ANKARA YILDIRIM BEYAZIT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MALWARE DETECTION USING MACHINE LEARNING AND EVOLUTIONARY ALGORITHMS

Ph.D. Thesis by

Gülsade KALE

Department of Computer Engineering

July, 2024

ANKARA

MALWARE DETECTION USING MACHINE LEARNING AND EVOLUTIONARY ALGORITHMS

A Thesis Submitted to

The Graduate School of Natural and Applied Sciences of

Ankara Yıldırım Beyazıt University

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering, Department of Computer Engineering

by

Gülsade KALE

July, 2024

ANKARA

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MALWARE DETECTION USING MACHINE LEARNING AND EVOLUTIONARY ALGORITHMS"** completed by **GÜLSADE KALE** under the supervision of **PROF. DR. FATİH VEHBİ ÇELEBİ** and **ASSOC. PROF. DR. GAZİ ERKAN BOSTANCI** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Fatih Vehbi ÇELEBİ
Supervisor

Assoc. Prof. Dr. Gazi Erkan BOSTANCI
Co-Supervisor

Asst. Prof. Dr. Mustafa YENİAD
Jury Member

Assoc. Prof. Dr. Murat AYDOS
Jury Member

Assoc. Prof. Dr. Serdar GÜZEL
Jury Member

Asst. Prof. Dr. Koray AÇICI
Jury Member

Prof. Dr. Sadettin ORHAN
Director
Graduate School of Natural and Applied Sciences

ETHICAL DECLARATION

I hereby declare that, in this thesis which has been prepared in accordance with the Thesis Writing Manual of Graduate School of Natural and Applied Sciences,

- All data, information and documents are obtained in the framework of academic and ethical rules,
- All information, documents and assessments are presented in accordance with scientific ethics and morals,
- All the materials that have been utilized are fully cited and referenced,
- No change has been made on the utilized materials,
- All the works presented are original,

and in any contrary case of above statements, I accept to renounce all my legal rights.

Date: 16/07/2024 **Signature:**

Name & Surname: **Gülsade KALE**

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisors, Prof. Dr. Fatih Vehbi ÇELEBİ and Assoc. Prof. Dr. Gazi Erkan BOSTANCI, for their invaluable guidance, advice, support, and motivation throughout my Ph.D. studies. Their mentorship has been instrumental in shaping my research and academic growth.

I am also deeply thankful to all the members of my jury for their insightful suggestions and constructive feedback, which have significantly enriched my work. My sincere thanks go to my friends and colleagues for their continuous encouragement and support.

Above all, I wish to convey my boundless appreciation to my family, especially my mother Şenel KALE, my father İbrahim KALE, and all my dear siblings, whose unwavering support and motivation have been a constant source of strength throughout the duration of my Ph.D. journey and my entire life. Their belief in me has been fundamental to my achievements.

2024, 16 July **Gülsade KALE**

MALWARE DETECTION USING MACHINE LEARNING AND EVOLUTIONARY ALGORITHMS

ABSTRACT

As cyber threats grow in complexity and frequency, the need for robust and adaptive malware detection mechanisms is critical due to their impact on national security and economic stability. Given malware's evolving nature to evade detection, selecting effective parameters by examining the interactions between malware characteristics is crucial. Traditional signature-based systems are becoming inadequate as malware adapts to new technologies. To address this, a new system is developed in this thesis that focuses on malware behavior and feature relationships. Multi-objective genetic algorithms (MOGAs) are employed to identify critical features for detection, which are then utilized by machine learning (ML) algorithms within a hybrid framework to accurately detect and classify malware.

The objective of this thesis is to determine the optimal feature selection and classification methods that yield the highest accuracy within the Cuckoo Sandbox environment. Specifically, classifiers such as the J48 Decision Tree (J48), Reduced Error Pruning Tree (REP Tree), Adaptive Boosting Model 1 (AdaboostM1), Multilayer Perceptron (MLP), and Naive Bayes (NB) were evaluated. As a result of the analysis, the feature set was reduced from 335 to 200 by considering the relationships between features, resulting in a high accuracy of 93.33% and a performance improvement of 40% due to the reduction in the number of features. The obtained metrics were meticulously compared and evaluated with respect to the employed algorithms and methodologies. Furthermore, Mc Nemar’s test was utilized to assess the performance of different malware detection classifiers by comparing their correct and incorrect classifications. The results from Mc Nemar’s test indicated significant improvements, highlighting the effectiveness of the proposed system.

Keywords: Malware analysis, malware classification, feature selection, machine learning, multi-objective genetic algorithms.

MAKİNE ÖĞRENMESİ VE GENETİK ALGORİTMALAR KULLANILARAK ZARARLI YAZILIM TESPİTİ

ÖZET

Siber tehditlerin karmaşıklığı ve sıklığı arttıkça, ulusal güvenlik ve ekonomik istikrar üzerindeki etkileri nedeniyle sağlam ve uyarlanabilir kötü amaçlı yazılım tespit mekanizmalarına duyulan ihtiyaç kritik önem taşımaktadır. Kötü amaçlı yazılımların tespitten kaçmak için gelişen doğası göz önüne alındığında, kötü amaçlı yazılım özellikleri arasındaki etkileşimleri inceleyerek etkili parametrelerin seçilmesi çok önemlidir. Kötü amaçlı yazılımlar yeni teknolojilere uyum sağladıkça geleneksel imza tabanlı sistemler yetersiz kalmaktadır. Bunu ele almak için, bu tezde kötü amaçlı yazılım davranışına ve özellik ilişkilerine odaklanan yeni bir sistem geliştirilmiştir. Tespit için kritik özellikleri belirlemek üzere çok amaçlı genetik algoritmalar (MOGA'lar) kullanılmakta ve bunlar daha sonra kötü amaçlı yazılımları doğru bir şekilde tespit etmek ve sınıflandırmak için hibrit bir çerçeve içinde makine öğrenimi (ML) algoritmaları tarafından kullanılmaktadır.

Bu tezin amacı, Cuckoo Sandbox ortamında en yüksek doğruluğu sağlayan en uygun özellik seçimi ve sınıflandırma yöntemlerini belirlemektir. Özellikle, J48 Karar Ağacı (J48), Azaltılmış Hata Budama Ağacı (REP Ağacı), Uyarlamalı Güçlendirme Modeli 1 (AdaboostM1), Çok Katmanlı Algılayıcı (MLP) ve Naive Bayes (NB) gibi sınıflandırıcılar değerlendirilmiştir. Analiz sonucunda, özellikler arasındaki ilişkiler göz önünde bulundurularak özellik seti 335'ten 200'e indirilmiş, %93,33 gibi yüksek bir doğruluk oranı ve özellik sayısındaki azalmaya bağlı olarak %40'lık bir performans artışı elde edilmiştir. Elde edilen metrikler, kullanılan algoritmalar ve metodolojiler açısından titizlikle karşılaştırılmış ve değerlendirilmiştir. Ayrıca, farklı kötü amaçlı yazılım tespit sınıflandırıcılarının doğru ve yanlış sınıflandırmalarını karşılaştırarak performanslarını değerlendirmek için Mc Nemar's testi kullanılmıştır. Mc Nemar's testinden elde edilen sonuçlar, önerilen sistemin etkinliğini vurgulayan önemli gelişmeler göstermiştir.

Anahtar Kelimeler: Kötü amaçlı yazılım analizi, kötü amaçlı yazılım sınıflandırması, özellik seçimi, makine öğrenmesi, çok amaçlı genetik algoritmalar.

CONTENTS

NOMENCLATURE

Abbreviations

FN	False Negative
FP	False Positive
GHz	Gigahertz
TN	True Negative
TP	True Positive
USD	United States Dollars

Acronyms

AdaboostM1	Adaptive Boosting Model 1
ANN	Artificial Neural Network
ANUBIS	Extended Analysis of Network and Host Behavior in Internet Security
API	Application Programming Interface
AV	Antivirüs
BDT	Boosted Decision Trees
BN	Bayesian Network
BSA	Buster Sandbox Analyzer
CD	Compact Disc
CNN	Convolutional Neural Network
DBEA	Decomposition-Based Evolutionary Algorithm
DF	Document Frequency
DLL	Dynamic Link Library
DNS	Domain Name Server
DT	Decision Tree
FS	Fisher Score
FT	Functional Tree
GA	Genetic Algorithm
GR	Gain Ratio
ID3	Iterative Dichotomiser 3
IDS	Intrusion Detection System
IG	Information Gain
IP	Internet Protocol

J48	J48 Decision Tree
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbor
LMT	Logistic Model Trees
MAE	Mean Absolute Error
MALGENE	Malware Generation
MD5	Message Digest 5
ML	Machine Learning
MLP	Multilayer Perceptron
MOEA	Multi Objective Evolutionary Algorithm
MOGA	Multi Objective Genetic Algorithm
NB	Naive Bayes
NBT	Naive Bayes Tree
OS	Operating System
PAES	Pareto Archieved Evolution Strategy
PCAP	Packet Capture
PDF	Portable Document Format
PE	Portable Executable
PIN	Personal Identification Number
RANDOM	A Simple Random Algorithm, Often Used As A Baseline
ReLU	Rectified Linear Unit
REPTree	Reduced Error Pruning Tree
RF	Random Forest
RGB	Red Green Blue
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RSO	Reactive Search Optimization
RVEA	Reference Vector Guided Evolutionary Algorithm
SHA1	Secure Hashing Algorithm 1
SVM	Support Vector Machine
URL	Uniform Resource Loader
USB	Universal Serial Bus
VB	Visual Basic
WEKA	Waikato Environment for Knowledge Analysis

xGBoost	eXtreme Gradient Boosting

Greek Letter Symbol

K	Kappa

Latin Letter Symbols

H0	Null Hypothesis
H1	Alternative Hypothesis
Z	Z Score

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the development of technology in the world and the spread of the internet, globalization has spread to every field and the transition to the information economy has made information, the greatest power, our most valuable asset. Therefore, it becomes a necessity to ensure that information and communication technologies and information systems used for the production, processing, analysis, distribution and storage of information are well protected. Especially the security of critical infrastructures is of national and international importance. Nowadays, attacks on systems, personal and corporate data, applications and critical infrastructures are increasing and becoming more professional. The building blocks of these attacks are malware. At this point, it is a necessity to detect and prevent malware that causes great damage to the world economy. Based on projections from Cybersecurity Ventures, it is anticipated that the worldwide annual expenditure associated with cybercrime will escalate to $9.5 trillion United States Dollar (USD) in the year 2024. Concurrently, the escalating financial toll stemming from cybercrime-induced damages is forecasted to elevate to $10.5 trillion USD by the year 2025 [1].

The malware threat is a form of malicious code designed to cause harm. Its primary purpose is to breach user privacy and steal sensitive information. As such, the use of malware in nefarious activities is constantly evolving. The most significant threat lies in the security researchers, who are of high priority concern.

The conventional methods utilized in malware analysis such as signature-based and static malware analysis necessitate the involvement of human analysts for manual inspection of the malicious code. However, the exponential increase in the production of malicious code has resulted in antivirus vendors encountering thousands of new malware files on a daily basis [2].

The examination of vast quantities of documents using this method is difficult due to obfuscation, polymorphism and other factors. Therefore, it is vital to have a reliable

and automated analysis capability in order to combat this danger. The proliferation of advanced tools has led to a rise in cyber threats that are highly specific, persistent and difficult to detect. These new-generation attacks employ sophisticated malware that is personalized, stealthy and often zero-day, in contrast to traditional malware which was broad, known and open. Upon infiltration, these malicious agents conceal themselves, reproduce and incapacitate the security measures of the host. Subsequently, they establish contact with their command-and-control servers to receive additional directives, which may entail the seizure of data, the contamination of other systems and the facilitation of reconnaissance.

The Antivirus (AV) software had the primary objective of preventing, detecting and removing malicious software. This initiative primarily centered on content-based signatures to automatically classify and analyze malware samples. Regrettably, these methodologies are fundamentally vulnerable to imprecisions caused by polymorphic and metamorphic techniques [3]. Consequently, the Intrusion Detection System (IDS) and AV have been unsuccessful in achieving complete triumph in malware analysis. Consequently, there is a widespread use of dynamic malware analysis tools to automate the classification of malware and benign samples. Dynamic malware analysis entails the execution of binaries within a controlled environment to extract the necessary information for the identification of malicious behavior.

An efficient method to uncover the behavior of malware upon infecting a system is to execute it in a controlled setting and record all the alterations made to the system and network during the analysis process. The analytical approach referred to as dynamic analysis is useful for gaining rapid insights into malware by executing it within a dynamic analysis sandbox for a brief period. Nevertheless, it is not universally effective for all types of malware. Some malware can detect that it is operating within a sandbox and subsequently cease or postpone its execution. Additionally, certain malware may be unable to complete all of its tasks within the analysis timeframe or may attempt to deceive analysts by refraining from performing any malicious or suspicious actions. Despite its limitations, dynamic analysis is widely utilized in the field of malware analysis due to the vast number of new malware that are discovered on a daily basis, necessitating rapid and automated analysis. Owing to their numerous

advantages, researchers predominantly favor free and open-source sandboxes in their investigations [4] In our thesis research, we utilized Cuckoo Sandbox, a well-known open-source automated malware analysis platform, to capture and analyze malware behavior.

Upon successful implementation of the laboratory environment utilizing Cuckoo Sandbox, the resulting analysis outcomes are subsequently employed for the purpose of machine learning algorithms. The utilization of such algorithms serves to enhance the efficiency and efficacy of malware analysis and classification. Prior to implementing machine learning algorithms, it is imperative to identify the dataset containing appropriate attributes. This phase of our thesis involves feature extraction and selection. The research primarily focuses on the feature selection method, which aims to reduce computational complexity while enhancing analytical performance.

Among the features obtained through dynamic analysis, we focused on the use of Application Programming Interface (API) and Dynamic Link Library (DLL) features. This choice is due to the importance of detecting and identifying malware indicators that emerge in behavior-based systems. APIs, which we refer to as system function calls, along with DLLs, summarize the events in malware behavior. Moreover, the efficacy of machine learning algorithms depends on the choice of relevant features integrated into these systems [5].

The process of feature selection can significantly decrease the redundant and irrelevant features present in the initial large feature space. As a result, it can enhance the detection rate and minimize the occurrence of false positives in the malware detection model. In our study, we have utilized multi-objective genetic algorithm-based feature selection.

Multi-objective genetic algorithms have emerged as valuable tools for feature selection. These algorithms aim to optimize multiple conflicting objectives simultaneously, such as enhancing classification accuracy while minimizing the quantity of selected features. They offer a versatile approach for identifying the most relevant features in malware samples, thereby enhancing the efficiency and effectiveness of automated analysis systems.

In this thesis, the multi objective evolutionary algorithm (MOEA) Framework was employed for the implementation of Genetic Algorithms (GAs). MOEA is a Java library freely available as open-source, specifically designed to support genetic algorithms. The genetic processes were executed utilizing various algorithms, namely Decomposition Based Evolutionary Algorithm (DBEA), Pareto Archived Evolution Strategy (PAES), Reactive Search Optimization (RSO), A simple random algorithm, often used as a baseline (RANDOM), and Reference Vector Guided Evolutionary Algorithm (RVEA) all of which are integrated within the MOEA platform. The PAES algorithm that gave the best results among the obtained values was selected and compared with other classical feature selection methods.

The Mc Nemar test is utilized to evaluate the performance of different malware detection classifiers by comparing their correct and incorrect classifications. This test determines if there is a statistically significant difference between the proportions of two related groups, specifically focusing on cases where the classifiers disagree. This method is particularly useful as malware evolves, rendering traditional signature-based detection systems inadequate. By examining correct and incorrect classifications, the Mc Nemar test ensures that differences in classifier performance are not due to random chance but reflect true differences in their ability to detect malware effectively. Based on the results in our paper, the Mc Nemar test revealed significant improvements [6,7].

1.1 Related Works

Malicious software creators have developed numerous methods for identifying the presence of malware analysis systems. However, dynamic malware analysis has effectively countered most of the techniques employed by hackers, including evasion, obfuscation and polymorphism.

Numerous online tools presently utilize dynamic analysis methods to produce reports that are comprehensible to humans. The analysis system must possess an adequate malware representation, which is subsequently employed for classification, either through similarity measurement or feature vectors. The proliferation of new malware samples being submitted to antivirus vendors on a daily basis necessitates the use of automated methods to reduce the number of samples that require in-depth human

analysis. Machine learning-based techniques, in particular, have been extensively utilized in the literature for automated malware analysis and classification. The following section will outline the various works that have been undertaken by the authors.

Shultz et al. [8] pioneered the use of static features in malware detection by utilizing multiple classifiers. Their research involved the application of three distinct types of static features, namely Portable Executable (PE), strings and byte sequence n-grams. The dynamic link library (DLL) information was extracted from the PE header of a binary and was utilized to create three distinct feature vectors. These vectors were designed to indicate whether a DLL had been utilized, whether a particular function within a DLL had been invoked and the number of distinct function calls made within each DLL. The researchers utilized encoded strings present in a binary as a feature and transformed binary files into hexadecimal codes to extract byte sequences for n-gram features. These three features were employed to classify new binaries as either malicious or clean using various classifiers. Although string features produced the highest accuracy, the optimum detection rate was attained by employing byte sequence n-grams.

The work of served as an inspiration and motivation for others to explore similar methods for classifying malware [9] further developed the byte sequence n-grams technique, resulting in improved detection of malicious binaries using classifiers such as Support Vector Machine (SVM), Decision Tree (DT) and their boosted versions [10] extracted API calls from the PE header of binary files, similar to the approach used by Schultz et al. [8] and utilized them as features for classifying binaries as either malicious or benign.

Tian et al. [11] conducted a more sophisticated analysis of static and reverse engineering tasks to extract characteristics from binaries. The initial step involved disassembling the binary, followed by the extraction of the length and frequency of function names. The utilization of function name length features for malware classification among various malware families has been observed. The results indicate that function name length is a crucial feature in discriminating between malware families.

Static features are commonly utilized in machine learning to identify or categorize malware. However, these features have certain limitations. It is assumed by the authors that the malware is not encrypted or unpacked and that static features can be promptly extracted from the binary. Malware often comes in packed or encrypted forms, which can sometimes make it impossible to fully unpack or decrypt. Additionally, there are various obfuscation techniques that can hinder the process [12].

Dynamic analysis refers to the systematic examination of a binary file through its execution in a regulated environment, with the aim of ascertaining its malicious or benign nature. This regulated environment may take the form of a customized sandbox that provides virtual, emulated or baremetal environments or a single computer with dynamic analysis tools.

Bailey et al. [13] have utilized different techniques to classify malware based on their behavior and features. This includes creating high-level behavior profiles, using hierarchical clustering, and employing SVM for classification. Some studies [14,15,16] have focused on capturing system calls and their parameters, while others have integrated static and dynamic features. Overall, these techniques have shown promising results in accurately identifying and categorizing malware [17,18,19].

Bayer et al. [14] have introduced a clustering technique for identifying and grouping malware in a scalable manner. Their approach employs the Extended Analysis of Network and Host Behavior in Internet Security (ANUBIS) system, which utilizes taint tracking to analyze the behavior of the malware. However, the trace dependence is a constraint for this method. Dynamic data tainting is an additional issue that needs to be addressed, as it allows for the injection of malicious binaries. Certain types of malware are only activated during specific actions or events, such as setting up a specific time to explore. These limitations present significant challenges for this framework. The utilization of this approach is accompanied by the drawback of its susceptibility to evasion techniques, which cannot be circumvented.

Syarif et al. [20] have provided an analysis of static and dynamic malware approaches. The static approach involves analyzing the malware without running it on the system, whereas the dynamic approach involves analyzing it while it is running. This paper

aims to bring to light the major advantages and disadvantages of both techniques and their diverse applications.

The static approach employs manual inspection, which is inadequate against techniques such as evasion, obfuscation and polymorphism. On the other hand, advanced static approach yields superior results by providing precise information on malicious programs. The basic dynamic analysis yields DLL infections in the system, providing information on the malware's actions. In contrast, the advanced dynamic malware analysis approach reveals more severe actions executed by the malicious code, such as disabling the firewall. The amalgamation of these two methodologies yields a superior resolution to identifying malicious software operations.

Dilung et al. [21] have devised an automated method, known as Malware Generation (MALGENE), for extracting analysis evasion signatures. This method utilizes bioinformatics algorithms to automatically detect evasive behavior in the form of system calls.

Data mining approaches are employed to ascertain the data events and call events for the purpose of identifying the behavioral patterns of malware. These findings are instrumental in the development of evasion signatures, which prove valuable in the examination of malware. The approach of system call alignment with parameter selection is a prominent technique employed in this methodology. Various levels of experimentation, including hypervisor and emulation, are conducted.

In their study, Philip O'Kane et al. [22] introduced a method for identifying malware through the application of N-gram analysis techniques. The technique involves examining the structure of a program, including its characters, strings and bytes. The researchers conducted an experiment that utilized a histogram of operational code (opcode) density, which was obtained through dynamic analysis. A reference model has been established using support vector machine to assess the efficacy of feature reduction techniques. However, the interrelationships among the features are intricate, rendering the use of simplistic filtering methods impractical. In order to achieve improved outcomes, the experiment has utilized the Eigen subspace analysis technique.

Konrad et al. [23] presented an automated analysis framework that utilizes prototype-based feature vectors for clustering and classifying malware samples. The primary objective of the author is to identify unknown malware with reduced runtime. The proposed framework employs clustering, classification and incremental analysis to carry out dynamic malware analysis. It has been observed that most of the existing research papers have limitations when dealing with larger datasets. The implementation of incremental analysis resulted in a decrease in runtime for processing larger amounts of data.

In reference to the machine learning methods, the related works are delineated.

Gavrilut et al. [24] endeavored to create a detection system utilizing various modified perceptron algorithms. The accuracy of these algorithms ranged from 69.90% to 96.18%. It should be noted that the algorithms with the highest accuracy also produced the greatest number of false-positives. The algorithm with the greatest accuracy resulted in 48 false positives. The algorithm that displayed the most balanced performance, with both appropriate accuracy and a low false-positive rate, achieved an accuracy score of 93.01%.

Singhal et al. [25] have presented a detection method using a modified Random Forest (RF) algorithm in conjunction with Information Gain (IG) for improved feature representation. It is important to note that the dataset used in this study exclusively comprised of portable executable files, which simplifies the process of feature extraction. The outcome of this method was a 97% accuracy rate with a false positive rate of 0.03.

Baldangombo et al. [26] presented extraction techniques utilizing PE headers, DLLs and API functions and methods, employing Naive Bayes, J48 Decision Trees and Support Vector Machines. The J48 algorithm demonstrated the highest accuracy overall, achieving 99% accuracy with the PE header feature type and hybrid PE header & API function feature type and 99.1% accuracy with the API function feature type.

In Alazab et al.'s [27] study, the API functions were utilized for feature representation. The Support Vector Machines algorithm with normalized polykernel yielded the most

favorable outcome, resulting in a precision rate of 97.6% and a false-positive rate of 0.025.

Shafiq et al. [28] developed the "PE-Miner" framework to detect new malware in real time by extracting structural features from executables. They used outlier analysis to select 189 features from the PE header of files. Feature selection was done using Redundant Feature Removal, Principal Component Analysis and Haar Wavelet Transform. The system achieved a 99% detection rate and a false positive rate of less than 0.5% in tests using VX Heavens and Malfease datasets.

Ye et al. [29] introduced two systems for identifying malware. The first system uses objective-oriented association mining and has a detection rate of 92%. The second system utilizes string analysis within a program's body and has a detection rate of 93.8%."

Aim of the Sharma et al.'s [30] study is to detect unknown malware by utilizing a machine learning technique that involves analyzing the frequency of opcode occurrence. The researchers utilized the Microsoft Malware Classification Challenge dataset for this purpose. The study involved comparing the top 20 features obtained from feature selection methods such as Fisher score, information gain, gain ratio, chi-square and symmetric uncertainty. The researchers conducted a study on various classifiers in the Waikato Environment for Knowledge Analysis (WEKA) GUI-based machine learning tool. They discovered that five of them, namely RF, Logistic Model Trees (LMT), Naive Bayes Tree (NBT), J48 Graft and REPTree, are capable of detecting malware with nearly 100% accuracy.

In 2007, Elovici et al. [31] utilized the PE and Fisher Score (FS) approach for selecting features. They employed Artificial Neural Network (ANN), Bayesian Network (BN) and Decision Tree algorithms with 5-grams, top 300 features and FS. Additionally, they applied BN and DT algorithms using PE, which resulted in an accuracy of 95.8%.

In 2008, Moskovitch et al. [32] employed a filters-based approach to select features. They utilized Gain Ratio (GR) and Fisher Score for feature selection and evaluated the performance of Artificial Neural Networks (ANN), DT, NB and Adaboost.M1

(Boosted DT and Boosted NB) and SVM classifiers were utilized to achieve an accuracy rate of 94.9%.

In 2008, Moskovitch et al. [33] proposed a methodology that utilized n-gram (1,2,3,4,5,6 gram) of opcodes as features and incorporated Document Frequency (DF), GR and FS feature selection techniques. They employed various classification algorithms such as ANN, DT, Boosted DT, NB and Boosted NB. Among these, ANN, DT and Boosted Decision Trees (BDT) exhibited superior performance, while maintaining a low level of false positive rate.

In 2011, Santos et al. [34] inferred that the process of supervised learning necessitates labelled data. Consequently, they proposed the use of semi-supervised learning as a means of detecting previously unknown malware. They conducted a study on the frequency of operational code appearances. They utilized the information gain method for feature selection and employed various classifiers, including DT, k-nearest neighbor (KNN), BN and SVM. The results showed that SVM outperformed the other classifiers, with an accuracy rate of 92.92% for one opcode sequence length and 95.90% for two opcode sequence length [35].

In 2012, Shabtai et al. [36] employed the use of opcode n-gram pattern feature to identify the most effective feature for their study. To achieve this, they utilized DF, G-mean and Fisher Score method. They employed multiple classifiers, with RF emerging as the most successful, having achieved an accuracy of 95.146%.

In 2016, Ashu et al. [37] introduced a new method for accurately identifying unknown malware. The approach involved examining the occurrence of opcodes and grouping the executables. Thirteen classifiers found in the WEKA machine learning tool were analyzed, with the RF, LMT, NBT, J48 and Functional Trees (FT) classifiers being studied in greater detail. The authors achieved a malware detection accuracy of over 96.28%.

In 2016, Sahay et al. [38] categorized executables based on their malware size using the Optimal k-means clustering algorithm. These categories were then used as effective features to train various classifiers such as NBT, J48, LMT, FT and RF. The

purpose of this training was to accurately identify unknown malware. The study revealed that the proposed approach has the capability to detect unknown malware with an accuracy rate of up to 99.11%.

In 2016, Ahmadi et al. [39] utilized the Microsoft malware dataset and incorporated hex dump-based features such as n-gram, Metadata, entropy, image representation and string length. Additionally, they extracted features from disassembled files including Metadata, Symbol frequency, opcodes, register, among others. They employed the eXtreme Gradient Boosting (XGBoost) classification algorithm and achieved a detection accuracy of approximately 99.8%.

In reference to the completed thesis, when we discuss the works related to it, we define them as follows.

Kateryna [40] proposed a methodology for the classification and detection of malware in a virtual environment through the utilization of machine learning techniques.

Yakup [41] has suggested a technique for the automatic recognition and categorization of specific malware. This would be accomplished through the utilization of machine learning on behavioral and memory characteristics obtained from dynamic and memory analysis outcomes.

If we summarize the studies carried out so far, there are differences in the relevant studies in terms of the preferred or created data sets, the applied analysis detection systems, the tools used in these analysis systems, the features extracted from the analysis results, the applied algorithm and feature selection methods.

In some articles, the feature selection part is not addressed effectively. At the same time, the relationships between features, which we call feature interaction, were not taken into account when performing the feature selection process. In some articles, malware and clean-ware files were not used in a balanced manner, and the malware that made up the dataset consisted of a single family. Again, although high accuracy is achieved in some studies, high false positive rates are also achieved. This highlights the need for improved feature selection and interaction processes.

CHAPTER 2

THEORITICAL BACKGROUND

This chapter offers a comprehensive elucidation of malware, machine learning algorithms, and genetic algorithms. A detailed exposition of these three major subjects is essential to fully comprehend their application in the thesis study.

2.1 Malware

This title provides an overview of malware, including its various types that are commonly encountered. Additionally, the title touches on the current techniques used for detecting malware, as well as the evasion methods commonly employed by malware creators.

2.1.1 Taxonomy of Malware

Malware can be classified based on various characteristics, but usually according to its family type. Bontchev et al. [42] proposed naming conventions for malware that took into account the taxonomy of these files. Due to the fast-paced development of malware, the previous naming convention needed to be revised to adapt to the constantly changing landscape of malware [43]. Typically, the standard name given to a particular type of malware consists of four components, as illustrated in Figure 2.1. However, the exact format of the naming convention, including the use of delimiters, may vary among organizations.

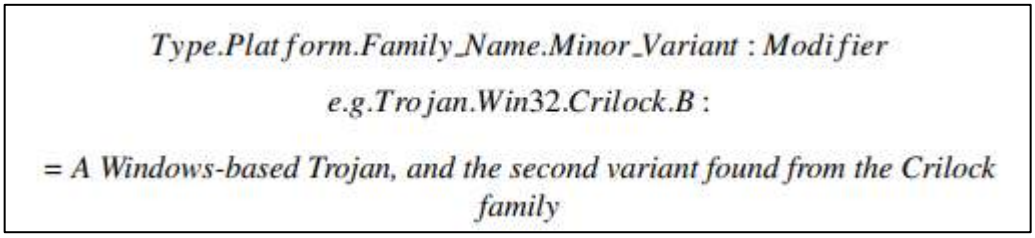

Figure 2.1 Naming convention for malware

The key components of the nomenclature are Type and Family, as they establish the overarching distinctions between classifications. According to the initial nomenclature guidelines established by Bontchev et al. [42], it was exceedingly difficult to precisely define a malware family, but it was feasible to group malware variants with similar structures together under a family label, such as MyDoom or Sasser. Malware is classified based on its type, which is determined by analyzing the behavior of the malware and its impact on the targeted system.

The development of a taxonomy system for malware is a challenging task due to the frequent overlap of categories and multiple mechanisms implemented. McGraw and Morrisett [44] suggests using the family attribute as a definition for malware. The Stuxnet cyber-weapon, known for its attack on the Iranian nuclear program, falls into multiple categories of malware. Stuxnet used an infected Universal Serial Bus (USB) drive to carry a worm module and a lnk file, exploiting vulnerabilities that allowed automatic execution of lnk files on USB drives. The worm module contains the payload and a rootkit module that hides the malicious activities and codes from the user [45]. Due to the complexity of this attack, it may be challenging to provide a comprehensive description of the malware using conventional methods. Nonetheless, utilizing the family variable enables a descriptive designation of the entire malware.

2.1.2 Types of Malware

There exist various kinds of malware that can be distinguished based on their attributes such as their mode of execution, their internal composition and their means of proliferation. They are as follows.

2.1.2.1 Virus

Viruses, as a form of malware, have the capability to replicate and contaminate other programs. Their dissemination may occur internally within a computer system or externally to other systems via removable media such as USB drives or Compact Discs (CDs) [46]. However, viruses are incapable of propagating through networks. The malware that spreads through networks is referred to as a "worm," while viruses can

be categorized based on their concealment approach, such as encrypted, polymorphic and metamorphic malware.

An encrypted virus is a form of malware that utilizes encryption techniques to encrypt the entire body of the malware. The decryption code is included within the malware's body. The detection method based on signatures is ineffective in identifying encrypted malware due to the inability to extract the signature from the malware. The decryption of the malware body occurs solely during execution, rendering signature-based detection useless. However, the presence of a signature for the decryption code within the executable can be utilized to detect such malware.

The polymorphic virus is a type of malware that changes its decryption code after each infection. This change is achieved through morphing, which alters the internal structure of the code without affecting its functionality [47]. This resolves the issue of encrypted malware and makes signature-based detection virtually impractical.

The metamorphic virus is an advanced iteration of the polymorphic virus. This malicious software alters its internal structure after each execution, while retaining its overall functionality. Diverse techniques are employed to modify the inner architecture of the malicious software.

The Subroutines Permutation technique involves the reordering of subroutine definitions in malware code, while preserving the sequence of subroutine calls. This results in the creation of structurally distinct code that is functionally identical.

In the technique of instruction reordering, a program's instructions are reorganized while ensuring that inter-instruction dependencies are met. In other words, instructions that do not rely on previous instructions can be rearranged to create modified versions of the program. This process can be accomplished through adherence to established guidelines [46, 47].

The process of instruction equivalence involves replacing an instruction or a series of instructions with an instruction that has the same function. This generates variations of the code that are functionally equivalent [48].

The technique of register renaming is a basic form of code obfuscation. It involves the renaming of all registers utilized within the code and producing modified versions of the code. This results in a more complex and difficult-to-understand version of the original code [46].

The method of inserting Junk Code involves the incorporation of non-functional instructions into software code, without impacting the program's execution [49]. These instructions, commonly known as dead code or junk code, are introduced to aid in the obfuscation of the code.

One of the techniques employed is subroutine interleaving, which involves merging or splitting subroutines to produce morphed copies. Another method is the creation of spaghetti code, which involves the frequent use of jump and go-to instructions. Through the integration of various techniques, a substantial quantity of morphed code copies can be effectively generated [46].

2.1.2.2 Worm

The characteristics of worms bear similarities to those of viruses. Both worms and viruses have the ability to self-replicate. However, there are two fundamental differences between them. Firstly, worms are stand-alone and do not require pre-existing executable code to infect a system. Secondly, viruses rely on an existing program to perform their functions. Additionally, it should be noted that worms can self-propagate throughout networks without the assistance of existing programs, unlike viruses. One of the most infamous worms from the 1980s is the Morris worm, which caused significant disruption to the internet for a considerable period [46].

2.1.2.3 Trojan

A Trojan is a form of malicious software that disguises itself as a harmless program, while covertly executing concealed malicious actions. An archetypal example of a Trojan would be a password login application that requests a user's login credentials and records their password via keylogging in the background. A prevalent form of Trojan is the phishing scam that deceives users into inputting their login credentials on a counterfeit login page. Upon doing so, the Trojan would exhibit an error message

indicating an invalid password and subsequently present the authentic login page [46]. Another instance of a Trojan is a counterfeit anti-virus program that promises to eliminate viruses, yet engages in malicious behavior in the background. The research employs various Trojan viruses such as Trojan.Zbot [50], Trojan.ZeroAccess [51], Trojan.Cleaman [52] and more.

2.1.2.4 Backdoor

A Backdoor is a means of circumventing regular security protocols. An instance of such a Backdoor that we have examined is Harebot. Harebot.Backdoor facilitates remote entry to the compromised system. The malware tries to sneakily carry out actions to block resources and disrupt the system's efficiency. It doesn't have the ability to spread on its own [53].

2.1.2.5 Adware

Adware is a form of malicious software designed to display unwanted advertisements to users. It often infiltrates systems through deceptive tactics such as bundled software installations or misleading advertising banners. Once installed, adware tracks user activity and displays targeted advertisements based on browsing history or other collected data. These advertisements can disrupt the user experience, slow down system performance, and potentially lead to privacy breaches.

2.1.2.6 Downloader

Downloader is a form of malware designed to infiltrate computer systems and facilitate the unauthorized download and installation of additional malicious software or unwanted applications. Downloaders are a significant threat to cybersecurity as they can serve as a gateway for the introduction of more harmful malware, such as ransomware, spyware, or banking trojans.

2.1.2.7 Dropper

A dropper is a form of malicious software designed to covertly deliver and install additional malware payloads onto target systems. It typically operates by exploiting

vulnerabilities in software or social engineering techniques to infiltrate systems undetected.

2.1.2.8 Keylogger

Keyloggers, a variant of rootkit malware, intercept keystroke events from the keyboard and store them in a log file, thereby capturing sensitive information such as usernames, Personal Identification Numbers (PINs), and passwords, subsequently transmitting them to malicious attackers without arousing user attention [54].

2.1.2.9 Ransomware

Ransomware constitutes a form of malevolent software meticulously crafted to encrypt files or lock computer systems, effectively rendering them inaccessible to users. Once installed, ransomware encrypts files on the infected system, making them inaccessible to the user. Subsequently, the assailant requests a ransom payment, typically in cryptocurrency, with the promise of furnishing the decryption key or unlocking the affected system.

2.1.3 Malware Detection Techniques

Several detection techniques are commonly employed to identify malware. These techniques involve extracting various characteristics of malware, which are subsequently used in statistical analyses on opcode sequences, pattern mining, similarity-based techniques and others.

The progression of malware detection methods can be likened to a competition between AV companies and malware creators. As AV vendors implement new techniques to identify malware, the malware writers develop new techniques to evade detection algorithms. The present article presents a classification of the conventional malware detection techniques, as shown in Figure 2.2.

The main categories of analysis are outlined below:

2.1.3.1 Signature-Based Analysis

The method of signature-based detection is dependent on comparing the payload of a program with a collection of pre-learned regular expressions extracted from malware that has been stored in a repository. This technique requires considerable human effort and a substantial time investment to extract, store, and disseminate these patterns [55-56].

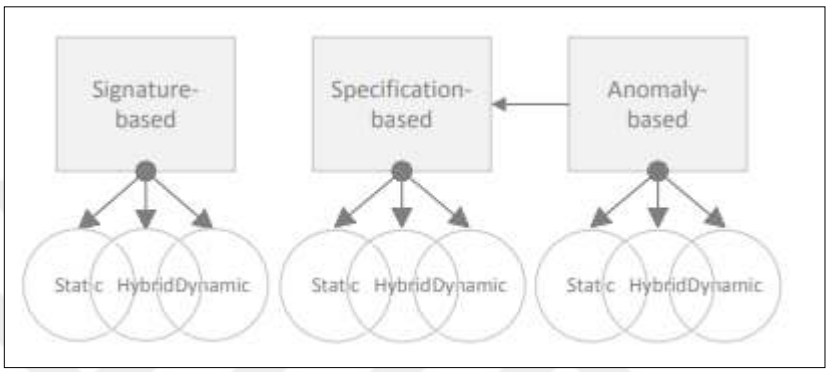

Figure 2.2 Malware detection methods

According to researchers at Symantec, a reputable antivirus company, the process of creating signatures is inefficient and prone to errors [57]. The authors presented a study on an automated signature extraction system, which showed significant improvements in the time it takes to generate candidate signatures from a dataset of 46,288 malicious samples. The researchers reported that it took them 1,278 minutes (equivalent to over 21 hours) to generate candidate signatures, while deriving final signatures took between 5 and 17 minutes. The process of detecting regular expressions often takes place in a static manner, which presents significant drawbacks. It is not effective when it comes to unidentified regular expressions and it is incapable of searching through disguised code for these strings. This makes signature-based methods less effective in the face of rapidly increasing malware.

2.1.3.2 Anomaly-Based Analysis

Anomaly-based detection systems categorize behaviors of a file or system that are regarded as standard and any deviations from these are deemed as anomalies. A model of standard behavior or structure of the file or system is created during a training phase

[58]. During the monitoring phase, any deviation from the established baselines in training is identified [59]. This could be, for instance, a poisoned Portable Document Format (PDF) that substantially differs from the standard or anticipated structure of a clean PDF. Anomaly detection is a useful tool for identifying unusual patterns in data. For example, a typical movie file may be around 700MB in size, while a one-hour TV show file is expected to be roughly 350MB in size. If a file is only 120KB, it would be considered suspicious. One of the primary advantages of anomaly detection is its ability to detect previously unknown threats, also known as zero-day threats. Anomaly-based systems do not require prior knowledge of all atypical traits for detection, as they are trained on typical traits. Nevertheless, these systems have limitations such as a tendency towards generating false-positive ratings and the need for extensive features to properly model the system being examined [59].

2.1.3.3 Specification-Based Analysis

The detection method of specification-based detection is a derivative of anomaly detection that aims to tackle the problem of high false-positive rates. The approach involves developing a set of rules that approximate the system's requirements rather than its implementation [58]. The task of accurately modeling a vast system is a challenging one. As a result, the use of specification-based detection may suffer from a disadvantage similar to that of an anomaly-based system. This is because the model may not fully capture the intricate behaviors of a complex system.

Each of these analyses can be performed using one of three methods: static, dynamic, or hybrid analysis.

2.1.3.4 Static Analysis Method

Static analysis is a method of examining code and data structures without program execution [56]. It can be done on source code or compiled binary representation to determine the program's functionality [60]. Hash representations of executables like Message Digest 5 (MD5) or Secure Hashing Algorithm 1 (SHA1) can be compared to databases of known malware. Call graphs can show the structure of software and the connections between functions." The analysis of strings involves detecting elements

such as Uniform Resource Loaders (URLs), Internet Protocol (IP) addresses, command line options, PE files and passwords [61].

Static analysis techniques provide the benefit of safely analyzing malware samples without the need for execution. Unlike dynamic analysis, which involves running the code, static analysis allows for a detailed inspection of all code paths since the entire codebase is accessible. However, these techniques may fall short in identifying complex malware designed to remain hidden until executed. Analyzing all code paths can be misleading if the code includes superfluous dead code added to disguise the file's true purpose. Additionally, obfuscation techniques, used to obscure the real code, often prevent the actual code from being revealed during static analysis since the code is never run [62].

2.1.3.5 Dynamic Analysis Method

Dynamic analysis entails the execution of the file being scrutinized. It involves the extraction of data during memory entry, while the program is running and after it has finished executing. This approach circumvents several obfuscation techniques that impede static analysis methods. The dynamic environment may be classified as native, wherein there is no clear distinction between the host and the analysis environment since the analysis is conducted within the native operating system of the host. However, this approach poses challenges in malware analysis as it may lead to adverse consequences on the host system. Software exists that can capture a snapshot of the pristine native environment, such as DeepFreeze [63]. This captured image can be restored after each incidence of malware execution. The restoration process can be time-consuming since it involves restoring the entire system to its original point [64]. Another method is emulation, where the guest environment is controlled by software that emulates hardware, all under the host's control. When utilizing emulation, there exists a possibility of performance problems since it is reliant on the underlying host architecture. In contrast, a virtual machine furnishes an isolated guest environment that is controlled by the host. In an ideal setting, the genuine essence of the subject is exposed and can be scrutinized. Cyber attackers have developed techniques to avoid detection in emulated, simulated or virtualized environments, including anti-debugging, anti-instrumentation and anti-virtual machine tactics [65]. Dynamic

approaches have a disadvantage of requiring more time to execute a program compared to static tools like IDAPro, which can disassemble an executable quickly [66].

2.1.3.6 Hybrid Analysis Method

Hybrid techniques combine static and dynamic analysis in their detection algorithms. Roundy and Miller [67] used parsing for pre-execution static analysis and dynamic execution for obfuscated code. The authors have created an algorithm that uses both dynamic and static techniques to instrument obfuscated code.

2.1.3.7 Dynamic Analysis Tools

Several freely available or open-source dynamic analysis tools can be utilized to extract API calls or instruction execution logs. These tools include:

Ether malware analysis tool is an open-source tool that has been created using hardware virtualization extensions. It operates entirely outside of the target operating system, which means that there is no detection of in-guest software components. Therefore, there are no issues with incomplete or inaccurate emulation. Many of the contemporary viruses have the ability to detect a debugger or a virtual environment while being executed. Nonetheless, the Ether malware analysis tool has effectively surmounted this issue [68].

API Logger is a straightforward tool that records all API calls that meet the criteria outlined in the inclusion and exclusion lists. The inclusion list identifies the libraries that should be incorporated, while the exclusion list identifies the libraries or functions that may be disregarded [69].

WinAPIOverride is a sophisticated software for monitoring APIs. It bridges the gap between traditional API monitoring software and debuggers. Its ability to manually extract API calls by controlling the program's flow during execution makes it extremely valuable.

API Monitor is a software that facilitates the monitoring and control of API calls performed by processes or applications. It is imperative to note that this tool must run

within a virtual machine to analyze malware and cannot function within a sandboxed environment [70].

Buster Sandbox Analyzer(BSA) is a dynamic analysis tool designed to detect malicious activities in processes. It closely monitors system changes such as registry, file-system and port modifications [71]. The Buster Sandbox Analyzer tool operates within a Sandbox environment that effectively shields the system from malware infection during its execution. Specifically, the Sandboxie application is employed for this purpose [72].

There exist additional frameworks, including Detours and DynInst, for tracing Windows API. Additionally, various tools utilize different methods to trace Windows API functions in-guest, as previously mentioned. There are various tools available that serve the same purpose as Buster Sandbox Analyzer, which is to track activities in sandboxed environments. Examples of such tools include CWSandbox and Norman Sandbox, as mentioned in reference [68].

2.2 Machine Learning Methods

This paper presents a theoretical foundation of machine learning techniques that are essential for comprehending their practical application. It begins with an overview of the field of machine learning, followed by a detailed explanation of the methods pertinent to this study. The aforementioned techniques comprise J48, RepTree, AdaboostM1, MLP and NB.

2.2.1 Machine Learning Basics

Machine Learning has emerged as a distinct area of Computer Science due to the progress in KNN methodologies and techniques. It is a subset of Artificial Intelligence, focusing on the system's ability to learn from its own operations. Arthur Samuel coined the term "field of study that gives computers the ability to learn without being explicitly programmed" in 1959. T Mitchell [73] provided a more precise definition, stating that a computer program is considered to learn from experience when it pertains to a specific category of tasks and performance measure. The program's performance in tasks should improve with experience.

The fundamental concept behind any machine learning task is to train the model using a specific algorithm to execute a designated task, such as classification, clusterization or regression. This process entails utilizing the input dataset to conduct training, following which the resultant model is utilized to make predictions. The outcome of this model is contingent on the initial task and implementation. Potential use cases include forecasting the price of an unknown house using information about its attributes such as size, number of rooms and price; categorizing a set of new medical images as healthy or containing tumors based on two datasets consisting of healthy and tumor images; and arranging a collection of animal pictures into multiple clusters.

In order to gain a more comprehensive comprehension, it is advisable to familiarize oneself with the overall workflow of the machine learning process, as depicted in Figure 2.3.

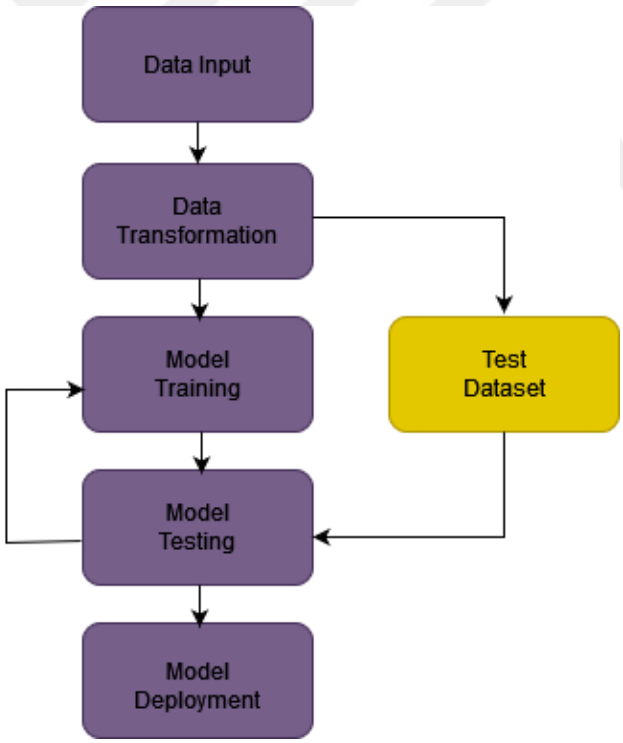

Figure 2.3 General workflow of machine learning algorithm

The process can be broken down into five distinct stages. First, the data is loaded from a file and stored in memory. Then, the loaded data undergoes transformation, clearing, and normalization to make it suitable for the algorithm. This includes converting the

data to a consistent range and format. Additionally, feature extraction and selection are performed at this stage. Data is divided into a training set and a test set. The model is built using the training set and then evaluated using the test set. The model is tested using test data and the results are used to build a new model. The best model is selected based on a defined number of iterations or achieving the desired result.

2.2.1.1 Feature Extraction

In order to utilize the algorithm effectively, it is necessary to extract the attributes from the input data. For instance, in the case of housing prices, the data can be represented as a multidimensional matrix with columns indicating the attributes and rows containing the numerical values for these attributes. This allows for proper processing of the data by the algorithm. In the context of an image, the data can be expressed as the Red-Green-Blue (RGB) value of every single pixel.

The characteristics of the data are known as features, while the feature vector is the matrix. The act of obtaining data from the files is referred to as feature extraction. The aim of feature extraction is to obtain a collection of significant and non-duplicative data. It is imperative to comprehend that the features ought to epitomize the significant and pertinent data regarding our dataset, as an accurate prediction cannot be made without it. Therefore, feature extraction is frequently a non-evident task that necessitates extensive experimentation and investigation. In addition, it is highly specialized in nature, making it difficult to apply general techniques in this field.

Possible transformations in feature extraction process are:

Normalization: A process that involves dividing an image x into its constituent parts, where x is represents the number of pixels with color i, by the total count of pixels present in the image. This technique is used to encode the distribution of the image and remove any dependence on its size. This translates into the formula: $x'=x\|\|x\|\|$ [74].

Standardization: A necessary step when dealing with comparable objects that have features measured on different scales. For instance, in the context of housing prices, there may be discrepancies in the scales used for different features. For example, the feature 'room size' may be measured on an integer scale that ranges up to 5, while

'house size' may be measured in square meters. In order to ensure accurate comparisons between these features, it is important to standardize the scales used for each feature. While both values may undergo comparison, addition and multiplication, the outcome would lack reasonability in the absence of normalization. The following scaling is often done:

$x'_i = (x_i - \mu_i)/\sigma_i$, where μ_i and σ_i are the mean and the standard deviation of feature x_i over training examples [74].

Non-linear expansions: They may be necessary in certain scenarios where increasing the dimensionality of data can be beneficial, despite the typical goal of reducing it. This approach is particularly useful for intricate problems that require more complex solutions beyond first-order interactions to achieve accurate results.

2.2.1.2 Feature Selection

A crucial criterion for a satisfactory feature set is non-redundancy. The existence of redundant features, which depict the same information and redundant information attributes that are closely interdependent, can introduce bias into the algorithm and, consequently, yield an imprecise outcome.

Furthermore, in cases where the input data exceeds the algorithm's capacity due to an excessive number of features, it can be converted into a condensed feature vector with a reduced number of features. Feature selection involves the process of decreasing vector dimensions. The aim of this process is to extract pertinent information from the original dataset, enabling its utilization without any loss of accuracy. Ultimately, selected features should provide a comprehensive overview of the initial data set, allowing it to be used more efficiently.

2.2.1.3 Supervised and Unsupervised Learning

Thus far, the fundamental principles of machine learning, particularly emphasizing the availability of preliminary data for training models, have been explored. However, this is not always the case. In this context, the two primary methodologies in machine learning, namely supervised and unsupervised learning, are introduced.

Supervised Learning is a type of machine learning that relies on labeled data. This approach involves working with a pre-existing dataset that contains data samples and their corresponding outcomes. For instance, the housing prices case is a prime example of supervised learning. In this scenario, the initial dataset comprises information on houses, including their attributes and their respective prices. The model has undergone training utilizing a dataset where it possesses knowledge of the correct outcomes. Instances of supervised learning comprise of regression and classification predicaments:

Regression is a technique used to predict values based on previous observations. This is achieved by analyzing the values of the samples from the training set. It is generally accepted that if the output is a real number or is continuous, then it is a regression problem.

Classification is the process of predicting the class of an unknown sample based on labeled data, with limited and typically small possible outputs. If the output is a discrete or categorical variable, then the problem can be classified as a classification problem.

In comparison to Supervised Learning, Unsupervised Learning does not involve the initial labeling of data. Its objective is to identify patterns in unsorted data instead of predicting a value. Clustering is a typical subclass of Unsupervised Learning.

The process of clustering involves identifying concealed patterns within unclassified data and segregating it into clusters based on their similarities. For instance, one may discover distinct customer segments within the customer base of an e-commerce store.

2.2.2 Classification Methods

From a machine learning perspective, the detection of malware can be viewed as either a classification or clusterization problem. The goal is to cluster unknown types of malware into several distinct clusters based on specific properties that are identified by the algorithm. In contrast, the issue of identifying malware can be simplified through the use of a model trained on a broad range of malicious and non-malicious files. This approach involves classification, which reduces the problem to a matter of

sorting samples into distinct categories. By limiting the number of possible classes, we can achieve greater accuracy in identifying the correct category for a given sample, particularly for known malware families. As a result, classification algorithms are more effective than clusterization in this context. This section will provide the theoretical foundation for all the techniques employed in this thesis.

2.2.2.1 Reduced Error Pruning Tree

REPTree, or Reduced Error Pruning Tree, is a decision tree algorithm used for classification and regression tasks. This algorithm is widely preferred due to its practicality and efficiency. REPTree is based on the C4.5 algorithm but is a simpler and faster version. The primary goal of the algorithm is to quickly construct decision trees with high accuracy and to prevent overfitting. This is achieved through pruning, which enhances the model's generalization ability [75].

The working principle of REPTree relies on random sampling and pruning techniques. The algorithm splits the dataset into random subsets and constructs a decision tree for each subset. These trees are then pruned to reduce the error rate. Pruning reduces the complexity of the model and enhances its generalization performance. Additionally, the algorithm employs the principle of variance reduction, which balances the model's performance across different datasets, thus preventing overfitting. An important feature of REPTree is its ability to work with continuous data, allowing it to be applied to a wide range of applications [76].

The applications of the REPTree algorithm are extensive. For instance, it is used in financial forecasting, customer segmentation, gene expression analysis in bioinformatics, and the development of marketing strategies. The algorithm's speed and efficiency enable it to handle large datasets, making it a preferred tool in data mining projects. Furthermore, the interpretability of REPTree's results enhances transparency in decision-making processes, allowing users to understand the algorithm's output more easily [77].

2.2.2.2 Adaptive Boosting

AdaBoostM1, also known as Adaptive Boosting, is a widely used ensemble learning algorithm for classification tasks. AdaBoostM1 combines weak classifiers to form a strong classifier by weighting each weak classifier and giving more importance to incorrectly classified examples, thereby increasing classification accuracy. This algorithm was initially proposed by Freund and Schapire and is considered a revolutionary method in the field of machine learning [78].

The working principle of AdaBoostM1 relies on iteratively combining weak classifiers. Initially, each example in the dataset is given equal weight. After training a weak classifier, the weights of incorrectly classified examples are increased, while the weights of correctly classified examples are decreased. This process continues for a specified number of iterations. At each step, a new weak classifier is added, and the final classifier is formed by a weighted majority vote of these weak classifiers. An important feature of AdaBoostM1 is its ability to minimize errors in the tree structure, thereby enhancing the model's generalization capability [79].

The applications of AdaBoostM1 are extensive. For instance, it has been successfully applied in fields such as face recognition, text classification, medical diagnosis, and financial forecasting. The flexibility and accuracy of the algorithm enable it to perform well even on large datasets. Additionally, AdaBoostM1 can be easily integrated with other machine learning algorithms, making it a preferred tool in many data science and engineering projects. In summary, AdaBoostM1 is regarded as a significant tool in machine learning and can be used in a wide range of applications [80].

2.2.2.3 Naive Bayes

The Naive Bayes algorithm is a machine learning classification method that utilizes the Bayes Theorem. It is applicable to both binary and multi-class classification problems and operates on the assumption that each feature is independent of the others. The Naive Bayes method assesses the likelihood of each feature independently, without considering any correlations and utilizes the Bayes Theorem to make

predictions. This approach is termed "naive" because in practical scenarios, features frequently exhibit some degree of interdependence.

In order to comprehend the Naive Bayes algorithm, it is essential to first acquaint oneself with the notions of class probabilities and conditional probabilities.

a. Class Probability refers to the probability of a particular class in a dataset. It quantifies the likelihood of a randomly selected item from the dataset belonging to a specific class.

b. Conditional Probability refers to the likelihood of a specific feature value occurring, given the associated class.

One of the benefits of utilizing this approach is its straightforwardness and ease of comprehension. Moreover, it exhibits proficiency in handling data sets with unimportant features as the probabilities of their influence on the outcome are insignificant. The algorithm that uses the NB model assumes that the features are independent of each other, which means that the occurrence of one feature does not affect the occurrence of any other feature. Consequently, these dependencies are disregarded when making predictions. Additionally, this algorithm generally exhibits efficient performance regarding resource consumption, as it solely requires the calculation of the probabilities of the features and classes without necessitating the discovery of any coefficients, unlike other algorithms. As has been previously stated, a significant limitation of this approach is that it treats each feature as independent, despite the fact that this is often not the case [81].

2.2.2.4 J48 Decision Tree

Decision trees are data structures with a tree-like structure, as their name suggests. They are created using the training dataset and are then utilized for making predictions on the test data. The objective of this algorithm is to attain the most precise outcome while minimizing the number of decisions that need to be made. Decision trees can be employed for both classification and regression issues.

The Iterative Dichotomiser 3 (ID3) algorithm for decision trees is popular and relies on Entropy and Information Gain to measure uncertainty in the data. The entropy of a coin toss is uncertain and unpredictable, but if a coin with two heads is used, the entropy becomes zero because the outcome can be accurately predicted [73].

The decision tree method gained popularity due to its simplicity, ability to handle large datasets and effectively managing noise in datasets. Additionally, it has an advantage over other algorithms, such as SVM or KNN, as it operates in a "white box" manner, providing clear visibility into how the outcome is achieved and which decisions were made along the way. These aforementioned facts have resulted in its widespread usage for various purposes, including medical diagnosis, spam filtering, security screening and other related fields [73].

2.2.2.5 Multilayer Perceptron

The MLP is one of the fundamental and widely used types of artificial neural networks. MLP is a feedforward neural network that can have one or more hidden layers. Due to its ability to model nonlinear relationships, it is commonly employed in various classification and regression tasks. The core components of MLP include an input layer, one or more hidden layers, and an output layer. Each neuron processes inputs using specific weights and an activation function, producing an output [82].

The operation of MLP relies on feedforward and backpropagation algorithms. During the feedforward phase, data flows from the input layer through the hidden layers to the output layer. In the backpropagation phase, the error between the predicted and actual values is computed and used to update the weights in the network. This process is repeated until the model's error rate is minimized. Common activation functions used in MLP include Rectified Linear Unit (ReLU), sigmoid, or tanh, which enhance the model's learning capacity [83].

The applications of MLP are extensive. For instance, it has been successfully applied in areas such as handwriting recognition, image processing, speech recognition, and natural language processing. Due to its high performance on large datasets and its flexible architecture, MLP is frequently chosen in data science projects. Furthermore,

MLP's ability to integrate with other machine learning algorithms makes it a valuable tool across a wide range of applications. In summary, MLP is regarded as a significant tool in machine learning and can be effectively used in various practical scenarios [84].

2.2.3 Cross-Validation

The deficiency of the accuracy evaluation techniques in machine learning models is that they are incapable of forecasting the model's performance on unseen data. To address this limitation, cross-validation is employed whereby the original dataset is split into subsets. The training of the model is conducted on the largest portion of the dataset, followed by its evaluation on the smaller portion. Cross-validation can be categorized into three distinct classes.

The holdout method involves the division of a dataset into two distinct parts, namely, a training set and a test set. The model is trained using the training set, after which it is evaluated using the test set, which has not been previously exposed to the model. The mean absolute test error is computed using the errors that are obtained from the model and it is used for evaluating the model. This method has the advantage of being highly efficient. However, the evaluation outcome is heavily influenced by the method used to select the test set, as there is usually a high level of variance. Consequently, the assessment outcome may vastly vary among diverse test sets.

The k-fold method is an enhancement of the holdout method, where k subsets are selected and the holdout method is performed k times. Each iteration uses one subset as the training set and the remaining subsets as the test set. The k-fold method is a more advanced version of the holdout method, where the dataset is divided into k subsets or folds. The model is trained on k-1 folds and tested on the remaining fold. The process is repeated k times, with each fold serving as the testing set once. The average error is calculated over all k runs to reduce variance and ensure accuracy remains consistent across different datasets. The holdout method is simpler and faster than the method being discussed.

The leave-one-out technique is a variation of the k-fold approach, where the model is trained on all data points except for one, which is used for testing in each iteration of

the holdout technique. This method minimizes variance but increases computational complexity [85].

This chapter provides essential background information on machine learning, which is necessary to understand the practical implementation of the project. It explains the concepts of feature set, feature extraction, selection methods and the machine learning algorithms that will be used in the practical component. The selected algorithms include J48, REP Tree, AdaboostM1,MLP and NB.

2.2.4 Genetic Algorithm

The Genetic Algorithm is a significant Heuristic Algorithm that emulates Darwin's theory of evolution. It is an Evolutionary Algorithm that identifies the optimal solution through natural selection and crossover. The Genetic Algorithms begin by randomly generating initial individuals to form an initial population. Each individual is composed of a variable Gene that represents a solution to the given problem and is encoded by Chromosome. Various encoding methods are utilized in optimizing distinct problems. Schemata can be expressed through binary or real-valued representations. The selection of representation holds substantial importance in the effective implementation of Genetic Algorithms. The objective function serves to articulate the issue in Genetic Algorithms and the degree of fitness is derived by evaluating each individual against the object [86].

Genetic Algorithm design must incorporate three crucial operators: Selection, Crossover and Mutation. The selection operator involves the meticulous process of selecting individuals for the subsequent generation from the current generation. The selection operator is typically engineered to probabilistically choose good solutions, meaning individuals with high Fitness Values, while discarding poor solutions. Meanwhile, the crossover operator entails gene recombination, wherein two parental chromosomes are recombined to produce fresh individuals for use in the subsequent generation. In the realm of genetic algorithms, there exist significant crossover techniques, namely the one-point and two-point crossover. Mutation operator, on the other hand, involves modifying one or multiple gene values that are randomly chosen from the present chromosome. The genetic algorithm involves a generational process

that utilizes genetic operators to gradually evolve candidate solutions towards approximate solutions. With each iteration, the solutions converge more closely towards an optimal solution that meets the given constraints. Once the process is terminated, the optimum solution is obtained to solve the problem at hand [30].

CHAPTER 3

PROPOSED APPROACH

The aim of the study is to determine the most accurate and performant algorithm by using relevant feature extraction technique and selecting the influential and least number of features to detecting malware in a shorter time. This section delves into the practical aspects of implementing the system. As depicted in Figure 3.1, the process entails various stages starting with the collection of data, followed by the extraction of features from the modified Cuckoo sandbox result, selection of relevant and the least features for the machine learning algorithms, devising an evaluation method and concluding with the implementation process.

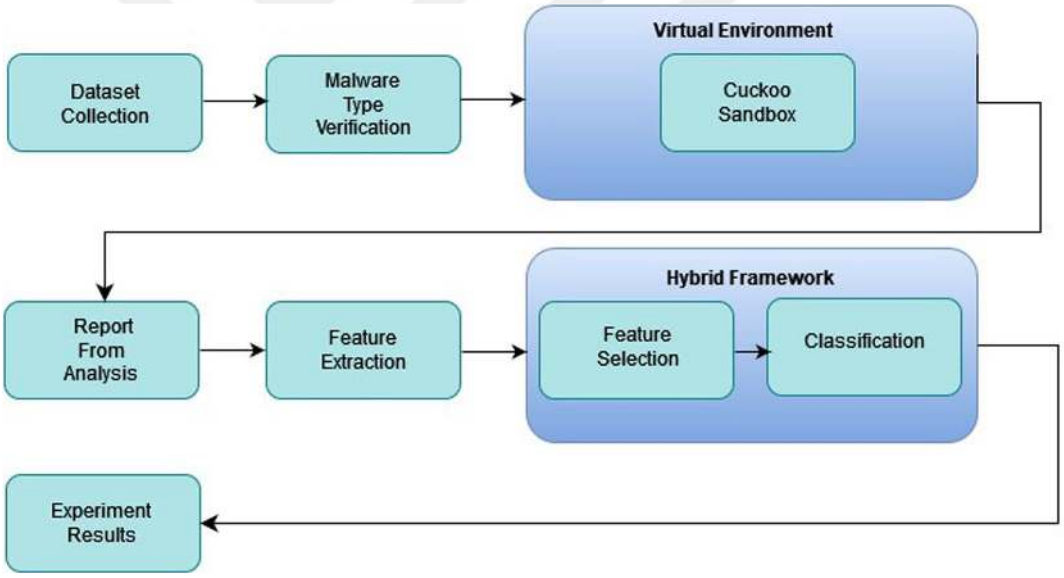

Figure 3.1 Flow chart of the system

3.1 Data

In this research, a collection of malware files totaling approximately 1 TB was obtained from virusshare.com. Virusshare.com is a malware sample repository that grants access to live malicious code for security researchers, forensic analysts, incident responders and individuals with a keen interest in the subject matter. In addition,

several other types of malware were obtained from various internet addresses in order to conduct tests for feature extraction and selection, with the aim of implementing the system with well-chosen inputs. The benign files primarily consisted of .exe format software installers, as well as system, utility, and application files predominantly from Windows 7 and Windows 10 operating systems.

A curated 1 TB collection of malware files was carefully examined to include eight distinct types of malicious software, ensuring a balanced dataset. The selected files for this dataset are in the Win32 Portable Executable (PE) format, which is a standard executable format compatible with all processor architectures and operating system versions. To maintain balance, the dataset comprises 100 malicious software samples and 100 benign executable files. The malware files were classified into eight distinct categories, with 50 being trojans, 10 being worm, 11 being adware, 5 being backdoors, 6 being downloaders, 6 being a dropper, 6 being a keylogger and 6 being ransomware. These various types of malware were identified through an analysis of each malware on the online service of VirusTotal. VirusTotal conducts inspections on items using more than 72 antivirus scanners and URL/domain blacklisting services, along with to employing various tools to extract signals from the content under examination. All users have the capability to choose a file from their computer through their web browser and transmit it to VirusTotal.

The 200 files undergo individual analysis in Cuckoo Sandbox and the outcomes are saved in a localized area for the purpose of extracting features and selecting elements to construct the pertinent dataset. The analysis of each file with Cuckoo Sandbox produced approximately 600 MB of data. In total, the 200 files generated around 117 GB of data in JavaScript Object Notation (JSON) format.

3.2 Implementation

In this stage, the research plan is created and put into action.

The entire implementation process can be summarized in the following stages:

1. Setting up the Lab Environment with Cuckoo Sandbox

2. Extracting Features (using Python 2.7)

3. Selecting Features (using MOGA)

4. Implementing machine learning methods (using WEKA)

5. Assessing the outcomes

Further in this chapter, each of these stages will be examined thoroughly.

3.2.1 Installation of Lab Environment with Cuckoo Sandbox

The focus of this study centers on Cuckoo Sandbox and its targeted usage. In order to effectively apply machine learning algorithms to any given problem, it is imperative to represent the data in a suitable format. Thus, in this study, Cuckoo Sandbox was utilized for this purpose. The reports produced by the sandbox that detail the behavioral data of each sample were subjected to preprocessing to extract malware features. However, comprehending the sandbox's functionality and report structure is crucial.

Cuckoo Sandbox is an open-source tool for analyzing malware that provides a detailed behavioral report for any file or URL within seconds. As per the Cuckoo Foundation [87], the current list of file formats that are supported includes a range of items such as Visual Basic (VB), Windows executables, PDF documents and almost all other formats.

The modular architecture of Cuckoo is highly customizable, facilitating its utilization as a standalone application or integration into larger frameworks.

Cuckoo's infrastructure includes a host machine and multiple guest machines. When a new file is submitted, it is assigned a virtual environment and executed, with all system activities documented.

Figure 3.2 illustrates that the sandbox produces a report that delineates all the actions undertaken by the file in the system.

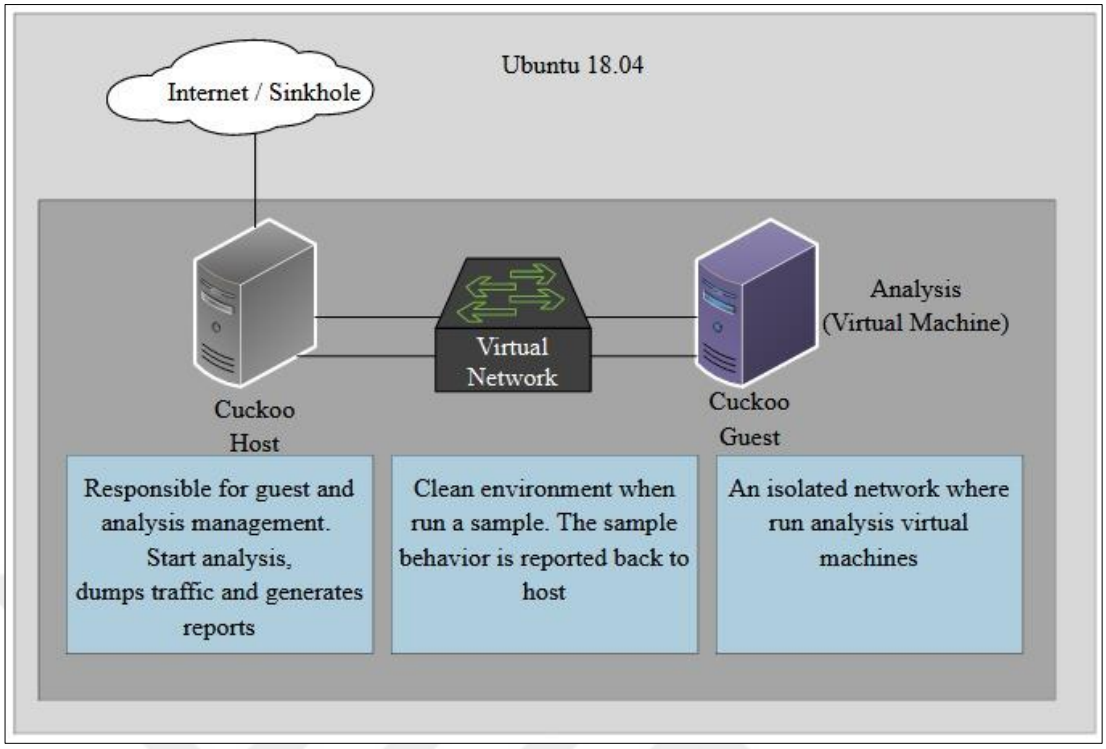

Figure 3.2 Cuckoo Sandbox architecture

The report is presented in the JSON file format and currently has the ability to identify the subsequent characteristics [87]:

- Traces of calls performed by all processes spawned by the malware
- Files being created, deleted and downloaded by the malware during its execution
- Memory dumps of the malware processes
- Network traffic trace in the Packet Capture (PCAP) format
- Screenshots that were taken during the execution of the malware
- Full memory dumps of the machines

Upon analyzing the behavior of a file, Cuckoo Sandbox utilizes pre-defined signatures to determine the degree of maliciousness. This aspect of the sandbox's functionality serves as a means to compare the efficacy of machine learning techniques with existing signature-based methods.

A JSON file example can be found in Figure 3.3.

```
{
    "category": "system",
    "status": 1,
    "stacktrace": [],
    "api": "NtClose",
    "return_value": 0,
    "arguments": {
        "handle": "0x000000cc"
    },
    "time": 1530616004.265,
    "tid": 2960,
    "flags": {}
},
{
    "category": "file",
    "status": 1,
    "stacktrace": [],
    "api": "SetFilePointer",
    "return_value": 2198528,
    "arguments": {
        "file_handle": "0x000000c8",
        "move_method": 0,
        "offset": 2198528
    },
    "time": 1530616004.265,
    "tid": 2960,
    "flags": {}
},
```


Figure 3.3 Sample of a JSON file

In order to establish the system depicted in Figure 3.4, the initial step is to install Ubuntu onto a high-performance computer. Subsequently, the most current version of VirtualBox must be installed onto Ubuntu. The operating system that is to be executed on VirtualBox is the 32-bit version of Windows 7. The installation process of the Cuckoo Sandbox on Ubuntu involves using the terminal and making specific configurations. The following instructions will guide you through the process.

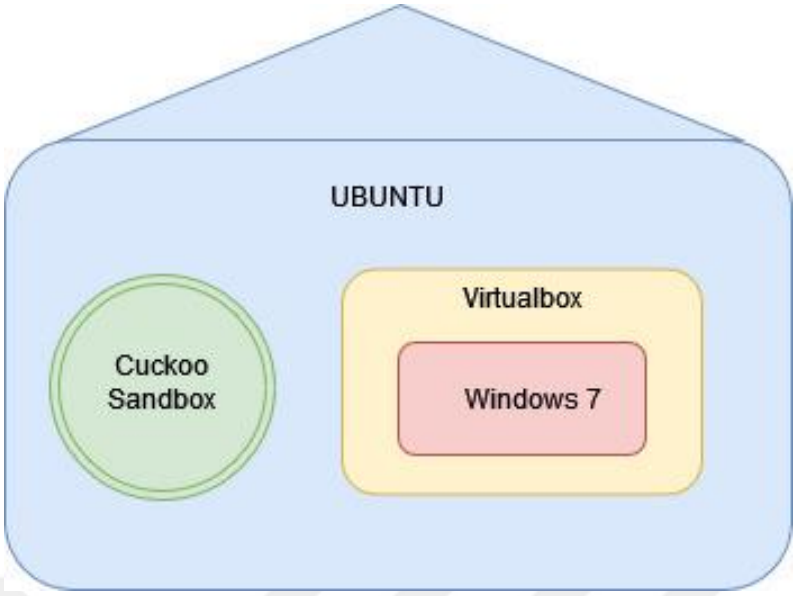

Figure 3.4 Malware lab implementation

In order to obtain accurate malware behavioral reports and ensure complete operation of all malware functions, it is imperative to properly configure Cuckoo Sandbox. In real-world scenarios, various malware samples exploit distinct vulnerabilities that may be present in specific software products. Hence, it is crucial to incorporate a diverse array of amenities within the virtual machines generated by the sandbox.

3.2.2 Setup Configuration

Cuckoo sandbox requires a minimum of two machines for its operation, namely the host and one or more guests.

The Host:

- Ubuntu 18.04 (latest Ubuntu version recommended)
- 2Gb RAM (minimum)
- VirtualBox (latest version)
- Cuckoo Sandbox 2.0 rc-1 (latest cuckoo version)

The Guest :

- WindowsXP (or Windows 7)-Installed software in the virtual machine

Preparing the Host:

- Installing Python and dependencies
- Configuring Tcpdump
- Installing Yara
- Installing Pydeep
- Installing Volatility
- Installing Cuckoo

Preparing the Guest:

- Installing VirtualBox
- Creating a Virtual Machine
- Installing and Configuring Guest OS
- Taking a snapshot
- Configuring Cuckoo
- Start Cuckoo

The following commands were executed in the terminal to configure the sandbox in Figures 3.5 through 3.8;

Preparing the Host

Installing Python and dependencies

$ sudo apt-get install git mongodb libffi-dev build-essential python-django python python-dev python-pip python-pil python-sqlalchemy python-bson python-dpkt python-jinja2 python-magic python-pymongo python-gridfs python-libvirt python-bottle python-pefile python-chardet tcpdump -y

Configuring Tcpdump

Tcpdump normally requires root privileges. However since Cuckoo is not going to run as root we need to configure it properly.

$ sudo setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump

Verify the results of the last command:

$ getcap /usr/sbin/tcpdump
/usr/sbin/tcpdump = cap_net_admin,cap_net_raw+eip

Installing Yara

$ sudo apt-get install autoconf libtool libjansson-dev libmagic-dev libssl-dev -y
$ wget https://github.com/plusvic/yara/archive/v3.4.0.tar.gz -O yara-3.4.0.tar.gz
$ tar -zxf yara-3.4.0.tar.gz
$ cd yara-3.4.0
$ ./bootstrap.sh
$ ./configure --with-crypto --enable-cuckoo --enable-magic
$ make
$ sudo make install

Validate the installation:

$ yara -v
yara 3.4.0

To build and install the yara-python extension:

$ cd yara-python
$ python setup.py build
$ sudo python setup.py install

Validate the installation

$ pip show yara-python

Figure 3.5 Commands runned from terminal to configure the sandbox-1

Name: yara-python
Version: 3.4.0
Location: /usr/local/lib/python2.7/dist-packages
Requires:

Installing Pydeep

Pydeep depends on ssdeep 2.8+

$ wget http://sourceforge.net/projects/ssdeep/files/ssdeep-2.13/ssdeep-2.13.tar.gz/download -O ssdeep-2.13.tar.gz
$ tar -zxf ssdeep-2.13.tar.gz
$ cd ssdeep-2.13
$ ./configure
$ make
$ sudo make install

Validate the installation

$ ssdeep -V

Then proceed by installing pydeep:

$ pip install pydeep

Validate that the package is installed:

$ pip show pydeep

Name: pydeep
Version: 0.2
Location: /usr/local/lib/python2.7/dist-packages
Requires:

Installing Volatility

Volatility is a tool used for forensic analysis on memory dumps. As stated in the Cuckoo configuration guide it is optional but you can might miss some rootkits if not installed.

$ pip install openpyxl
$ pip install ujson

Figure 3.6 Commands runned from terminal to configure the sandbox-2

$ pip install pycrypto
$ pip install distorm3
$ pip install pytz

To install volatility type the following commands:

$ git clone https://github.com/volatilityfoundation/volatility.git
$ cd volatility
$ python setup.py build
$ python setup.py install

Validate the installation by typing:

$ python vol.py -h

Installing Cuckoo

As a final step of our preparation we are going to download and install Cuckoo.

Download the latest Cuckoo sandbox version from the following site and create a folder.

https://cuckoosandbox.org/

Create a user to for Cuckoo.

$ useradd cuckoo

Make sure that the user who is going to run cuckoo is the owner of the files.

$ chown -R cuckoo:cuckoo <path/to/cuckoo/folder>

Preparing the guest

Issue the following **as cuckoo user**.

Installing VirtualBox

$ sudo apt-get install vitualbox

Add cuckoo user to the vboxusers group:

$ sudo usermod -a -G vboxusers cuckoo

Creating a Virtual Machine

Download windowsxpsp3.iso or windows7.iso according to your preferred options

(Notice that you might need to change the path ***/home/cuckoo/windowsxpsp3.iso*** for the iso according to your environment.)

Figure 3.7 Commands runned from terminal to configure the sandbox-3

```
$ vboxmanage createvm --name "windowsxp" --ostype WindowsXP --register
$ vboxmanage modifyvm "windowsxp" --memory 1000 --acpi on --boot1 dvd --nic1 nat
$ vboxmanage createhd --filename "windowsxp.vdi" --size 12000
$ vboxmanage storagectl "windowsxp" --name "IDE Controller" --add ide --controller PIIX4
$ vboxmanage storageattach "windowsxp" --storagectl "IDE Controller" --port 0 --device 0 --type hdd --medium "windowsxp.vdi"
$ vboxmanage storageattach "windowsxp" --storagectl "IDE Controller" --port 0 --device 1 --type dvddrive --medium /home/cuckoo/windowsxpsp3.iso
$ vboxmanage hostonlyif create
$ vboxmanage modifyvm "windowsxp" --nic1 hostonly
$ vboxmanage modifyvm "windowsxp" --hostonlyadapter1 vboxnet0
```

Create a share for the guest as we are going to need it in order to transfer the cuckoo agent from the host to the guest. (You can either do it manually: Create a shared folder and copy cuckoo agent.py to it)

```
$ mkdir -p /home/cuckoo/VirtualBox VMs/windowsxpshare
$ vboxmanage sharedfolder add "windowsxp" --name "windowsxpshare" --hostpath /home/cuckoo/VirtualBox VMs/windowsxpshare --automount
$ cp /home/cuckoo/cuckoo/agent/agent.py /home/cuckoo/VirtualBox VMs/windowsxpshare/agent.py
```

We used a host-only adapter for the guest VM in order to isolate it from the rest of the network. However we are going to need internet connectivity from the VM in order for the analysis to work. We can achieve that by adding the following rules to the iptables.

```
$ sudo iptables -A FORWARD -o eth0 -i vboxnet0 -s 192.168.56.0/24 -m conntrack --ctstate NEW -j ACCEPT
$ sudo iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
$ sudo iptables -A POSTROUTING -t nat -j MASQUERADE
$ sudo sysctl -w net.ipv4.ip_forward=1
```

Figure 3.8 Commands runned from terminal to configure the sandbox-4

Installing and Configuring Guest Operating System (OS)

Virtual machine is launched and installed OS.

$ vboxmanage startvm "windowsxp"

Enough time until the installation process was completed is expected. Once finished, network settings are configured from the window shown in Figure 3.9.

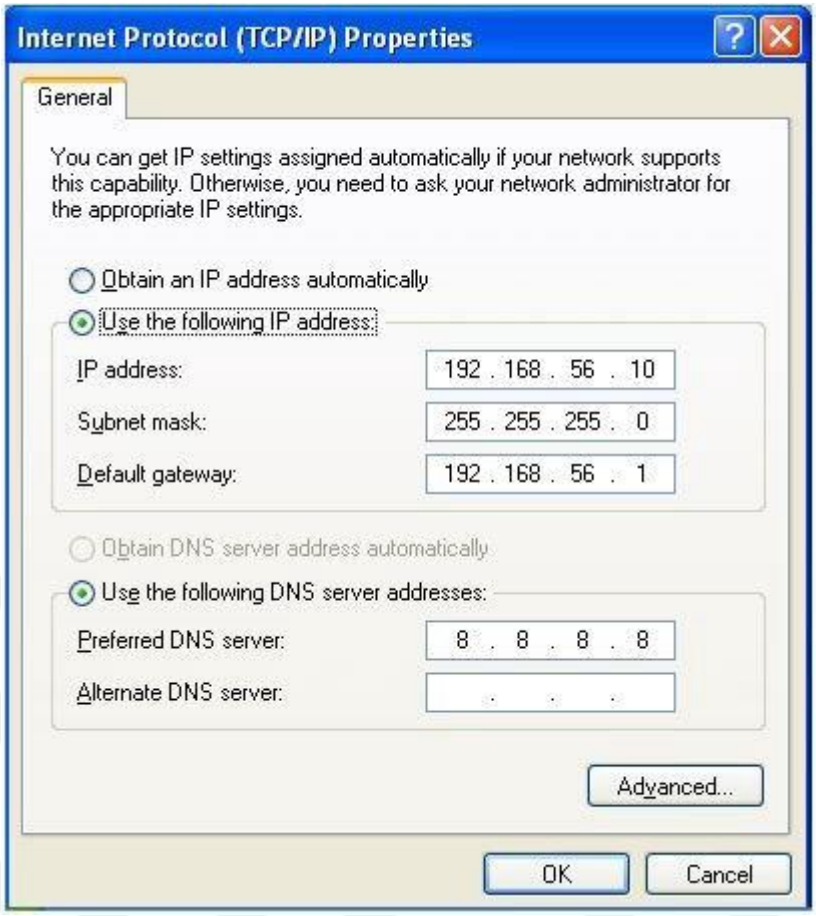

Figure 3.9 Windows firewall and automatic updates

It is necessary to disable both the Windows Firewall and Automatic Updates in order to have a virtual machine that is as vulnerable as possible for analysis. However, it should be noted that this may not apply in all situations.

VirtualBox Guest Additions is downloaded and installed through the VirtualBox interface.

Prior to being ready for Cuckoo, the guest requires certain configurations to be installed. These include the following software that must be downloaded and installed on the guest:

Python 2.7 https://www.python.org/downloads/release/python-2711/

Python Imaging Library for windows http://effbot.org/downloads/PIL-1.1.7.win32-py2.7.exe

The final step involves the installation of the Cuckoo Agent, which operates an XMLRPC server that is continuously listening for connections.

The share is mounted from Windows and the file agent.py is moved to the Documents directory in C drive and Settings/All Users/Start Menu/Programs/Startup. This will allow for the automatic initialization of the agent upon the powering on of the virtual machine. Alternatively, the agent can be launched manually each time.

Taking a snapshot

Upon completion of the Guest configuration, it is crucial to take a snapshot with the guest VM powered on and the agent running. Failure to do so will hinder Cuckoo's ability to conduct the analysis. Alternatively, can be manually obtained a snapshot and restore it through the VirtualBox interface. The relevant commands are shown in Figure 3.10.

```
$ vboxmanage snapshot "windowsxp" take "snapshot1" --pause
$ vboxmanage controlvm "windowsxp" poweroff
$ vboxmanage snapshot "windowsxp" restorecurrent
```

Figure 3.10 Commands of taking a snapshot

Configuring Cuckoo

It is necessary to configure at least the following files for Cuckoo to function correctly. Commands of configuring Cuckoo are shown in Figure 3.11.

cuckoo.conf

```
machinery = virtualbox
[resultserver]
ip = 192.168.56.1 #This is the IP address of the host
port = 2042 #leave default unless you have services running
```

auxiliary.conf

```
[sniffer]
# Enable or disable the use of an external sniffer (tcpdump) [yes/no].
enabled = yes
# Specify the path to your local installation of tcpdump. Make sure this
# path is correct.
# You can check this using the command: whereis tcpdump
tcpdump = /usr/sbin/tcpdump
# Specify the network interface name on which tcpdump should monitor the
# traffic. Make sure the interface is active.
# The ifconfig command will show you the interface name.
interface = vboxnet0
```

virtualbox.conf

```
machines = windowsxp
[windowsxp]
label = windowsxp
platform = windows
ip = 192.168.56.10 # IP address of the guest
snapshot = snapshot1 # name of snapshot
```

reporting.conf

```
[mongodb]
enabled = yes
```

Figure 3.11 Commands of configuring Cuckoo

Start Cuckoo

To initiate cuckoo, the subsequent commands are executed on the host.

$ cd /home/cuckoo/cuckoo

$ python cuckoo.py

Web interface is started:

$ cd /home/cuckoo/cuckoo/web

$ python manage.py runserver 0.0.0.0:8000

Web interfaces of cuckoo are shown in Figures 3.12 through 3.15.

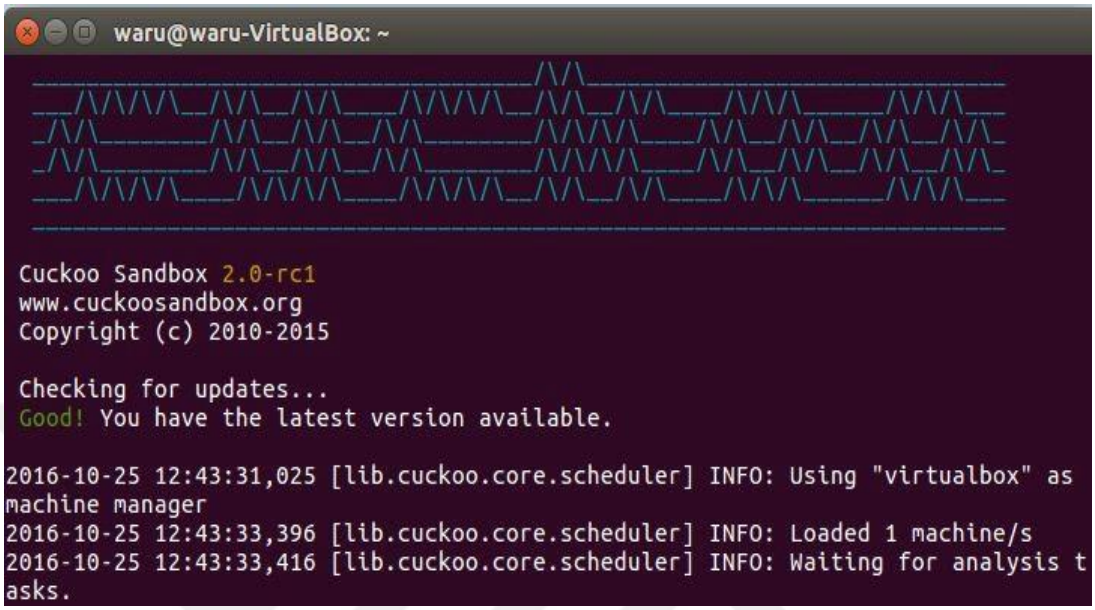

Figure 3.12 Web interface of start Cuckoo

Web browser is opened and typed host ip and port to access Cuckoo's web interface. (Ex: 0.0.0.0:8000)

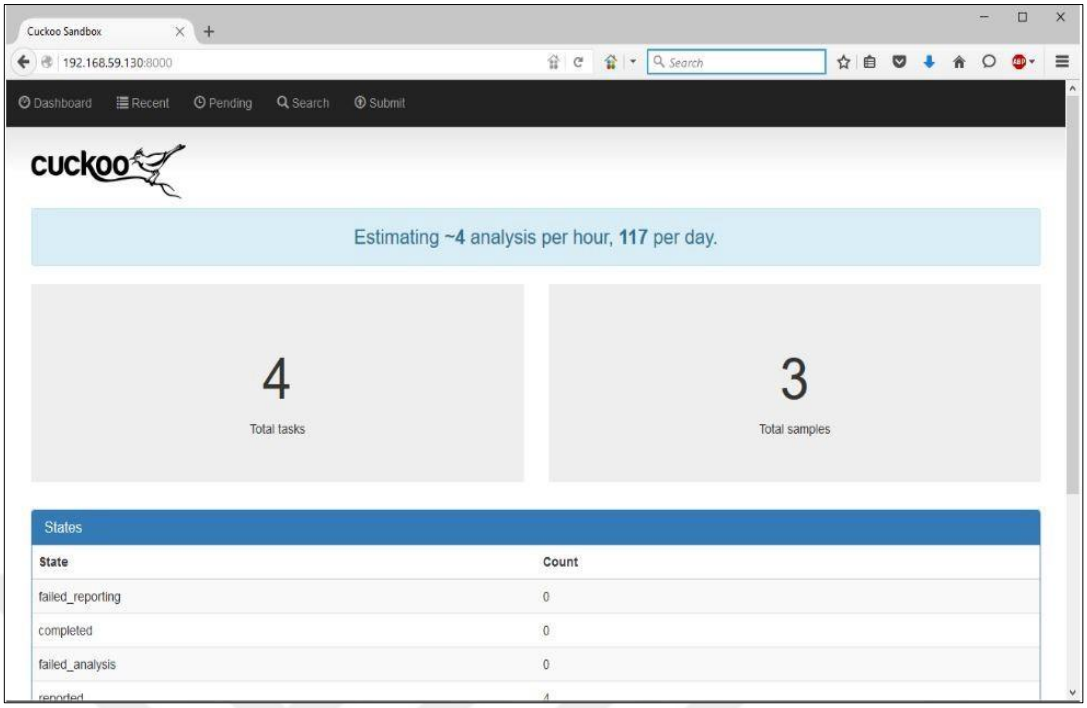

Figure 3.13 Web interface of Cuckoo-1

A file can be submitted through web interface and the results can be checked.

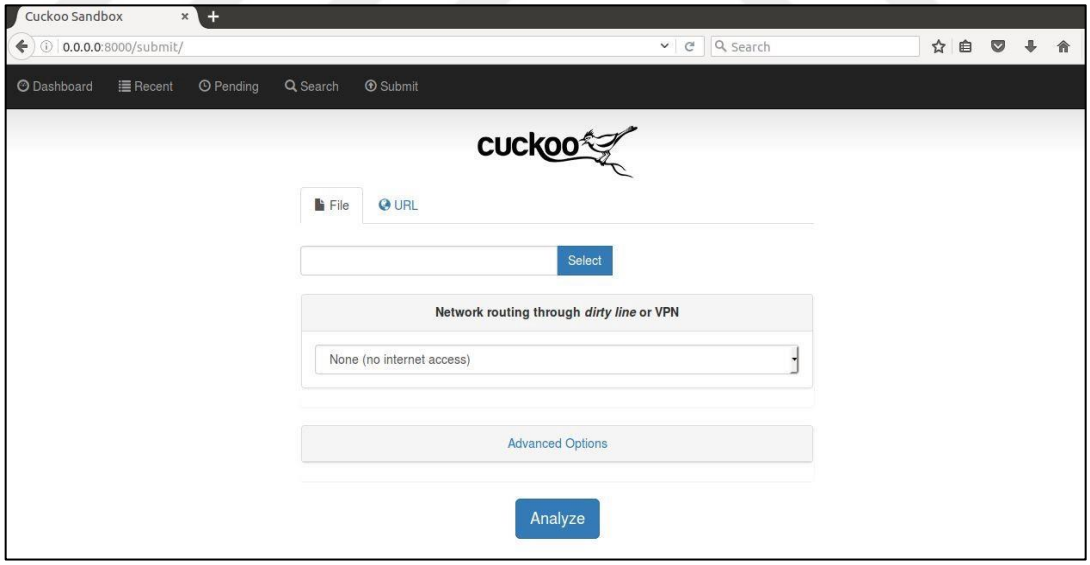

Figure 3.14 Web interface of Cuckoo-2

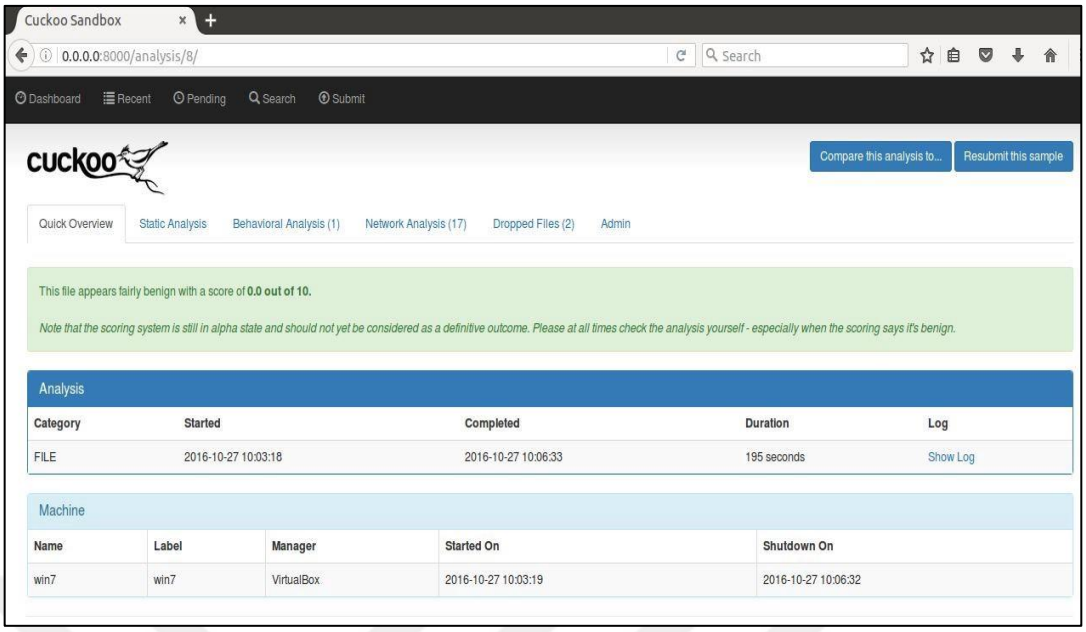

Figure 3.15 Web interface of Cuckoo-3

3.2.3 Reports and Features

In order to utilize machine learning algorithms for the problem, it is imperative to determine the nature of the data that needs to be extracted and the criteria for selecting it based on its pertinence.

Certain works within the field are implementing the utilization of string properties or file format properties as a foundation for the extraction of features. One example of this is the analysis of Windows-based malware samples, where the information found within the PE headers is frequently employed as a core element for examination. The implementation of format-specific feature extraction may not be the most optimal solution due to the significant variation in the formats of the files being analyzed [88].

Other studies make use of n-grams, which refer to overlapping substrings that are collected in a sliding-window manner. Byte n-grams, for example, consist of fixed-sized windows that slide one byte at a time. Meanwhile, word n-grams and character n-grams are commonly employed in various fields such as natural language processing, information retrieval and text mining [89].

The use of byte n-grams as a feature for malware detection has its drawbacks. The primary challenge is the enormity of the set of byte n-grams derived from both malware and benign programs. It is impractical to apply classification techniques directly to this set, as stated by Reddy and Pujari in 2006. Furthermore, this methodology restricts the capacity to detect polymorphic malware. As a consequence, specimens that exhibit identical behavior will yield distinct strings and, consequently, diverse n-grams.

Due to the aforementioned reasons, it has been determined in this study that the actual behavior of the files will be relied upon, as monitored by the sandbox.

This section will discuss the features that should be utilized in our work from the list provided above.

3.2.3.1 Files

The reports provide details on files that have been opened, written to and created. This type of information is valuable for predicting the malware family, as any malware files will cause significant modifications to the file system. The file modification approach is generally effective for accurately classifying malware in most cases. However, it may not be sufficient for distinguishing between different families of ransomware. Ransomware encrypts all files on a system, making up the majority of its feature set. The differences between ransomware families lie in the smaller portion of the feature set that contains malware settings. Making predictions based on this data would be very difficult.

3.2.3.2 Registry Keys

The Windows registry contains important system settings and application information. When a program runs, it makes many changes to the registry. Cuckoo reports provide information on registry key actions, such as opening, reading, writing or deleting. This data is useful for identifying system changes caused by malware and detecting malware.

3.2.3.3 Mutexes

The term "mutex" refers to "Mutual Exclusion." It denotes a program object that enables multiple threads to access a shared resource. Upon program initiation, a unique mutex is created with a distinct name. The names of mutexes can serve as effective identifiers for different malware samples. Nevertheless, the families' approach cannot yield precise results on a significant scale due to the significantly lower number of mutexes generated per sample compared to the dataset. Consequently, even a slight alteration related to the bug or non-initiated process can result in a significant deviation in the prediction outcomes.

3.2.3.4 Processes

The typical way to identify a particular malware sample is by its process name. However, this method is not often effective in identifying the malware family since the process names usually match the sample hash. An alternative approach for the malware sample is to inject itself into the system process, which renders the feature unsuitable for family identification purposes.

3.2.3.5 IP Addresses and Domain Name Server (DNS) Queries

Cuckoo provides information about network traffic in PCAP format, which can be used to extract data about IP addresses and DNS queries. This helps identify attackers' command and control servers and accurately determine the malware family in most cases." Attackers often change the domain names or IP addresses of their servers and may use spoofing techniques, so relying solely on this information is not always reliable.

3.2.3.6 API Calls and DLL Files

API stands for Application Programming Interface and refers to a set of tools that enable communication between software components. API calls made during malware execution are associated with a specific process. The documentation provides an overview of all the events that take place in the operating system, including file

activities, registry functions, mutex operations and processes. This includes various API calls such as OpenFile, OpenFileEx, CreateFile, CopyFileEx and so on.

In software engineering, DLL (Dynamic Link Library) files are external libraries containing code and data that can be utilized by multiple programs simultaneously. These files are dynamically linked during runtime, allowing software applications to share resources and functionality without redundantly including the same code in each program. DLL files serve various purposes, including providing common functionality, facilitating code reuse, and enhancing modularization in software development. These libraries are essential components of the Windows operating system and are widely used in software development across various platforms to improve efficiency, maintainability, and scalability of applications.

3.2.3.7 Memory

Cuckoo Sandbox provides the capability to integrate with Volatility, enabling the initiation of memory analysis tasks with a chosen number of plugins once the dynamic analysis task has been completed. This feature is particularly useful in detecting advanced targeted attacks that employ various obfuscation and stealth techniques, which cannot be fully identified without conducting memory analysis. The output of the Volatility plugins is included in the analysis report alongside the Cuckoo Sandbox.

The present thesis utilizes the features of API calls and DLL files obtained from the results of Cuckoo Sandbox. As per the findings of prior research, API calls offer more precise outcomes in the identification of malware.

3.2.4 Feature Extraction

In our implementation, the reports are first processed by the sandbox and then stored locally. These reports are subsequently utilized as input to the feature extraction script, which generates a csv file using Python. Based on the feature extraction process, the resulting dataset consists of 335 attributes (features) and 1 class (either malware or benign). The Python code parses solely the API calls and DLL files from the JSON-formatted sandbox output.

Below is the compilation of the extracted features in Figure 3.16 and Figure 3.17:

Figure 3.16 The list of the features extracted-1

Section,NtOpenDirectoryObject,NtOpenFile,NtOpenKey,NtOpenKeyEx,NtOpenMutant,NtOpenProcess,NtOpenSection,NtOpenThread,NtProtectVirtualMemory,NtQueryAttributesFile,NtQueryDirectoryFile,NtQueryInformationFile,NtQueryKey,NtQuerySystemInformation,NtQueryValueKey,NtQueueApcThread,NtReadFile,NtResumeThread,NtSetContextThread,NtSetInformationFile,NtSetValueKey,NtTerminateProcess,NtUnmapViewOfSection,NtWriteFile,OLE32dll,OLEACCdll,OLEAUT32dll,OleInitialize,OpenSCManagerA,OpenSCManagerW,OpenServiceA,OpenServiceW,OskSupportdll,POWRPROFdll,PRF,PROPSYSdll,PSAPIDLL,PSHEDdll,Process32FirstW,Process32NextW,REGAPIdll,RPCRT4dll,ReAgentdll,ReadProcessMemory,RegCloseKey,RegCreateKeyExA,RegCreateKeyExW,RegEnumKeyExA,RegEnumKeyExW,RegEnumKeyW,RegEnumValueW,RegOpenKeyExA,RegOpenKeyExW,RegQueryInfoKeyW,RegQueryValueExA,RegQueryValueExW,RegSetValueExA,RegSetValueExW,RtlAddVectoredContinueHandler,SETUPAPIdll,SHELL32dll,SHGetFolderPathW,SHGetSpecialFolderLocation,SHLWAPIdll,SLCdll,SPPdll,SearchPathW,Secur32dll,SendNotifyMessageW,SetEndOfFile,SetErrorMode,SetFileAttributesW,SetFilePointer,SetFileTime,SetUnhandledExceptionFilter,SetWindowsHookExA,ShellExecuteExW,SizeofResource,SpectrumSyncClientdll,SspiClidll,StartServiceA,StartServiceW,StateRepositoryCoredll,TAPI32dll,TQUERYDLL,UIAutomationCoreDLL,UNATTENDDLL,UNPShareddll,URLMONDLL,USER32dll,USERENVdll,UnhookWindowsHookEx,UuidCreate,UxThemedll,VDMDBGdll,VERSIONdll,VSSAPIDLL,VirtDiskdll,WDSCOREdll,WIMGAPIDLL,WININETdll,WINMMdll,WINSPOOLDRV,WINSTAdll,WINTRUSTdll,WLDAP32dll,WMsgAPIdll,WOFUTILdll,WS2_32dll,WSARecv,WSASend,WSASocketW,WSAStartup,WSOCK32dll,WTSAPI32dll,WinJsondll,WriteProcessMemory,XmlLitedll,__exception__,bcddll,bcryptdll,bind,certclidll,chakradll,clbdll,closesocket,cngsys,combasedll,connect,creduidll,d2d1dll,d3d11dll,d3d12dll,dbghelpdll,dcompdll,dmEnrollEngineDLL,dpxdll,dwmapidll,dxgidll,gdiplusdll,getaddrinfo,gethostbyname,getsockname,iedkcs32dll,iertutildll,kdcomdll,ktmw32dll,msIsodll,mscoreedll,msdeltadll,msdrmdll,msidll,msrpcsys,msvcp110_windll,msvcp_windll,ncryptdll,netwphelperdll,newdevdll,ntdlldll,omadmapidll,pdhdll,policymanagerdll,profapidll,send,setsockopt,shfolderdll,skcidll,snmpapidll,socket,twinapiappcoredll,ulibdll,unbcldll,vrfcoredll,webservicesdll,werdll,wlanapidll,class

Figure 3.17 The list of the features extracted-2

3.2.5 Feature Selection

The objective of feature selection is to eliminate irrelevant features from the feature set when it becomes excessively large. Larger feature sets are more difficult to work with, but some features in the set may not contribute to the algorithm's decision and can thus be eliminated.

Feature selection methods can generally be classified into three main types: filtering, wrapper, and embedded methods [74].

Filtering methods use statistical scoring to determine which features to keep in a dataset based on their scores. Wrapper methods involve testing various feature combinations using a prediction model to determine the combination that yields the highest accuracy. The embedded methods are techniques that assess the features utilized during the model's development.

Feature selection is a technique utilized for eliminating redundant and irrelevant features, thereby enhancing the accuracy of prediction. The rationale behind incorporating multi-objective genetic algorithms alongside conventional feature selection methods in this paper is to account for interactions between features. These algorithms are specifically employed to address the complexity of feature interactions, thereby enhancing the effectiveness of the feature selection process [90].

The utilization of Multi-Objective Genetic Algorithms is intended for executing the selection of features and the application of machine learning techniques. A program in Java has been developed to employ the Genetic Algorithms through the utilization of the MOEA Framework. As shown in Figure 3.18, the optimization algorithms utilized in this paper include PAES, DBEA, RSO, RANDOM and RVEA.

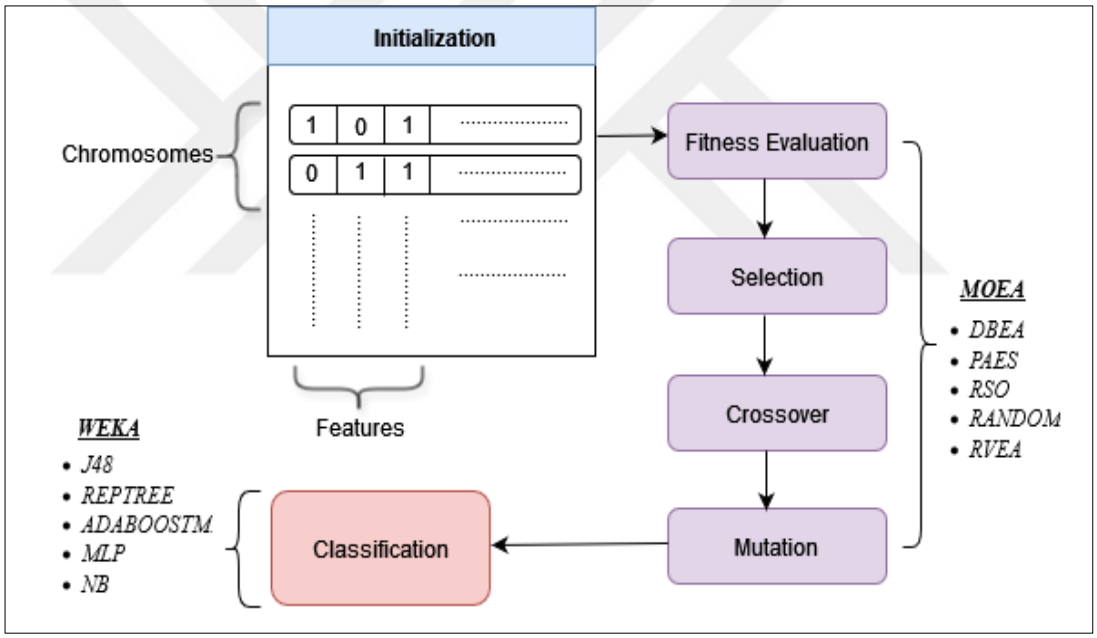

Figure 3.18 Combining of multi objective genetic algorithms and classification methods

The PAES algorithm produced the optimal outcome with a 93,33% accuracy rate and the elimination of 135 attributes, using the REP Tree classifier. The feature count was reduced from 335 to 200 following the selection model.

3.2.6 Application of Machine Learning Methods

After extracting and selecting features, the next step is to use machine learning methods such as J48, AdaBoostM1, Rep Tree, Multilayer Perceptron and Naive Bayes.

The approach involves analyzing malware samples dynamically and submitting the analysis reports to a database. The characteristics are extracted and feature vectors are generated for each sample. The malware samples are then divided into training and testing sets, adopting a split ratio of 70% for training and 30% for testing. This division is achieved using a train-test split methodology rather than cross-validation. The final section of the task involves testing the classifier by classifying malware samples in the test set. We use WEKA for traditional feature selection methods and JAVA program for multi-objective feature selection methods in classification task with machine learning algorithms. General model for malware classification is shown in in Figure 3.19.

Figure 3.19 Malware Classification Process

CHAPTER 4

EXPERIMENTAL RESULTS

The described system has been implemented and tested across various conditions. In each testing scenario, the resulting shape is evaluated to determine its similarity to the target shape. The program was executed on a computer equipped with an Intel Core i7-7700HQ processor (2.8 GHz, 4 cores) and 16 GB of memory. All tests were conducted with an average duration of 48 hours, and the outcomes were meticulously recorded.

4.1 Evaluation

Ten metrics used for measuring the performance of a system are Kappa, Mean Absolute Error (MAE), Root Mean Square Error (RMSE), False Positive (FP) Rate, Precision, Recall, F-Measure and Accuracy. Kappa is calculated as follows Equation 4.1.

$$K = \frac{p_0 - p_e}{1 - p_e} \quad \textbf{(4.1)}$$

In this case, p_e represents the likelihood of agreement altering, while p_0 stands for the agreement observed in rates. If the observed and anticipated values align, k will equal 1; otherwise, it will be ≤0. Any value of k that surpasses 0 indicates that the outcomes are more favorable than mere coincidence.

The MAE calculates the average difference between the actual values and the predicted values. Unlike MAE, it is less affected by extreme values and does not magnify errors. It provides a measure of the deviation between predictions and the real output. However, because MAE uses the absolute value of the difference, it does not indicate whether the predictions are too low or too high. There is no doubt about the accuracy of error interpretation, as it corresponds directly to the degree of the variable. Unlike Mean Squared Error (MSE), MAE is not able to be differentiated. Mathematically, its represented in Equation 4.2.

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \check{y}_j| \tag{4.2}$$

Where:

y_j: ground-truth value

$\hat{y}_j$: predicted value from the regression model

N: number of datums

The square root of the average of the squared difference between the predicted and target values is known as the RMSE. It is essentially the square root of the MSE. By taking the square root, it maintains the differentiable property of MSE while also penalizing smaller errors. This allows for smoother error interpretation, as the scale is now the same as the random variable. Additionally, the RMSE is less likely to be affected by outliers due to the normalization of scale factors. This can be represented mathematically using Equation 4.3.

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2} \tag{4.3}$$

The False Positive Rate (FPR), or fallout, is a measure of how many negative data points are incorrectly labeled as positive. Put simply, a higher FPR means more negative data points are being misclassified. This can be represented mathematically using Equation 4.4.

$$FPR = \frac{\text{FP}}{FP+TN} \tag{4.4}$$

$$Precision = \frac{\text{TP}}{TP+FP} \tag{4.5}$$

$$Recall = \frac{\text{TP}}{TP+FN} \tag{4.6}$$

$$F1-Score=\frac{2\times Precision\times Recall}{Precision+Recall} \quad \textbf{(4.7)}$$

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN} \quad \textbf{(4.8)}$$

where, Eqs. (4.4)-(4.8), the true positive values are denoted as TP, false positive values as FP, while TN and FN represent the values of true negative and false negative calculations, respectively.

Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Correlation Attribute Evaluation is shown in Table 4.1.

Table 4.1 Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Correlation Attribute Evaluation (Before-After Selection)

			J48	**AdaboostM1**	**RepTree**	**MultilayerPerceptron**	**NaiveBayes**
Evaluator with CorrelationAttributeEval	**Before Feature Selection**	**Kappa**	*0.7966*	*0.7059*	*0.7952*	*0.7059*	*0.331*
		MAE	*0.1243*	*0.1692*	*0.1604*	*0.1634*	*0.3167*
		RMSE	*0.2637*	*0.2715*	*0.2692*	*0.3639*	*0.5627*
		FP Rate	*0,109*	*0,123*	*0,115*	*0,123*	*0,367*
		Precision	*0,901*	*0,888*	*0,906*	*0,888*	*0,712*
		Recall	*0,900*	*0,850*	*0,900*	*0,850*	*0,683*
		F-Measure	*0,900*	*0,849*	*0,899*	*0,849*	*0,659*
		Accuracy	*% 90*	*% 85*	*% 90*	*% 85*	*% 68,33*
	After Feature Selection	**Kappa**	*0.7966*	*0.7952*	*0.7952*	*0.6429*	*0.331*
		MAE	*0.1243*	*0.1662*	*0.1604*	*0.1797*	*0.3167*
		RMSE	*0.2637*	*0.2701*	*0.2692*	*0.3819*	*0.5627*
		FP Rate	*0,109*	*0,115*	*0,115*	*0,150*	*0,367*
		Precision	*0,901*	*0,906*	*0,906*	*0,870*	*0,712*
		Recall	*0,900*	*0,900*	*0,900*	*0,817*	*0,683*
		F-Measure	*0,900*	*0,899*	*0,899*	*0,814*	*0,659*
		Accuracy	*% 90*	*% 90*	*% 90*	*% 81,67*	*% 68,33*

Table 4.1 presented the quantitative metric results before and after feature selection for J48, AdaboostM1, Rep Tree, Multilayer Perceptron, and Naive Bayes. The Kappa values remained the same for J48, Rep Tree, and Naive Bayes. This means that there is no improvement in Kappa value after feature selection. When the Kappa value of

AdaboostM1 is analyzed, the Kappa value before feature selection is 0.7059, while after feature selection it is 0.7952. With feature selection, the kappa value of AdaboostM1 is improved by approximately 12.65%. After evaluating the Kappa results of the Multilayer Perceptron after feature selection, there is a reduction in the Kappa rate.

When machine learning methods are evaluated in terms of their MAE and RMSE values to investigate the effectiveness of feature selection, the J48 produces the best results, which are 0.1243 and 0.2637, respectively. Although J48 provides the best results, it should not be overlooked that the same results are obtained with feature selection. On the other hand, before feature selection, the MAE and RMSE values of AdaboostM1, Rep Tree, Multilayer Perceptron, and Naive Bayes are calculated as being 0.1692 to 0.2701, 0.1604 to 0.2692, 0.1634 to 0.3639, and 0.3167 to 0.5627, respectively. After feature selection, The MAE and RMSE rates of AdaboostM1, Rep Tree, Multilayer Perceptron, and Naïve Bayes are 0.2715 to 0.2701, 0.2692 to 0.2692, 0.3639 to 0.3819, and 0.5627 to 0.5627, respectively. When the MAE values given in Table 4.1 are statistically assessed, no improvement is observed in the results of all methods except Multilayer Perceptron.

FP Rate values of other algorithms except Naive Bayes classifier vary between 0.10-0.11. The FP Rate value of the Naive Bayes classifier is 0.367. There is no significant change in FP Rate values after feature selection. In this respect, other algorithms except Naive Bayes classifier perform well. When Table 4.1 is analyzed, it is clearly seen that the Precision, Recall, and F-Measure rates of machine learning methods vary between 0.683 and 0.906. A significant improvement is seen in the metric values of AdaboostM1 with feature selection. More specifically, while the Precision, Recall, and F-Measure values of AdaboostM1 before feature selection are 0.888, 0.850, and 0.849, these values are obtained as 0.906, 0.900, and 0.899 after feature selection. Following the feature selection, AdaboostM1 shows enhancements of 2.03%, 5.88%, and 5.89% in Precision, Recall, and F-Measure values, respectively. When considering Accuracy rates, it is recognized that feature selection does not impact other methods except for AdaboostM1 and Multilayer Perceptron. The accuracy of AdaboostM1 is increased while the accuracy of Multilayer Perceptron is decreased after feature selection.

Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Info Gain Attribute Evaluation is shown in Table 4.2.

Table 4.2 Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Info Gain Attribute Evaluation (Before-After Selection)

			J48	AdaboostM1	RepTree	MultilayerPerceptron	NaiveBayes
Evaluator with InfoGainAttributeEval	Before Feature Selection	**Kappa**	*0.7966*	*0.7059*	*0.7952*	*0.7059*	*0.331*
		MAE	*0.1243*	*0.1692*	*0.1604*	*0.1634*	*0.3167*
		RMSE	*0.2637*	*0.2715*	*0.2692*	*0.3639*	*0.5627*
		FP Rate	*0,109*	*0,123*	*0,115*	*0,123*	*0,367*
		Precision	*0,901*	*0,888*	*0,906*	*0,888*	*0,712*
		Recall	*0,900*	*0,850*	*0,900*	*0,850*	*0,683*
		F-Measure	*0,900*	*0,849*	*0,899*	*0,849*	*0,659*
		Accuracy	*% 90*	*% 85*	*% 90*	*% 85*	*% 68,33*
	After Feature Selection	**Kappa**	*0.7377*	*0.7059*	*0.7059*	*0.7377*	*0.331*
		MAE	*0.1944*	*0.1833*	*0.1715*	*0.154*	*0.3167*
		RMSE	*0.3213*	*0.3058*	*0.2946*	*0.2834*	*0.5627*
		FP Rate	*0,109*	*0,123*	*0,123*	*0,109*	*0,367*
		Precision	*0,897*	*0,888*	*0,888*	*0,897*	*0,712*
		Recall	*0,867*	*0,850*	*0,850*	*0,867*	*0,683*
		F-Measure	*0,866*	*0,849*	*0,849*	*0,866*	*0,659*
		Accuracy	*% 86,67*	*% 85*	*% 85*	*% 86,67*	*% 68,33*

Table 4.2 displayed the quantitative metric outcomes before and after feature selection using Info Gain Attribute Evaluation for J48, AdaboostM1, Rep Tree, Multilayer Perceptron, and Naive Bayes. The Kappa values for AdaboostM1 and Naive Bayes remain unchanged, indicating no enhancement in Kappa values after feature selection. For Multilayer Perceptron, the Kappa value increased from 0.7059 to 0.7377 after feature selection, showing an improvement of around 4.50%. Conversely, the Kappa rates for J48 and Rep Tree decreased following feature selection. When evaluating machine learning techniques based on their MAE, RMSE, and FP rates to assess the impact of feature selection, there is only an improvement in the values of Multilayer Perceptron.

Upon analyzing Table 4.1, it is evident that the Precision, Recall, and F-Measure scores of classifiers range from 0.683 to 0.90. Similar to the previous metrics, only the

Multilayer Perceptron's values are improved. In accuracy metric values, a very small improvement rate of 1.96% is occurred in Multilayer Perceptron.

Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Evaluator with Chi Square Attribute Evaluation is shown in Table 4.3.

Table 4.3 Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Chi Square Attribute Evaluation (Before-After Selection)

			J48	AdaboostM1	RepTree	MultilayerPerceptron	NaiveBayes
Evaluator with ChiSquareAttributeEval	Before Feature Selection	Kappa	*0.7966*	*0.7059*	*0.7952*	*0.7059*	*0.331*
		MAE	*0.1243*	*0.1692*	*0.1604*	*0.1634*	*0.3167*
		RMSE	*0.2637*	*0.2715*	*0.2692*	*0.3639*	*0.5627*
		FP Rate	*0,109*	*0,123*	*0,115*	*0,123*	*0,367*
		Precision	*0,901*	*0,888*	*0,906*	*0,888*	*0,712*
		Recall	*0,900*	*0,850*	*0,900*	*0,850*	*0,683*
		F-Measure	*0,900*	*0,849*	*0,899*	*0,849*	*0,659*
		Accuracy	*% 90*	*% 85*	*% 90*	*% 85*	*% 68,33*
	After Feature Selection	Kappa	*0.7377*	*0.7059*	*0.7059*	*0.7377*	*0.331*
		MAE	*0.1944*	*0.1833*	*0.1715*	*0.1571*	*0.3167*
		RMSE	*0.3213*	*0.3058*	*0.2946*	*0.2874*	*0.5627*
		FP Rate	*0,109*	*0,123*	*0,123*	*0,109*	*0,367*
		Precision	*0,897*	*0,888*	*0,888*	*0,897*	*0,712*
		Recall	*0,867*	*0,850*	*0,850*	*0,867*	*0,683*
		F-Measure	*0,866*	*0,849*	*0,849*	*0,866*	*0,659*
		Accuracy	*% 86,67*	*% 85*	*% 85*	*% 86,67*	*% 68,33*

Table 4.3 showed the results of quantitative metrics before and after feature selection using Info Gain Attribute Evaluation for various machine learning algorithms. In Kappa values of other algorithms except Multilayer Perceptron classifier, there is a reduction. the Kappa value for Multilayer Perceptron is increased from 0.7059 to 0.7377, showing a 4.50% improvement. When assessing MAE, RMSE, and FP rates of classifiers, after feature selection, Multilayer Perceptron had an improvement of 3.86% for MAE, 20.02% for RMSE, and 11.38% for FP rate, respectively. In terms of Accuracy rates, it is acknowledged that feature selection has no effect on other classifiers except Multilayer Perceptron.

Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Evaluator with Genetic Algorithm is shown in Table 4.4.

Table 4.4 Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Genetic Algorithm (Before-After Selection)

			J48	AdaboostM1	RepTree	MultilayerPerceptron	NaiveBayes
Evaluator with Genetic Algorithm	Before Feature Selection	**Kappa**	*0.7966*	*0.7059*	*0.7952*	*0.7059*	*0.331*
		MAE	*0.1243*	*0.1692*	*0.1604*	*0.1634*	*0.3167*
		RMSE	*0.2637*	*0.2715*	*0.2692*	*0.3639*	*0.5627*
		FP Rate	*0,109*	*0,123*	*0,115*	*0,123*	*0,367*
		Precision	*0,901*	*0,888*	*0,906*	*0,888*	*0,712*
		Recall	*0,900*	*0,850*	*0,900*	*0,850*	*0,683*
		F-Measure	*0,900*	*0,849*	*0,899*	*0,849*	*0,659*
		Accuracy	*% 90*	*% 85*	*% 90*	*% 85*	*% 68,33*
	After Feature Selection	**Kappa**	*0,8671*	*0,8344*	*0,8671*	*0,7312*	*0,2933*
		MAE	*0,1148*	*0,1262*	*0,1046*	*0,1741*	*0,3333*
		RMSE	*0,2234*	*0,2362*	*0,2139*	*0,3241*	*0,5772*
		FP Rate	*0,1212*	*0,1515*	*0,1212*	*0,140*	*0,241*
		Precision	*0,8709*	*0,8437*	*0,8709*	*0,881*	*0,7692*
		Recall	*1*	*1*	*1*	*0,867*	*0,3704*
		F-Measure	*0,9310*	*0,9152*	*0,9310*	*0,865*	*0,5*
		Accuracy	*% 93,33*	*% 91,67*	*% 93,33*	*% 86,67*	*% 66,67*

The Kappa, MAE, RMSE, FP Rate, Precision, Recall, F-Measure and Accuracy values with Evaluator based on Genetic Algorithm is shown in Table 4.4. First of all, when the Kappa values are analyzed, there is a visible improvement in the metric values of classifiers. While the kappa values of J48, AdaboostM1, Rep Tree, Multilayer Perceptron, and Naive Bayes before feature selection are 0.7966, 0.7059, 0.7952, 0.7059, and 0.331, respectively, these values are obtained as 0,8671, 0,8344, 0,8671, 0,7312, and 0,2933 after feature selection. By using the genetic algorithm in feature selection, the improvement rates are calculated as 8.85%, 18.20%, 9.04%, 3.58%, and 8.11%, respectively, for J48, AdaboostM1, Rep Tree, and Multilayer Perceptron. When classifiers are evaluated in terms of their MAE and RMSE values to investigate the effectiveness of feature selection with genetic algorithm, the Rep Tree produces the best results, which are 0.1046 and 0.2139, respectively. After feature selection, improvements of 34.79% and 20.54% are achieved in the MAE and RMSE values of

Rep Tree, respectively. Before feature selection, the MAE and RMSE values of J48, AdaboostM1, Multilayer Perceptron, and Naive Bayes are calculated as being 0.1243 to 0.1148, 0.1692 to 0.1262, 0.1634 to 0.1741, 0.3167 to 0.3333, respectively. After feature selection, the MAE and RMSE rates are 0.2637 to 0.2234, 0.2715 to 0.2362, 0.3639 to 0.3241, and 0.5627 to 0.5772, respectively. The FP rates of J48, AdaboostM1, Rep Tree, Multilayer Perceptron, and Naive Bayes are 0.109-0.1212, 0.123-0.1515, 0.115-0.1212, 0.123-0.140, and 0.367-0.241, respectively.

The effectiveness of feature selection is investigated by evaluating the classifiers based on their Precision, Recall, and F-Measure values. Except for Naive Bayes, there is a significant improvement in the Recall, and F-Measure values of the other classifiers.

While Accuracy rates of J48, AdaboostM1, and Rep Tree are 90%, 85%, and 90% before feature selection, these rates are calculated as 93,33%, 91,67%, and 93,33% after feature selection. It is clearly seen from these values that a satisfactory improvement in accuracy values is achieved with multi-objective genetic algorithm-based feature selection.

4.2 Accuracy, Number of Selected Features and Time Taken to Test

Accuracy, in the context of classification, denotes to the ratio of accurately identified instances, including both true positives and true negatives, in relation to the total number of instances retrieved. It represents a balanced evaluation metric that considers the overall performance of a classification model. Furthermore, in the accompanying table comparing the accuracy, number of selected features and the time taken to test, we aim to provide a comprehensive analysis of the trade-offs between feature selection and the performance of the machine learning algorithms with taken account of times used.

Accuracy and Time Taken to Test values for Correlation Attribute Evaluation is shown in Table 4.5.

Table 4.5 Accuracy and Time Taken to Test values for Correlation Attribute Evaluation

	Evaluator with CorrelationAttributeEval					
	Before Feature Selection			*After Feature Selection*		
	Accuracy	*Time Taken to Test*	*Number of Features*	*Accuracy*	*Time Taken to Test*	*Number of Seleceted Features*
J48	*% 90*	*0.04 seconds*	*335*	*% 90*	*0.04 seconds*	*298*
AdaBoostM1	*% 85*	*0.07 seconds*	*335*	*% 90*	*0.04 seconds*	*298*
REPTree	*% 90*	*0.04 seconds*	*335*	*% 90*	*0.03 seconds*	*298*
MultilayerPerceptron	*% 85*	*0.07 seconds*	*335*	*% 81,67*	*0.07 seconds*	*298*
NaiveBayes	*% 68,33*	*0.15 seconds*	*335*	*% 68,33*	*0.06 seconds*	*298*

When evaluated in terms of Evaluator with Correlation Attribute Eval, J48 algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %90 and time taken to test is 0.04 seconds. AdaBoostM1 algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %90 and time taken to test is 0.04 seconds. REP Tree algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %90 and time taken to test is 0.03 seconds. Multilayer Perceptron algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %81 and time taken to test is 0.07 seconds. Naive Bayes algorithm accuracy is %68 and time taken to test is 0.15 seconds before feature selection. After feature selection accuracy is %68 and time taken to test is 0.06 seconds.

J48 and REP Tree algorithms have the highest accuracy value. While the accuracy value of the AdaBoostM1 algorithm was low before feature selection, it increased to 90% after feature selection. In this respect, the benefit of feature selection work is seen. Naive Bayes algorithm remains the lowest accuracy.

In terms of Time Taken to Test, the time taken by other algorithms is almost the same, except for the Naive Bayes algorithm. The Time Taken to Test value of the Naive Bayes algorithm is slightly higher. In this respect, it lags behind other algorithms.

Number of selected features of all algorithms are decreased to 298 with % 11.04 increases of performance.

Accuracy and Time Taken to Test values for Info Gain Attribute Evaluation is shown in Table 4.6.

Table 4.6 Accuracy and Time Taken to Test values for Info Gain Attribute Evaluation

	Evaluator with InfoGainAttributeEval					
	Before Feature Selection			*After Feature Selection*		
	Accuracy	*Time Taken to Test*	*Number of Features*	*Accuracy*	*Time Taken to Test*	*Number of Seleceted Features*
J48	*% 90*	*0,04 seconds*	*335*	*% 86,67*	*0,03 seconds*	*155*
AdaBoostM1	*% 85*	*0.07 seconds*	*335*	*% 85*	*0,03 seconds*	*155*
REPTree	*% 90*	*0.04 seconds*	*335*	*% 85*	*0,03 seconds*	*155*
MultilayerPerceptron	*% 85*	*0.07 seconds*	*335*	*% 86,67*	*0,06 seconds*	*155*
NaiveBayes	*% 68,33*	*0.15 seconds*	*335*	*% 68,33*	*0.05 seconds*	*155*

When evaluated in terms of Evaluator with Info Gain Attribute Eval, J48 algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %87 and time taken to test is 0.17 seconds. AdaBoostM1 algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %85 and time taken to test is 0.17 seconds. REP Tree algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %85 and time taken to test is 0.13 seconds. Multilayer Perceptron algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %87 and time taken to test is 0.20 seconds. Naive Bayes algorithm accuracy is %68 and time taken to test is 0.15 seconds before feature selection. After feature selection accuracy is %68 and time taken to test is 0.20 seconds.

Accuracy values of J48 and REP Tree algorithms decrease after feature selection. The accuracy value of the Multilayer Perceptron algorithm increases after the feature selection. There is no change in the accuracy values of AdaBoostM1 and Naive Bayes algorithms.

In terms of Time to Test, the Time to Test value of all algorithms is decreasing.

Number of selected features of all algorithms are decreased to 155 with % 53.73 increases of performance.

Accuracy and Time Taken to Test values for Chi Square Attribute Evaluation is shown in Table 4.7.

Table 4.7 Accuracy and Time Taken to Test values for Chi Square Attribute Evaluation

	Evaluator with ChiSquareAttributeEval					
	Before Feature Selection			*After Feature Selection*		
	Accuracy	*Time Taken to Test*	*Number of Features*	*Accuracy*	*Time Taken to Test*	*Number of Seleceted Features*
J48	*% 90*	*0,04 seconds*	*335*	*% 86,67*	*0,05 seconds*	*155*
AdaBoostM1	*% 85*	*0.07 seconds*	*335*	*% 85*	*0,05 seconds*	*155*
REPTree	*% 90*	*0.04 seconds*	*335*	*% 85*	*0,05 seconds*	*155*
MultilayerPerceptron	*% 85*	*0.07 seconds*	*335*	*% 86,67*	*0,03 seconds*	*155*
NaiveBayes	*% 68,33*	*0.15 seconds*	*335*	*% 68,33*	*0.06 seconds*	*155*

When evaluated in terms of Chi Square Attribute Eval, J48 algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %87 and time taken to test is 0.11 seconds. AdaBoostM1 algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %85 and time taken to test is 0.08 seconds. REP Tree algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %85 and time taken to test is 0.06 seconds. Multilayer Perceptron algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %87 and time taken to test is 0.08 seconds. Naive Bayes algorithm accuracy is %68 and time taken to test is 0.15 seconds before feature selection. After feature selection accuracy is %68 and time taken to test is 0.09 seconds.

Accuracy values of J48 and REP Tree algorithms decrease after feature selection. The accuracy value of the Multilayer Perceptron algorithm increases after feature selection.

There is no change in the accuracy values of AdaBoostM1 and Naive Bayes algorithms.

In terms of Time to Test, there is a slight increase in the times of J48 and REP Tree algorithms. Time to Test values are decreased in the other algorithms.

Number of selected features of all algorithms are decreased to 155 with % 53.73 increases of performance.

Accuracy and Time Taken to Test values for Genetic Algorithm is shown in Table 4.8.

Table 4.8 Accuracy and Time Taken to Test values for Genetic Algorithm

	Evaluator with Genetic Algorithm					
	Before Feature Selection			*After Feature Selection*		
	Accuracy	*Time Taken to Test*	*Number of Features*	*Accuracy*	*Time Taken to Test*	*Number of Seleceted Features*
J48	*% 90*	*0,04 seconds*	*335*	*% 93,33*	*0,01 seconds*	*203*
AdaBoostM1	*% 85*	*0.07 seconds*	*335*	*% 91,67*	*0,01 seconds*	*168*
REPTree	*% 90*	*0.04 seconds*	*335*	*% 93,33*	*0,014 seconds*	*200*
MultilayerPerceptron	*% 85*	*0.07 seconds*	*335*	*% 86,67*	*0,03 seconds*	*143*
NaiveBayes	*% 68,33*	*0.15 seconds*	*335*	*% 66,67*	*0,022 seconds*	*185*

When evaluated in terms of Genetic Algorithm, J48 algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %33,33 and time taken to test is 0.01 seconds. AdaBoostM1 algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %91,67 and time taken to test is 0.01 seconds. REP Tree algorithm accuracy is %90 and time taken to test is 0.04 seconds before feature selection. After feature selection accuracy is %93,33 and time taken to test is 0.014 seconds. Multilayer Perceptron algorithm accuracy is %85 and time taken to test is 0.07 seconds before feature selection. After feature selection accuracy is %86,67 and time taken to test is 0.03 seconds. Naive Bayes algorithm accuracy is %68 and time taken to test is 0.15 seconds before feature selection. After feature selection accuracy is %66,67 and time taken to test is 0.022 seconds.

The highest accuracy obtained from results after feature selection is %93,33. J48 and REP Tree algorithms have the same accuracy. Only the accuracy value of Naive Bayes decreases after feature selection. The accuracy value of the J48, REP Tree, AdaBoostM1 and Multilayer Perceptron algorithms increases after feature selection.

In terms of Time to Test, the Time to Test value of all algorithms is decreasing

4.3 Mc Nemar's Test Results

The Mc Nemar's test [91,92], a variant of chi square test, is a non-parametric statistical method utilized for analyzing matched pairs of data. According to Mc Nemar's test, two algorithms can result in four possible outcomes, as illustrated in Table 4.9. To quantify these differences, the Mc Nemar's test employs the z score as delineated in Eq. (4.9).

$$z = \frac{\left(\left|N_{sf} - N_{fs}\right| - 1\right)}{\sqrt{N_{sf} + N_{fs}}} \tag{4.9}$$

Table 4.9 Possible results of two algorithms

	Algorithm A failed	**Algorithm A succeeded**
Algorithm B failed	N_{ff}	N_{sf}
Algorithm B succeeded	N_{fs}	N_{ss}

In Tables 4.10, Table 4.11, Table 4.12 and Table 4.13, the arrowheads (←, ↑) indicate which classifier exhibited superior performance in the provided datasets. Z scores, presented alongside the arrowheads, serve as a measure of the statistical significance of the results [93].

The null hypothesis (H0) in this experimental design posits that different classifiers exhibit similar performance, while the alternative hypothesis (H1) asserts otherwise, indicating that at least one of the classifiers performs differently, as depicted in Eq. (4.10) and Eq. (4.11) [93].

$$H_0 : C_1 = C_2 = C_3 = C_4 = C_5 \tag{4.10}$$

$$H_1 : \exists C_i : C_i \neq C_j, (i,j) \in (1,2,3,4,5), i \neq j \quad \textbf{(4.11)}$$

Mc Nemar's Test Results for Comparison After Feature Selection with Correlation Attribute Eval is shown in Table 10.

Table 4.10 Mc Nemar's Test Results for Comparison After Feature Selection with Correlation Attribute Evaluation

	AdaBoostM1	REP Tree	Multilayer Perceptron	Naive Bayes
J48	0	0	←1.10	←2.75
AdaBoostM1		0	←1.03	←2.91
REP Tree			←1.03	←2.91
Multilayer Perceptron				←1.42

Based on the results of the Mc Nemar's test for the nominal datasets (Table 4.10), it is evident that J48 has yielded significantly superior outcomes compared to the Multilayer Perceptron and Naive Bayes classifiers (accepting hypothesis H1 with a confidence level exceeding 99.5%). Additionally, the AdaBoostM1 classifier has demonstrated notably better performance than the Multilayer Perceptron and Naive Bayes classifiers, while the REP Tree classifier has also exhibited significantly better results in comparison. Furthermore, the Multilayer Perceptron has shown significantly superior outcomes compared to the Naive Bayes classifier.

Mc Nemar's Test Results for Comparison After Feature Selection with Info Gain Attribute Evaluation is shown in Table 4.11.

Table 4.11 Mc Nemar's Test Results for Comparison After Feature Selection with Info Gain Attribute Evaluation

	AdaBoostM1	REP Tree	Multilayer Perceptron	Naive Bayes
J48	←0,70	0	0	←2,08
AdaBoostM1		0	↑0,70	←1,6
REP Tree			0	←1,83
Multilayer Perceptron				←2,08

After examining the results of the Mc Nemar's test for the nominal datasets (Table 4.11), it can be inferred that J48 has yielded significantly superior outcomes compared to the AdaBoostM1 and Naive Bayes classifiers (accepting hypothesis H1 with a

confidence level exceeding 99.5%). Additionally, the AdaBoostM1 classifier has demonstrated notably better performance than the Naive Bayes classifiers, while the Multilayer Perceptron classifier has exhibited significantly better results compared to the AdaBoostM1 classifiers. Furthermore, the REP Tree classifier has shown significantly superior outcomes compared to the Naive Bayes classifiers. Moreover, the Multilayer Perceptron has demonstrated significantly better results compared to the Naive Bayes classifier.

Mc Nemar's Test Results for Comparison After Feature Selection with Chi Square Attribute Eval is shown in Table 4.12.

Table 4.12 Mc Nemar's Test Results for Comparison After Feature Selection with Chi Square Attribute Evaluation

	AdaBoostM1	REP Tree	Multilayer Perceptron	Naive Bayes
J48	←0,7	0	0	←2,08
AdaBoostM1		0	↑0,7	←1,6
REP Tree			0	←1,83
Multilayer Perceptron				←2,08

Upon reviewing the Mc Nemar's test outcomes for the nominal datasets (Table 4.12), it can be inferred that J48 has demonstrated significantly superior performance compared to the AdaBoostM1 and Naive Bayes classifiers, with hypothesis H1 being accepted at a confidence level exceeding 99.5%. Furthermore, the AdaBoostM1 classifier has exhibited notably better results than the Naive Bayes classifiers, while the Multilayer Perceptron classifier has shown significantly better outcomes compared to the AdaBoostM1 classifiers. Additionally, the REP Tree classifier has demonstrated significantly superior results compared to the Naive Bayes classifiers. Moreover, the Multilayer Perceptron has exhibited significantly better performance compared to the Naive Bayes classifier.

Mc Nemar’s Test Results for Comparison After Feature Selection with Genetic Algorithm is shown in Table 4.13.

Table 4.13 Mc Nemar’s Test Results for Comparison After Feature Selection with Genetic Algorithm

	AdaBoostM1	REP Tree	Multilayer Perceptron	Naive Bayes
J48	←1,3	↑0,3	←1,7	←3,05
AdaBoostM1		↑1,4	←1,5	←2,89
REP Tree			←1,8	←3,17
Multilayer Perceptron				←2,12

Upon examining the results of the Mc Nemar's test for the nominal datasets (Table 4.13), it is apparent that J48 has yielded significantly superior outcomes compared to the AdaBoostM1, Multilayer Perceptron, and Naive Bayes classifiers, with hypothesis H1 being accepted at a confidence level exceeding 99.5%. Moreover, the REP Tree classifier has demonstrated significantly better results than J48, AdaBoostM1, Multilayer Perceptron, and Naive Bayes classifiers. Additionally, the AdaBoostM1 classifier has exhibited notably better performance than the Multilayer Perceptron and Naive Bayes classifiers, while the Multilayer Perceptron has shown significantly better outcomes compared to the Naive Bayes classifier.

CHAPTER 5

CONCLUSION

5.1 Summary

In this study, machine learning algorithms such as AdaBoostM1, REP Tree, Multilayer Perceptron, Naive Bayes, and J48 were evaluated using Mc Nemar's test. The test was used to determine the accuracy of these algorithms in correctly identifying the class of a given instance in a dataset. This non-parametric test provided a new approach to evaluate classification algorithms. The results indicated that REP Tree algorithm outperformed the other methods in both nominal and numerical data. On the other hand, Naive Bayes performed poorly in both types of data. Additionally, the results of Mc Nemar's test were found to align with the Kappa statistic and RMSE, confirming the reliability of the method.

When evaluating different algorithms for accuracy, time taken to test and number of selected features, it was found that J48 and REP Tree had the highest accuracy values. The AdaBoostM1 algorithm's accuracy increased after feature selection with Genetic algorithm and CorrelationAttributeEval, while Naive Bayes remained the lowest in Genetic algorithm. In terms of time taken to test, Genetic algorithm performance is better than other feature selection methods.

The number of features selected was one of our goals. Therefore, it is important for the performance of the system to keep the accuracy value high and to remove as much of the features that do not contribute to classification as possible. In the light of the results of this paper, it can be seen that the highest number of attributes selected for machine learning algorithms is 298 using CorrelationAttributeEval. The lowest number of attributes is 143 with using the genetic algorithm.

By using multi-objective genetic algorithm, the number of features was reduced and the accuracy values obtained as a result of classification were increased. Since the

multi-objective genetic algorithm also takes into account the interaction between features, it has significant differences compared to classical feature selection methods.

5.2 Future Work

The research presented in this thesis opens several avenues for further investigation. First, exploring the integration of additional machine learning algorithms, such as deep learning models, could provide deeper insights and potentially improve detection accuracy. In particular, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) could be employed to capture intricate patterns and temporal dependencies in malware behavior. Second, extending the current study to include real-time malware detection and response systems would be beneficial. These future directions will not only build upon the current findings but also address the evolving challenges in the field of malware detection and cybersecurity.

Secondly, the dataset could be expanded by incorporating additional malware samples and benign files to evaluate any potential alterations in the performance of classification models and extracted features

REFERENCES

[1] Cybersecurity Ventures. *Cybercrime to cost the world $9.5 trillion usd annually in 2024*. Retrieved February 1, 2024, from https://world.einnews.com/pr_news/664091567/report-cybercrime-to-cost-the-world-9-5-trillion-usd-annually-in-2024-according-to-cybersecurity-ventures. 2023.

[2] Aygör, D., Aktan, E. Kötücül yazılımların tespitinde imza temelli ve dinamik analiz yöntemlerinin zayıflıkları: Örnek olay çalışması. Gazi Universitesi Muhendislik Mimarlık Fakultesi Dergisi, *37*, (1) 305–316, 2021.

[3] Irshad, A., Maurya, R., Dutta, M. K., Burget, R., Uher, V. Feature optimization for run time analysis of malware in windows operating system using machine learning approach. In 42nd International Conference on Telecommunications and Signal Processing (TSP), 255–260, IEEE, (Budapest, 2019).

[4] Kale, G., Bostanci, E., Celebi, F. V. Differences between free open source and commercial sandboxes. In International Conference on Cyber Security and Computer Science (ICONS'18), 95-98, (Safranbolu, 2018).

[5] Kaur, R., Singh. M. Hybrid real-time zero-day malware analysis and reporting system. International Journal of Information Technology and Computer Science (IJITCS), *8,* (4) 63–73, 2016.

[6] Japkowicz, N., Shah, M. *Evaluating learning algorithms: A classification perspective*. Cambridge: Cambridge University Press. August 2011.

[7] Dietterich, T.G. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation, *10*, (7) 1895–1923, 1998.

[8] Schultz, M. G., Eskin, E., Zadok, F., Stolfo, S. J. Data mining methods for detection of new malicious executables. In Proceedings 2001 IEEE Symposium on Security and Privacy (S&P 2001), 38-49, IEEE, (California, 2001).

[9] Kolter, J. Z., Maloof, M.A. Learning to detect malicious executables in the wild. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04), 470-478, (Washington, 2004).

[10] Ye, Y., et al. An intelligent PE-malware detection system based on association mining. Journal in Computer Virology, *4*, (4) 323-334, 2008.

[11] Tian, R., Batten, L. M., Versteeg, S. C. Function length as a tool for malware classification. In 3rd International Conference on Malicious and Unwanted Software (MALWARE), 69-76, IEEE, (California, 2008).

[12] Moser, A., Kruegel, C., Kirda, E. Limits of static analysis for malware detection. In Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), 421-430, IEEE, (Florida, 2007).

[13] Bailey, M., et al. Automated classification and analysis of internet malware. In Recent Advances in Intrusion Detection: 10th International Symposium (RAID 2007), *10*, 178-197, Springer Berlin Heidelberg, (Gold Goast, Australia, 2007).

[14] Bayer, U., et al. Scalable, behavior-based malware clustering. In Proceedings of the Network and Distributed System Security Symposium (NDSS 2009), *9*, 8-11, (San Diego, California, 2009).

[15] Konrad, R., et al. Learning and classification of malware behavior. Detection of Intrusions and Malware, and Vulnerability Assessment, 108-125, Springer Berlin Heidelberg, (Berlin, 2008).

[16] Tian, R., et al. Differentiating malware from cleanware using behavioural analysis. In 2010 5th International Conference on Malicious and Unwanted Software Malicious and Unwanted Software (MALWARE), 23-30, IEEE, (New Jersey, 2010).

[17] Islam, R. et al. Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications, *36*, (2) 646-656, 2013.

[18] Anderson, B., Storlie, C., Lane, T. Improving malware classification: bridging the static/dynamic gap. In Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence (AISec'12), 3-14, (Raleigh, 2012).

[19] Mohaisen, A., Alrawi, O., Mohaisen, M. Amal: High-fidelity, behavior-based automated malware analysis and classification. Computers & Security, *52*, 251-266, 2015.

[20] Yusirwan, S., Prayudi, Y., Riadi, I. Implementation of malware analysis using static and dynamic analysis method. International Journal of Computer Applications, *117*, (6) 975-8887, 2015.

[21] Kirat, D., Vigna, G. MalGene: Automatic extraction of malware analysis evasion signature. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15), 769-780, (Denver, 2015).

[22] O'Kane, P., et al. SVM training phase reduction using dataset feature filtering for malware detection. IEEE Transactions on Information Forensics and Security, *8*, (3) 500-509, 2013.

[23] Rieck, K., et al. Automatic analysis of malware behavior using machine learning. Journal Of Computer Security, *19*, (4) 639-668, 2011.

[24] Dragos, G., et al. Malware detection using machine learning. In Proceedings of the International Multiconference on Computer Science and Information Technology, *4*, 735-741, (Mragowo, Poland, 2009).

[25] Singhal, P., Raul, N. Malware detection module using machine learning algorithms to assist in centralized security in enterprise networks. International Journal of Network Security & Its Applications (IJNSA), 4, (1) 61-67, 2012.

[26] Baldangombo, U., Jambaljav, N., Horng, S. J. A static malware detection system using data mining methods. International Journal of Artificial Intelligence & Applications (IJAIA), *4*, (4) 113-126, 2013.

[27] Mamoun, A., et al. Zero-day malware detection based on supervised learning algorithms of API call signatures. In Proceedings of the 9-th Australasian Data Mining Conference (AusDM 11), *121*, 171-182, (Ballarat, Australia, 2011).

[28] Shafiq, M. Z., et al. Pe-miner: Mining structural information to detect malicious executables in realtime. In Proceedings of the 12th International Symposium on Recent Advancesin Intrusion Detection (RAID '09), 121-141, (Saint Malo, France, 2009).

[29] Ye, Y., Wang, D., Li, T., and Ye, D. Imds: Intelligent malware detection system. In Proccedings of the 13th ACM International Conference on Knowlege Discovery and Data Mining (SIGKDD 2007), 1043-1047, 2007.

[30] Sharma, S., Rama Krishna, C., Sahay, S.K. Detection of advanced malware by machine learning techniques, In Soft Computing: Theories and Applications, Advances in Intelligent Systems and Computing, *742*, 333-342, Springer, (Singapore, 2019).

[31] Elovici, Y., et al. Applying machine learning techniques for detection of malicious code in network traffic. In Annual Conference on Artificial Intelligence (KI 2007), *30*, 44-50, Springer Berlin Heidelberg, (Osnabrück, Germany, 2007).

[32] Moskovitch, R., et al. Unknown malcode detection and the imbalance problem. Journal in Computer Virology, 5, (4) 295–308, 2009.

[33] Moskovitch, R., et al., Unknown malcode detection using OPCODE representation. In Intelligence and Security Informatics: First European Conference (EuroISI 2008), *5376*, 204-215, Springer Berlin Heidelberg, (Esbjerg, Denmark, 2008).

[34] Santos, I., Nieves, J., Bringas, P. G. Semi-supervised learning for unknown malware detection. In International Symposium on Distributed Computing and Artificial Intelligence, *91*, 415-422, Springer Berlin Heidelberg, 2011.

[35] Santos, I., et al. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences, *231*, 64–82, 2013.

[36] Shabtai, A., et al. Detecting unknown malicious code by applying classification techniques on OpCode patterns. Security Informatics, *1*, (1), 2012.

[37] Sharma, A., Sahay, S. K. An effective approach for classification of advanced malware with high accuracy. International Journal of Security and its Applications, *10*, (4) 249–266, 2016.

[38] Sahay, S.K., Sharma, A. Grouping the executables to detect malwares with high accuracy. Procedia Computer Science, 78, 667–674, 2016.

[39] Ahmadi, M., et al. Novel feature extraction, selection and fusion for effective malware family classification. In In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY'16), 183–194, (New Orleans, USA, 2016).

[40] Chumachenko, K. Machine learning methods for malware detection and classification. Bachelor's Thesis, University of Applied Sciences of Southeastern Finland. 2017.

[41] Korkmaz, Y. Automated detection and classification of malware used in targeted attacks via machine learning. Bachelor's Thesis, Graduate School of Engineering and Science of Bilkent University. 2015.

[42] Lee, S., et al. Malware response naming scheme for security control service. In 2020 International Conference on Information and Communication Technology Convergence (ICTC), 1549-1552, IEEE, (Jeju Island, South Korea, 2020).

[43] FitzGerald, N. A virus by any other name: Towards the revised caro naming convention. Proc. AVAR, 141–166, 2002.

[44] McGraw, G., Morrisett, G. Attacking malicious code: A report to the infosec research council. IEEE Software, *17*, (5) 33–41, 2000.

[45] Sembiring, Z. Stuxnet Threat analysis in SCADA (Supervisory Control And Data Acquisition) and PLC (Programmable Logic Controller) systems. Journal of Computer Science, Information Technology and Telecommunication Engineering, *1*, (2) 96-103, 2020.

[46] Aycock, J. *Computer viruses and malware*. *Advances in information security (1st ed.).* New York, NY, USA: Springer-Verlag. 2006.

[47] Rad, B. B., Masrom, M., Ibrahim, S. Camouflage in malware: from encryption to metamorphism. International Journal of Computer Science and Network Security, *12*, (8) 74-83, 2012.

[48] Walenstein, A. et al. The design space of metamorphic malware. In Proceedings of the 2nd International Conference on Information Warfare (ICIW 2007), 241-248, 2007.

[49] Al Daoud, E., Jebril, I., Zakaibeh, B. Computer virus strategies and detection methods. International Journal of Open Problems in Computer Science and Mathematics, *1*,(2) 122-129, 2008.

[50] Jagan, S., et al. A meta-classification model for optimized zbot malware prediction using learning algorithms. Mathematics, *11,* (13), 2023.

[51] Rawat, R. S., Diwakar, M., Verma, P. ZeroAccess botnet investigation and analysis. International Journal of Information Technology, *13*, (5) 2091-2099, 2021.

[52] Symantec Corparation. Trojan.Cleaman. Retrieved July 15, 2017, from http://www.symantec.com/security_response/writeup.jsp?docid=2012-050103-5441-99. 2015.

[53] Li, Y., et al. Backdoor attacks to deep learning models and countermeasures: A survey. IEEE Open Journal of the Computer Society, *4*, (99) 134-146, 2023.

[54] Singh, A., et al. Keylogger detection and prevention. In Journal of Physics: Conference Series, 3rd International Conference on Computational & Experimental Methods in Mechanical Engineering (ICCEMME), *2007*, (1):012005, (Uttar Pradesh, India, 2021).

[55] Ellis, D. R., et al. A behavioral approach to worm detection. In Proceedings of the 2004 ACM Workshop on Rapid Malcode, 43-5, 2004.

[56] Egele, M., et al. A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys (CSUR), *44*, (2):6 1-42, 2008.

[57] Griffin, K., et al. Automatic generation of string signatures for malware detection. In 12th International Symposium on Recent Advances in Intrusion Detection (RAID 2009), *12*, 101–120, Springer Berlin Heidenberg, (Saint-Malo, France, 2009).

[58] Thunga, S. P., Neelisetti, R. K. Identifying metamorphic virus using n-grams and hidden markov model. In 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2016–2022, IEEE, (Kochi, 2015).

[59] Idika, N., Mathur, A. P. A survey of malware detection techniques. Purdue University, *48*, (Indiana, 2007).

[60] Sikorski, M., Honig, A. *Practical malware analysis: The hands- on guide to dissecting malicious software (1st ed.).* San Francisco, California, USA: No Starch Press. 2012.

[61] Namanya, A. P., Disso, J. P., Awan, I. Evaluation of automated static analysis tools for malware detection in portable executable files. In 31st UK Performance Engineering Workshop (UKPEW 2015), *17*, 81-96, (University of Leeds, UK, 2015).

[62] Willems, C., Holz, T., Freiling, F. Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy, *5*, (2) 32-39, 2007.

[63] Zolkipli, M. F., Jantan, A. Secure Environment Platform for Host-based Dynamic Malware Analysis using DeepFreeze. In International Conference on Computer Application and Education Technology (ICCAET2011), 164-167, (Beijing, China, 2011).

[64] Ligh, M., et al. *Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code*. Indianapolis, Indiana: Wiley Publishing. 2010.

[65] Bulazel, A., Yener, B. A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium, 1-21, 2017.

[66] Eagle, C. *The IDA Pro Book: The unofficial guide to the world's most popular disassembler*. San Francisco, CA, USA: No Starch Press. 2011.

[67] Kevin, A. R., Barton, P. M. Hybrid analysis and control of malware. In Recent Advances in Intrusion Detection: 13th International Symposium (RAID 2010), *13*, 317-338, Springer Berlin Heidelberg, (Ottawa, Ontario, Canada, 2010).

[68] Dinaburg, A., et al. Ether: malware analysis via hardware virtualization extensions. In Proceedings of the 15th ACM conference on Computer and communications security (CCS 2008), 51-62, (Virginia, 2008).

[69] Andreopoulos, W. B. Malware detection with sequence-based machine learning and deep learning. Malware Analysis Using Artificial Intelligence and Deep Learning, 53-70, 2021.

[70] D'Elia, D. C., et al. Designing robust API monitoring solutions. IEEE Transactions on Dependable and Secure Computing, *20*, (1) 392-406, 2021.

[71] Damodaran, A., et al. A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, *13*, 1-12, 2017.

[72] Helmy, T., Keshta, I., Rashed, A. Performance evaluation of system resources utilization with sandboxing applications. In Information Science and Applications, 475-482, Springer Berlin Heidelberg, 2015.

[73] Mitchell, T. M. Does machine learning really work?. AI magazine, *18*, (3) 11-20, 1997.

[74] Guyon, I., Elisseeff, A. An introduction to feature extraction. In Feature extraction: foundations and applications, 207, 1-25, Springer Berlin Heidelberg, (Zurich, 2006).

[75] Witten, I. H., et al. *Data mining: Practical machine learning tools and techniques*. Cambridge, USA: Morgan Kaufmann. 2016.

[76] Quinlan, J. R. Improved use of continuous attributes in C4.5. Journal of Artificial Intelligence Research, *4*, 77-90, 1996.

[77] Han, J., Pei, J., Tong, H. *Data mining: concepts and techniques (3th ed.)*. USA: Morgan Kaufmann. 2022.

[78] Freund, Y., Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, *55*, (1) 119-139, 1997.

[79] Hastie, T., Tibshirani, R., Friedman, J. *The elements of statistical learning: data mining, inference, and prediction, 2*, 1-758. New York: Springer. 2009.

[80] Rokach, L. Ensemble-based classifiers. Artificial Intelligence Review, 33, (1-2) 1-39, 2010.

[81] Bishop, C. M., Nasrabadi, N. M. *Pattern recognition and machine learning, 4,* (4) 1-738. New York: Springer. 2006.

[82] Goodfellow, I., Bengio, Y., Courville, A. *Deep Learning*. Cambridge, Massachusetts, ABD: MIT Press. 2016.

[83] Nielsen, M. A. *Neural Networks and Deep Learning*. San Francisco, CA, USA: Determination Press. 2015.

[84] LeCun, Y., Bengio, Y., Hinton, G. Deep learning. Nature, *521*, (7553) 436-444, 2015.

[85] Moore, A. W., Schneider, J., Deng, K. Effient locally weighted polynomial regression predictions. In Proceedings Fourteenth International Conference on Machine Learning, 236-244, 1997.

[86] Siddiqui, M., Wang, M. C., Lee, J. Detecting trojans using datamining techniques. In Wireless Networks, Information Processing and Systems: International Multi Topic Conference (IMTIC 2008), 400-411, Springer Berlin Heidelberg, (Jamshoro, Pakistan, 2008).

[87] Cuckoo Foundation. 2015. *Cuckoo Sandbox Book*. Retrieved February 15, 2020, from https://cuckoo.readthedocs.io/en/latest/introduction/what/. 2020.

[88] Hung, P. V. An approach to fast malware classification with machine learning technique. Bachelor's Thesis, Faculty of Environment and Information Studies, Keio University. 2011.

[89] Reddy, D. K. S., Pujari, A. K. N-gram analysis for computer virus detection. Journal in Computer Virology, *2*, 231-239, 2006.

[90] Govardhan, P., Prasad, P. Survey on approaches to feature selection. International Journal of Advanced Engineering, Management and Science (IJAEMS), 265–269, 2017.

[91] McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. Psychometrika, *12*, (2) 153–157, 1947.

[92] Clark, A. F., Clark, C. Performance characterization in computer vision a tutorial. N (Eds.): FBook performance characterization in computer vision a tutorial/(Citeseer, 1999, edn.). 1999.

[93] Bostanci, B., Bostanci, E. An evaluation of classification algorithms using mc nemar's test. In Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012), *1*, 15-26, Springer, (India, 2013).