

LEVERAGING FEATURE ENGINEERING FOR MALWARE DETECTION USING ENSEMBLE LEARNING

BY

RAED KATIB

A Thesis Presented to the
COLLEGE OF GRADUATE AND INTERDISCIPLINARY STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SECURITY & INFORMATION ASSURANCE

MARCH 2024

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **RAED KATIB** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SECURITY & INFORMATION ASSURANCE**.

Thesis Committee

Dr. Ahcene Bounceur (Adviser)

Prof. Mohammed Hammoudeh (Member)

Dr. Farag Azzedin (Member)

Dr. Motaz Alfarraj
Department Chairman

Dr. Suliman Al-Homidan
Dean of Graduate Studies

Date



©Raed Katib
2024

To my family

ACKNOWLEDGMENTS

First and foremost, I would like to thank Allah who gave me the ability and power to work and finish this long research paper for the completion of my master of science degree.

A special thanks to Dr. Danish Vasan who supported me since the first day that I started thinking about a research problem for my thesis. I would like to thank him for his continuous technical and research support. I would also like to thank Prof. Mohammed Hammoudeh and Dr. Ahcene Bounceur for providing me with continuous feedback and directions for the theoretical work in the thesis.

I would like to also thank the coordinator, Dr. Sajjad Mahmoud, for his advice and directions for the thesis process and the plan requirements. I would like to thank the Information and Computer Science department for providing me the opportunity to learn and conduct research under the direction of some of the best professors.

Finally, I would like to thank my family, my parents, and my wife for their continuous support during the research period. Their support allowed me to complete the thesis and finish the journey.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vi
LIST OF TABLES	x
LIST OF FIGURES	xiv
LIST OF ABBREVIATIONS	xvii
ABSTRACT (ENGLISH)	xix
ABSTRACT (ARABIC)	xxi
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	4
1.2 Aim and Objectives	6
1.3 Contributions	7
CHAPTER 2 BACKGROUND	9
2.1 Malware Analysis Methods	9
2.1.1 Static Malware Analysis	10
2.1.2 Dynamic Malware Analysis	11
2.2 Portable Executable Format	12
2.2.1 PE Files Overview	12
2.2.2 PE file Structure	13

CHAPTER 3	LITERATURE REVIEW	17
3.1	Recent Malware Detection Methods	17
3.1.1	Static Analysis-based Malware Detection	18
3.1.2	Dynamic Analysis-based Malware Detection	21
3.1.3	Visualization-based Malware Detection	25
3.2	Malware Evasion Techniques	28
3.2.1	Traditional Obfuscation Methods	28
3.2.2	Machine Learning Evasion Techniques	29
3.3	Limitations of Existing Malware Detection Methods	32
CHAPTER 4	FEATURE ENGINEERING AND ANALYSIS	35
4.1	Selected Features	36
4.2	Features Overview	37
4.3	Features Processing	41
4.4	Features Visualization	43
4.5	Single-Feature Analysis	46
4.6	Information Gain Implementation	49
CHAPTER 5	MLMD-AFE SPECIFICATIONS	51
5.1	MLMD-AFE Overview	51
5.2	Feature Extraction and Preprocessing	54
5.3	Features Engineering and Analysis	55
5.3.1	PCA Visualization	56
5.3.2	Single-Feature set Analysis	58
5.3.3	IG Ranking and Selection	60
5.4	Machine Learning Model Selection	62
5.4.1	Decision Tree	63
5.4.2	Random Forest	63
5.4.3	AdaBoost	64
5.4.4	Gradient Boosting	65
5.4.5	XGBoost	65

5.4.6	SVM	66
CHAPTER 6 EVALUATION AND DISCUSSION		68
6.1	MLMD-AFE Dataset	68
6.2	Evaluation Measures	72
6.3	Experiment Setup	74
6.4	Training, Validation, and Testing	75
6.5	Baseline Results	75
6.6	Performance Evaluation	76
6.7	Computational Overhead Evaluation	79
6.8	Discussion	81
6.9	MLMD-AFE Limitations	82
6.10	General Limitations	83
CHAPTER 7 CONCLUSION AND FUTURE WORK		85
7.1	Conclusion	85
7.2	Future Work	87
APPENDIX A FULL RESULTS OF EACH FEATURE CATEGORY		90
APPENDIX B NUMERICAL RESULTS FOR THE IMPLEMENTA-		
TION OF THE FINAL SELECTED FEATURES		93
APPENDIX C RESULTS INCLUDING BYTE AND BYTE EN-		
TROPY HISTOGRAMS		96
C.1	Byte Histogram Inclusion	96
C.2	Byte Entropy Histogram Inclusion	97
C.3	Byte Histogram and Byte Entropy Histogram Inclusion	101
REFERENCES		104
VITAE		121

LIST OF TABLES

2.1	Sections within a PE file with the description of the content of each section and its purpose.	16
3.1	A comparison of recent literature and our work.	31
3.2	A summary of the reviewed articles with the base features used, the classifier(s), pros, cons, contribution, and limitation of each reviewed article.	34
4.1	Selected Features, the number of features per feature type, and their Descriptions	37
4.2	The best results of running machine learning models for each feature group.	48
4.3	The results of running the SVM model for each feature group.	49
4.4	Threshold values and their associated number of features for each IG implementation.	50
6.1	The baseline results of training the six models on all the extracted features.	76
6.2	The best results achieved when 20% of the features are used.	77
6.3	The number of features per feature set after the final feature selection phase.	78
6.4	The Computational time required for the baseline implementation, the Machine Learning for Malware Detection Based on Advanced Feature Engineering (MLMD-AFE) implementation, and the saved computational time after applying MLMD-AFE.	81

A.1	The results of running the six machine learning models for byte histogram.	90
A.2	The results of running the six machine learning models for byte entropy histogram.	91
A.3	The results of running the six machine learning models for the strings.	91
A.4	The results of running the six machine learning models for general file information.	91
A.5	The results of running the six machine learning models for the headers.	91
A.6	The results of running the six machine learning models for section information.	92
A.7	The results of running the six machine learning models for imported libraries.	92
A.8	The results of running the six machine learning models for exported libraries.	92
A.9	The results of running the six machine learning models for data directories.	92
B.1	The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various Information Gain (IG) thresholds for the final features.	94
B.2	The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the final features.	94
B.3	The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the final features.	94
B.4	The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the final features.	94
B.5	The numerical values for the accuracy, precision, recall, and F1-score of applying the Support Vector Machine (SVM) model to evaluate the various IG thresholds for the final features.	94

B.6	The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the final features.	95
C.1	The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the features with byte histogram.	97
C.2	The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the features with byte histogram.	98
C.3	The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the features with byte histogram.	98
C.4	The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the features with byte histogram.	98
C.5	The numerical values for the accuracy, precision, recall, and F1-score of applying the SVM model to evaluate the various IG thresholds for the features with byte histogram.	98
C.6	The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the features with byte histogram.	99
C.7	The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the features with byte entropy histogram.	100
C.8	The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the features with byte entropy histogram.	100
C.9	The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the features with byte entropy histogram.	100

C.10	The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the features with byte entropy histogram.	100
C.11	The numerical values for the accuracy, precision, recall, and F1-score of applying the SVM model to evaluate the various IG thresholds for the features with byte entropy histogram.	101
C.12	The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the features with byte entropy histogram.	101
C.13	The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram. . . .	102
C.14	The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram. . . .	102
C.15	The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram. . . .	103
C.16	The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.	103
C.17	The numerical values for the accuracy, precision, recall, and F1-score of applying SVM model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.	103
C.18	The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram. . . .	103

LIST OF FIGURES

2.1	The file structure of the 32-bit PE file. The image is obtained from Wikipedia Creative Commons [1].	14
4.1	An example of the features used in JSON format.	39
4.2	The PCA visualization of the four numerical raw features.	45
4.3	The PCA visualization of all the features after processing.	47
5.1	An overview of the three main phases in MLMD-AFE method.	52
5.2	The process followed by the PCA visualization step in feature engineering and analysis.	57
5.3	The single feature set analysis process with each step, the machine learning models used for cross-validation, and the feature complicity information as output.	59
5.4	An overview of the three main phases in MLMD-AFE method.	60
6.1	The steps performed for the MLMD-AFE dataset collection and validation.	69
6.2	A sample VirusTotal results of a malicious PE sample.	71
6.3	The results of applying IG using the six machine learning models with different percentages of features used at each time.	77
C.1	The results of applying IG using the six machine learning models with different percentages of features used at each time including the byte histogram.	97

C.2	The results of applying IG using the six machine learning models with different percentages of features used at each time including the byte entropy histogram.	99
C.3	The results of applying IG using the six machine learning models with different percentages of features used at each time including the byte and byte entropy histograms.	102

LIST OF ABBREVIATIONS

API	Application Programming Interfaces
AutoML	Automated Machine Learning
Bi-LSTM	Bidirectional Long Short-Term Memory
BoWs	Bag of Words
CFGs	Control Flow Graphs
CNN	Convolutional Neural Network
COFF	Common Object File Format
DLLs	Dynamic Link Libraries
DNN	Deep Neural Network
DQN	Deep Q-Network
ELF	Executable Linkable Format
FCG	Function Call Graph
GANs	Generative Adversarial Networks
GCN	Graph Convolution Network
GNN	Graph Neural Network
IG	Information Gain

IoT	Internet of Things
IPT	Intel Processor Trace
IRL	Improved Reinforcement Learning
KNN	K-Nearest Neighbor
MLMD-AFE	Machine Learning for Malware Detection Based on Advanced Feature Engineering
OpCode	Operational Code
PCA	Principal Component Analysis
PE	Portable Executable
RFE	Recursive Feature Elimination
RL	Reinforcement Learning
RRF	Regularized Random Forest
SAFE	Self-Attentive Function Embeddings
SVM	Support Vector Machine

THESIS ABSTRACT

NAME: Raed Katib

TITLE OF STUDY: Leveraging Feature Engineering for Malware Detection
Using Ensemble Learning

MAJOR FIELD: Security & Information Assurance

DATE OF DEGREE: March 2024

Malware variants pose a persistent challenge for machine learning classifiers. Existing methods used two main approaches: rely on a single feature or employ many features. This thesis proposes Machine Learning for Malware Detection Based on Advanced Feature Engineering (MLMD-AFE), an approach that integrates advanced feature engineering and optimized ensemble learning models to detect malware variants in Windows. A feature engineering approach that analyzes and selects the best features for malware detection is introduced. A recent malware dataset is collected and analyzed to evaluate MLMD-AFE thoroughly. The application of MLMD-AFE to well-known feature sets showed that the number of features could be reduced by 80% while maintaining the same accuracy of 99.09% and F1-Score of 99.06%. MLMD-AFE showed its effectiveness in reducing the number of features, optimizing the learning

process, and minimizing the computational overhead in malware detection.

ملخص الرسالة

الاسم: رائد فاروق كاتب

عنوان الدراسة: الاستفادة من هندسة الميزات لاكتشاف البرامج الضارة باستخدام التعلم المجمع

التخصص: الأمن وتأمين المعلومات

تاريخ الدرجة العلمية: مارس 2024

تشكل متغيرات البرامج الضارة تحديًا مستمرًا لمصنفات التعلم الآلي. تستخدم الأساليب الحالية طريقتين رئيسيتين: الاعتماد على ميزة واحدة أو استخدام العديد من الميزات. تقترح هذه الأطروحة التعلم الآلي المتقدم لاكتشاف البرامج الضارة استنادًا إلى هندسة الميزات المتقدمة (MLMD-AFE)، وهو نهج يدمج هندسة الميزات ونماذج التعلم المجموعة المحسنة لاكتشاف متغيرات البرامج الضارة في Windows. تم تقديم أسلوب هندسة الميزات الذي يقوم بتحليل واختيار أفضل الميزات لاكتشاف البرامج الضارة. يتم جمع مجموعة بيانات حديثة من البرامج الضارة وتحليلها لتقييم MLMD-AFE بدقة. أظهر تطبيق MLMD-AFE على مجموعات الميزات المعروفة أنه يمكن تقليل عدد الميزات بنسبة 80% مع الحفاظ على نفس الدقة البالغة 99.09% ودرجة F1 البالغة 99.06%. أظهر MLMD-AFE فعاليته في تقليل عدد الميزات، وتحسين عملية التعلم، وتقليل العبء الحسابي في اكتشاف البرامج الضارة.

CHAPTER 1

INTRODUCTION

In the evolving landscape of digital communication, the Internet facilitated seamless information exchange to support diverse objectives. That evolution made the Internet an indispensable medium for information exchange. The amount of information collected and used on the Internet has exponentially grown and become a crucial high-value asset. Unfortunately, the allure value of information assets made them a compulsive target for cybercriminals exposing computers and networks to various cyberspace threats. The spectrum of cybercriminal activities, ranging from simple scam emails to sophisticated advanced malware, dramatically extended. Infiltrating through the file system, malware might take the form of deceptive files that exploit a system upon execution. Malware might masquerade as benign files obtained from a variety of sources, ranging from a public website on the Internet to an attachment in an email. The number of malware-based attacks that occurred in 2022 is estimated to be 5.5 billion [2]. Notably, the global landscape is exposed to 1 billion variations of malware [3]. The number is expected to increase with the continuous development of variations and signature changes.

Recognizing the substantial risk posed by malware across cyberspace, advanced malware detection tools and methodologies have emerged. Within the industry, malware detection primarily relies on antivirus or antimalware software i.e., McAfee [4] and advanced firewalls i.e., Fortinet [5]. However, the sophistication of advanced malware coerced researchers to explore alternative detection techniques. Particularly, machine learning and deep learning-based classification have gained prominence in malware classification. The utilization of distinctive features extracted from applications to distinguish between malware and benign software gained ground [6], [7], [8]. Leveraging Deep learning models based on the distinctive features of the software has been proven to detect unseen malware variants more efficiently compared to traditional antimalware methods.

Various features have evolved and proved their effectiveness in the detection of different malware variants. Features such as byte sequence [9], Portable Executable (PE) images [10], and other file information have been used to detect and classify malware. Feature extraction from applications for malware detection can be performed using two approaches: static analysis and dynamic analysis. Static analysis extracts the features from the application without execution based on the file information, headers, and source code [11]. Conversely, dynamic analysis obtains the features from the application by running it within a controlled and sandboxed environment [12]. The extracted features serve as the foundation for constructing a machine learning model for malware detection.

Despite the development of malware detection machine learning models, the emergence of new variants and advanced obfuscation techniques poses a persistent chal-

lenge. Innovative obfuscation techniques [13] are employed to modify the code while preserving the malware functionality to elude the existing malware detection models. Obfuscation techniques are used to conceal the code to evade the detection. Metamorphic obfuscation, for example, [14] modifies the code itself without encryption. Whereas polymorphic obfuscation [15] applies encryption and data appending. Obfuscation techniques are used widely for the development of new malware variants that have emerged recently. Aside from obfuscation, advanced adversarial techniques are designed explicitly to target and evade deep learning detection models [16], [17], [18], [19]. These adversarial techniques, such as the FeaGAN [20] method that combines Generative Adversarial Networks (GANs) with Reinforcement Learning (RL), proved their effectiveness in bypassing the current deep learning models, achieving a high success rate in evasion.

In this thesis, Ensemble Learning for Malware Detection Based on Feature Engineering (MLMD-AFE) is proposed as a malware detection and classification approach that integrates an advanced feature engineering and analysis process with optimized machine learning models. Initially, a comprehensive review of existing methodologies utilized in malware detection tailored explicitly for Windows systems is conducted. Recognizing the need for better distinctive features for the detection of new malware as the current models are evaded by new variants and evasion techniques, the research focuses on the application of advanced feature engineering techniques on the feature vectors extracted from executable malware binaries. The feature engineering step is crucial for the augmentation and refinement of the features used in the classification model design. Subsequently, ensemble learning models that optimize the learning

process of the selected features are designed. This optimization is achieved based on the feature vectors obtained through the MLMD-AFE feature engineering process. To evaluate MLMD-AFE, a dataset on the most recent malware and benign files available is constructed, ensuring that MLMD-AFE model is trained on the latest advancements in malware. Finally, different machine learning models are selected, constructed, and evaluated based on the advanced feature engineering and analysis outcomes for the optimization of malware detection.

1.1 Motivation

Malware (Malicious Software) is a broad term used to describe software that exploits the device to perform malicious actions. It affects the integrity, the confidentiality, and the functionality of the attacked system. Malware has myriad classes based on its functionality [21], [22], intention, and target i.e., Viruses [23], Worms [24], Trojans, and Ransomware [25]. Variants of malware from the same class are further divided into families [26]. Malware coders exploit obfuscation techniques to develop variants of their malware to evade detection [27]. A malware that performs system deletion, for example, can be rewritten in hundreds of variants with different signatures. Malware variants pose a challenge to the current antimalware solutions as new variants render them incapable of keeping track of new signatures [28], [29].

Considering malware advancement and the development of new variants, the current malware detection methods based on machine learning require further improvements. The selection of a proper set of features that is resilient against evasion methods

deployed by the new malware variants. Refining feature engineering and constructing a machine learning model requires focusing on the analysis of specific features that exhibit characteristics intrinsic to the file. Malware is written to attack certain CPU architectures or operating systems such as Windows or Linux. Windows and Linux use different binaries for application execution where PE and Executable Linkable Format (ELF) formats are used, respectively. Features extracted from different formats differ. Thus, the selection of features tailored toward a specific system would lead to better performance in the detection. In this work, the focus is on the features that could be extracted from Windows applications since it is the most used operating system. Furthermore, this research focuses on the use of static PE file features as the static analysis was proven to be more efficient, safer, and faster in the detection of malware [30].

The prime challenge for the construction of a resilient machine learning model for malware detection is the selection of features that resist adversarial modification attempts. The current approaches in machine learning model construction exhibit two predominant strategies: rely on a single feature for model construction [31] or employ a plethora of features that can be extracted from applications without proper feature engineering and selection [32]. The single-feature-based models have proven to be vulnerable as adversaries focus on the modification of specific application elements to evade the model. Conversely, the multi-feature models are susceptible to the noise generated by the less critical features used in the model construction. The majority of the existing research focused on the development of advanced deep learning models while less attention was given to feature engineering [31], [33]. On the other hand,

other researchers focused on feature engineering and did not design a strong detection and classification model [34], [35]. Constructing a resilient deep learning model capable of withstanding adversarial attacks demands efforts in both feature engineering and the construction of the deep learning model. MLMD-AFE proposed in this thesis will address the limitations within existing methods that make use of all the possible features and improve their performance.

1.2 Aim and Objectives

The aim of the study is to advance the current malware detection techniques for executable files within the Windows system. This aim can be achieved by designing a machine learning classifier that optimizes the learning of the features chosen using the advanced feature engineering process in MLMD-AFE.

The following are the objectives of our research:

1. Review and critically analyze existing malware detection techniques, the current features used for malware detection, and the adversarial attacks against the current methods used in Windows systems available in the literature.
2. Construct a balanced dataset with samples enough for generalization with the latest advancements in malware that target Windows systems and a set of benign executables based on Windows 11 executables.
3. Propose an advanced feature engineering approach that includes feature extraction, feature selection, normalization, and feature analysis that evaluates each

feature deeply to select the most effective features for a given set of features.

4. Evaluate and analyze the selected features based on the effectiveness and computational overhead in the detection of malware on Windows Systems
5. Train and test six machine learning models based on the feature engineering process results.
6. Achieve a higher detection rate, or at least maintain the same rate, while reducing the number of features used and the computational overhead of the feature extraction, feature processing, and model training.

1.3 Contributions

MLMD-AFE is expected to improve the detection of malware executable files in Windows Systems by the integration of efficient feature engineering and machine learning model design. Overall, the expected contribution from this research is the following:

1. A review of the most recent literature for PE format, Windows malware detection, and Adversarial attacks against malware detection deep learning models.
2. A balanced PE dataset with the latest advancements in malware that target Windows systems and a set of benign executables based on Windows 11.
3. An advanced feature engineering and analysis approach that can be used to perform feature analysis and selection.

4. A set of efficiently engineered features that effectively distinguish malware and benign executables in Windows systems based on the combined characteristics of each feature.
5. A machine learning model that optimizes the use of the selected features, detects malware effectively and is resilient against the latest malware variants released recently.

The rest of the thesis is organized as follows: Chapter 2 presents the background required to understand the work conducted in the thesis. Chapter 3 presents the most recent malware detection methods used in the literature and the evasion techniques and methods employed by adversaries to evade the detection of Windows-based PE files. Chapter 4 presents the feature engineering and preprocessing of features within this thesis. Chapter 5 presents MLMD-AFE, the proposed method, and its technical specifications for performing feature engineering and analysis, feature selection, and machine learning model construction in detail. Chapter 6 presents the experimental evaluation of the designed method, its implementation, its results, and its limitations. Finally, Chapter 7 concludes and summarizes the work done in this thesis and eventually presents future directions and suggestions.

CHAPTER 2

BACKGROUND

In this chapter, the technical background of the terms and methods used in this thesis is explained. This chapter aims to establish a common ground to understand the rest of the thesis thoroughly. First, malware analysis methods are explained. Both static analysis and dynamic analysis are introduced with examples. Second, this thesis focuses on the detection of PE malware. Understanding the PE format and the information included within executable files is essential for understanding this work. Thus, the PE file format is explained.

2.1 Malware Analysis Methods

Malware analysis is the process of deeply examining the functionality, behavior, attributes, and impact of malware files on the computer or network. The goal of performing malware analysis is to identify the malware characteristics and impact to take actions that would protect computers and networks. Performing malware analysis can be done in two main methods: static and dynamic analysis. Malware analysis serves

as the basis for many important tasks such as malware detection based on the static or dynamic analysis of files.

Malware detection can be achieved either through static or dynamic analysis. Features are extracted from the Windows malware files either using static or dynamic analysis for the detection of malware. In general, Windows files contain information that can be extracted and used as features for the malware classification problem such as PE information and headers [36], Application Programming Interfaces (API) and System calls [37], [38], n-gram bytes [9], Operational Code (OpCode) [39], and others [40], [41].

Extracted features can be used in raw format to train classifiers, or they can be analyzed, reduced, reshaped, or presented in a different format such as images to train different classifiers [42]. Raw features can be transformed into statistical [43], structural [44], and other forms that combine multiple processed features. In essence, features processing and transformation's main goal is to improve the detection and classification capabilities of malware. Many researchers conducted feature selection and extraction to build malware classifiers. In the next subsections, static analysis and dynamic analysis are briefly introduced.

2.1.1 Static Malware Analysis

Static analysis is the process of analyzing the code without executing any instructions. It can be used to examine the source code, the file information, the file headers and format information, and any additional information that can be obtained without execution [45]. Static analysis can be applied for many purposes such as the detection

of vulnerabilities, assessing the quality of code, or the identification of certain characteristics that would be used for malware detection. It can be used as a security tool, performance optimization tool, and quality assurance tool. This work will examine the application of static analysis for malware files.

Static malware analysis is the process of applying static analysis for malware files to identify their characteristics, behavior, expected harm, and the best responsive action for the analyzed malware [46]. In static malware analysis, certain file characteristics that are obtained without code execution are associated with malware. The analysis of such characteristics would help in the creation of malware detection models. Some of the advantages of applying static malware analysis are the ability to scale for a huge number of malware samples easily, the high level of protection since unknown code will not be executed, and the early detection of malware [46].

2.1.2 Dynamic Malware Analysis

Unlike static analysis, dynamic analysis involves the execution of a program while monitoring and examining the behavior and the changes performed by the program to understand its functionality. Similar to static analysis, dynamic analysis can be used to find vulnerabilities, detect software errors, and highlight performance issues [47]. In addition, dynamic analysis helps in identifying runtime errors that cannot be easily detected by static analysis. In some cases, the static analysis of a program would not show any issues, but the dynamic analysis would show that there is an unexpected behavior during the run time. This thesis will examine the application of dynamic analysis for malware detection.

Dynamic malware analysis is the process of executing a malware sample to monitor its behavior and the damage it causes to the system. Intuitively, dynamic malware analysis is performed within an isolated environment where the examined malware would not cause device corruption or the spread of malware across the network. Dynamic malware analysis examines certain characteristics obtained by monitoring the behavior of the malware or the changes to the system that can distinguish malware from benign files [48]. One of the advantages of applying dynamic analysis is that obfuscated and new variants of malware would not bypass the detection as in the case of static analysis. Unlike static characteristics, the dynamic characteristics of files are hard to temper as the functionality of the file would change.

2.2 Portable Executable Format

2.2.1 PE Files Overview

The PE file format is the prominent format used for Windows files. It includes the information needed by the Windows Operating system to understand the requirements needed for the execution of the file [10]. The PE file format presents many files including the Dynamic Link Libraries (DLLs) files, executable Windows files, object files, and other Windows-based files. DLLs are files that hold codes needed for the execution of programs in the Windows System. DLLs are called by other programs in order to support their execution. Generally, the code included in DLLs is frequently used to support the execution of many programs. Thus, these files include executable codes that contain instructions for the system operation. It was included in its own format

to enhance the system performance and reduce memory usage as many programs use it simultaneously. In addition, DLLs include systems libraries that are essential for the operating system to work.

Exetables are programs utilized in Windows. executable files have the .exe file extension. It includes a code that is transformed into a format ready for execution by the Windows system. These files are created by compiling source code written using other high-level programming languages such as C++ or Java. Then, files are stored in the binary format and are ready for execution. executable files are compiled to operate in Windows-based systems only; they would not run if were used in another operating system. The other format, object code, includes machine code and the metadata from a source code. Object files cannot be directly executed. It requires additional steps before the execution. Nonetheless, all the described files would share the need for the PE file headers to provide the system with execution instructions.

2.2.2 PE file Structure

PE format includes a high number of headers arranged in a certain order followed by additional sections as shown in Figure 2.1. The figure presents the full structure of the 32-bit PE fike. It starts with the DOS Header, the Common Object File Format (COFF) header, and then an optional header with additional information. First, the DOS header is a common header that exists at the beginning of the PE file. The latest PE specifications by Windows no longer support the DOS header, but it might be used for backward compatibility [49].

Second, the COFF is a format that existed before the creation of the PE format.

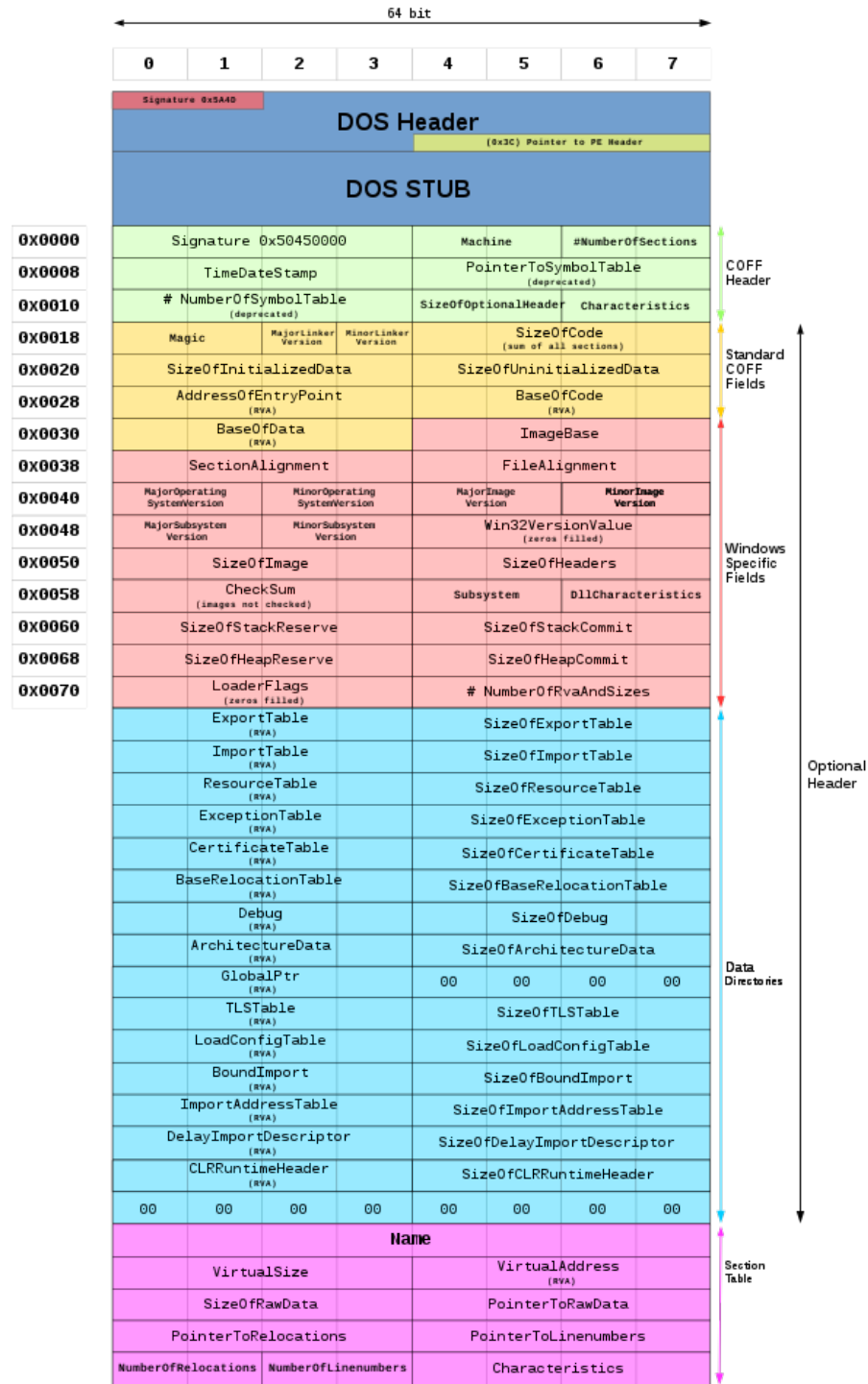


Figure 2.1: The file structure of the 32-bit PE file. The image is obtained from Wikipedia Creative Commons [1].

It was based on the very old computers in the 1970s [50]. This header includes information needed for understanding the remaining structure of the file. It starts with a signature that identifies the file type. In this thesis, this signature will always refer to the PE format. After the signature, the machine type for the execution of the file is identified. Since the PE format targets Windows, the machine field provides information about the instruction set architecture (x86, x64, ARM, etc.), not the operating system.

Other variables include the number of sections, timestamps, pointers to tables, number of symbol tables, size of the optional header, and the characteristics flag. The number of sections describes the number of sections related to the PE file, while the timestamp includes information about the creation or compilation date. The symbol pointer and table include information related to the variables defined or referenced within the PE file. The size of the optional header field contains information about the next header in the PE format. Lastly, the characteristics flag specifies the attributes of the file. After the COFF header, the PE format contains an optional file header called the optional header. The optional header includes information that extends the COFF header. Information about the subsystem, the DLLs imported by the file, the linker version, and additional supporting fields are found in the optional header section.

After the PE headers, the PE file format includes the sections of the file. Sections are part of the PE files that include data or code that can be mapped and executed using the Windows loader. Many sections can be found in the PE format that are summarized in Table 2.1. The sections are considered the body of the file which in-

Table 2.1: Sections within a PE file with the description of the content of each section and its purpose.

Section	Description
.txt	Contains executable code that is executed by the file.
.data	Contains initialized global and static variables used by the executable file.
.bss	Contains uninitialized global and static variables.
.rsrc	Contains resources such as images, strings, and others used by the file.
.idata	Contains information about the imported functions.
.edata	Contains information about the exported functions.
.reloc	Contains relocation information that can be used to adjust memory allocation if needed.
.debug	Contains debugging information.

cludes all the instructions, references to both initialized and uninitialized static and global variables, resources utilized by the executable such as images and strings, information about the imported and explored functions, and relocation information. The relocation information describes how the memory can be adjusted when the file is executed if needed. Lastly, the debug section contains information related to debugging the executable file.

In summary, the PE format consists of a collection of headers and sections that are combined to form an executable file that can run on Windows systems. The headers contain information that guides the operating system for handling the file, while the sections contain the code and additional data and resources utilized by the file when executed.

CHAPTER 3

LITERATURE REVIEW

In this chapter, a review of the malware detection techniques utilized for Windows is performed. Recent methods were reviewed and critically analyzed. After the detection models review, the evasion techniques used against the existing detection methods to evade the detection in the literature are presented. Finally, The strengths and weaknesses of the current methods are highlighted in a comparison table. Section 3.3 presents the limitations of existing methods and highlights the contribution of this thesis and its role in addressing the limitations.

3.1 Recent Malware Detection Methods

This section focuses on reviewing the latest malware detection methods available in the literature. In Section 3.1, the analysis methods that can be utilized for malware detection were described. This section is categorized according to the analysis methods. First, the static analysis-based malware classifiers are reviewed. The methods that are executed without running the file are described. Second, the dynamic analysis-

based features are described. Dynamic analysis-based features focus on the detection of malware by monitoring the runtime behavior. Additionally, a visualization-based malware detection section is included. This section utilizes dynamic or static features for the detection. However, it does not require any domain expertise or deep understanding of the malware detection context. The main goal of the review is to identify the current state-of-the-art features used for malware detection in Windows systems and identify the limitations within the existing models.

3.1.1 Static Analysis-based Malware Detection

Static features are extracted from the malware executable or binary code without executing it. As the code would not be executed, static features can be obtained from the source code and the PE format information in Windows systems. Raw features obtained can be further processed to make useful features. It includes instruction-level features like n-grams and OpCode, statistical features such as frequencies and skewness, Structural features such as Control Flow Graphs (CFGs) or execution paths, and semantic features such as strands and embedding [51]. Various researchers have developed methods based on static features for malware classification.

PE format includes several headers with content that can be used as raw features for malware classification. Several studies utilized PE headers for the extraction and selection of static features. Rizvi et al. [32] proposed PROUD-MAL that utilizes an unsupervised malware classification approach based on the executable file format header information. They parsed the PEs and extracted 35 unique features based on the Information contained in the PE only. Although their method was effective in the

detection, the feature selection process based on the attention block did not reveal feature importance. Their work could be further optimized with a white-box feature selection technique prior to the application of the attention block technique.

Jurecek and Lorencz [52] utilized the distance metric for the development of an improved K-Nearest Neighbor (KNN) malware detection model. Their idea is to utilize the distance metric for selecting the optimal feature set from the PE file metadata. The features were extracted from the PE headers and sections. After computing the distance metrics for the features, KNN was utilized for the classification. Their work has shown good performance in the detection. However, their method lacks the application of all the possible optimization criteria they specified.

Brown et al. [53] leveraged three major static analysis-based datasets for the application of Automated Machine Learning (AutoML) for malware detection. Their focus was on evaluating and comparing the performance of AutoML with the existing deep learning methods for malware detection based on static analysis features. AutoML is an automated machine learning model that optimizes the performance of a model given a set of features to achieve the highest possible results. Although the results achieved using AutoML are better, the application of AutoML lacks the analysis and the deep understanding of the features.

Srastika et al. [35] used the Burota method to reduce and select optimal features from PE headers. Their work included 27 features selected for the analysis. Burota feature ranking and selection was applied to the 27 features where only the top 20 features were utilized for obtaining the best performance. The top 20 features were then used for the evaluation of multi-year datasets, where the performance of the

proposed method is downgraded with newly released malware variants. Kamboj et al. [54] focused on the detection of malware files downloaded from the internet by analyzing static features such as the optional header, the load configuration size, and other unspecified features. Their work focused on testing various machine learning models on a set of selected features. Their feature selection and analysis methods are not described. Only the full number of features and the number of features used in documented.

Another feature that was used extensively in malware detection is OpCode. OpCode presents instructions in the .txt section of the source code of the file. It includes all the instructions that a program performs when used. Jeon and Moon [33] proposed OpCode sequences from binary executables as features. They defined a fixed number of Opcodes with specific lengths from each file. For each OpCode sequence vector, the used classifier classifies it solely where the sum of classifications is used for the final classification. They used two classifiers for the classification. An opcode Convolutional autoencoder was used at the OpCode level, while a recurrent neural network was used for the final classification. Nevertheless, their method can be easily evaded since a single feature is used. Other recent work that utilized OpCode combined it with other static features.

Bokolo et al. [39] extracted bytecode frequency from the byte sequences, OpCode sequences, and section frequency and integrated them as static features for malware detection. Bag of Words (BoWs) were then utilized to generate a set of 256 features, where the frequency of occurrence for each feature is counted for all the samples. Bokolo et al. [39] tested the generated feature vector using eight machine and deep

learning classifiers. The results in their work were not high compared to other malware detection models. Moreover, the authors did not perform proper feature selection and analysis before designing the model.

Park et al. [55] extracted the byte sequences from the raw binaries of PE files integrated with additional file content-based features. Features such as Byte Histograms, Byte Entropy Histograms, Strings, and others were utilized in addition to the raw byte sequences. The raw sequences were transformed into wavelets using a novel wavelet transformation method designed to process and select the proper features. In essence, the wavelet coefficients, statistical features from the wavelet transforms, additional statistical features, and the file content-based features were merged into a large feature vector utilized for general malware detection. Training the extracted feature vector using machine learning showed its effectiveness in the detection of malware. However, this method focused on cross-platform detection, achieving lower results than other Windows-based methods. A customized Windows-based malware detection would provide better results.

3.1.2 Dynamic Analysis-based Malware Detection

Similar to Static analysis, dynamic analysis can be conducted to extract dynamic features from the malware. Dynamic features refer to the features obtained from the malware by observing the application behavior after running it within a controlled environment [12]. Dynamic features might include registry keys, API calls, system calls, and other malware traces [51]. Since dynamic features require the execution of the code, it is more difficult to obtain and apply at a large scale compared to static

features. Thus, less work was done for malware classification using dynamic features. Nevertheless, many researchers have used dynamic features for malware detection recently.

Chen et al. [56] proposed graph-based features where applications are reverse-engineered, and functions are collected to build a function call graph. On top of the generated graphs, their method extracts the embedding for each collected function using the NUM embedding, the Asm2Vec, and Self-Attentive Function Embeddings (SAFE) forming Function Call Graph (FCG) vectors. Their implementation of FCGs included all the extracted API calls, including less useful ones as features. Similarly, Kargarnovin et al. [31] used the presentation of executable files to detect Windows malware. They extracted API calls and strings from executable files and utilized the BoWs approach to generate 1000-length feature vectors that contain API calls and string sentences. They also tested deep learning classifier evasion techniques. Both Chen et al. [56] and Kargarnovin et al. [31] tested their features using a Graph Convolution Network (GCN) and did not perform feature analysis or reduction.

Chen et al. [44] used parameter-augmented API call sequences that combine raw API calls with executable embeddings as features for the classification. Their main focus was the development of CruParamer, a Deep Neural Network (DNN) model that optimizes the learning of the selected features. They performed ensemble learning by combining Text-Convolutional Neural Network (CNN) with CruParamer to optimize the detection. Nevertheless, they used an old dataset without further analysis of the selected features. Anand et al. [34] performed a complete review and analysis of API call sequences as features for Windows-based ransomware. They applied four

different feature reduction techniques including Boruta, Regularized Random Forest (RRF), Recursive Feature Elimination (RFE), and Genetic Algorithm to analyze and select optimal API calls as features. Although their analysis is comprehensive, their work is limited to feature analysis with weak classification and specific malware class scope.

Amer and Zelinka [57] focused on creating a method that reduced the API call sequences to generate smaller API features that are more distinctive. The purpose of their feature generation technique is to reduce the long API call sequences to a much smaller feature without the additional noise caused by the removed parts. The reduction method they utilized focused on finding a correlation between the API call sequences and the malware sequence. Based on their assumption, Amer and Zelinka [57] used a word embedding technique that takes the API sequences and generates an embedding model. Essentially, a similarity matrix is generated based on the embedding models that would be used for creating benign and malicious feature clusters. The word embedding model was limited due to the need for extensive computational power to generate the embedding models, train a model, and test the model

Another example of API calls used is the Javaheri et al. [58] method. They extracted sequences of API and system calls at the kernel level for Windows. Their focus was on addressing the most recent metamorphic malware generation methods by deceiving the malware in a sandboxed environment and extracting the features while it runs. The extracted API calls are then integrated with the API calls within the PE header to make a full API dataset. Metamorphic malware is generated using the Genetic Algorithm. The proposed model was tested on DNN, J48, and JRip.

Nevertheless, their method is difficult to adopt in real and time-efficient scenarios and its accuracy is not high enough.

Li et al. [59] highlighted that the majority of API calls-based malware detection models focus on the API names rather than the semantic meaning of API calls based on the API attributes. They designed a feature generation method that integrates the API names and arguments while providing a semantic meaning presented in a graph of API call sequences. Transforming the API sequences into graphs provides semantic information based on the structural information of each graph. The proposed method was evaluated using a Graph Neural Network (GNN) model achieving good results. The limitation of this method is that it requires huge computational power for each step.

Similar to Li et al. [59], Zhang et al. [60] utilized API call names and arguments as features for malware detection. The main goal of their research is to apply feature engineering and learning properly to the extracted API names and arguments for an optimized detection rate. In the proposed model, the hashing trick method was utilized to encode the extracted features into a fixed length and unique feature vector for all the samples. The name of the API call, the category, and its arguments were included in a long feature vector with 102 features included. The engineered feature vector was then fed to train multiple CNN models for the generation of highly distinctive features for malware detection. Then, a Bidirectional Long Short-Term Memory (Bi-LSTM) is evaluated and compared with existing methods. The major limitation of this method is that it requires deep analysis of dynamic features that might pose a huge risk in the case of analyzing unknown malware variants.

Qiang et al. [61] examined the API calls-based dynamic malware detection methods and highlighted that API calls are not robust against obfuscation methods. Alternatively, they proposed the development of classifiers based on the control flow traces. Qiang et al. [61] utilized Intel Processor Trace (IPT) for capturing control flow traces at low hardware level. The captured packets were decoded, transformed into behavioral byte sequences, and embedded into vectors. The vectors were used to design a deep learning model that combines CNN and Bi-LSTM while optimizing the learning process. The major limitation of this approach is that the detection rate on unseen malware samples is low.

3.1.3 Visualization-based Malware Detection

Visualization-based malware detection relies on static analysis or the transformation of dynamic features into static features as base features. The base features are then transformed into visualized representations that would be distinctive for each file. Visualization-based approaches might perform additional preprocessing techniques on the features before transforming them into images. However, the general usage of this type of feature does not include preprocessing. The features are transformed into images without the need for domain expertise to design the detection model. Understanding how to solve image-based problems is sufficient for applying visualization-based malware detection. For this reason, visualization techniques were separated from the static and dynamic analysis categories.

Researchers transformed different features into images to perform malware detection. Shaukat et al. [62] proposed a model that transforms PE file information into

images within the RGB channel. They utilize transfer learning based only on visualized static features to extract deep features and fine-tune a deep learning model. In order to generate the images, they transformed the PE files within the dataset to raw byte binaries. The raw byte binaries were used without any modifications or analysis to generate the images utilized in their work. Moreover, API call sequences and OpCodes were also extracted and utilized to generate the other dimensions of the images without requiring any analysis or feature engineering. For this reason, Shaukat et al. [62] transfer learning method contains weakly correlated features that would lead to weaker model performance as the images would contain a high amount of noise.

Shaukat et al. [63] utilized visualization-based features based on static analysis to create and optimize a deep learning model for malware detection. Their proposed solution transformed the PE file into raw byte sequences that are then clustered into a group of 8 bits. The 8-bit group is then transformed into one pixel with the three-dimensional RGB image. These images are then used as features by applying a transfer learning approach based on a customized deep learning model that would compute new 1-dimensional feature vectors. Thus, the deep learning model is only used for the extraction of features, not the detection. Finally, the extracted features are fed into a 1D SVM model that would make the final prediction. This approach has many limitations. First, the computation of the model requires high power consumption. Second, although the author claims that the model would not require domain expertise in security, it requires highly skilled researchers in the application of image-based problems.

Huang et al. [64] proposed two methods that visualize PE files into images that

would not require domain expertise. Namely, they proposed static-based and hybrid-based visualization techniques. For the static visualization, the byte frequency information was integrated with the application’s general Information. They visualized the general application’s Information from the PE header within the blue channel, the byte information frequency within the red channel, and pad the green channel with zeroes in the RGB spectrum. In their second visualization method, the static visualization approach into one image with a dynamic-based visualization approach. Their dynamic approach worked by capturing API call sequences from malware by running them inside a sandboxed environment. Essentially, they applied transfer learning using the VGG16 network. Their experiment was on a small, unbalanced dataset and could easily be evaded as it relies on two easy-to-temper features.

Kumar and Jenet [65] designed a method that detects malware based on the textural features extracted from grey-scale images generated from the malware raw binaries. Thus, this method fully transforms the malware detection problem into an image-based problem where descriptors are utilized to extract features. The method utilized three local descriptors and one global descriptor for the extraction of the textural features. After textural features extraction, a hybrid model that integrates machine learning with CNN for training and testing. Their work has shown to be highly effective in the detection of obfuscated malware. Nevertheless, the features used were weak and could be easily modified as grey-scale images contain much less information compared to the other existing features.

3.2 Malware Evasion Techniques

In this Section, A review of the evasion techniques used against malware detection methods in the literature is presented. First, the traditional obfuscation techniques are described. Techniques such as packing, metamorphic obfuscation, and polymorphic obfuscation are described first. Second, advanced evasion techniques that modify files to evade the detection of machine learning-based models are presented. The goal of this section is to present the threats to the current detection methods and highlight the need for better detection models. Thus, this section provides a summary of some well-known and the latest evasion techniques. It will not include a deep explanation of the technical specifications, the implementation requirements, and explanations for the model operation. Models are listed with only the general technique used with the results of applying them for malware evasion.

3.2.1 Traditional Obfuscation Methods

Evasion techniques are the techniques used by adversaries to evade the detection of their malware by the existing security measures. Evasion can be achieved using several methods that modify the content or headers of the malicious file. One popular evasion technique is packing. Packing aims to hide the portable executable header information to avoid detection by static feature-based methods [58]. There are many types of packers available for use as open-source or commercial products such as Petite [66], UPX [67], ASPack [68], Yoda [69], Detect It Easy [70], and ExeSealth [71]. Recent research considered these packers and optimized methods for the detection of similar

malware [72]. Nevertheless, more advanced techniques have been developed to evade the detection of current machine and deep learning detection models.

3.2.2 Machine Learning Evasion Techniques

Several studies reviewed the existing methods that focus on Windows malware evasion techniques. Galloro et al. [73] studied the statistics of the past ten years for malware evasion techniques evolution and showed that evasive samples are increasing over time. Moreover, evasive techniques are improving with the advancement of new detection methods. Ling et al. [74] reviewed the current adversarial attacks against PE files and the countermeasures that might improve the robustness of malware classifiers. They highlighted numerous adversarial attacks that target Windows PE detectors. Demetrio et al. [75] developed three types of attacks against machine learning detectors for PE files. They named their attacks Full DOS, Extend, and Shift. They combined the three attacks to manipulate the DOS header in the PE format. Then, the DOS header is extended with the content in the first section shifted. In their work, they considered previous attacks and focused on improving the evasion of malware for the current machine learning-based detectors. Their developed attacks showed their effectiveness in evading machine learning models.

Li and Li [76] adopted an Improved Reinforcement Learning (IRL)-based approach to generate adversarial executables that evade machine learning models for malicious executable file detection. Their method is based on the Deep Q-Network (DQN) and contains an engine that automatically generates adversarial samples based on a set of features given to the engine. It adjusts the malicious code to evade detection based on

the given features. Similarly, Quertier et al. [77] utilized an IRL-based approach employing DQN and achieved good evasion results. Lucas et al. [78] proposed a method that modifies the malicious code with functional instructions that are part of the original binary code to evade the detection of DNN and commercial antimalware tools. Their method proved to be effective in evading many DNN models and antimalware tools. To et al. [20] proposed FeaGAN, a metamorphic malware creation method based on the integration of GANs and RL. Their model takes advantage of both the GANs and RL to design an ensemble learning model for metamorphic malware generation. Their method showed an almost 100% evasion rate.

Fasci et al. [79] proposed a GANs-based approach that targets visualization-based malware classifiers. Their proposed method works by the utilization of a substitute detector, a general generator, and a blackbox detector. The purpose of the blackbox detector is to mimic the real-world scenario where an executable is provided, transformed into an image, analyzed, and then the prediction is made. On the other hand, the substitute detector is used to generate labels based on known variants where additional information is given. The substitution detector is used to make predictions based on the images and then update the weights for the generator to create undetectable visualization patterns that evade visualization-based malware detectors.

Table 3.1: A comparison of recent literature and our work.

Work	Year	Dataset type/ date	Model Analysis	Feature Analysis	Feature Reduction	Latest Variants	ML Evasion Techniques
[56]	2023	VirusShare (n.d.)	✓	✓	×	×	×
[31]	2023	VirusShare (n.d.)	✓	×	×	✓	✓
[39]	2023	Microsoft Kag- gle (2015)	✓	×	×	×	×
[32]	2022	Honeyd collected (2022)	✓	Attention Block	×	×	×
[55]	2022	binaries (n.d.)	×	×	×	×	×
[44]	2022	Windows ex- ecution traces (2019)	✓	×	×	×	×
[34]	2022	PEs (n.d.)	×	Boruta, RRF, RFE, Genetic Algorithm	✓	×	×
[35]	2022	PEs (2021)	×	Boruta ANOVA	✓	×	×
[58]	2021	PEs (n.d.)	×	×	×	✓	✓
[52]	2021	Windows files ()	×	RFE	✓	×	×
[9]	2021	Microsoft Kag- gle (2015) Drebin (2014)	×	RFE	✓	×	×
[64]	2021	PEs (2020)	×	×	×	×	×
[80]	2021	VirusShare (n.d.)	×	×	×	×	×
[33]	2020	VirusShare (n.d.)	✓	×	×	×	×
[81]	2020	NITRSCCT (2019)	×	heatmaps	✓	×	×
[82]	2020	Microsoft Kag- gle (2015)	×	×	×	×	×
MLMD- AFE	2024	PEs (2023)	✓	PCA	✓	✓	×

3.3 Limitations of Existing Malware Detection Methods

Additional static and dynamic analysis methods were reviewed and found similar to the methods that were reviewed in this section [83], [84], [85], [86], [82], [87]. Table 3.2 shows a summary of several methods that were reviewed with their proposed methods, pros, cons, and limitations. In essence, a few deficiencies are noticed within the current detection models. First, existing static methods were not tested against the latest malware variants. Many methods reviewed in the previous section were introduced to evade the detection of machine and deep learning classifiers. Based on our review, static methods are less effective against new variants [58]. Second, the majority of the studies that focused on building a good model failed to address the feature analysis and engineering part. On the contrary, studies that focused on feature engineering did not perform enough analysis for the selected model. Third, many methods used old datasets that might not present sufficient evidence for the ability to detect current malware. As previously mentioned, new malware variants, obfuscation, and evasion techniques are being developed frequently. Table 3.1 shows a summary of reviewed methods with comparison characteristics.

This research focuses on addressing the deficiencies in the current literature. First, the use of a single feature source is addressed by using multiple static feature sources to make the features resistant to evasion attempts. The selected features are tested against the latest malware variants available on the web. The use of multiple static features would improve the robustness of the generated feature vectors. Second, MLMD-

AFE aims to perform both advanced feature engineering and the construction of an optimized model based on the feature engineering process outcomes. Existing work focused either on feature engineering or model selection and optimization, leading to average results. Third, a unique most recent malware dataset is constructed to test MLMD-AFE effectively. The last rows in Table 3.2 and Table 3.1 show the MLMD-AFE compared to existing work.

Table 3.2: A summary of the reviewed articles with the base features used, the classifier(s), pros, cons, contribution, and limitation of each reviewed article.

Work	Feature	Classifier	Pros	Cons	Contribution	Limitations
[56].	API Calls, Embedding	GCN	Combined API calls and embeddings	Did not consider adversarial samples	GCN model using FCG vectors	FCG built without proper feature analysis. Obfuscation not tested.
[31]	API Calls, Strings	GCN	Utilized FCGs for semantic meaning	Robust against appending attack	GCN model based on API calls and strings	Weak feature analysis
[39]	ByteCode, OpCode Section	Multi ML, DNN	8 classifiers tested	Raw features used	Comparative study of 8 classifiers	Old dataset. No contribution to feature selection.
[32]	PE info	FANN	Unsupervised learning, novel feature selection	Blackbox feature selection method	Deep unsupervised learning model (PROUD-MAL)	Good features cannot be identified using the model
[55]	Raw Binary	XGBoost	Multi-platform, novel features	Better results for Windows	Cross-platform malware detection	Obfuscation not tested. Weak Windows results.
[44]	API Calls, Embedding	Text-CNN, BiLSTM	Embedding provides semantic meaning	Raw features given to the model	CruParamer as DNN model based on API sequences	Old dataset. Obfuscation not tested. Features not optimized.
[34]	API Calls	RF, GBM, Adaboost, SVM	Complete feature analysis	Only for ransomware, low accuracy	Comprehensive analysis of API calls	Weak classifiers analysis. Obfuscation not tested.
[35]	PE info	XGBoost, RF, KNN	Effective feature reduction	Accuracy reduced with recent samples	Feature selection to enhance ML classifiers	Obfuscation not tested.
[58]	API & Sys Calls	DNN, J48, JRip	Evaluation against evasion techniques	Hard to scale with dynamic features	Method for unpacking and extracting features	Difficult to apply. Accuracy not high enough
[52]	PE info	KNN	Simple and fast	Requires a long time to compute	Malware detection based on PSO	Obfuscation not tested.
[9]	n-grams bytes	RF	Dataset independent, cross-platform	Susceptible to benign code insertion	Dataset-agnostic malware classification	Old Dataset. Obfuscation not tested. Lacks analysis.
[64]	PE info, Bytes	CNN-VGG16	No expertise required	Easily evaded with simple features	Transfer learning method based on VGG16	Small, unbalanced, old dataset. Obfuscation not tested. Easy to evade.
[88]	Fuzzified features, n-gram	Fuzzy RF, SVM	Novel faster method	Low accuracy	Boosted RF classifier based on fuzzy PE features	Feature analysis required for distinctive vectors
[33]	OpCode	CRNN	Easy to implement, sufficient for low power devices	Single feature use	CRNN for Windows malware detection	Easy to evade. Obfuscation not tested.
[81]	API & Sys Calls	SVM	Short training time	Weak experiment	Incremental malware detection	Weak analysis. Obfuscation not tested.
[82]	FCG, OpCode, PE info	RF	Integrates two integration vectors	Only tested for malware	RF Classifier using FCGV and PE-based features	Old dataset. Obfuscation not tested. Features not analyzed.
MLMD-AFE	PE info, Bytes	DNN	-	-	Ensemble Learning model based on advanced feature engineering	-

CHAPTER 4

FEATURE ENGINEERING AND ANALYSIS

In this Chapter, the feature engineering and analysis process of MLMD-AFE is described. The goal of the feature engineering process is to analyze a set of a huge number of features and then exclude the unimportant features. First, the feature sets used in this thesis are described. The reasoning behind its selection and the description and its source are presented. Second, the selected feature sets are described in detail. Each feature is described with the parameters, the number of features per set, and the extraction source. Third, the processing steps for each feature set are described in detail. Fourth, the PCA visualization of the features is presented with the analysis of the features based on the graphs. The raw feature and the processed feature are compared. Fifth, the single feature set analysis is performed for each feature set selected and evaluated for malware detection individually. Finally, the IG implementation for the selected features is described.

4.1 Selected Features

For the purpose of malware detection, the features extracted using the EMBER extractor are selected for analysis. EMBER [89] is a large-scale dataset that contains nine features used for the creation of a machine learning model for malware detection. The dataset was created by collecting rich information from the PE file headers, sections, and raw binaries. In essence, nine types of features were extracted from the malware. Four format-agnostic features from the raw binaries including the byte histogram, the byte-entropy histogram, the strings, and the data directories. Moreover, five parsed features were extracted from the PE file header including general file information, header information, imported functions, exported functions, and section information.

This thesis utilizes the EMBER extractor to extract similar features from a newly constructed dataset. EMBER extractor was chosen for many reasons. First, the majority of static features described in the literature can be easily extracted using it. Second, unlike other papers reviewed in Chapter 3, the selected extractor would extract more than 2,000 features, whereas other researchers utilized much fewer features. Thus, the use of a huge number of features that cover the majority of static analysis-based features is a suitable option for the application of MLMD-AFE. Third, the EMBER model is considered one of the state-of-the-art models. Nevertheless, the majority of the features utilized by EMBER can be removed to improve the effectiveness of the model.

EMBER model was trained using the LightGBM classifier and is considered a

Table 4.1: Selected Features, the number of features per feature type, and their Descriptions

Feature Type	# of features	Description
Bytes Histogram	256	A histogram presentation of the bytes
Bytes Entropy	256	A histogram presentation of the bytes entropy
Strings	104	Information about the strings used within the executable
General Information	10	General Information about the file
Headers Information	62	File header information
Section Information	255	Information about each section in the PE file
Imported libraries	1280	Imported DLLs
Exported libraries	128	Exported DLLs
Data Directories	30	Directories presented within the executable file
Total	2381	All the features

state-of-the-art malware classifier. Moreover, recent research has used some feature types previously utilized by EMBER to detect malware such as byte strings [31], byte entropy [64], and other features. Although EMBER achieved high results, their experiment lacked a detailed analysis of the selected features. Further, their work was conducted in 2018 when less advanced malware was present. MLMD-AFE extends and analyzes the features used by EMBER to improve the detection of the most recent malware. EMBER was used to detect unpacked malware with a total of 2381 features. The feature vector they used combined all the feature sets without excluding any feature.

4.2 Features Overview

The selected features included nine groups of raw features previously used in malware detection based on the PE file format static information. Figure 4.1 shows a sample of the extracted features from a benign sample in the JSON format. It includes the Byte Histogram, the Byte Entropy, stings, general file information, section information,

headers, imported libraries, exported libraries, and data directories. Also, the SHA256 of the files is included for reference. Each feature within the identified feature sets can be utilized individually for malware detection. However, the combination of different feature sets would improve the detection rate.

The following is the description of each of the selected feature sets:

1. Byte Histogram: The byte histogram is the representation of the counts of the byte values for a given PE file. The Byte Histogram contains 256 integer values for the count that are considered features. Nonetheless, each integer by itself, if used solely, would be meaningless since the 256 integers make up the full histogram.
2. Byte Entropy Histogram: the byte entropy Histogram presents the bytes in relation to a chosen entropy window that is compared to the values of the bytes. Similar to the byte histogram, the byte entropy histogram contains 256 features used in combination as a single feature to preserve its semantic information.
3. Strings: Strings that exist within the PE files are another important feature used for malware detection. The extracted information related to strings include the number of stings in the PE file, and average length of the strengths for a given PE file, the entropy computed across all the strengths, and a histogram that presents the characters of the strings within the PE file. A total of 104 entries are used as string features
4. General File Information: The general file information includes file size, the virtual size, a boolean if the file has debug enabled, the number of exported

```
{
  "sha256": "46b0590e52830cfa736bc0aa72ba5f7ca90bcc50f830c6aecbf7e6c1bc5f2e30",
  "histogram": [175839, 19844, ..., 1977, 17821],
  "byteentropy": [0, 0, 4133, 8413],
  "strings": {
    "numstrings": 3870,
    "avlength": 9.090697674418605,
    "printabledist": [1166, 51, 131, 60],
    "printables": 35181,
    "entropy": 5.8876729011535645,
    "paths": 2,
    "urls": 25,
    "registry": 0,
    "MZ": 6 },
  "general": {
    "size": 944504,
    "vsize": 1056768,
    "has_debug": 1,
    "exports": 5,
    "imports": 91,
    "has_relocations": 1,
    "has_resources": 1,
    "has_signature": 1,
    "has_tls": 1,
    "symbols": 0 },
  "header": {
    "coff": {
      "timestamp": 1689039820,
      "machine": "AMD64",
      "characteristics": [
        "LARGE_ADDRESS_AWARE",
        "DLL",
        "EXECUTABLE_IMAGE" ] },
    "optional": {
      "subsystem": "WINDOWS_GUI",
      "dll_characteristics": [
        "HIGH_ENTROPY_VA",
        "NX_COMPAT",
        "DYNAMIC_BASE" ],
      "magic": "PE32_PLUS", ... } },
  "section": {
    "entry": ".text",
    "sections": [
      {
        "name": ".text",
        "size": 672256,
        "entropy": 6.371279255210566,
        "vsize": 672020,
        "props": [
          "CNT_CODE",
          "MEM_EXECUTE",
          "MEM_READ"
        ] ... } ],
    "imports": { "KERNEL32.dll": ["SizeofResource", ...] ... },
    "exports": ["MSE_AB48_Destroy", ...],
    "datadirectories": [
      {
        "name": "EXPORT_TABLE",
        "size": 224,
        "virtual_address": 834288, ... },
    ]
  }
}
```

Figure 4.1: An example of the features used in JSON format.

libraries, the number of imported libraries, a boolean if relocation is enabled, a boolean if resources are available, a boolean if the file has a signature, a boolean if the file has TLS enabled, and the number of symbols. This category contains 10 items that are used as features.

5. Headers: PE files contain the COFF and an optional header. In this feature category, the timestamp, targeted machine, and strings present image characteristics are taken from the COFF header, while DLL characteristics, code size, header sizes, and other information as shown in Figure 4.1. A total of 62 features are extracted under this category.
6. Section Information: The PE file format contains different sections, such as .txt, .data, .rdata, .reloc, .idata, .rsrc, .pdata, .debug, and .bss. The name, size, entropy value, virtual size, and properties of each section are extracted and used as features under this category. Adding all the features extracted from all the sections, 255 features are used from the section information.
7. Imported Functions: When the PE file is executed, it imports functions that are used to support the execution of the file. Some combinations of functions might be used to perform malicious actions, and thus the imported functions are crucial for the detection of malware. For the import features, EMBER [89] uses the hashing trick to find unique library sets with 256 entries, adding it to 1024 entries that capture individual functions to have a total of 1280 features.
8. Exported Functions: Export functions include the functions that a given file would export when executed. It might reveal information about adversarial

behavior based on the actions performed by the exported functions and libraries.

Thus, exported functions play a major role in malware detection. Similar to the imported functions, the exported function uses the hash trick but with only 128 entries as features.

9. Data Directories: Data Directories are features introduced in the second version of EMBER [89]. It contains the size and virtual address of 15 data directories within the PE file, including the EXPORT_TABLE, IMPORT_TABLE, DEBUG, GLOBAL_PRT, and other directories. Further description of each category can be found in the EMBER dataset [89]. This category contains a total of 30 features.

Algorithm 1 Normalize Histogram

Require: *raw_file*: Array of a histogram (counts)

Ensure: *normalized*: Normalized array

1: *counts* $\leftarrow$ np.array(*raw_obj*, dtype = np.float32)

2: *sum* $\leftarrow$ counts.sum()

3: *normalized* $\leftarrow \frac{\text{counts}}{\text{sum}}$

4: **return** *normalized*

4.3 Features Processing

Since the nature of different categories is different, each feature group was processed using different processing methods. The byte histogram and the byte entropy histogram were normalized using the average value to make the weights of each feature consistent among all the samples. Algorithm 1 shows the normalization technique used for the normalization of both feature types. The algorithm takes all the entries

within the histogram, sums them, and then divides each entry with the sum of all the entries.

Strings contained some features that did not require normalization and a histogram that presented the appearance of characters within each string. The characters' histogram was normalized using the average appearance value to make it consistent among all the samples. It used the same algorithm for the byte histogram and the byte entropy histogram. The other string features included counts and other numerical statistics about the PE file. No processing steps were applied to these features since their weights are constant among all the files.

The general file information categories were used as it is without further processing since the numbers did not require normalization. Features such as file size, file virtual size, debug availability, and counts were added to create a total of 10 numerical features. The header information contained numerical and categorical features. The numerical features were extracted from the raw JSON file without applying processing techniques. Similar to the general file information, this type of information is consistent among all the files. On the other hand, the categorical features were transformed for a numerical representation by applying the feature hasher function [90]. The feature hasher is a function provided by the scikit-learn Python library that transforms a given string into a corresponding numerical value which can be utilized for machine learning classifiers. This function is designed specifically to create unique numerical presentations for strings to avoid misleading the classifier.

The imported section was divided into two parts: the functions and the libraries parts. For the library parts, a list of 256 unique library bins was created using the

hashing trick. The hashing trick, implemented using the feature hasher function, takes all the libraries used by the sample and generates a unique numerical presentation for the libraries within the specified number of bins. In the MLMD-AFE case, the number of selected bins is 256. The functions part of the imported section was processed similarly to the library part. However, since functions are user-defined and might occur more frequently, the number of bins specified for the functions is 1024. It was calculated using the hashing trick with the same procedure followed for the library part.

The exported section contained only functions with no libraries. Compared to the imported section, the number of exported functions from PE files in general is much lower than the imported functions. Thus, only 128 bins were used for the exported libraries. The calculation of bins and the numerical transformation were done using the hashing trick similar to the imported functions. Lastly, data directories did not require any processing steps since each entry in the feature set contains a unique numerical value that is weighted similarly across all the files.

4.4 Features Visualization

The goal of the visualization step is to visualize each category using PCA transformation to show the effectiveness of the processing techniques used. The graphs of the raw features are computed and compared with the graphs of the processed features. PCA visualization also shows the uniqueness of the combinations of features when used to detect PE malware without combining it with other features. Before plotting

the features, Principal Component Analysis (PCA) is applied to each feature category, including the numerical raw features and all the processed features. PCA transformed the huge number of features into a smaller number of features with the first features having much higher weight compared to the last feature. The graphs presented using the PCA visualization present the first two transformed features that contain the rich feature information.

Figure 4.2 shows the visualization of the numerical raw features. Out of the nine features, four were considered numerical as it does not require transforming categories into numerical formats. The raw representation shows that the majority of the benign flies have similar byte distribution with a few outliers while the malicious samples vary. Moreover, it shows that the byte entropy histogram does not differentiate between malicious and benign samples. The string and byte visualization shows that there is a difference in the identification of malware and benign samples. Nonetheless, the string visualization shows a higher intersection between the two categories.

Figure 4.3 shows all the features after performing processing. Comparing the histogram and byte-entropy histogram after processing with the previous figure before visualization, both have higher variances where malicious samples are more distinctive. Moreover, strings became more unique after processing which would lead to a higher detection rate. The general file format was used without performing processing steps, and thus the visualization is similar.

The other categorical features were analyzed using PCA to test their effectiveness in the detection of malicious PE files. The headers and imported libraries show a high difference between the malicious and benign samples. Nonetheless, there is a high

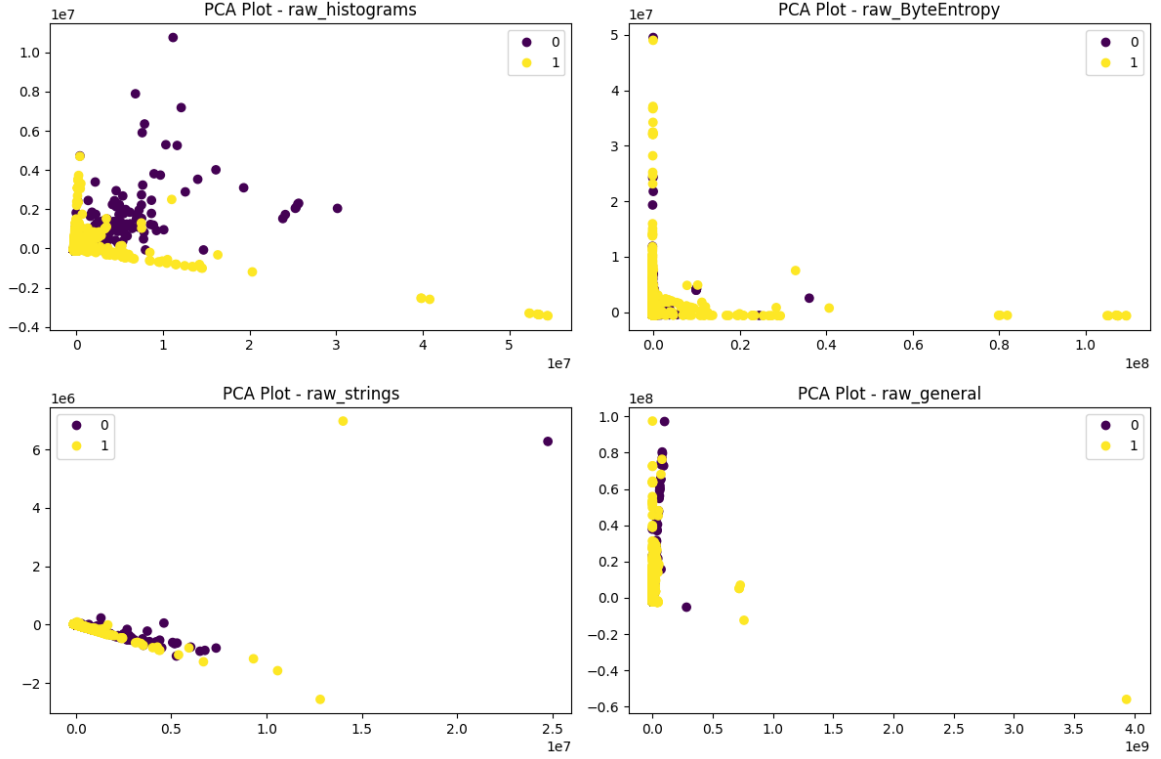


Figure 4.2: The PCA visualization of the four numerical raw features.

intersection where malicious and benign samples are indistinguishable. On the other hand, the section, exported libraries, and data directories categories do not show a unique representation that shows the difference between malicious and malware samples. Section information does not show any malicious samples on the graph, meaning that the combinations of features within the section information would lead to similar features for both benign and malicious samples.

While the PCA visualization implies that some features cannot be used for malware detection as it does not show uniqueness between malicious and benign samples, these features might be used as supporting features to improve the performance of other features. Moreover, some of the features within the categories might be useful but not the whole set when combined. The visualization shows that byte histograms, byte

entropy histograms, strings, header information, and imported libraries can achieve better detection rates compared to the general file information, the section information, the exported libraries, and the data directories when used individually.

Based on the PCA visualization analysis, byte histograms and byte entropy histograms are excluded from the selection process. The reason behind excluding them is that their values are unique enough to work as a standalone feature. Combining them with other features would result in either lowering the detection rate or not affecting the detection. Although these two feature sets were excluded, they will be implemented as an additional experimental evaluation to validate the claim made by the PCA analysis. The other features are included in the selection phase. Although some features might have been shown as unuseful based on the PCA visualization, their values are not presented explicitly. Thus, there might be strong features within these categories. Further analysis is performed by testing each category using six models to evaluate all the feature sets' capability in malware detection.

4.5 Single-Feature Analysis

Each of the feature categories was used to train six machine learning models to evaluate their effectiveness. five out of the six models were ensemble learning models including Decision Tree, Random Forest, AdaBoost, Gradient Boosting, and XGBoost. The reason for choosing ensemble learning models is due to the complexity of the features used. To confirm that ensemble learning models are better, the Support Vector Machine (SVM) model was used to show that its performance is not good compared to the

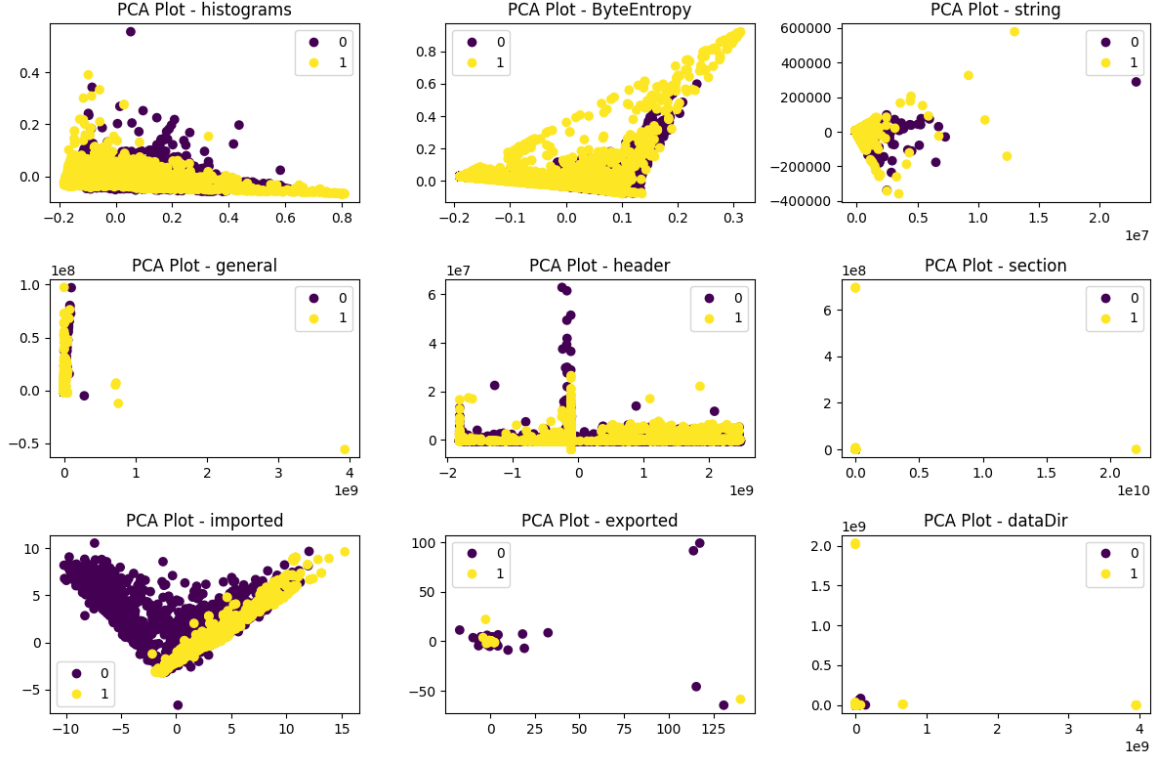


Figure 4.3: The PCA visualization of all the features after processing.

other selected models. The models were trained using a five-fold (5K) cross-validation to tune the hyperparameters in each model. Cross-validation was performed to ensure that the results were not biased toward the training/testing split. The best results for each model are presented in Table 4.2. The best results presented in this table were achieved using the XGBoost ensemble learning model. The full result of all the models for each feature group is presented in Appendix A.

The results show that the majority of feature categories used achieved results close to 95% with the exception of exported functions. The exported libraries had very low performance, indicating that the majority of the features within this category are not useful. The header information achieved the best results with an accuracy of 96.86% and an F1-score of 97.87%, indicating that headers might be strong as a standalone

Table 4.2: The best results of running machine learning models for each feature group.

Feature	Accuracy	Precision	Recall	F1-Score
Byte Histogram	96.67	96.71	96.40	96.55
Entropy Histogram	95.78	96.11	95.12	95.61
Strings	97.02	97.38	96.43	96.90
Headers	97.93	97.80	97.91	97.86
General Information	95.71	95.64	95.48	95.56
Section Information	96.80	96.92	96.45	96.68
Imported Libraries	93.04	93.93	91.52	92.71
Exported Libraries	69.20	61.22	99.01	75.66
Data Directories	97.63	97.88	97.21	97.54

feature type for malware detection. Since the results of the other features were high, additional experiments that combine each feature type might lead to an improved detection rate.

Since ensemble learning performs complex operations that might extremely reduce the weights of certain features, SVM was used to evaluate the performance of each category. Table 4.3 shows the results of the SVM model. When using SVM, only the byte histogram, byte entropy histogram, and imported libraries achieved results close to 90%. Referring to the PCA visualization in Figure 4.3, the models with high variance between benign and malicious samples in the visualization achieved the highest results in SVM. Thus, the byte and byte entropy histograms are better in the detection of malware when used individually. Moreover, the majority of important features for the detection are within the Imported libraries category. Since the other categories achieved high results using the best ensemble model, it indicates that many significant features were misleading the model when trained on SVM. The analysis and comparison of the ensemble learning and traditional machine learning implementation highlights the complexity of the features. Combining the PCA visualization outcomes

Table 4.3: The results of running the SVM model for each feature group.

Feature	Accuracy	Precision	Recall	F1-Score
Byte Histogram	88.89	85.14	93.28	89.03
Entropy Histogram	89.36	86.23	92.83	89.40
Strings	61.34	71.50	33.43	45.49
Headers	67.72	62.82	81.39	70.90
General Information	66.17	77.91	41.87	54.42
Section Information	53.86	82.86	5.27	8.72
Imported Libraries	92.81	94.28	90.63	92.42
Exported Libraries	67.95	60.25	99.05	74.93
Data Directories	60.76	93.80	20.17	33.17

with the single feature analysis, the current feature set, excluding the byte histogram and entropy byte histogram, is considered complex and requires ensemble learning with optimized parameters to achieve a high detection rate.

4.6 Information Gain Implementation

IG is implemented using the final feature sets specified by the PCA visualization and the single feature analysis findings. The selected feature sets are processed and aggregated to form a large inclusive feature vector. IG is implemented on the full feature vector to rank and sort the features based on their distinctive value for the given malware detection problem. After ranking the features, thresholds are set each time to select a different number of features. Each time the feature number is reduced, evaluation is performed using the new smaller feature set. The process is repeated for the values presented in table 4.4.

Table 4.4: Threshold values and their associated number of features for each IG implementation.

Threshold	Number of Features
1.0	2,381
0.8	1,904
0.5	1,190
0.3	714
0.2	476
0.1	238
0.08	190
0.05	119
0.03	71
0.01	23

CHAPTER 5

MLMD-AFE SPECIFICATIONS

This section describes the proposed MLMD-AFE methodology specifications and how it can be applied to optimize the detection rate and performance in malware detection. First, a general overview of MLMD-AFE is presented that describes the three main phases and the steps within each phase. Then, a detailed description of each phase and the technical details behind the selection of each component are presented.

5.1 MLMD-AFE Overview

This research proposes MLMD-AFE that machine learning models based on a feature engineering and analysis process on features extracted from PE files. The review in Chapter 3 showed that static features are more efficient and easier to scale when working on malware detection problems. Thus, MLMD-AFE will utilize static analysis-based features for the purpose of enhancing the detection performance. To overcome the limitations in the literature where a single feature source is used, MLMD-AFE will use features from multiple parts of the file to extract feature sets that are difficult

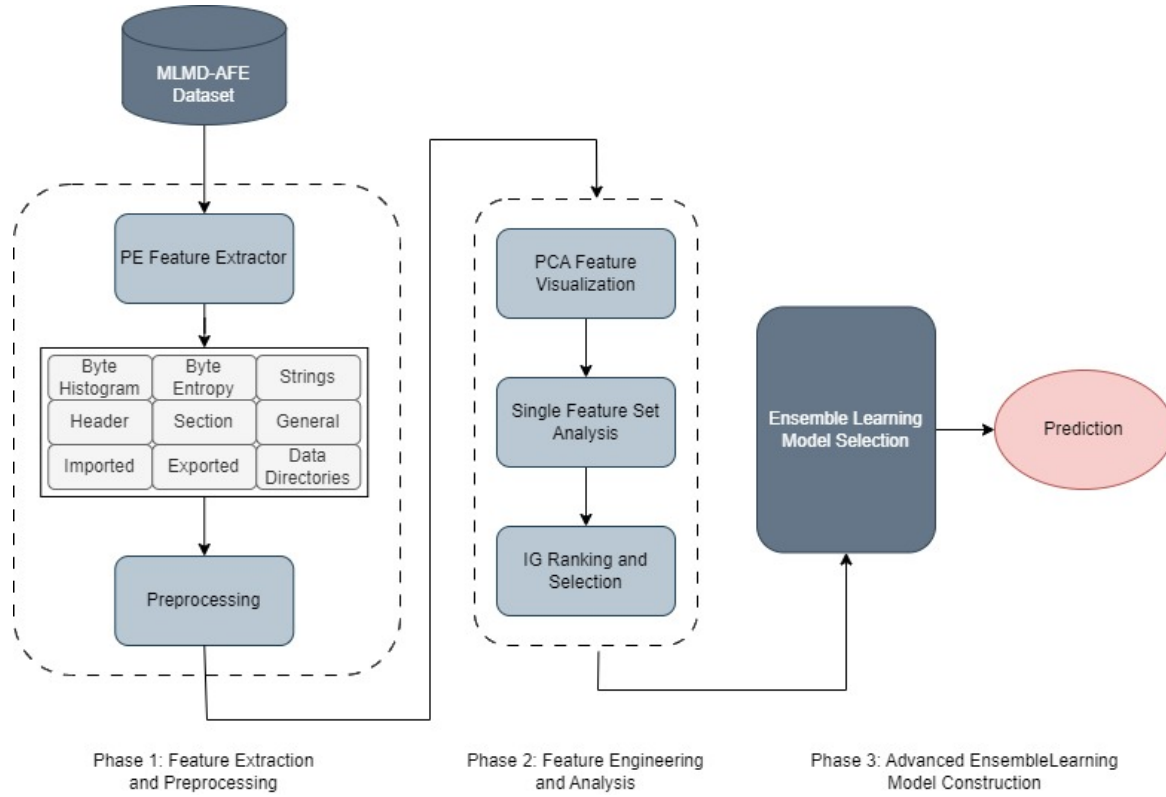


Figure 5.1: An overview of the three main phases in MLMD-AFE method.

to temper. Indeed, EMBER [89] features will be used as base features. Section 4.1 described the features provided by EMBER in detail. EMBER features consist of nine different types of features that were proven to be effective in the detection of malware. The features are extracted from the byte code and the PE headers for Windows PE files. MLMD-AFE method aims to apply advanced feature engineering on the selected features, that were interpreted as highly effective in malware detection by the research community, to make more resilient feature vectors based on the latest malware in the wild. MLMD-AFE aims to reduce the high number of the selected features by analyzing the quality of the features to make the model capable of detecting recent malware variants effectively.

Figure 5.1 illustrates the main phases of the MLMD-AFE approach. Initially,

MLMD-AFE employs a PE feature extractor to extract the feature sets from the PE dataset. The extracted features are preprocessed and prepared for further analysis. Subsequently, feature engineering and reduction are applied to yield an optimized feature set capable of accurate malware detection. Based on the feature engineering outcomes, some features are eliminated while others are reduced. The PCA algorithm will be implemented to perform dimensionality reduction for the feature and further analyze them. Subsequently, the IG algorithm for feature ranking and selection will be implemented to rank the features based on importance. The algorithm has been effective in the selection of features. Based on the analysis, the best feature set is chosen and analyzed for validation. Finally, distinctive features are selected and used to construct advanced machine learning models. Each machine learning is finetuned and optimized according to the selected features to maximize the detection rate while maintaining reasonable computational overhead.

The detailed steps for MLMD-AFE are subdivided into three main phases as follows:

1. Feature Extraction and Preprocessing
2. Feature Analysis and Engineering
3. Machine Learning Model Selection and Optimization

In Phase 1, a PE feature extractor is used to extract the nine feature types from the PE dataset. The EMBER extractor based on the LIEF project is utilized for the extraction of the features. The extracted features are then preprocessed and prepared

for further quantitative analysis. Missing information, corrupted files, and mislabeled files are removed during this phase. In Phase 2, each feature type is critically analyzed solely in the context of malware detection and classification. Each feature type is further visualized using the PCA algorithm. Based on the visualization and analysis, the distinctive feature types are selected and used in building the model. In Phase 3, machine learning models are designed while considering making it effective in grasping the final feature set while optimizing the results by fine-tuning the hyperparameters.

5.2 Feature Extraction and Preprocessing

The EMBER extractor [91] is used to extract features from PE files from the binary code and the PE headers. MLMD-AFE uses EMBER code with LIEF [92] version 0.10.0. Since EMBER was released earlier based on an older version of LIEF, a library used to instrument PE format, our extracted features might differ from the original EMBER database. The original EMBER data used LIEF version 0.9.0 while ours uses 0.10.0. The EMBER extractor targets PE headers and the binary code of executable files to extract nine feature sets.

After feature extraction, feature preprocessing is performed on each feature set based on the nature of the features, the complexity of the features, the best approach to interpret the feature, and the best practices for preprocessing features. For instance, the first two categories of features contain histograms. Histograms might weigh differently across different files based on their size. Normalization is applied in the preprocessing step to ensure that each bin of the histogram is weighted equally

across all the PE files. On the other hand, categorical variables are transformed into numerical vectors that comparably present them to utilize their use in machine learning model construction. The detailed feature preprocessing steps are presented in Chapter 4, the feature engineering chapter. Chapter 4 included a summary of the number of features contained within each category and the total number of features extracted and preprocessed using EMBER.

5.3 Features Engineering and Analysis

The selected feature sets are analyzed deeply using advanced techniques to perform the feature selection and implementation effectively. The goal of this phase is to remove the features that might downgrade the performance of MLMD-AFE and reduce the number of features to minimize the computational overhead while preserving the performance of the model. The removal of less critical features would improve the performance of the model by reducing the highly correlated features fed to the machine learning model. Different techniques are utilized to optimize the feature selection process. For instance, PCA is used to visualize the correction of each feature group in terms of the specified categories. Then, each feature is analyzed individually to analyze its complexity. Lastly, IG is used to rank the final features and achieve the optimized feature set.

Feature Engineering and Analysis processes can be summarized in three main steps:

1. PCA Visualization

2. Single-Feature set Analysis

3. IG Ranking and Selection

Before building the main machine learning detection model, each feature is analyzed as a standalone feature. This step would help us understand the limitations of each feature and how MLMD-AFE can overcome them by combining multiple features based on complexity and correlation. First, each feature type is isolated and analyzed in the context of malware detection and classification. The correlation of the feature and its significance to the detection of malware is further explored to decide if it is critical for the detection process. Feature types are visualized using the PCA algorithm to highlight the correlations or differences between the two classes. Subsequently, different combinations of feature types are then used for the analysis after ranking them using IG. All the possible combinations of feature types will be tested and analyzed while optimizing a neural network model learning for all the combinations to efficiently select the proper set of features and neural network model. The next subsections describe each step in detail.

5.3.1 PCA Visualization

PCA visualization involves the visualization of transformed features into 2D plots that can help in analyzing the quality of feature sets. This step includes six main steps. Figure 5.2 shows the overall process followed in the PCA visualization step. It contains the six essential processes and the feature exclusion list as output. First, each feature set is transformed using PCA dimensionality reduction approach into

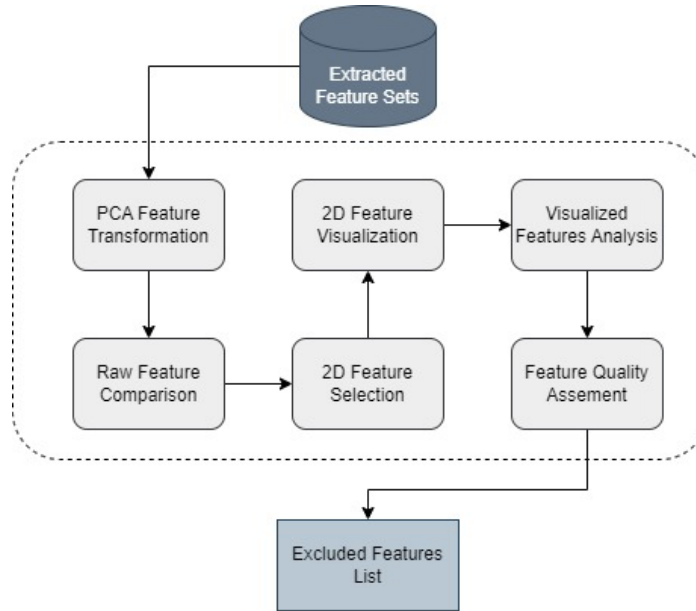


Figure 5.2: The process followed by the PCA visualization step in feature engineering and analysis.

a smaller set that preserves the high-quality aspects of each feature. Second, the transformation is applied for both the raw and processed features to assess the pre-processing effectiveness. Only the numerical raw features are transformed using PCA since categorical features require preprocessing prior to the transformation. Second, the first two entries of the PCA transformations are selected for the analysis. Third, the two representative features are visualized using a 2D plot. The 2D plot contains the features based on the labels for the purpose of distinguishing and analyzing how different categories are presented. Fifth, graphs are visually analyzed and interpreted. If a graph shows that features are clearly separated linearly, simple machine learning models might be sufficient for the detection. Conversely, ensemble learning or deep learning models are required to match the complexity of the features.

Finally, the quality of each feature in the detection is assessed based on the visualization. Features that contain huge and complex variances are effective in the

detection better when used without combining them with other features. On the other hand, features that contain minor or small differences are effective in the detection of malware when combined with other features. Lastly, features that show no differences are not good in the detection. It might contain a very small number of useful features, but overall, the feature category is considered ineffective. Based on the PCA visualization and analysis, a list of excluded features is created. The excluded features are the features that were found to be unnecessary or ineffective when combined with the other feature sets.

5.3.2 Single-Feature set Analysis

The Single feature set analysis step involves the analysis of each feature set individually to understand the feature quality and complexity. This is done by training different models with different complexities using each feature set individually. First, the feature sets are separated to create nine feature vectors per file that would be used for the training. Second, the 5-K fold cross-validation and training are performed to measure the detection rate for each feature set. Two different types of machine learning models are used, namely, traditional machine learning and ensemble learning algorithms. The ensemble learning models involve additional steps for learning the features while excluding some features. Traditional machine learning models would learn and detect based on all the features without applying further steps. Thus, comparing the results of applying ensemble learning and traditional machine learning reveals the complexity of the features.

Third, feature assessment is performed based on the results of the cross-validation

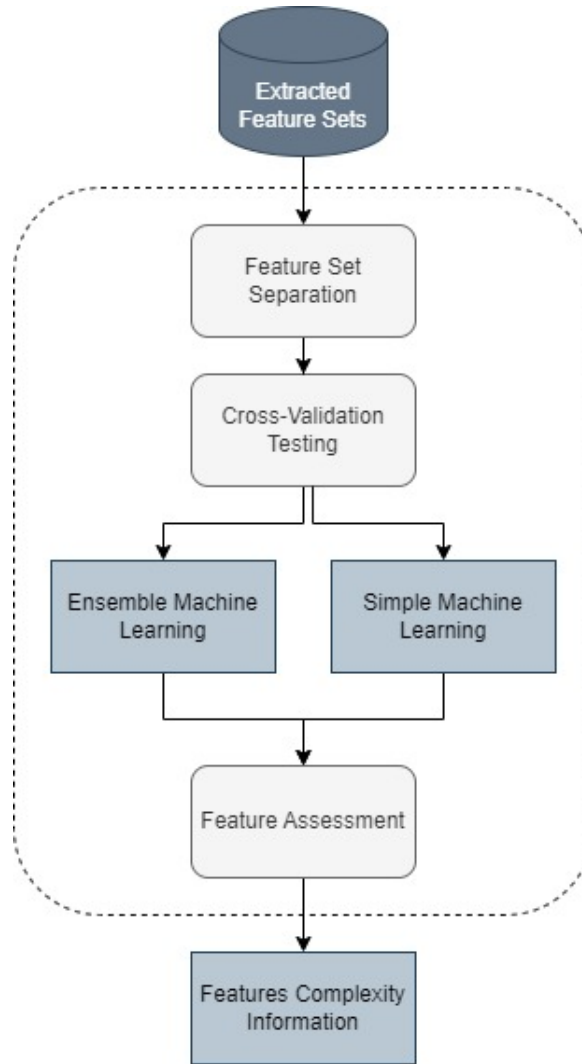


Figure 5.3: The single feature set analysis process with each step, the machine learning models used for cross-validation, and the feature complicity information as output.

testing step. In essence, the single feature analysis step provides MLMD-AFE with information related to the complexity of each feature set. The complexity of each feature set is further used in the final feature set selection and the machine learning model selection phases. Figure 5.3 summarized the process followed during the single feature analysis step. As shown, the main goal of this step is to measure the complexity of each feature set.

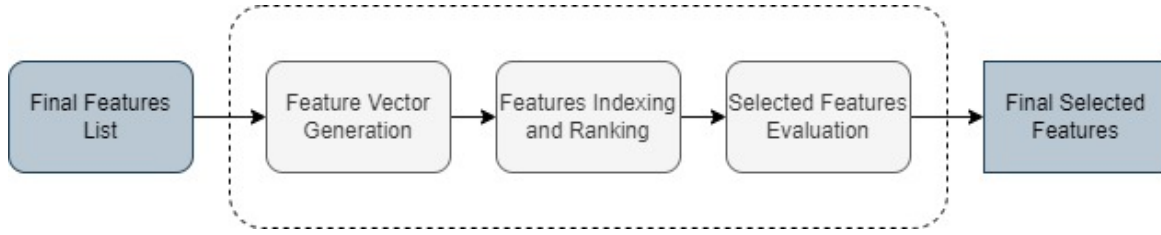


Figure 5.4: An overview of the three main phases in MLMD-AFE method.

5.3.3 IG Ranking and Selection

Based on the analysis performed in the previous two steps, a final feature set is selected that excludes unnecessary and low-quality feature sets. The Final feature list is fed to the IG algorithm for feature ranking and selection. IG is a feature selection method that ranks the features based on the amount of information provided by a feature that makes it distinctive when performing the detection. In other words, IG helps in the selection of features that contain more information that is helpful in the detection. Thus, the higher the feature rank, the higher the correlation between the feature and the category detection.

IG is utilized by MLMD-AFE to rank the features and select the top features for the detection of malware, aiming to improve the performance of the detection by removing the less unique features and keeping the more distinctive features. From the feature sets, feature sets that contain semantic meaning when combined with other features in the group are excluded from the selection process. The byte histogram and the byte entropy histogram are an example of an excluded feature set from the IG feature selection process. As described in Section 4.2, the 256 features in both the byte entropy and the byte histograms present the distribution as a histogram. Using part of the histogram while removing others would violate the semantic meaning of

the features and would lead to meaningless features. Thus, IG feature selection is performed on final categories that preserve the semantic meaning of features.

After the selection of ranking features, features are combined and indexed to analyze the results after ranking and reducing the number of features. Finally, features are reduced and evaluated varying from 100% to 1% of the original number of features the detection accuracy and other evaluation metrics are computed for performance measurement. The goal is to identify the percentage of features that optimize the results. Also, the point where the performance of the model is downgraded is identified.

After the selection of critical and distinctive features, the final feature set is used as input for the selected machine learning model. In essence, the feature engineering and analysis phase will contain three main outcomes: the excluded features list, the feature complexity information, and the final ranking of the features. The feature complexity information is used in the machine learning model selection and construction phase. Choosing the proper model for the training is crucial to improving the detection rate while minimizing the computational overhead. The final ranking of the features is also utilized by the selected machine learning models. As described in the IG section, different percentages of the features are utilized and the performance is reported.

$$G(D; f) = - \sum_{i=1}^n p_i \log_2 p_i + \sum_{v \in \{0,1\}} \frac{|D_v|}{|D|} \sum_{i=1}^n \frac{|D_{iv}|}{|D_v|} \log_2 \left(\frac{|D_v|}{|D_{iv}|} \right) - \sum_{i=1}^k w_i \log_2 w_i \quad (5.1)$$

Equation 5.1 shows the mathematical notation used by IG for the calculation of the ranking value for each feature. Given a dataset D with n classes and given feature f, the IG calculates the information gain (G) values based on the equation. After the

calculation, the values of IG are sorted in descending order for the ease of thresholding in the training and testing process.

5.4 Machine Learning Model Selection

The feature engineering and analysis phase shows that the feature sets utilized in the MLMD-AFE method are complex and would require ensemble learning models. Thus, MLMD-AFE utilizes five different ensemble learning models where the performance of each model is fine-tuned and optimized based on the feature sets obtained from the feature engineering and analysis phase. With the five ensemble learning models, an additional traditional machine learning model is added. The reason behind adding this model is to show that ensemble learning best fits the complexity of features compared to the traditional model. The selected ensemble models are Decision Tree, Random Forest, AdaBoost, Gradient Boosting, and XGBoost. In addition, SVM is added for the experiments. Although these models are considered ensemble learning models as they are trained based on smaller models that are aggregated, they are still considered machine learning models as no aggregation of two models is performed. However, these models are marked as ensemble learning in this thesis to differentiate their behavior from other models.

The reason behind choosing different ensemble learning models is to evaluate the performance of the defined feature engineering and analysis procedure and test its application to various models rather than limiting the evaluation to a single classifier. Moreover, enhanced versions of the same classifiers were utilized to evaluate how

complex ensemble learning models are better in the detection based on MLMD-AFE's feature engineering and analysis phase.

5.4.1 Decision Tree

Decision Trees are a type of supervised machine learning model that can be utilized for classification problems [10]. A decision tree model works by creating a tree that splits into branches and nodes that would lead to making a decision and classifying a given entry based on the tree structure [10]. The tree starts with the root node and then splits into branches based on the variables and their weight in the classification problem. At each branch, nodes exist with a query about the value of a given variable. Based on the presented value, the node would make a choice of which branch to follow at each step. Each node is connected to a child and a parent node. The chain starts from the root node until the leaf node is reached, where the results based on the variable combinations are presented.

As the tree structure implies, not all the variables are used in the construction of the decision tree model. The model will design the tree based on the underlying correlation of variables within the given malware detection problem. Thus, decision trees are considered an advanced type of machine learning.

5.4.2 Random Forest

Random Forest is a supervised machine learning algorithm used in classification problems. The random forest classifier combines multiple predictions from many smaller decision trees. It aggregates the results of the decision trees, takes the average, and

makes the prediction [93]. Since this model creates many small decision trees and combines them, it can achieve better results in large-scale settings compared to the single decision tree.

Random Forest Classifier works by specifying the number of decision trees within the model. Other parameters such as the maximum depth can be specified to make more resilient trees [94]. For instance, the maximum depth will customize the maximum number of nodes that can be split before reaching the leaf node that was described in the decision tree subsection 5.4.1.

5.4.3 AdaBoost

Adaboost is another supervised learning model that works by creating a large model based on all the given features. Then, the model is re-trained multiple times based on the results achieved from the full model training [95]. In each round, the model changes the weights of the misclassified samples and learns again to optimize the learning process.

Adaboost works by training many models called weak learners. The weak learners are trained to get an accuracy that is slightly higher than when the prediction is guessed randomly [96]. The assumption is that the creation of a meta-model from the sum of predictions of the weak learners would result in a higher detection rate. Thus, the AdaBoost model will change the weights of the features that misguided the classifier and include only weak learners that meet the specified condition.

5.4.4 Gradient Boosting

Gradient Boosting works similarly to the Adaboost model. It functions by combining the results obtained by many weak learners with results higher than random guessing. The weak learners are combined to create a strong learner that achieves better results in the prediction [97]. Gradient Boosting works by mapping the prediction values of a given dataset entry to the associated feature value while minimizing the loss function of the prediction.

In basic terms, Gradient Boosting works by solving an optimization problem that minimizes the loss based on the mapping of the prediction value to the given feature. In order to perform the estimation, ensemble models such as a decision tree are utilized [98]. The optimization problem is solved using the weak learners and then summed iteratively to construct the final model. The limit where the model would stop iterating is specified when implemented.

5.4.5 XGBoost

eXtreme Gradient Boosting (XGBoost) is another supervised ensemble learning model based on decision trees and the gradient boosting model. Unlike Gradient Boosting, XGBoost only uses decision trees as its base classifier for solving the optimization problem [99]. Similar to Gradient Boosting, the goal of the algorithm is to map the prediction to the associated features while minimizing the loss function. Nevertheless, the XGBoost algorithm contains an additional regularization parameter to reduce the step size when working on the optimization problem.

XGBoost reduced the complexity of applying the optimization when using the Gradient Boosting model. It uses a shorter decision tree structure that would result in much shorter training time and less computational overhead [100]. XGBoost model also introduced randomization to the selection of the features to reduce overfitting. Its main goal, compared to gradient boosting, is to achieve better results while reducing the model complexity.

5.4.6 SVM

SVM is a traditional, supervised machine learning model that aims to solve complex data classification problems. SVM fits a given dataset within a mathematical equation where the prediction is the output of the equation [101]. Within the default implementation, SVM fits the features using a linear equation ($y=xm+b$). Nonetheless, nonlinear versions are available to fit more complex feature structures with multiple classes.

Unlike ensemble learning models, SVM is much simpler and uses all the features to fit the model linearly. It is expected not to have high results using it, but it was added for validation purposes. Based on the complexity of the features within the malware detection problem, SVM is expected to behave abnormally as it cannot fit the complex intersections between the features.

In MLMD-AFE, the hyperparameters of all the models were fine-tuned based on the information obtained from the feature engineering and analysis phase. The number of features used, the ranking procedure, and the model complexity information were used to select the appropriate parameters for each model. The modifications were

done for the main parameters related to the model structure itself. Other parameters that include a high level of customization were not utilized.

CHAPTER 6

EVALUATION AND DISCUSSION

6.1 MLMD-AFE Dataset

Malware variants are deployed frequently to evade existing detection techniques. The evolution of new malware and obfuscation techniques poses a challenge to the collection of the most recent malware samples. We explored the existing datasets within the literature and found that they are outdated, not publicly available, unbalanced, or do not match the format required for MLMD-AFE. Furthermore, the publicly available datasets are small or focused on specific malware classes. Due to these constraints, a comprehensive study and review of existing malware resources is performed for the construction of a dataset. The goal is to construct a balanced and up-to-date collection of executable malware and benign files for the evaluation of MLMD-AFE. The process for dataset construction is presented in Figure 6.1.

The first step in the dataset construction is to obtain files from open-source web-

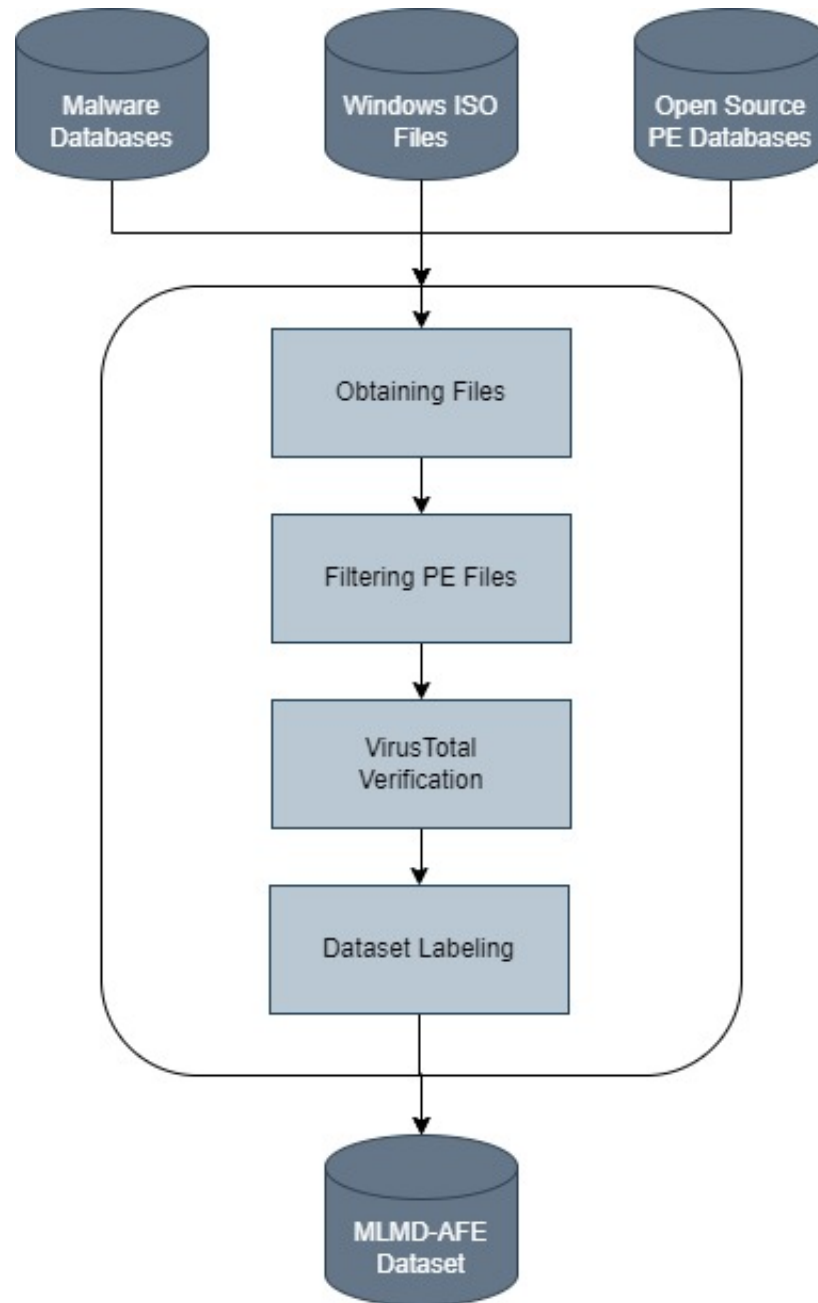


Figure 6.1: The steps performed for the MLMD-AFE dataset collection and validation.

sites, malware databases, and Windows ISO files. Files were collected from many malware sources that are publicly available for research purposes. Malware samples were mainly collected from Virus-Share [102] and Malware-Bazaar [103]. Both are popular malware-hosting websites that are updated daily with emerging and revisited malware samples, including new variants of old malware samples and new malware types. These websites contain various formats and types of malware. From both websites, malware files were collected for a span of 35 days starting from 15 December 2023 until 18 January 2024. All the files that were released during that period were collected and stored as part of the dataset construction process. After obtaining the files, PE files were filtered out and the remaining files were removed. Files that have the wrong extension or were marked as PE but aren't were excluded.

The benign samples were collected from Windows 11 and Windows 10 ISO images. It included distinctive PE files that were not copies and variations of the same file. More benign samples will be collected from trusted websites such as CNET [104] and legitimate websites that contain new executable files for Windows-based systems. Next, the collected PE files are filtered and verified using VirusTotal Manualy [105]. Each PE file was uploaded to the VirusTotal website and analyzed manually to ensure proper classification of samples in the constructed dataset. The data of the first submission, the number of detection engines that classified the sample as malicious, and the format of the malware were analyzed prior to the inclusion of a malware sample.

The first submission data was used instead of the creation date as the creation date can be modified or based on an older variant. For instance, a few samples with

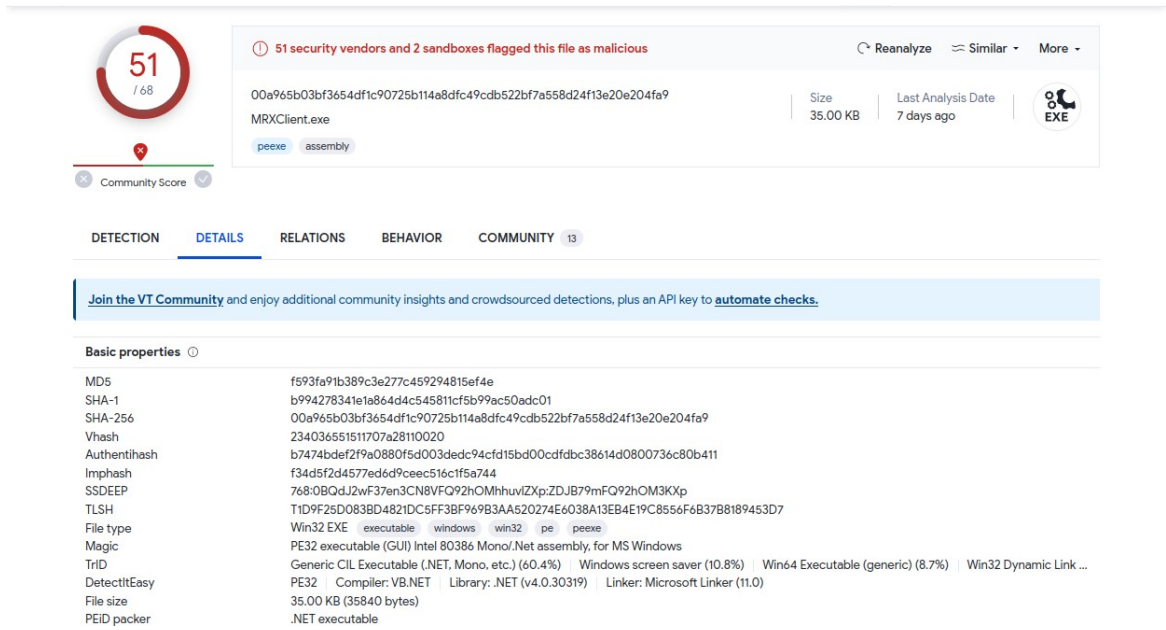


Figure 6.2: A sample VirusTotal results of a malicious PE sample.

the year 2082 were found during the analysis. Moreover, malware samples that were identified by the majority, where 50% of the engines marked it, as malicious were included. Samples with less score were analyzed manually based on the data included in VirusTotal [105] to confirm if the file is malware. Figure 6.2.

The total number of newly collected samples was 4,232 released in the last quarter of 2023 and the first month of 2024. An additional 1,039 samples that were reused but not released in 2023 were added to include a diverse set of malware samples. A total of 5,271 malware PE files were collected and will be used. The benign PE samples collected from Windows 11, Windows 11, and benign websites are 5,635. It included PE files released at different periods. In essence, the dataset size is 10,907 with 5,635 benign and 5,271 malicious samples.

6.2 Evaluation Measures

The well-known metrics used for the measurement of deep learning models are used in our experiments. We compute metrics including Precision, Recall, F1-Score, and Accuracy for the designed model. The aforementioned models are calculated using four standard measures: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The formula for each metric are as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} \quad (6.1)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (6.2)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (6.3)$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6.4)$$

The accuracy is the ratio of the number of correctly identified samples over the total number of samples. Accuracy is considered the most straightforward evaluation metric that is used widely. Precision focuses on measuring the percentage of the correctly identified samples from all the positive predictions. Thus, precision focuses on the sensitivity to the false positive of the model. The lower the false positive, the higher the precision. A high precision value indicates that the model is likely to

identify positive samples correctly. On the other hand, recall, also known as sensitivity, focuses on the number of correctly identified proportional to the total number of actual positive instances within the samples. It is used to evaluate the model's ability to detect all the positive instances in the samples. A high recall value indicates that the model is able to capture actual positive instances efficiently. The F1 score combines the recall and precision scores to provide a number that is used to balance between precision and recall scores. In essence, the F1 score relies on the combined values of precision and recall. When its value is 1, both the precision and recall are 1. When its value is close to 0, the precision and recall would have a low value.

Aside from the performance metrics The computational overhead of MLMD-AFE is calculated before and after applying the feature engineering process. The feature engineering process changes the time taken and the resources required at different stages. (1) The feature extraction time, (2) the features processing time, and (3) the model training time are measured and compared to show the effectiveness of MLMD-AFE. The feature extraction time is the time required to extract all the features used within the model. After applying the feature engineering and analysis phase, the number of features is reduced where a few sets are excluded, leading to a shorter extraction time. Similarly, the processing time is reduced as the features are less than the original implementation.

The training time would also change for two reasons. First, the number of features used during the training phase is reduced. The ensemble learning models will work on a much smaller number of features, leading to improved performance. Second, the optimization of the selected models based on the feature engineering and analysis find-

ings led to improved performance. Thus, the time for each step is measured after the implementation of MLMD-AFE and then compared with the baseline implementation for the features used prior to applying any feature engineering techniques.

6.3 Experiment Setup

The evaluation will be conducted in a Windows 11 operating system with 11th Gen Intel i7-11800H CPU with 8 cores. The experiment will be conducted without the use of the GPU to evaluate the computational overhead of the CPU only. The training, testing, and visualization steps were performed using the Windows 11 operating system to use the full CPU power. The dataset collection, feature extraction, and feature processing were performed within an isolated virtual machine since Windows malware files were used and might corrupt the original device.

Dataset collection was performed within a virtual machine running Kali Linux 2023.3 version. The hosting application is VMware Workstation PRO version 17.0.0. The isolated virtual machine was assigned 4 CPU cores and 4 Gigabytes of RAM. the dataset was collected manually within the virtual machine. For the extraction of features, the EMBER extractor was installed inside the virtual machine. Since EMBER requires a customized environment, a virtual environment was created using Anaconda version 23.7.4. Python 3.7 was installed within the virtual environment since it is required to run the EMBER code using LIEF version 0.10.0. The customized environment was used only for feature extraction and processing. Once these steps are finished, JSON and NUMPY files are downloaded to the hosting Windows machine

for full CPU utilization. The work on the Windows system was conducted in a virtual environment using Anaconda 23.3.1 and Python 3.10.0. All the libraries used were with the latest version on the installed Anaconda and Python versions.

6.4 Training, Validation, and Testing

The training of the selected features is performed using six models. Five ensemble models were utilized for improved detection while one machine learning model was used for performance comparison. 5-k fold cross-validation is performed to tune the model and compute the results without bias. In the 5-k cross-validation evaluation, the dataset is divided into five folds. Then, the training and testing are performed five times. At each time, four folds are assigned as a training set while the remaining one fold is used for testing. The results from running the cross-validation are reported for each fold. The advantage of applying the 5-K cross-validation is to avoid overfitting and data biases. After running the 5-K fold cross-validation, the average results are computed and reported for all the performance metrics specified in Section 6.2. The average values are used for the evaluation of MLMD-AFE.

6.5 Baseline Results

The original EMBER [89] implementation and evaluation were done using a feature vector that combines all the nine feature categories. To measure the effectiveness of the MLMD-AFE method, the results and computational overhead of the full model are computed using the AMLMD-FE dataset introduced in this paper. Although the origi-

Table 6.1: The baseline results of training the six models on all the extracted features.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	97.50	97.80	97.00	97.40
Random Forest	98.47	98.50	98.33	98.41
Adaboost	98.10	97.94	98.14	98.04
Gradient Boosting	98.41	98.46	98.25	98.36
SVM	67.74	62.82	81.54	70.96
XGBoost	99.06	99.15	98.90	99.02

nal implementation of EMBER utilized the Gradient Boosting Decision Tree (GBDT), the baseline implementation uses the same ensemble learning models used for MLMD-AFE evaluation for fair and direct comparison. Table 6.1 presents the results using all the features to train the six models chosen in this work. The highest results were achieved using the XGBoost model. A comparison between the performance of all the models will be conducted after performing feature selection and reduction using MLMD-AFE for the generalization of the feature engineering process.

6.6 Performance Evaluation

The accuracy, precision, recall, and F1 score of each model are computed using different numbers of features. The features were ranked using IG and reduced the number of used features to find the best combination of features to improve the performance of the malware detection model. For instance, the results when 80% of the features were used are computed and analyzed. The number of features at each iteration was as defined in Section 4.6. The results at each iteration are calculated and plotted using the final feature set specified by the feature engineering and analysis phase in the MLMD-AFE method. Figure 6.3 shows the results of the model after performing

Table 6.2: The best results achieved when 20% of the features are used.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	97.89	97.95	97.69	97.82
Random Forest	98.82	98.97	98.58	98.77
Adaboost	98.21	98.09	98.22	98.15
Gradient Boosting	98.34	98.37	99.20	98.28
SVM	67.78	62.86	81.52	70.98
XGBoost	99.09	99.09	99.03	99.06

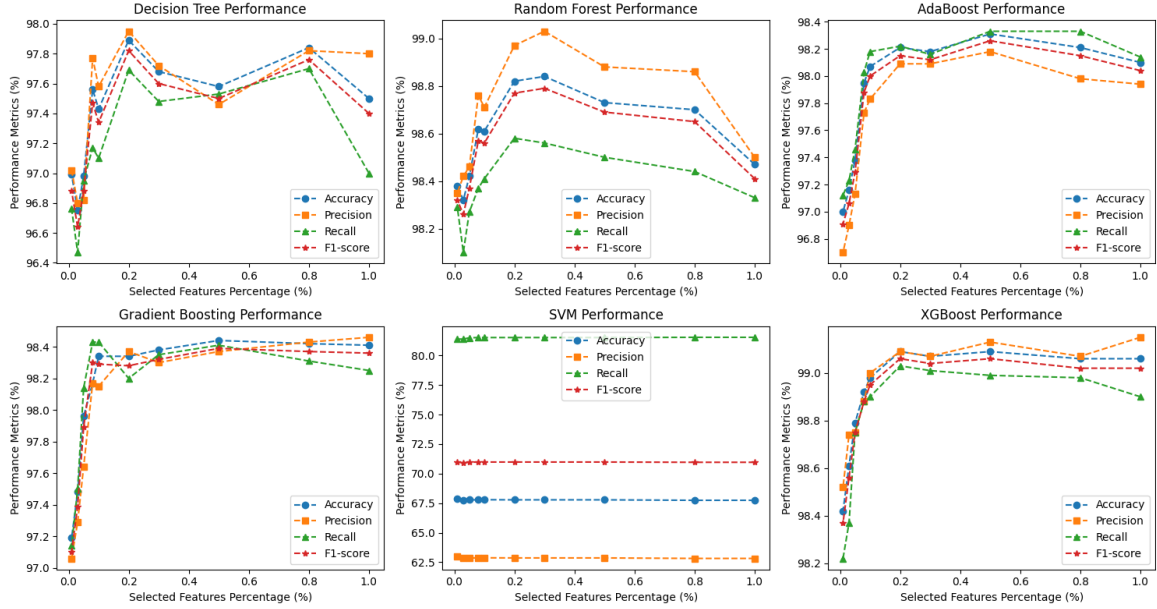


Figure 6.3: The results of applying IG using the six machine learning models with different percentages of features used at each time.

feature selection. The percentage of features is reduced each time varying from 100% of the features to 0.01% of the features to find the optimal point where the results are optimized. This figure only shows the results using the selection features excluding the byte histogram and the byte entropy histogram. The full detailed numbers for each iteration are available in Appendix B, while the best results at 20% of the features is presented in Table 6.2

The highest results were achieved after excluding these two categories. The reduced model was able to achieve an accuracy of 90.09% and an F1-score of 99.06%

Table 6.3: The number of features per feature set after the final feature selection phase.

Feature Type	# of features
Bytes Histogram	0
Bytes Entropy	0
Strings	102
General Information	7
Headers Information	28
Section Information	36
Imported libraries	174
Exported libraries	3
Data Directories	23
Total	373

after reducing 80% of the features, compared to the baseline that achieved 90.07% and 99.02% accuracy and F1-score, respectively. The baseline implementation included 2,381 features from the nine feature set categories, while the MLMD-AFE implementation included only 373 features as presented in Table 6.3. The final evaluation of the features and their numbers validates that the implemented MLMD-AFE’s feature engineering and analysis phase successfully reduced the number of features while maintaining the performance of the model in malware detection.

The analysis of the complexity of the features in the feature engineering and analysis phase indicated that the traditional machine learning models would behave abnormally when applied to such a complex set of features. In Figure 6.3, the SVM model achieved almost constant results for all four performance measures. Although the scores were not exactly the same, the behavior of the model using the selected features is due to the inability to fit a linear model on the complex final features. Thus, the selection of ensemble learning models was essential to achieve a high detection rate.

In the evaluation, the byte histogram and entropy byte histogram were excluded from the final feature set during the feature engineering and analysis phase. For validation purposes, additional experiments were done using different combinations of features including all the feature sets. Our experiments show that the results after excluding the byte histogram and the entropy byte histogram are better. Appendix C shows the results of computing the six models using the other combinations of features. Four different experiments were conducted. The features are tested in four settings: (1) the selection features after IG, (2) the selection features after IG combined with the byte histogram, (3) the selection features after IG combined with the byte entropy histogram, and (4) the selection features after IG combined with the byte and byte entropy histograms. The first setting is presented in this chapter as the best-achieved results while the other features are in Appendix C.

6.7 Computational Overhead Evaluation

The time required for running the extraction phase, the processing phase, and the training and testing phases were measured for the baseline and the final model after choosing the best number of features. Table 6.4 shows the time taken by the baseline model at each step and the time taken by the model using 20% of the features. It also shows the difference in the computational time required to complete each step and the total time required by each model to complete the full process, starting from the extraction phase until the end of the testing phase.

Based on the measured time, the implemented method, MLMD-AFE, managed

to reduce the time required for the extraction phase by 24.75% from the original time taken by the baseline features. Moreover, the time required for processing the extracted features was reduced by 5%. For each implemented model, the time required for the training and testing was reduced significantly. Gradient boosting is the most computationally expensive model that was utilized by MLMD-AFE. The original implementation required 135.99 seconds to complete the training which would be multiplied by five for the 5-K cross-validation implementation. After applying MLMD-AFE, Gradient boosting required only 25.88 seconds to train and test the model based on the selected reduced features set. In essence, about 81% of the time and computational resources required for running the gradient boosting were reduced after the implementation of MLMD-AFE’s feature engineering and ensemble learning model optimization process.

Out of the six selected models, the XGBoost model outperformed the other models in the detection accuracy, precision, recall, and F1-score. XGBoost originally did not require as high computational power as other ensemble learning models. However, the time required for training was reduced significantly from 4.36 seconds to 1.21 seconds, which preserves 72.25% of the computational resources and the time required for the training. Moreover, for the overall process, starting from the feature extraction phase until the final prediction results, the time and the computational power required for the best model, XGBoost, were reduced by 24.75% from the baseline implementation.

Table 6.4: The Computational time required for the baseline implementation, the MLMD-AFE implementation, and the saved computational time after applying MLMD-AFE.

Implementation	Model	Extraction (s)	Processing (s)	Training & Testing (s)	Total Time (s)
Baseline	Decision Tree	2,096.15	10.55	5.84	2122.54
	Random Forest			7.60	2114.3
	Gradient Boosting			135.99	2242.69
	Adaboost			48.92	2155.62
	SVM			84.45	2191.15
	XGBoost			4.36	2111.06
MLMD-AFE	Decision Tree	1577.43	9.98	1.09	1588.50
	Random Forest			3.16	1590.57
	Gradient Boosting			25.88	1613.29
	Adaboost			9.63	1597.04
	SVM			15.56	1602.97
	XGBoost			1.21	1588.62
Computational Difference	Decision Tree	518.72	0.57	4.75	524.04
	Random Forest			4.44	523.73
	Gradient Boosting			110.11	629.40
	Adaboost			39.29	558.58
	SVM			68.89	588.18
	XGBoost			3.15	522.44

6.8 Discussion

The implementation of MLMD-AFE showed that the high number of features that were previously utilized by one of the state-of-the-art malware detection models, EMBER [89], were effective but computationally expensive. MLMD-AFE showed that the baseline features extracted by EMBER could be reduced by 80% while maintaining the same accuracy, precision, recall, and F1-score. The evaluation in this paper included six models in order to measure the effectiveness of the feature engineering and analysis process over multiple models for generalization purposes.

Based on the findings, MLMD-AFE are capable of reducing huge feature sets into much smaller feature sets while maintaining the same results. The implementation of MLMD-AFE over multiple models that were optimized based on the feature en-

gineering and analysis phase showed that MLMD-AFE is capable of improving the performance of any model and is not bound to a specific model. In this thesis, additional steps were taken to customize MLMD-AFE for the malware detection and classification problem. In essence, MLMD-AFE can be applied to solve general feature selection and reduction problems within any field, but it requires customization to fit within the model context.

6.9 MLMD-AFE Limitations

Although the proposed approach, MLMD-AFE, achieved good results in reducing the number of features while maintaining the detection rate, it still has many limitations. this section discusses the limitations and weaknesses of MLMD-AFE. The following are the major limitations of this work:

Principal Component Analysis Limitation: During the feature engineering process, PCA visualization was utilized to understand the complexity of each feature category. The use of PCA helped in understanding the features' complexity using the given features. However, the visualization of features might not work in some cases. For instance, the visualization of the exported libraries in this thesis did not clarify whether the features are good to use in complex models. Thus, PCA effectiveness depends on the nature of the features and could not be applied in some cases where the features cannot be visualized clearly or have too close values.

Information Gain Limitation: IG was utilized for ranking the selected features. In the process, byte histograms and byte entropy histograms were excluded. The

reason behind excluding them is their semantic meaning where all the features within the category work as one feature. Thus, IG can be utilized to rank the features based on their importance only when the features are not related and do not have shared semantic value. For instance, ranking the pixels in an image as features would not be an effective solution since all the pixels are used to make the images. Nonetheless, corners might be chosen to make the features unrelated to the position and other parts. Similarly, the IG ranking step in MLMD-AFE can be applied only when the features are unrelated to the other close features.

EMBER Extractor Limitation: EMBER extractor allowed this work to deeply analyze a huge set of features and apply the proposed approach. Although the EMBER extractor can extract 2,371 features, these features can be extended using other features in the literature. This thesis focused on the application of the advanced feature engineering method and evaluating the proposed method. Thus, additional features can be added to perform a comprehensive feature selection of existing static features.

6.10 General Limitations

This section highlights the general limitations of the methods designed for malware detection using machine learning. Although these issues are general, they are still not addressed properly by MLMD-AFE.

Limitation to Windows-based files: MLMD-AFE implementation focused on the detection of malware in Windows-based systems. The features utilized were ex-

tracted from the PE format and the raw binaries. The application of MLMD-AFE into other operating systems requires the selection of other features and re-implementing them.

Static features-based model: MLMD-AFE utilized static analysis-based features for the detection of malware. Although static features are efficient and better for large-scale settings, new techniques that aim to evade static approaches are being developed. The use of static features only would make this model exposed in the future. The integration of dynamic analysis-based features with the current features would improve the performance of MLMD-AFE.

Machine Learning Evasion: The robustness of machine learning models cannot be proved unless it was tested against advanced obfuscation and evasion techniques. MLMD-AFE was only tested against the most recent malware variants. It was not tested against evasion techniques designed to evade machine learning specifically. This work did not test evasion techniques since their implementation is not publicly available and designing them is out of the scope.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

This thesis focuses on the integration of advanced feature engineering and analysis techniques with ensemble learning models. It aims to improve the performance of existing malware detection methods by applying advanced feature engineering and analysis. Subsequently, ensemble learning models are designed and optimized according to the findings of the feature engineering and analysis. A comprehensive review of the existing malware detection methods and features used in the literature utilized explicitly for Windows systems was performed. Based on the current literature limitations, the focus of the proposed method, MLMD-AFE, is on the application of advanced feature engineering techniques on feature vectors extracted from raw malware binaries. Features will be selected and refined to construct an ensemble learning model that optimizes the learning process based on the feature complexity and require-

ments. The extracted features will be analyzed using MLMD-AFE feature engineering process and evaluated prior to the model selection and optimization.

MLMD-AFE design included three main phases. Namely, the feature extraction and processing phase, the feature engineering and analysis phase, and the model selection and optimization phase. In the feature extraction and processing phase, a feature extractor is utilized to extract the features from the PE files. EMBER feature extractor was selected to perform the extraction. EMBER extractor extracted nine feature sets that can be utilized individually to detect malware. The extracted features are then processed according to the semantic value of the feature, the type of the feature, and best practices for the feature processing. Each feature set was processed individually based on the requirements.

In the second phase, feature engineering and analysis are performed. Features were first visualized using the PCA transformation to transform the most useful information in the whole feature set into the first two transformed features. Second, each feature set was analyzed and evaluated individually using five ensemble learning models and one traditional machine learning model. Lastly, IG was utilized to rank the selected features according to the findings of the previous steps.

In the last phase, ensemble learning models were selected based on the feature complexity identified in the feature engineering and analysis phase. The ensemble learning model parameters were tuned based on the feature complexity and requirements. First, a baseline model is computed by evaluating the ensemble learning models. Second, the IG ranking was set using different threshold values to evaluate the effectiveness of reduced selected values.

The evaluation of the proposed method was performed on a newly constructed dataset with the latest advances in malware variants. The dataset was collected between December 2023 and January 2024. All the collected PE files were analyzed and verified manually before the evaluation. Essentially, MLMD-AFE managed to reduce the number of features by 80% compared to the baseline model that was implemented previously by EMBER. Moreover, the computational power and time of MLMD-AFE was reduced significantly. For the best ensemble learning model, XGBoost, 72.25% of the computational power required to train the baseline model were preserved using MLMD-AFE. Overall, the time and computational power required for the feature extraction, processing, and model training were reduced by 25.75%.

7.2 Future Work

The effectiveness of MLMD-AFE in the feature selection and reduction for malware detection allows further utilization where more advanced problems are addressed. In this thesis, only the most recent variants were addressed but not the latest research findings for obfuscation techniques that evade malware detection. Moreover, the scope of the work included Windows PE files while excluding many malware types. With the advancement of malware that attacks various operating systems, MLMD-AFE can be utilized to address some of these challenges. In this section, future work and implementation that extends the work in this thesis is presented.

Malware Family Classification: The implementation of MLMD-AFE focused on addressing the detection of malware, meaning it would only classify the file as

either malicious or benign. In real-world scenarios, malware can take various forms. Each malware type has different characteristics and methods for taking action after the detection. Although MLMD-AFE was implemented for the detection only, future work would include the application of MLMD-AFE for the classification of malware families.

Hybrid Feature Sources: The scope of this thesis included the application of MLMD-AFE to a set of high number of features to select features, reduce features, and optimize a model based on the findings. Thus, features were obtained only from static analysis. Although MLMD-AFE was effective in the detection of malware, it was not tested against all the obfuscation scenarios that target static analysis. The integration of other features, such as API calls features obtained using dynamic analysis, can be tested in addition to the current feature set for improved robustness.

Obfuscation and Evasion Techniques Robustness: MLMD-AFE is shown to be effective in the detection of the most recent malware variants with the latest techniques. However, new methods are designed and tailored specifically to target malware detection models. This paper did not include testing these methods on the designed model. Future work would include applying advanced obfuscation and evasion techniques to the AMLMD-FE dataset to evaluate its robustness against such attacks. Additionally, the inclusion of new feature sources such as dynamic analysis-based features would be tested to improve the detection of obfuscated malware.

Applying MLMD-AFE to Different Domains: Although this work was tailored and designed specifically to address malware detection based on static analysis, the feature engineering and analysis process can be applied to any domain in machine

learning. In future work, MLMD-AFE can be tested in other security problems to improve the performance of the existing models. For instance, MLMD-AFE can be applied on Internet of Things (IoT)'s networks where huge amounts of traffic that include features are available. MLMD-AFE would be used to evaluate and analyze the traffic and then design an optimized detection model based on the analysis.

APPENDIX A

FULL RESULTS OF EACH FEATURE CATEGORY

This Appendix presents the full results for running the analysis of each feature set as identified in section 4.5. The best results for each feature set were documented in the feature engineering and analysis part. Moreover, the SVM results for each feature set were also documented. Thus, this section only includes the full experiment followed to achieve the results reported in Chapter 6. The Tables below present the six models evaluated for each feature set. Decision Tree, Random Forest, AdaBoost, Gradient Boosting, SVM, and XGBoost were utilized for the nine feature categories.

Table A.1: The results of running the six machine learning models for byte histogram.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	92.64	91.65	93.26	92.45
Random Forest	96.26	96.81	95.41	96.10
Adaboost	92.51	92.26	92.24	92.25
Gradient Boosting	94.25	94.03	94.08	94.05
SVM	88.89	85.14	93.28	89.03
XGBoost	96.67	96.71	96.40	96.55

Table A.2: The results of running the six machine learning models for byte entropy histogram.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	91.98	92.05	91.31	91.67
Random Forest	94.60	96.10	92.58	94.31
Adaboost	90.89	91.01	90.04	90.52
Gradient Boosting	93.07	93.95	91.56	92.74
SVM	89.36	86.23	92.83	89.40
XGBoost	95.78	96.11	95.12	95.61

Table A.3: The results of running the six machine learning models for the strings.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	93.01	93.11	91.37	92.74
Random Forest	96.27	97.41	94.80	96.09
Adaboost	92.74	92.28	92.75	92.51
Gradient Boosting	94.45	94.80	93.66	94.22
SVM	61.34	71.50	33.43	45.49
XGBoost	97.02	97.38	96.43	96.90

Table A.4: The results of running the six machine learning models for general file information.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	94.93	94.78	94.74	94.76
Random Forest	95.72	95.62	95.52	95.57
Adaboost	94.00	94.28	93.27	93.76
Gradient Boosting	94.92	95.04	94.42	94.73
SVM	66.17	77.91	41.87	54.42
XGBoost	95.71	95.64	95.48	95.56

Table A.5: The results of running the six machine learning models for the headers.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	97.30	97.41	97.00	97.20
Random Forest	97.94	98.13	97.59	97.86
Adaboost	94.61	93.81	95.12	94.46
Gradient Boosting	96.22	96.02	96.17	96.09
SVM	67.72	62.82	81.39	70.90
XGBoost	97.93	97.80	97.91	97.86

Table A.6: The results of running the six machine learning models for section information.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	94.57	95.81	92.87	94.30
Random Forest	97.03	97.25	96.59	96.92
Adaboost	92.55	92.89	91.61	92.24
Gradient Boosting	93.38	93.11	93.21	93.16
SVM	53.86	82.86	5.27	8.72
XGBoost	96.80	96.92	96.45	96.68

Table A.7: The results of running the six machine learning models for imported libraries.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	90.68	91.32	89.22	90.25
Random Forest	93.22	94.54	91.25	92.87
Adaboost	90.61	91.44	88.90	90.15
Gradient Boosting	90.69	91.96	88.48	90.18
SVM	92.81	94.28	90.63	92.42
XGBoost	93.04	93.93	91.52	92.71

Table A.8: The results of running the six machine learning models for exported libraries.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	65.34	58.28	99.60	73.53
Random Forest	70.10	61.96	98.88	76.18
Adaboost	67.68	60.02	99.22	74.80
Gradient Boosting	67.40	59.82	99.24	74.64
SVM	67.95	60.25	99.05	74.93
XGBoost	69.20	61.22	99.01	75.66

Table A.9: The results of running the six machine learning models for data directories.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	96.45	96.45	96.22	96.34
Random Forest	97.45	97.87	96.83	97.35
Adaboost	94.40	97.73	93.63	94.17
Gradient Boosting	95.83	95.59	95.79	95.69
SVM	60.76	93.80	20.17	33.17
XGBoost	97.63	97.88	97.21	97.54

APPENDIX B

NUMERICAL RESULTS FOR THE IMPLEMENTATION OF THE FINAL SELECTED FEATURES

Chapter 6 presented graphs plotting each model's results. The accuracy, precision, recall, and F1 scores were plotted for different thresholds. This Appendix presents the numerical results for the graphs shown in Chapter 6. The tables below present the performance metrics for each model with the various tested thresholds. The 1.0 threshold was omitted as it was replaced with the baseline results for each model.

Table B.1: The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the final features.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	97.84	97.58	97.68	97.89	97.43	97.56	96.98	96.75	96.99
Precision	97.82	97.46	97.72	97.95	97.58	97.77	96.82	96.80	97.02
Recall	97.70	97.53	97.48	97.69	97.10	97.17	96.95	96.47	96.76
F1-score	97.76	97.50	97.60	97.82	97.34	97.47	96.88	96.64	96.88

Table B.2: The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the final features.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.70	98.73	98.84	98.82	98.61	98.62	98.42	98.32	98.38
Precision	98.86	98.88	99.03	98.97	98.71	98.76	98.46	98.42	98.35
Recall	98.44	98.50	98.56	98.58	98.41	98.37	98.27	98.10	98.29
F1-score	98.65	98.69	98.79	98.77	98.56	98.57	98.37	98.26	98.32

Table B.3: The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the final features.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.21	98.31	98.18	98.21	98.07	97.95	97.38	97.16	97.00
Precision	97.98	98.18	98.09	98.09	97.83	97.73	97.13	96.90	96.70
Recall	98.33	98.33	98.16	98.22	98.18	98.03	97.46	97.23	97.12
F1-score	98.15	98.26	98.12	98.15	98.00	97.88	97.29	97.06	96.91

Table B.4: The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the final features.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.42	98.44	98.38	98.34	98.34	98.17	97.96	97.48	97.19
Precision	98.43	98.37	98.30	98.37	98.15	98.17	97.64	97.29	97.06
Recall	98.31	98.41	98.35	98.20	98.43	98.43	98.14	97.50	97.14
F1-score	98.37	98.39	98.32	98.28	98.29	98.30	97.89	97.39	97.10

Table B.5: The numerical values for the accuracy, precision, recall, and F1-score of applying the SVM model to evaluate the various IG thresholds for the final features.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	67.74	67.74	67.78	67.78	67.78	67.79	67.78	67.78	67.86
Precision	62.82	62.82	62.86	62.86	62.86	62.87	62.86	62.86	62.97
Recall	81.54	81.54	81.52	81.52	81.52	81.52	81.52	81.50	81.39
F1-score	70.96	70.96	70.98	70.98	70.98	70.98	70.98	70.97	71.00

Table B.6: The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the final features.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	99.06	99.09	99.07	99.09	98.98	98.92	98.79	98.61	98.42
Precision	99.07	99.13	99.07	99.09	99.00	98.88	98.75	98.74	98.52
Recall	98.98	98.99	99.01	99.03	98.90	98.88	98.75	98.37	98.22
F1-score	99.02	99.06	99.04	99.06	98.95	98.88	98.75	98.56	98.37

APPENDIX C

RESULTS INCLUDING BYTE AND BYTE ENTROPY HISTOGRAMS

This Appendix includes the additional experiments performed by including the byte histogram and byte entropy histogram in the feature selection process. Three experiments are presented: (1) the experiments using IG feature ranking and selection with the byte histogram, (2) the experiments using IG feature ranking and selection with the byte entropy histogram, and (3) the experiments using IG feature ranking and selection with the byte histogram and the byte entropy histogram.

C.1 Byte Histogram Inclusion

This section shows the results of the evaluation of the features with the byte histogram feature set included. Figure C.1 shows the results of the experiments. Comparing

Figure C.1: The results of applying IG using the six machine learning models with different percentages of features used at each time including the byte histogram.

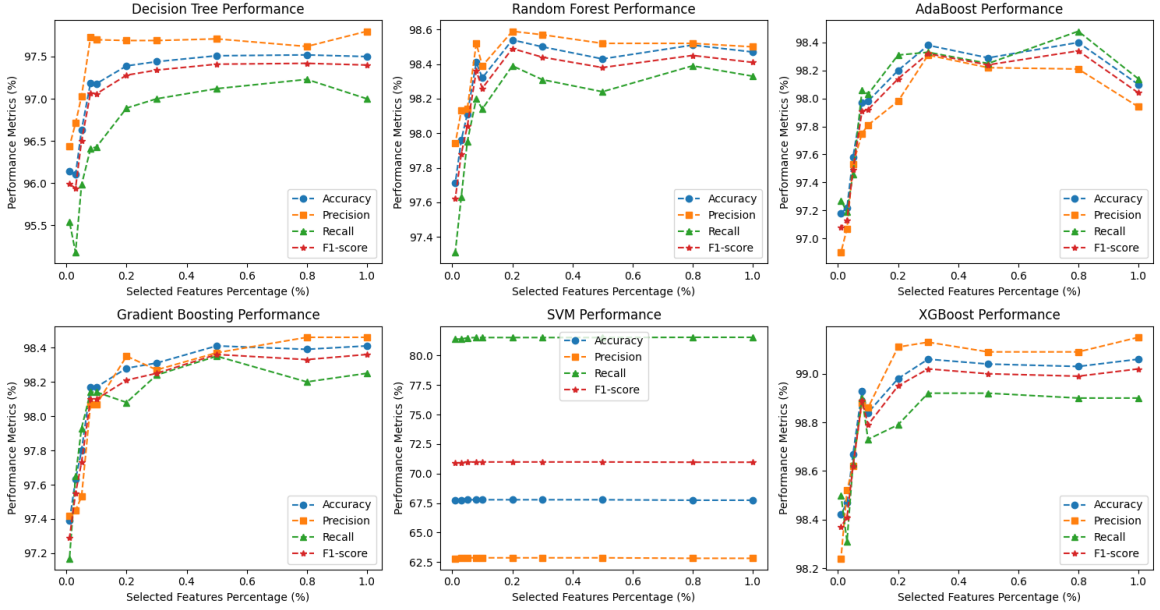


Table C.1: The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the features with byte histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	97.52	97.51	97.44	97.39	97.18	97.19	96.63	96.10	96.14
Precision	97.62	97.71	97.69	97.69	97.70	97.73	97.03	96.71	96.44
Recall	97.23	97.12	97.00	96.89	96.43	96.41	95.98	95.18	95.54
F1-score	97.42	97.41	97.34	97.28	97.06	97.07	96.50	95.94	95.99

these results with the results in Figure 6.3, the results were better after the exclusion of the byte histogram feature set from the experiment. Tables C.1 until C.6 shows the numerical results for the presented graphs.

C.2 Byte Entropy Histogram Inclusion

This section shows the results of the evaluation of the features with the byte entropy histogram feature set included. Figure C.2 shows the results of the experiments.

Table C.2: The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the features with byte histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.51	98.43	98.50	98.54	98.32	98.41	98.11	97.96	97.71
Precision	98.52	98.52	98.57	98.59	98.39	98.52	98.14	98.13	97.94
Recall	98.39	98.24	98.31	98.39	98.14	98.20	97.95	97.63	97.31
F1-score	98.45	98.38	98.44	98.49	98.26	98.36	98.04	97.88	97.62

Table C.3: The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the features with byte histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.40	98.29	98.38	98.20	97.98	97.97	97.58	97.22	97.18
Precision	98.21	98.22	98.31	97.98	97.81	97.75	97.53	97.07	96.90
Recall	98.48	98.25	98.33	98.31	98.03	98.06	97.46	97.19	97.27
F1-score	98.34	98.24	98.32	98.14	97.92	97.91	97.49	97.13	97.08

Table C.4: The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the features with byte histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.39	98.41	98.31	98.28	98.17	98.17	97.80	97.63	97.39
Precision	98.46	98.37	98.27	98.35	98.07	98.07	97.53	97.45	97.42
Recall	98.20	98.35	98.24	98.08	98.14	98.14	97.93	97.65	97.17
F1-score	98.33	98.36	98.25	98.21	98.10	98.10	97.73	97.55	97.29

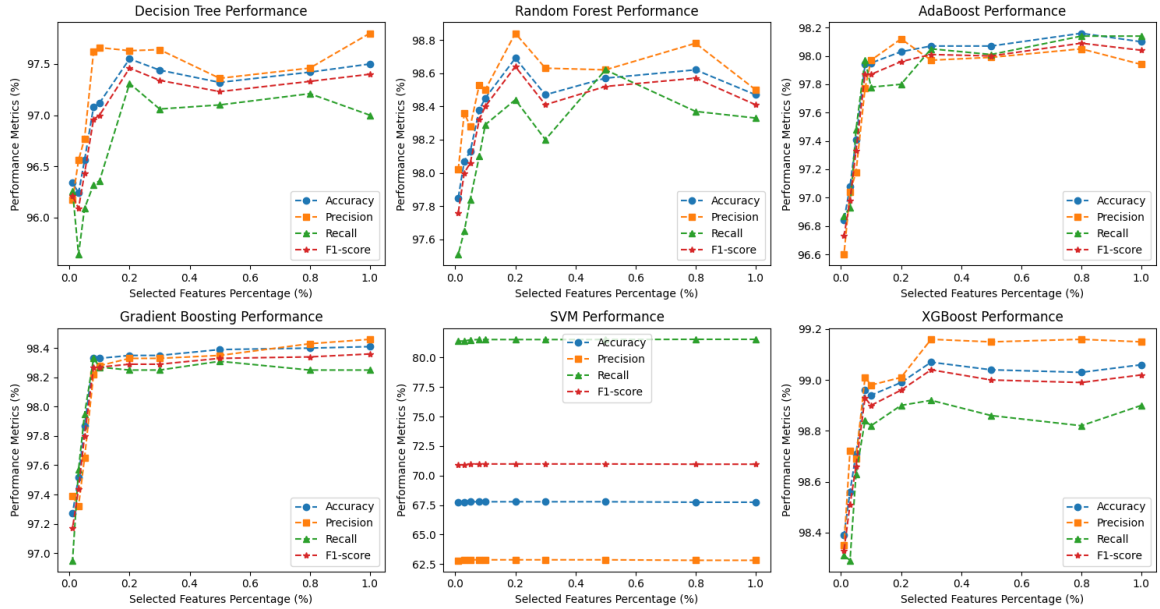
Table C.5: The numerical values for the accuracy, precision, recall, and F1-score of applying the SVM model to evaluate the various IG thresholds for the features with byte histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	67.74	67.74	67.78	67.78	67.78	67.78	67.78	67.72	67.71
Precision	62.82	62.82	62.86	62.86	62.86	62.86	62.86	62.82	62.81
Recall	81.54	81.54	81.52	81.52	81.52	81.52	81.52	81.39	81.39
F1-score	70.96	70.96	70.98	70.98	70.98	70.98	70.98	70.90	70.90

Table C.6: The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the features with byte histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	99.03	99.04	99.06	98.98	98.84	98.93	98.67	98.47	98.42
Precision	99.09	99.09	99.13	99.11	98.86	98.88	98.62	98.52	98.24
Recall	98.90	98.92	98.92	98.79	98.73	98.90	98.63	98.31	98.50
F1-score	98.99	99.00	99.02	98.95	98.79	98.89	98.62	98.41	98.37

Figure C.2: The results of applying IG using the six machine learning models with different percentages of features used at each time including the byte entropy histogram.



Comparing these results with the results in Figure 6.3, the results were better after the exclusion of the byte entropy histogram feature set from the experiment. Tables C.7 until C.12 shows the numerical results for the presented graphs.

Table C.7: The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the features with byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	97.42	97.32	97.44	97.55	97.12	97.08	96.56	96.24	96.34
Precision	97.46	97.36	97.64	97.63	97.66	97.62	96.77	96.56	96.17
Recall	97.21	97.10	97.06	97.31	96.36	96.32	96.09	95.64	96.26
F1-score	97.33	97.23	97.34	97.46	97.00	96.96	96.43	96.09	96.21

Table C.8: The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the features with byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.62	98.57	98.47	98.69	98.45	98.38	98.13	98.07	97.85
Precision	98.78	98.62	98.63	98.84	98.50	98.53	98.28	98.36	98.02
Recall	98.37	98.62	98.20	98.44	98.29	98.10	97.84	97.65	97.51
F1-score	98.57	98.52	98.41	98.64	98.40	98.32	98.06	98.00	97.76

Table C.9: The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the features with byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.16	98.07	98.07	98.03	97.95	97.94	97.41	97.08	96.84
Precision	98.05	97.99	97.97	98.12	97.97	97.77	97.18	97.04	96.60
Recall	98.14	98.01	98.05	97.80	97.78	97.97	97.48	96.93	96.87
F1-score	98.09	98.00	98.01	97.96	97.87	97.87	97.33	96.98	96.73

Table C.10: The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the features with byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.40	98.39	98.35	98.35	98.33	98.33	97.87	97.52	97.27
Precision	98.43	98.35	98.33	98.33	98.28	98.22	97.65	97.32	97.39
Recall	98.25	98.31	98.25	98.25	98.27	98.33	97.95	97.57	96.95
F1-score	98.34	98.33	98.29	98.29	98.27	98.27	97.80	97.44	97.17

Table C.11: The numerical values for the accuracy, precision, recall, and F1-score of applying the SVM model to evaluate the various IG thresholds for the features with byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	67.74	67.74	67.78	67.78	67.78	67.78	67.78	67.72	67.71
Precision	62.82	62.82	62.86	62.86	62.86	62.86	62.86	62.82	62.81
Recall	81.54	81.54	81.52	81.52	81.52	81.52	81.52	81.39	81.39
F1-score	70.96	70.96	70.98	70.98	70.98	70.98	70.98	70.90	70.90

Table C.12: The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the features with byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	99.03	99.04	99.07	98.99	98.94	98.96	98.71	98.56	98.39
Precision	99.16	99.15	99.16	99.01	98.98	99.01	98.69	98.72	98.35
Recall	98.82	98.86	98.92	98.90	98.82	98.84	98.63	98.29	98.31
F1-score	98.99	99.00	99.04	98.96	98.90	98.93	98.66	98.51	98.33

C.3 Byte Histogram and Byte Entropy Histogram

Inclusion

This section shows the results of the evaluation of the features with the byte histogram and the byte entropy histogram feature sets included. Figure C.3 shows the results of the experiments. Comparing these results with the results in Figure 6.3, the results were better after the exclusion of the byte histogram and the byte entropy histogram feature sets from the experiment. Tables C.13 until C.18 shows the numerical results for the presented graphs.

Figure C.3: The results of applying IG using the six machine learning models with different percentages of features used at each time including the byte and byte entropy histograms.

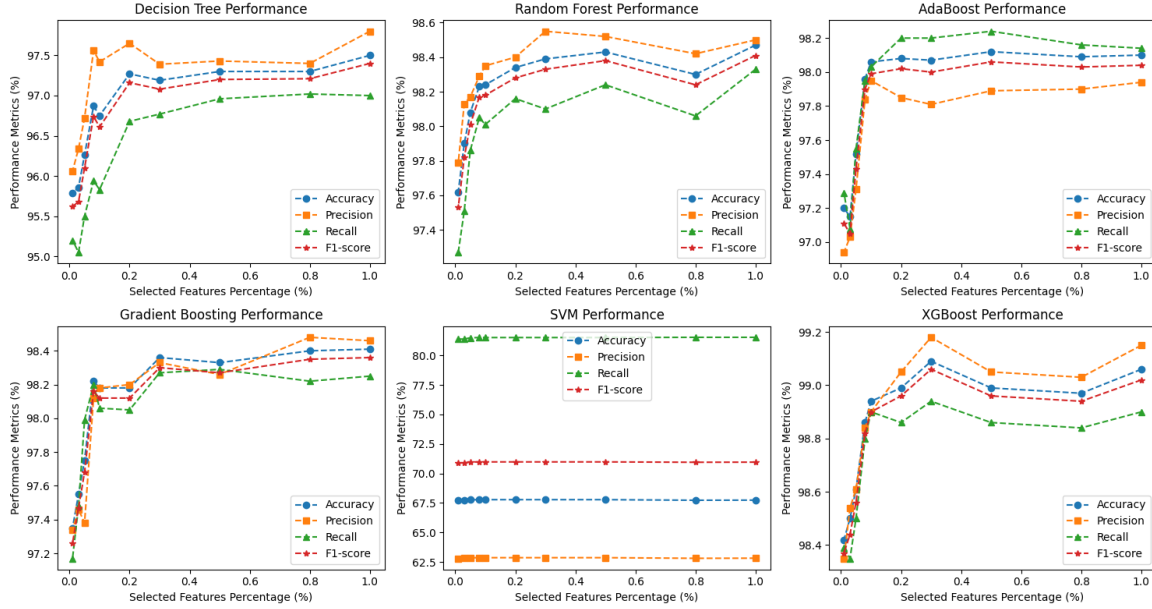


Table C.13: The numerical values for the accuracy, precision, recall, and F1-score of applying the decision tree model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	97.30	97.30	97.19	97.27	96.75	96.87	96.26	95.86	95.79
Precision	97.40	97.43	97.39	97.65	97.42	97.56	96.72	96.34	96.06
Recall	97.02	96.96	96.77	96.68	95.83	95.94	95.50	95.05	95.20
F1-score	97.21	97.20	97.08	97.16	96.61	96.74	96.10	95.68	95.62

Table C.14: The numerical values for the accuracy, precision, recall, and F1-score of applying the random forest model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.30	98.43	98.39	98.34	98.24	98.23	98.08	97.90	97.62
Precision	98.42	98.52	98.55	98.40	98.35	98.29	98.17	98.13	97.79
Recall	98.06	98.24	98.10	98.16	98.01	98.05	97.86	97.51	97.27
F1-score	98.24	98.38	98.33	98.28	98.18	98.17	98.01	97.82	97.53

Table C.15: The numerical values for the accuracy, precision, recall, and F1-score of applying the Adaboost model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.

Model	0.8	0.5	0.3	0.2	0.1	0.08	0.05	0.03	0.01
Accuracy	98.09	98.12	98.07	98.08	98.06	97.96	97.52	97.15	97.20
Precision	97.90	97.89	97.81	97.85	97.95	97.84	97.31	97.03	96.94
Recall	98.16	98.24	98.20	98.20	98.03	97.95	97.55	97.08	97.29
F1-score	98.03	98.06	98.00	98.02	97.99	97.90	97.43	97.05	97.11

Table C.16: The numerical values for the accuracy, precision, recall, and F1-score of applying the gradient boosting model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.

Model	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.05
Accuracy	98.40	98.33	98.36	98.18	98.18	98.22	97.75	97.55	97.35
Precision	98.48	98.26	98.33	98.20	98.18	98.12	97.38	97.46	97.34
Recall	98.22	98.29	98.27	98.05	98.06	98.20	97.99	97.48	97.17
F1-score	98.35	98.27	98.30	98.12	98.12	98.16	97.68	97.47	97.26

Table C.17: The numerical values for the accuracy, precision, recall, and F1-score of applying SVM model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.

Model	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.05
Accuracy	67.73	67.78	67.78	67.78	67.78	67.78	67.77	67.72	67.70
Precision	62.81	62.86	62.86	62.86	62.86	62.86	62.85	62.82	62.80
Recall	81.54	81.52	81.52	81.52	81.52	81.52	81.50	81.39	81.39
F1-score	70.95	70.98	70.98	70.98	70.98	70.98	70.97	70.90	70.89

Table C.18: The numerical values for the accuracy, precision, recall, and F1-score of applying the XGBoost model to evaluate the various IG thresholds for the features with byte histogram and byte entropy histogram.

Model	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.05
Accuracy	98.97	98.99	99.09	98.99	98.94	98.86	98.61	98.50	98.42
Precision	99.03	99.05	99.18	99.05	98.90	98.84	98.61	98.54	98.35
Recall	98.84	98.86	98.94	98.86	98.90	98.80	98.50	98.35	98.39
F1-score	98.94	98.96	99.06	98.96	98.90	98.82	98.56	98.44	98.37

REFERENCES

- [1] “Portable Executable,” Jan. 2024, page Version ID: 1194517052. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Portable_Executable&oldid=1194517052
- [2] “Malware - Statistics & Facts.” [Online]. Available: <https://www.statista.com/topics/8338/malware/>
- [3] N. Palatty, “30+ Malware Statistics You Need To Know In 2023,” May 2023, section: Security Audit. [Online]. Available: <https://www.getastra.com/blog/security-audit/malware-statistics/>
- [4] “McAfee.” [Online]. Available: <https://www.mcafee.com/>
- [5] “Fortinet.” [Online]. Available: <https://www.fortinet.com/>
- [6] H. Darabian, S. Homayounoot, A. Dehghantanha, S. Hashemi, H. Karimipour, R. M. Parizi, and K.-K. R. Choo, “Detecting Cryptomining Malware: a Deep Learning Approach for Static and Dynamic Analysis,” *Journal of Grid Computing*, vol. 18, no. 2, pp. 293–303, Jun. 2020.

- [7] T.-N. To, H. D. Hoang, P. T. Duy, and V.-H. Pham, “MalDEX: An Explainable Malware Detection System Based on Ensemble Learning,” in *2023 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, Oct. 2023, pp. 1–6, iSSN: 2770-6850.
- [8] T. M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, and B. S. Manjunath, “Malware Detection Using Frequency Domain-Based Image Visualization and Deep Learning,” Jan. 2021, arXiv:2101.10578 [cs].
- [9] R. Korine and D. Hendler, “DAEMON: Dataset/Platform-Agnostic Explainable Malware Classification Using Multi-Stage Feature Mining,” *IEEE Access*, vol. 9, pp. 78 382–78 399, 2021.
- [10] I. Almomani, A. Alkhayer, and W. El-Shafai, “An Automated Vision-Based Deep Learning Model for Efficient Detection of Android Malware Attacks,” *IEEE Access*, vol. 10, pp. 2700–2720, 2022, conference Name: IEEE Access.
- [11] M. Nachtigall, L. Nguyen Quang Do, and E. Bodden, “Explaining Static Analysis - A Perspective,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, Nov. 2019, pp. 29–32, iSSN: 2151-0830.
- [12] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, “Dynamic Malware Analysis in the Modern Era—A State of the Art Survey,” *ACM Computing Surveys*, vol. 52, no. 5, pp. 88:1–88:48, Sep. 2019.

- [13] J. A. Royo, M. S. Rubio, W. Fuertes, M. C. Cuervo, C. A. Estrada, and T. Toulkeridis, “Malware Security Evasion Techniques: An Original Keylogger Implementation,” in *Trends and Applications in Information Systems and Technologies*, ser. Advances in Intelligent Systems and Computing, J. Rocha, H. Adeli, G. Dzemyda, F. Moreira, and A. M. Ramalho Correia, Eds. Cham: Springer International Publishing, 2021, pp. 375–384.
- [14] K. Brezinski and K. Ferens, “Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits,” *Security and Communication Networks*, vol. 2023, p. e8227751, Sep. 2023, publisher: Hindawi.
- [15] C. Catalano, A. Chezzi, M. Angelelli, and F. Tommasi, “Deceiving AI-based malware detection through polymorphic attacks,” *Computers in Industry*, vol. 143, p. 103751, Dec. 2022.
- [16] R. Labaca-Castro, S. Franz, and G. D. Rodosek, “AIMED-RL: Exploring Adversarial Malware Examples with Reinforcement Learning,” in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*, ser. Lecture Notes in Computer Science, Y. Dong, N. Kourtellis, B. Hammer, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 37–52.
- [17] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, “Functionality-preserving Black-box Optimization of Adversarial Windows Malware,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3469–3478, 2021, arXiv:2003.13526 [cs].

- [18] J. Burr and S. Xu, “Improving Adversarial Attacks Against Executable Raw Byte Classifiers,” in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, May 2021, pp. 1–2.
- [19] H. Bostani and V. Moonsamy, “EvadeDroid: A Practical Evasion Attack on Machine Learning for Black-box Android Malware Detection,” Jun. 2023, arXiv:2110.03301 [cs].
- [20] T.-N. To, D. L. Kim, D. T. T. Hien, N. H. Khoa, H. D. Hoang, P. T. Duy, and V.-H. Pham, “On the Effectiveness of Adversarial Samples against Ensemble Learning-based Windows PE Malware Detectors,” Sep. 2023, arXiv:2309.13841 [cs].
- [21] “What Is Malware? - Definition and Examples.” [Online]. Available: <https://www.cisco.com/site/us/en/learn/topics/security/what-is-malware.html>
- [22] “What is malware and how cybercriminals use it.” [Online]. Available: <https://www.mcafee.com/en-us/antivirus/malware.html>
- [23] L. Wang, D. Xu, J. Ming, Y. Fu, and D. Wu, “MetaHunt: Towards Taming Malware Mutation via Studying the Evolution of Metamorphic Virus,” in *Proceedings of the 3rd ACM Workshop on Software Protection*, ser. SPRO’19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 15–26.
- [24] H. Salazar and C. Barria, “Classification and Update Proposal for Modern Computer Worms, Based on Obfuscation,” in *ITNG 2021 18th International Conference on Information Technology-New Generations*, ser. Advances in Intelligent

- Systems and Computing, S. Latifi, Ed. Cham: Springer International Publishing, 2021, pp. 49–57.
- [25] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, “A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions,” *ACM Computing Surveys*, vol. 54, no. 11s, pp. 238:1–238:37, Sep. 2022.
- [26] S. O’Shaughnessy and F. Breitingner, “Malware family classification via efficient Huffman features,” *Forensic Science International: Digital Investigation*, vol. 37, p. 301192, Jul. 2021.
- [27] C. Gao, M. Cai, S. Yin, G. Huang, H. Li, W. Yuan, and X. Luo, “Obfuscation-Resilient Android Malware Analysis Based on Complementary Features,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 5056–5068, 2023.
- [28] J. Zhang, Z. Qin, H. Yin, L. Ou, and Y. Hu, “IRMD: Malware Variant Detection Using Opcode Image Recognition,” in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2016, pp. 1175–1180, iSSN: 1521-9097.
- [29] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, “Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1103–1112, May 2017.

- [30] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, “IM-CFN: Image-based malware classification using fine-tuned convolutional neural network architecture,” *Computer Networks*, vol. 171, p. 107138, Apr. 2020.
- [31] O. Kargarnovin, A. M. Sadeghzadeh, and R. Jalili, “Mal2GCN: a robust malware detection approach using deep graph convolutional networks with non-negative weights,” *Journal of Computer Virology and Hacking Techniques*, Sep. 2023.
- [32] S. K. J. Rizvi, W. Aslam, M. Shahzad, S. Saleem, and M. M. Fraz, “PROUD-MAL: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable,” *Complex & Intelligent Systems*, vol. 8, no. 1, pp. 673–685, Feb. 2022.
- [33] S. Jeon and J. Moon, “Malware-Detection Method with a Convolutional Recurrent Neural Network Using Opcode Sequences,” *Information Sciences*, vol. 535, pp. 1–15, Oct. 2020.
- [34] P. M. Anand, P. Sai Charan, and S. K. Shukla, “A Comprehensive API Call Analysis for Detecting Windows-Based Ransomware,” in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, Jul. 2022, pp. 337–344.
- [35] Srastika, N. Bhandary, S. R. S, P. Honnavalli, and S. E, “An Enhanced Malware Detection Approach using Machine Learning and Feature Selection,” in *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Aug. 2022, pp. 909–914.

- [36] A. Sharma, S. Mohanty, and M. R. Islam, “An Experimental Analysis on Malware Detection in Executable Files using Machine Learning,” in *2021 8th International Conference on Smart Computing and Communications (ICSCC)*, Jul. 2021, pp. 178–182.
- [37] A. Irshad, R. Maurya, M. K. Dutta, R. Burget, and V. Uher, “Feature Optimization for Run Time Analysis of Malware in Windows Operating System using Machine Learning Approach,” in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, Jul. 2019, pp. 255–260.
- [38] A. Walker, R. M. Shukla, T. Das, and S. Sengupta, “Runs in the Family: Malware Family Variants Identification through API Sequence and Frequency Analysis,” in *2023 International Conference on Multimedia Computing, Networking and Applications (MCNA)*, Jun. 2023, pp. 55–61.
- [39] B. Bokolo, R. Jinad, and Q. Liu, “A Comparison Study to Detect Malware using Deep Learning and Machine learning Techniques,” in *2023 IEEE 6th International Conference on Big Data and Artificial Intelligence (BDAI)*, Jul. 2023, pp. 1–6.
- [40] A. Duby, T. Taylor, G. Bloom, and Y. Zhuang, “Detecting and Classifying Self-Deleting Windows Malware Using Prefetch Files,” in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2022, pp. 0745–0751.

- [41] M. Onoja, A. Jegede, J. Mazadu, G. Aimufua, A. Oyedele, and K. Olibodum, “Exploring the Effectiveness and Efficiency of LightGBM Algorithm for Windows Malware Detection,” in *2022 5th Information Technology for Education and Development (ITED)*, Nov. 2022, pp. 1–6.
- [42] A. Duby, T. Taylor, G. Bloom, and Y. Zhuang, “Evaluating Feature Robustness for Windows Malware Family Classification,” in *2022 International Conference on Computer Communications and Networks (ICCCN)*, Jul. 2022, pp. 1–10, iISSN: 2637-9430.
- [43] A. M. Radwan, “Machine Learning Techniques to Detect Maliciousness of Portable Executable Files,” in *2019 International Conference on Promising Electronic Technologies (ICPET)*, Oct. 2019, pp. 86–90.
- [44] X. Chen, Z. Hao, L. Li, L. Cui, Y. Zhu, Z. Ding, and Y. Liu, “CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 788–803, 2022.
- [45] E. Gandotra, D. Bansal, and S. Sofat, “Malware Analysis and Classification: A Survey,” *Journal of Information Security*, vol. 2014, Feb. 2014, publisher: Scientific Research Publishing.
- [46] M. Chikapa and A. P. Namanya, “Towards a Fast Off-Line Static Malware Analysis Framework,” in *2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, Aug. 2018, pp. 182–187.

- [47] T. Ball, “The concept of dynamic analysis,” *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 6, pp. 216–234, Nov. 1999.
- [48] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” *ACM Computing Surveys*, vol. 44, no. 2, pp. 6:1–6:42, Mar. 2008.
- [49] K. Bridge, “PE Format - Win32 apps,” Mar. 2023. [Online]. Available: <https://learn.microsoft.com/en-us/windows/win32/debug/pe-format>
- [50] M. Pietrek, “Inside Windows: Win32 Portable Executable File Format in Detail,” Oct. 2019. [Online]. Available: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2002/february/inside-windows-win32-portable-executable-file-format-in-detail>
- [51] A. Qasem, P. Shirani, M. Debbabi, L. Wang, B. Lebel, and B. L. Agba, “Automatic Vulnerability Detection in Embedded Devices and Firmware: Survey and Layered Taxonomies,” *ACM Computing Surveys*, vol. 54, no. 2, pp. 25:1–25:42, Mar. 2021.
- [52] M. Jureček and R. Lórencz, “Application of Distance Metric Learning to Automated Malware Detection,” *IEEE Access*, vol. 9, pp. 96 151–96 165, 2021.
- [53] A. Brown, M. Gupta, and M. Abdelsalam, “Automated machine learning for deep learning based malware detection,” *Computers & Security*, vol. 137, p. 103582, Feb. 2024.

- [54] A. Kamboj, P. Kumar, A. K. Bairwa, and S. Joshi, “Detection of malware in downloaded files using various machine learning models,” *Egyptian Informatics Journal*, vol. 24, no. 1, pp. 81–94, Mar. 2023.
- [55] S. Park, S. Jeon, and H. K. Kim, “HMLET: Hunt Malware Using Wavelet Transform on Cross-Platform,” *IEEE Access*, vol. 10, pp. 124 821–124 834, 2022.
- [56] Y.-H. Chen, S.-C. Lin, S.-C. Huang, C.-L. Lei, and C.-Y. Huang, “Guided Malware Sample Analysis Based on Graph Neural Networks,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4128–4143, 2023.
- [57] E. Amer and I. Zelinka, “A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence,” *Computers & Security*, vol. 92, p. 101760, May 2020.
- [58] D. Javaheri, P. Lalbakhsh, and M. Hosseinzadeh, “A Novel Method for Detecting Future Generations of Targeted and Metamorphic Malware Based on Genetic Algorithm,” *IEEE Access*, vol. 9, pp. 69 951–69 970, 2021.
- [59] C. Li, Z. Cheng, H. Zhu, L. Wang, Q. Lv, Y. Wang, N. Li, and D. Sun, “DMal-Net: Dynamic malware analysis based on API feature engineering and graph learning,” *Computers & Security*, vol. 122, p. 102872, Nov. 2022.
- [60] Z. Zhang, P. Qi, and W. Wang, “Dynamic Malware Analysis with Feature Engineering and Feature Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 1210–1217, Apr. 2020, number: 01.

- [61] W. Qiang, L. Yang, and H. Jin, “Efficient and Robust Malware Detection Based on Control Flow Traces Using Deep Neural Networks,” *Computers & Security*, vol. 122, p. 102871, Nov. 2022.
- [62] K. Shaukat, S. Luo, and V. Varadharajan, “A novel deep learning-based approach for malware detection,” *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106030, Jun. 2023.
- [63] ———, “A novel machine learning approach for detecting first-time-appeared malware,” *Engineering Applications of Artificial Intelligence*, vol. 131, p. 107801, May 2024.
- [64] X. Huang, L. Ma, W. Yang, and Y. Zhong, “A Method for Windows Malware Detection Based on Deep Learning,” *Journal of Signal Processing Systems*, vol. 93, no. 2, pp. 265–273, Mar. 2021.
- [65] S. Kumar and B. Janet, “Distinguishing malicious programs based on visualization and hybrid learning algorithms,” *Computer Networks*, vol. 201, p. 108595, Dec. 2021.
- [66] “Petite - Win32 Executable Compressor.” [Online]. Available: <https://www.un4seen.com/petite/>
- [67] “UPX: the Ultimate Packer for eXecutables - Homepage.” [Online]. Available: <https://upx.github.io/>
- [68] “ASPack Software - Application for compression, packing and protection of software.” [Online]. Available: <http://www.aspack.com/>

- [69] “Yoda’s Protector,” Apr. 2013. [Online]. Available: <https://sourceforge.net/projects/yodap/>
- [70] Hors, “Detect It Easy,” Dec. 2023, original-date: 2014-06-01T21:37:32Z. [Online]. Available: <https://github.com/horsicq/Detect-It-Easy>
- [71] “ExeStealth Protector - Software Copy Protection Tool for EXE, PDF, SWF, SCR, TXT and RTF files and documents.” [Online]. Available: <https://www.webtoolmaster.com/exestealth.htm>
- [72] B. Jung, S. I. Bae, C. Choi, and E. G. Im, “Packer identification method based on byte sequences,” *Concurrency and Computation: Practice and Experience*, vol. 32, no. 8, p. e5082, 2020.
- [73] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, “A Systematical and longitudinal study of evasive behaviors in windows malware,” *Computers & Security*, vol. 113, p. 102550, Feb. 2022.
- [74] X. Ling, L. Wu, J. Zhang, Z. Qu, W. Deng, X. Chen, Y. Qian, C. Wu, S. Ji, T. Luo, J. Wu, and Y. Wu, “Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art,” *Computers & Security*, vol. 128, p. 103134, May 2023.
- [75] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli, “Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection,” *ACM Transactions on Privacy and Security*, vol. 24, no. 4, pp. 27:1–27:31, Sep. 2021.

- [76] X. Li and Q. Li, “An IRL-based malware adversarial generation method to evade anti-malware engines,” *Computers & Security*, vol. 104, p. 102118, May 2021.
- [77] T. Quertier, B. Marais, S. Morucci, and B. Fournel, “MERLIN – Malware Evasion with Reinforcement LearnINg,” Mar. 2022, arXiv:2203.12980 [cs].
- [78] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, “Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes,” in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, May 2021, pp. 744–758, arXiv:1912.09064.
- [79] L. Saidia Fascí, M. Fisichella, G. Lax, and C. Qian, “Disarming visualization-based approaches in malware detection systems,” *Computers & Security*, vol. 126, p. 103062, Mar. 2023.
- [80] D. Li and Q. Li, “Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3886–3900, 2020.
- [81] P. Kishore, S. K. Barisal, and D. P. Mohapatra, “An incremental malware detection model for meta-feature API and system call sequence,” in *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2020, pp. 629–638.
- [82] Y. Zhang, Z. Liu, and Y. Jiang, “The Classification and Detection of Malware Using Soft Relevance Evaluation,” *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 309–320, Mar. 2022.

- [83] X. Ling, L. Wu, W. Deng, Z. Qu, J. Zhang, S. Zhang, T. Ma, B. Wang, C. Wu, and S. Ji, “MalGraph: Hierarchical Graph Neural Networks for Robust Windows Malware Detection,” in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022, pp. 1998–2007, iSSN: 2641-9874.
- [84] M. Ficco, “Malware Analysis by Combining Multiple Detectors and Observation Windows,” *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1276–1290, Jun. 2022.
- [85] D. Baig, M. U. Khan, D. Dancey, A. Abbas, M. Ali, and R. Nawaz, “Malware Detection and Classification along with Trade-off Analysis for Number of Features, Feature Types, and Speed,” in *2021 International Conference on Frontiers of Information Technology (FIT)*, Dec. 2021, pp. 246–251.
- [86] S. Tyagi, A. Baghela, K. M. Dar, A. Patel, S. Kothari, and S. Bhosale, “Malware Detection in PE files using Machine Learning,” in *2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON)*, Feb. 2023, pp. 1–6.
- [87] Namita, Prachi, and P. Sharma, “Windows Malware Detection using Machine Learning and TF-IDF Enriched API Calls Information,” in *2022 Second International Conference on Computer Science, Engineering and Applications (ICC-SEA)*, Sep. 2022, pp. 1–6.
- [88] F.-Q. Li, S.-L. Wang, A. W.-C. Liew, W. Ding, and G.-S. Liu, “Large-Scale Malicious Software Classification With Fuzzified Features and Boosted Fuzzy

- Random Forest,” *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 11, pp. 3205–3218, Nov. 2021.
- [89] H. S. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” Apr. 2018, arXiv:1804.04637.
- [90] “sklearn.feature_extraction.FeatureHasher.” [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.feature_extraction.FeatureHasher.html
- [91] “Elastic Malware Benchmark for Empowering Researchers,” Dec. 2023, original-date: 2018-04-11T17:48:15Z. [Online]. Available: <https://github.com/elastic/ember>
- [92] “Lief project,” Dec. 2023, original-date: 2017-03-16T14:34:53Z. [Online]. Available: <https://github.com/lief-project/LIEF>
- [93] G. Biau and E. Scornet, “A random forest guided tour,” *TEST*, vol. 25, no. 2, pp. 197–227, Jun. 2016. [Online]. Available: <https://doi.org/10.1007/s11749-016-0481-7>
- [94] “sklearn.ensemble.RandomForestClassifier.” [Online]. Available: <https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- [95] R. E. Schapire, “Explaining AdaBoost,” in *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, B. Schölkopf, Z. Luo, and V. Vovk,

- Eds. Berlin, Heidelberg: Springer, 2013, pp. 37–52. [Online]. Available: https://doi.org/10.1007/978-3-642-41136-6_5
- [96] “sklearn.ensemble.AdaBoostClassifier.” [Online]. Available: <https://scikit-learn/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>
- [97] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, “A comparative analysis of gradient boosting algorithms,” *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1937–1967, Mar. 2021. [Online]. Available: <https://doi.org/10.1007/s10462-020-09896-5>
- [98] “sklearn.ensemble.GradientBoostingClassifier.” [Online]. Available: <https://scikit-learn/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>
- [99] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 785–794. [Online]. Available: <https://dl.acm.org/doi/10.1145/2939672.2939785>
- [100] “XGBoost Documentation — xgboost 2.0.3 documentation.” [Online]. Available: <https://xgboost.readthedocs.io/en/stable/>
- [101] S. Suthaharan, “Support Vector Machine,” in *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective*

Learning, ser. Integrated Series in Information Systems, S. Suthaharan, Ed. Boston, MA: Springer US, 2016, pp. 207–235. [Online]. Available: https://doi.org/10.1007/978-1-4899-7641-3_9

[102] “VirusShare.com.” [Online]. Available: <https://virusshare.com/>

[103] “MalwareBazaar | Browse malware samples.” [Online]. Available: <https://bazaar.abuse.ch/browse/>

[104] “Free Software Downloads and Reviews for Windows, Android, Mac, and iOS – CNET Download.” [Online]. Available: <https://download.cnet.com/>

[105] “VirusTotal.” [Online]. Available: <https://www.virustotal.com/gui/home/upload>

VITAE

- Name: Raed Farouq Katib
- Nationality: Saudi
- Date of Birth: 18/09/1999
- Email: raeed.katib@gmail.com
- Permanent Address: 7870 Al Bahriyyah St., Al Awali Destrict, 24372 Makkah, Saudi Arabia
- M.S. in Security and Information Assurance from King Fahd University of Petroleum and Minerals
- B.S. in Information Science and Technology from Pennsylvania State University
- Research interest: Interested in applying machine and deep learning in solving information security problems.
- Research Papers released: A Federated Learning Approach to Banking Loan Decisions, Authors: Farag Azzedin, Mustafa Ghaleb, Yasser El-Alfy, Raed Katib, M. D. Hossain. Date: 2023