select the closest ones.

Artificial Neural Network. We used a Multilayer Perceptron (MLP) to represent an artificial neural network, which has a forward structure that maps a set of input data to a set of output data. MLP can be regarded as a directed graph composed of multiple node layers, and each layer is fully connected to the next layer. Each node is a neuron with a nonlinear activation function. MLP can be applied to complex non-linear problems and works well with a large dataset. The predictions are quick with high accuracy. The initial learning rate is set to 0.0007 to control the step size in updating the weights. Furthermore, we increased the alpha from default 0.0001 to 0.1 to fix the high variance (a sign of overfitting).

Adaptive Boosting. Adaptive Boosting is an iterative boosting algorithm. Its core idea is to train different classifiers (weak classifiers) for the same training set, and then combine these weak classifiers to form a stronger final classifier. Different classification algorithms can be used as weak classifiers. The weight of each classifier is fully considered by AdaBoost. We used the default hyperparameters for this technique.

Experiment Settings:

❶ **Dataset Distribution.** For our study, we divided the dataset into two categories packed and unpacked (benign/malicious). We also studied the ten packers separately and combined.

❷ **Handling Class Imbalance.** As shown in Figure 1, there are almost twice more malicious executables than the benign ones. To overcome the issue, we applied Synthetic Minority Oversampling Technique (SMOTE) [13] to oversample the minority class. SMOTE selects close samples by drawing a boundary line between those samples in their feature space and creates new samples along that line. Specifically, SMOTE selects a random example among samples from the minority class and finds its K nearest neighbor samples from the same class. All generated synthetic samples are based on a convex combination of those selected samples. In our experiments, we utilize SMOTE to resample all classes but the majority class using 5 nearest neighbors.

Table 4: Settings and hyperparameters used for building ML models. All settings follow the default implementation in scikit-learn library with the modifications highlighted in the table.

Machine Learning Algorithm	Hyperparameter
Support Vector Machine	*C: 10, kernel: 'rbf', degree: 3, gamma: 0.01, tol: 1e-3, cache_size: 200, max_iter: -1, decision_function_shape: 'ovr'*
Decision Tree Classifier	*criterion: 'gini', splitter: 'best', max_depth: 20, min_samples_split: 2, min_samples_leaf: 1*
Logistic Regression	*Penalty: l2, tol: 1e-4, C:10, solver: 'lbfgs', max_iter: 100, multi_class: 'auto'*
Random Forest	*n_estimators: 100, criterion: 'gini', min_samples_split: 2, min_samples_leaf: 1, max_features: 'auto'*
K-Nearest Neighbors	*n_neighbors: 4, weights: uniform', algorithm: auto', leaf_size: 40, p: 2, metric: 'minkowski'*
Artificial Neural Network	*hidden_layer_sizes: (100,), activation: 'relu', solver: 'adam', alpha: 0.1, batch_size: 'auto', learning_rate_init: 0.0007, learning_rate: 'constant', power_t: 0.5, max_iter: 200, tol: 1e-4, momentum: 0.9, validation_fraction: 0.1, beta_1: 0.9, beta_2: 0.999, epsilon: 1e-8, n_iter_no_change: 10, max_fun: 15000*
Adaptive Boosting	*n_estimators: 50, learning_rate: 1.0, algorithm: 'SAMME.R'*

❸ **Training/Testing Data Splitting.** Before machine learning algorithms are applied, the dataset is split into training and testing datasets in a stratified fashion with 70% and 30%, respectively.

❹ **Evaluation Metrics.** The machine learning results are compared based on precision, recall, and F1 score. The precision rate, Precision = TP/TP+FP where TP is the true positive and FP is the false positive, indicates how many of the samples whose predictions are positive, are truly positive samples. The Recall, Recall = TP/TP+FN where FN is the false negative, indicates how many samples with a positive true label are predicted by the model. F1-score is a weighted average of precision and recall, and it is calculated as F1-score = $(2 \times \text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall})$.

2.4 Feature Importance

To explain what features are essential in the prediction phase, we adopt Dalex [9] for each machine learning algorithm. Dalex is a combination of the Explainable Artificial Intelligence (XAI) tools in terms of the LOCO (leave-one covariate out) approach to generate explainabilities. Generally, the model and dataset to be explained are passed to the metric, the performance of the model is then measured by conducting new training processes with the newly-generated data and the inversion of each attribute of the data in an iterative and unitary way, hence each attribute measurement is crucial to the model regarding the performance. The metric is applied based on the features: static, dynamic, and hybrid (static+dynamic) with packed, unpacked, and whole dataset (packed+unpacked). The final result is extracted based on the intersection of the results of those features, and they are provided in Figure 2.

Observing dynamic feature importance based on the machine learning models, several features are found significant by the majority of models, such as *#modules_behavior, #pe_resources_static, #dll_loaded_behavior, #calls_behavior, #apistats_behavior, #networks, udp, hosts*. From the perspective of static feature importance, *comctl32.dll* is shown to be the most important feature by all models, while *ntdll.dll* is considered less important by only one model (*i.e.,* KNN). Considering the results of hybrid features, *#pe_resources_static, modules_behavior, #dll_loaded_behavior* are the most important features to the majority of machine learning models. Based on our observation, the dynamic features contribute highly to the correct prediction of the malware.

3 MLxPack: EXPERIMENTS AND RESULTS

In the experiments, we divided the dataset into four big categories: whole dataset, unpacked dataset, packed dataset, and ten packers' dataset. We apply the dynamic, static, and hybrid(*i.e.,* dynamic+static) features to build classifiers and obtain the final precision, recall, and F1 scores. Each experiment is performed ten times, and each time the dataset is randomly split to perform unbiased model evaluation, and then we report the average results.

3.1 Experiment 1: Packing-Absent Analysis

This experiment aims to evaluate the features and machine learning methods for detecting malware without the presence of packing. To this end, we use only unpacked benign/malware samples for extracting features and building machine learning models. The results are shown in Figure 3-(A).

Using Static Analysis. Without packed samples, the static analysis provided adequate results for all models. This experiment shows that MLP achieves good results with 91.3% precision, 87.6% recall, and 89.4% F1 score. RF and KNN achieved similar results. The results of LR and AdaBoost are relatively less than ideal, with the lowest precision, recall and F1 score.

Using Dynamic Analysis. Almost all models achieved better results using dynamic features compared to those achieved by static features. This indicates the ability of dynamic analysis to capture behavioral patterns that accurately detect malware. For instance, we can clearly observe that the best classifier, *i.e.,* RF, in this experiment with dynamic features has a precision of 95.2%, a recall score of 97%, and an F1 score of 96%. Compared with the static features experiment, the performance of RF classifier has improved by 2.52%, 7.68%, and 5.19% for the precision, recall, and F1 scores, respectively.

Using Hybrid Analysis(static+dynamic). Compared to the results of static and dynamic analyses, using features from both analyses enable machine learning models to achieve higher accuracy. RF and DT showed the best results among the seven classifiers with all 95%. KNN and AdaBoost also perform well as their precision, recall, and F1 scores are around 90%. SVC has the highest precision score with 99.1% but with a slightly low recall (74.4%) and F1 score of 85%. The result of MLP is still the lowest, with a precision score of 86.1%, a recall of 75.5%, and an F1 score of 79.6%.

Static

Features	RF	DT	SVM	LR	MLP	KNN	AdaBoost
relocs	●	●			●	●	●
ntdll.dll	●	●	●	●	●		●
shell32.dll	●	●	●		●	●	
comctl32.dll	●	●	●	●	●	●	●
advapi32.dll	●	●					●
user32.dll	●	●			●		●
ole32.dll	●	●					
oleaut32.dll	●	●					
nodes			●	●	●	●	
CC			●	●	●	●	

Dynamic

Features	RF	DT	SVM	LR	MLP	KNN	AdaBoost
apistats	●	●	●		●	●	●
dll_loaded	●	●	●	●		●	●
calls	●	●	●		●	●	●
modules	●	●	●	●	●	●	●
pe_imports		●	●	●	●	●	●
pe_resources	●	●	●	●	●	●	●
networks	●	●	●	●	●	●	●
UDP	●	●	●	●	●	●	●
Hosts	●		●	●	●	●	●
Domains	●	●					●

Hybrid

	Features	RF	DT	SVM	LR	MLP	KNN	AdaBoost
dynamic	pe_resources	●	●		●	●	●	●
	modules	●		●	●	●	●	●
	pe_imports	●		●				
	UDP		●		●	●		●
	calls			●		●	●	
	apistats			●		●	●	
	networks			●	●	●	●	●
	hosts							●
static	dll_loaded	●		●			●	●
	ntdll.dll	●				●		

Figure 2: The result of the feature importance metric, showing the feature importance results of static, dynamic, and hybrid features from left to right. RF, DT, SVM, LR, MLP, KNN and AdaBoost are the acronym of seven machine learning algorithms: Random Forest, Decision Tree, Support Vector Machine, Logistic Regression, Multilayer Perceptron, K-Nearest Neighbors and Adaptive Boosting, respectively. *comctl32.dll* and *ntdll.dll* are among the most important static features, while *UDP*, *modules*, and *networks* are among the most important dynamic features. *Modules* and *pe_resources* are highly-scored for hybrid methods.

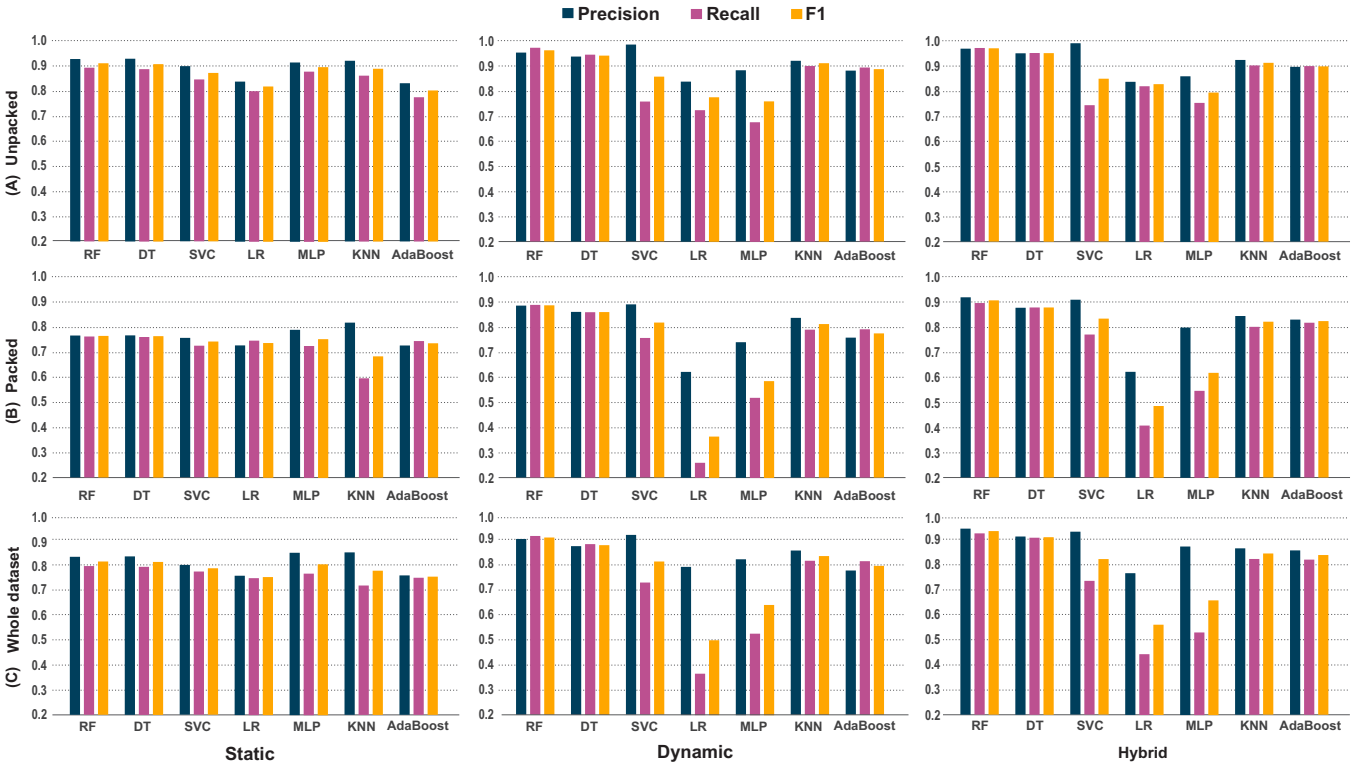

Figure 3: The Experimental results of applying hybrid features, dynamic features, and static features on unpacked dataset (A), packed dataset (B), and the entire dataset (C). Using combination of static and dynamic features improve the performance.

3.2 Experiment 2: Packing-Informed Analysis

This experiment evaluates the features and machine learning methods for detecting malware when only packed software exists. We used a dataset of packed software of benign and malware samples for ten packers. We applied the same feature extraction process for static and dynamic analyses and machine learning techniques. The results are shown in Figure 3-(B). All the experimental results in the packed dataset, *i.e.,* precision, recall, and F1-score, have declined compared to the unpacked dataset. In the process of extracting static features from the packed dataset, we observed that according to each packing tool, the structures of CFGs across samples are highly similar in degree and depth. Therefore, many of the static features were considered less significant. The result of this experiment shows that the packing tools hide many software features, making it difficult for analysts to extract valuable information.

Figure 4: The figure shows the results of applying hybrid, dynamic, and static features on different packers' dataset. The highlighted scores (in a brighter color) are the best results of each packer.

Using Static Analysis. All classifiers have similar precision, recall, and F1 scores with lower values compared to using unpacked samples. For example, the results of RF, DT, SVC, LR, and AdaBoost are all around 73%∼76%.

Using Dynamic Analysis. Using dynamic features, some classifiers achieved an improved performance compared to using static features. For example, RF and DT achieved F1-scores of approximately 85% and 88%. However, LR achieved the worst result, with a recall score of less than 26% and an F1-score of 36.3%.

Using Hybrid Analysis(static+dynamic). Using a combination of both static and dynamic features, all classifiers achieved higher performance (*i.e.,* 2% to 3% higher) compared to those obtained by dynamic features. In this experiments, LR and MLP were the lowest-performing models, with low recall and F1 scores, although the precision was 62.1% and 79.5%, respectively. RF showed the best performance with almost all scores around 90%.

3.3 Experiment 3: Packing-Oblivious Analysis

This experiment aims to evaluate the features and machine learning methods for detecting malware regardless of the knowledge of packing. To this end, we use the whole dataset for extracting features and building machine learning models. The results are shown in Figure 3-(C).

Using Static Analysis. In the experiments where static features were used, all the classifiers achieved detection F1-score in the range of 70% to 82%. Generally, classifiers maintain relatively similar performance. RF and DT are relatively the best classifiers.

Using Dynamic Analysis. In the experiments with dynamic features, RF, DT, KNN, and AdaBoost achieved detection F1-score over 80%, with RF getting the best results. LR and MLP achieved the worst results, where the precision of LR and MLP were 78.9% and 87.1%, respectively, and the recall scores were 36.4% and 52.8% for LR and MLP, respectively.

Using Hybrid Analysis(static+dynamic). When using a combination of features from both static and dynamic analyses, most classifiers, *i.e.,* RF, DT, KNN, and AdaBoost, provided high performance. As you can see in Figure 3-(C), the performance of these four classifiers reached over 80%. RF achieves the best results, precision of 94.2%, recall of 92.3%, and F1 score of 93.2%. On the contrary, the results of LR are not satisfactory, with the precision being 76%, recall and F1 being 44.2% and 56%, respectively.

3.4 Experiment 4: Packing-Specific Analysis

This experiment evaluates different analyses and machine learning methods for detecting packed malware using a specific packing tool. We used separate datasets of benign/malware samples for each of the ten considered packers, *i.e.,* Obsidium, Themida, PECompact, Petite, UPX, kkrunchy, MPRESS, tElock, PELock, and dolphin-dropper-3. The results are shown in Figure 4.

Using Static Analysis. Features such as relocs and imports libraries provide different values, which are considered as important features by several ML classifiers (see Figure 2), while the values of other features are less distinctive across samples. Figure 4 demonstrates the performance of the ML classifier based on the dataset of each packer. Unlike other results, the performance of ML classifiers on the dolphin and telock dataset is low. The low performance could be due to the indistinctive static features produced after packing, including the relocs and imports of libraries.

Using Dynamic Analysis. Among ML classifiers, superior performance can be seen in the results of RF, DT, and AdaBoost models. In the second place, KNN and SVC models performed better, while LR and MLP provided the lowest results.

Using Hybrid Analysis(static+dynamic). As the result of hybrid experiments, RF, DT and AdaBoost were the best models. They got relatively high precision, recall, and F1 scores. As we can observe in Figure 4, RF achieved the highest F1 score for almost all experiments. On the contrary, LR and MLP are not performing well.

4 RELATED WORK

There have been several types of research regarding malware detection based on static, dynamic, and both analyses.

Static analysis. In the static analysis, several features such as opcode sequences, functional call-graph, portable executable (PE) headers, byte sequences, are being extracted for malware detection and classification. In the paper [8], two distinct categories of features (strings and Portable Executable header information) from the APT1 dataset [11] were extracted for six different machine learning classifiers: Support Vector Machine, Logistic Regression, Random Forest, XGBoost, Logistic Regression/XGBoost (Hybrid), and Logistic Regression/Random Forest/Naïve Bayes (Hybrid). Additionally, to operate with those features, twelve malware detectors were implemented for each classifier, and the accuracy and execution time of those detectors were analyzed and compared. However, the experiments are conducted with a few PE header features that need to be taken into account. Roseline *et al.* [17] proposed oblique random forest ensemble learning that is considered a novel approach for malware classification by handling the binary and multiclass problem with a better geometric property. The effectiveness of the approach was provided by conducting experiments to calculate the accuracy and false positive rate on three malware classification datasets: Antivirus, ClaMP (Classification of Malware with PE headers), and Kaggle datasets.

Dynamic analysis. In the dynamic analysis, usually sandbox or virtual machine is utilized to extract information (network activities, system calls, and instruction sequences) about malware in that environment for the performed activity. For this analysis, there are different open-source or commercial sandboxes (cuckoo sandbox [18], Joe sandbox [14], *etc.*). Usman *et al.* [1] proposed a novel hybrid approach including dynamic malware analysis, machine learning, cyber threat intelligence, and data forensics. The purpose is to predict the IP reputation in its pre-acceptance stage and to categorize its associated zero-day attacks using Decision Tree (DT) on behavioral analysis. The effectiveness of their approach is shown in two ways: comparison of machine learning methods based on F1, precision, and recall scores; comparison of their entire reputation system with other existing systems. Even though their approach reduces the false alarm rate to some extent, it will not remove completely. The paper [20] proposed a system using cuckoo-based malware dynamic analysis to analyze malware automatically and quickly, to classify malicious attributes efficiently. The system is based on a semantic deep learning feature model to describe the multi-layered aggregation relationship of program semantics via a deep recursive neural network model and to build a malware semantic aggregation model.

Hybrid analysis. The analysis is a combination of static analysis and dynamic analysis to extract annotated disassembly listings and generate additional information about malware. In the paper [12], static properties and dynamic behavior analysis are used to classify malware on a publicly available source with 987 malicious files and 613 benign samples included. Falcon Sandbox [19] and a virtualized Windows 10 environment are leveraged to extract dynamic data on which three machine learning classifiers (random forests, gradient boosting, and neural networks) are analyzed. Their empirical experiments demonstrated a high F1 score for every model up to 98.9%. Huang *et al.* [15] introduced a malware detection technique that is based on deep learning. In the paper, static visualization images of static features and hybrid visualization images of both static

and dynamic analysis to train two convolutional neural networks (VGG16) are used. The effectiveness of the models on detecting unknown malware is tested on the dataset collected from the free malware package provided by virussign.com. As compared with the static approach, their hybrid approach performs better.

5 CONCLUSION

In this paper, we explored the following questions: ❶ Among the three types of malware analysis approaches, *i.e.,* static, dynamic, or combination of both, what is the most efficient and effective approach for packed malware analysis? ❷ What are the features that are important for machine learning progress? ❸ What are the machine learning techniques that provide consistent performance on malware detection? In the presence of packers, we first observed that static features are not very helpful for malware detection since most packers obfuscate the software. Incorporating both static and dynamic features helps improve the accuracy of packed malware detection. Using hybrid analysis, security analysts can obtain the benefits of both static and dynamic analysis, thus improving the accuracy of unknown malware detection. Observing the feature importance based on machine learning models, this research shows that several dynamic features were found to be significant, including behavioral and network analysis features. Based on our findings, dynamic features contribute more to the proper prediction of malware than static features. Regarding machine learning methods, our experimental results suggest that RF and DT have good performance across multiple datasets. We also observed that many packed malware samples incorporate an anti-debugging scheme to circumvent dynamic analysis. Furthermore, in the process of using CFG to extract static features, we can also observe that some packers produce similar CFGs for packed malware samples through CFG flattening, instruction substitutions, and code injections. Such challenges motivate future research on studying effective methods to study packed malware.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2021R1A2C1011198).

REFERENCES

[1] 2021. Intelligent Dynamic Malware Detection using Machine Learning in IP Reputation for Forensics Data Analytics. *Future Generation Computer Systems* 118 (2021), 124–141. https://doi.org/10.1016/j.future.2021.01.004

[2] Ahmed Abusnaina, Mohammed Abuhamad, Hisham Alasmary, Afsah Anwar, Rhongho Jang, Saeed Salem, Daehun Nyang, and David Mohaisen. 2021. DL-FHMC: Deep Learning-based Fine-grained Hierarchical Learning Approach for Robust Malware Classification. *IEEE Transactions on Dependable and Secure Computing* (2021).

[3] Ahmed Abusnaina, Hisham Alasmary, Mohammed Abuhamad, Saeed Salem, Daehun Nyang, and Mohaisen Aziz. 2019. Subgraph-based Adversarial Examples Against Graph-based IoT Malware Detection Systems. In *Proceedings of the 8th International Conference on Computational Data and Social Networks (CSoNet)*.

[4] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When Malware is Packin'Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. *Network and Distributed Systems Security (NDSS) Symposium 2020*.

[5] Hisham Alasmary, Ahmed Abusnaina, Rhongho Jang, Mohammed Abuhamad, Afsah Anwar, DaeHun Nyang, and David Mohaisen. 2020. Soteria: Detecting adversarial examples in control flow graph-based malware classifiers. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 888–898.

[6] Hisham Alasmary, Afsah Anwar, Ahmed Abusnaina, Abdulrahman Alabduljabbar, Mohammed Abuhamad, An Wang, Dae Hun Nyang, Amro Awad, and David Mohaisen. 2021. SHELLCORE: Automating Malicious IoT Software Detection Using Shell Commands Representation. *IEEE Internet of Things Journal* (2021).

[7] Hyrum S Anderson and Phil Roth. 2018. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).

[8] Neil Balram, George Hsieh, and Christian McFall. 2019. Static Malware Analysis Using Machine Learning Algorithms on APT1 Dataset with String and PE Header Features. *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, 90–95. https://doi.org/10.1109/CSCI49370.2019.00022

[9] Hubert Baniecki, Wojciech Kretowicz, Piotr Piatyszek, Jakub Wisniewski, and Przemyslaw Biecek. 2020. dalex: Responsible Machine Learning with Interactive Explainability and Fairness in Python. *CoRR* abs/2012.14406 (2020). https://arxiv.org/abs/2012.14406

[10] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. 2013. A survey on heuristic malware detection techniques. *The 5th Conference on Information and Knowledge Technology*, 113–120.

[11] Mandiant Intelligence Center. 2013. APT1: Exposing one of China's cyber espionage units. *Mandian. com* (2013).

[12] Rajchada Chanajitt, Bernhard Pfahringer, and Heitor Murilo Gomes. 2021. Combining Static and Dynamic Analysis to Improve Machine Learning-based Malware Classification. *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, 1–10. https://doi.org/10.1109/DSAA53316.2021.9564144

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[14] Joe Sandbox Cloud. 2018. Joe Sandbox. https://www.joesandbox.com/

[15] Xiang Huang, Li Ma, Wenyin Yang, and Yong Zhong. 2021. A method for windows malware detection based on deep learning. *Journal of Signal Processing Systems* 93 (2021), 265–273. Issue 2.

[16] Radare2. 2018. Radare2. http://www.radare.org/r/

[17] S Abijah Roseline and S Geetha. 2018. Intelligent Malware Detection using Oblique Random Forest Paradigm. *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 330–336. https://doi.org/10.1109/ICACCI.2018.8554903

[18] Cuckoo Sandbox. 2010. Cuckoo Sandbox – Automated Malware Analysis. https://www.cuckoosandbox.org/

[19] Falcon Sandbox. 2019. Falcon Sandbox. https://www.hybrid-analysis.com/

[20] Lele Wang, Binqiang Wang, Jiangang Liu, Qiguang Miao, and Jianhui Zhang. 2019. Cuckoo-based malware dynamic analysis. *International Journal of Performability Engineering* 15 (2019), 772. Issue 3.