AN ENHANCED APPROACH FOR MALWARE DETECTION BY UTILIZING COMPUTER VISION AND MEMORY FORENSIC

BİLGİSAYARLI GÖRÜ VE BELLEK ANALİZİNDEN YARARLANILARAK ZARARLI YAZILIM TESPİTİ İÇİN GELİŞMİŞ BİR YAKLAŞIM

Erşan TAHILLIOĞLU

Asst. Prof. Dr. Murat AYDOS

Supervisor

Asst. Prof. Dr. İlker KARA

Co-Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

2020

To My Beloved Family…

ABSTRACT

AN ENHANCED APPROACH FOR MALWARE DETECTION BY UTILIZING COMPUTER VISION AND MEMORY FORENSIC

Erşan TAHILLIOĞLU

Master of Science, Computer Engineering Department

Supervisor: Asst. Prof. Dr. Murat AYDOS

Co- Supervisor: Asst. Prof. Dr. İlker KARA

June 2020, 79 pages

The number of advanced malware, which have been released, has increased substantially year after year with the increase in the use of information systems. Due to the ransomware programs, which we have frequently heard of in recent years, the effects of malware have drawn attention. The detection and prevention of malware before resulting in destructive effects have been a subject of research.

Malware detection methods are inadequate for detecting the sophistical malware to which methods such as code obfuscation and packing have been applied. Portable executable files (PE) leave some traces in the memory, and this allows for examining the behavior of the malware by security experts.

In this thesis, malware were researched in detail, the studies conducted to detect malware were examined and a new method was proposed to detect the sophisticated malware and potential zero-day attacks. The proposed method in this study, dumped the memory

patterns of the malicious process and then visualized them as RGB images in different dimensions. The visualized memory dump data enabled the use of computer vision methods. With the GIST and HOG global descriptors, which are frequently used in the field of computer vision, feature extraction was performed over the created images and vectors were obtained as a result of this process. These vectors were classified with XGBoost, J48, Support Vector Machine, SMO, Random Forest machine learning algorithms. In this study, the success of the visualization technique and different machine learning algorithms in detecting malware was analyzed. According to the results of the study, the highest accuracy was obtained when the visualization method with the fixed width of 4096 was used. The success rate of this method was found to be 96.39%. This study shows that the method proposed in the thesis can be effective in detecting malware.

Keywords: Malware Analysis, Malware Visualization, Memory Analysis, Machine Learning

ÖZET

BİLGİSAYARLI GÖRÜ VE BELLEK ANALİZİNDEN YARARLANILARAK ZARARLI YAZILIM TESPİTİ İÇİN GELİŞMİŞ BİR YAKLAŞIM

Erşan TAHILLIOĞLU

Yüksek Lisans, Bilgisayar Mühendisliği

Tez Danışmanı: Dr. Öğr. Üyesi Murat AYDOS

Eş Danışman: Dr. Öğr. Üyesi İlker KARA

Haziran 2020, 79 sayfa

Yayınlanan gelişmiş zararlı yazılım sayısı, bilgi sistemlerinin kullanımındaki artışla beraber yıldan yıla ciddi bir şekilde artmıştır. Son yıllarda adını sıklıkla duyduğumuz ransomware yazılımından dolayı zararlı yazılımların neden olabileceği etkiler konusu dikkatleri çekmiştir. Zararlı yazılımlar, yıkıcı etkilere neden olmadan önce tespit edilip, önlenebilmesi araştırma konusu olmuştur.

Geçmiş önerilen zararlı yazılım tespit yöntemleri, kod karmaşıklaştırma ve paketleme gibi yöntemler uygulanmış gelişmiş zararlı yazılımları tespit etmekte yetersiz kalmaktadır. Çalıştırılabilir dosyalar bellek üzerinde bir takım izler bırakmaktadır ve bu durum uzmanlar tarafından zararlı yazılımların davranışını incelemeye imkan sağlamaktadır.

Bu tez çalışmasında, zararlı yazılımlar detaylı bir şekilde araştırılmış, zararlı yazılımları tespit etmek için yapılan çalışmalar incelenmiş ve gelişmiş zararlı yazılımları ve potansiyel sıfır gün saldırısını tespit etmeye yarayan yeni bir yöntem önerilmiştir. Bu çalışmada önerilen yöntem ile zararlı yazılımların bellek üzerinde bıraktığı izler elde

edilip, ardından renkli olarak farklı boyutlarda görselleştirmiştir. Görselleştirilen bellek verileri, bilgisayarlı görü yöntemlerinden faydalanılmasını sağlamıştır. Bilgisayarlı görü alanında sıklıkla kullanılan GIST ve HOG küresel tanımlayıcılarıyla, oluşturulan resimler üzerinden özellik çıkarımı yapılmıştır ve bu işlem sonucunda vektörler elde edilmiştir. Bu vektörler XGBoost, J48, Destek Vektör Makinası, Sıralı Minimal Optimizasyon, Rassal Orman Makina öğrenmesi algoritmaları ile sınıflandırılmıştır. Bu çalışmada görselleştirme tekniğinin ve farklı makina öğrenmesi algoritmalarının zararlı yazılım tespitindeki başarısı analiz edilmiştir. Yapılan çalışmanın sonuçlarına göre, en yüksek başarı oranı 4096 sabit genişlik görselleştirme yöntemi kullanıldığında elde edilmiştir. Bu yöntemin başarı oranı %96,39 olarak bulunmuştur. Bu çalışmasında tezde önerilen yöntemin zararlı yazılımları tespit etmekte etkili olabileceğini göstermektedir.

Anahtar Kelimeler: Zararlı Yazılım Analizi, Zararlı Yazılım Görselleştirme, Bellek Analizi, Makine Öğrenmesi

ACKNOWLEDGEMENTS

I would like to thank Dear Asst. Prof. Dr. Murat AYDOS, Asst. Prof. Dr. İlker KARA, Dr. Ahmet Selman BOZKIR, who has always guided me with their valuable contribution and criticisms at every stage of my thesis study and encouraged me to study and gave me confidence with endless patience, the jury members and my family who has always been with me.

Erşan TAHILLIOĞLU

June 2020, ANKARA

CONTENTS

FIGURES

TABLES

ABBREVIATIONS

AET	Advanced Evasion Technique
API	Application Programming Interface
BOW	Bag of Words
CNN	Convolutional Neural Network
DNN	Deep Neural Network
HOG	Histogram of Oriented Gradients
HPC	Hardware Performance Counter
IDS	Intrusion Detection System
LSTM	Long Short-Term Memory
MALWARE	Malicious Software
ML	Machine Learning
MLP	Multilayer Perceptron
OPENCV	Open Source Computer Vision
OS	Operating System
PE	Portable Executable
RF	Random Forest
SMO	Sequential Minimal Optimization
SVM	Support Vector Machines
VM	Virtual Machine
XGB	XGBoost

1. INTRODUCTION

1.1. Overview

Malware, which have existed since the early days of information systems have become more sophisticated over the years, year after year. With the incredible increase in the use of information systems, attackers develop malware for reasons such as stealing the information of people, enabling unauthorized access, making the system unusable and mainly earning illegally.

The definitions of malware are very close to each other. Some of these definitions are as follows: (1) Programs which are inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or otherwise annoying or disrupting the victim [1] and (2) Malware is the common name of the malware such as viruses, trojans, and ransomware [2]. Malware can be divided into types and examined according to their behaviors and propagation styles; virus, worm, trojan horse, backdoor, spam, ransomware, rootkit, keylogger, and spyware can be given as examples for these categories. Each type of malware has its own propagation method. Some of them are propagated via links on an e-mail, and others are propagated through executable files and instant messaging applications. Attackers can exploit the malware they have developed more quickly by finding the vulnerability of the web browser and the operating system or through social engineering techniques. With the development of the internet, the information spreads much more quickly, and that situation causes to increase the speed of the spreading required information that needed to develop malware. For this reason, approximately 145 million new malwares were developed only in 2019, and the change in the number of developed malware is shown by year in Figure 1.1 [3]. This high increase in the number of malware leads to great economic and reputation losses in many sectors such as defense industry, healthcare, banking, and entertainment sector. In September 2007, Israeli jets bombed Syria and they deactivated the Syrian radars via the malware they had developed before bombing it [4]. This has put forward that the harms of the malware will not only be economic, it can also lead to loss of a war in case of any war.

Since the tools and methods that are used change every passing day, malware become more sophisticated and more difficult to detect. The Creeper virus, which is considered

the first developed malware, was developed by Bob Thomas in 1971. The Creeper virus targeted the computers on Arpanet and was spread to other computers over the network [5]. The aim of Bob Thomas was to show that a program can be spread to other computers over the network. Different names such as black hats, hackers and crackers are attributed to the malware developers. To overcome these problems, automatic malware analysis tools were needed, and the analyses of the security researchers played a crucial role in detecting the first developed malware. If the security researchers consider the portable executable (PE) file (the executable file format of Windows), which they have analyzed, malicious, they add the hash of this PE file into the database. Later, they compare the hash of any PE file to the hashes in the database before analyzing it. If the hash of the suspicious file is found in the database, it is directly considered malicious. This method is called the signature-based malware analysis in the literature and it is inadequate to detect the malware which is released today. This analysis method is inadequate because the attackers can change the hash of the PE file by making any changes in the PE file.

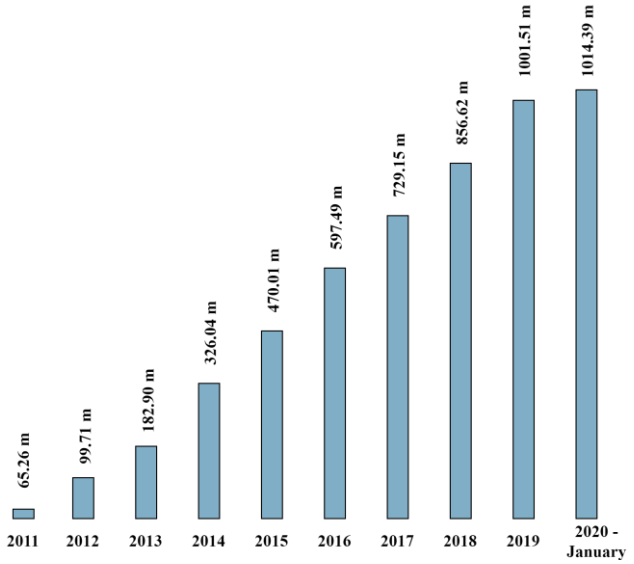


Figure 1.1. Number of malware that appeared between 2011 and 2020 [3]

The code obfuscation methods, which are frequently used for developing malware today, are some of the biggest difficulties in packers malware detection. When the malicious codes developed with these methods are made decompilers, it becomes difficult for people to understand them. Briefly, it makes the reverse engineering on the developed code difficult; therefore, malware cannot be detected when analyzed, and malware even

evade the analysis via the methods like the utilization of the hardware components by the malicious code injected into the kernel.

To overcome the deficiencies of the malware analysis methods, which are inadequate in the presence of methods such as code obfuscation and packing, the information in the memory is used. When the PE file is run, it leaves some traces in the memory, and these traces are live until the system is shut down. When a suspicious file is analyzed without being run, it becomes inadequate to detect the sophisticated malware which is resistant to analysis over the reverse engineering tools. For this reason, this trace left by the PE file in the memory gives valuable information for malware detection. This information in the memory is more robust against the methods such as obfuscation and packing, providing the opportunity of analysis independent of the operating system [6].

1.2. Motivation

The amount of shared information has also increased due to the decrease in the transmission time of the produced information. The wide popularization of the information systems attracts attackers, and attackers develop malware primarily to earn illegally. As the number of the released malware increases, information systems become seriously threatened, thus, the number of studies conducted by cybersecurity researchers for malware detection is also increasing. This is making it difficult to detect the released malware day by day. Within the scope of this thesis, the memory analysis method, which is more robust against the methods such as obfuscation that makes the malware detection difficult, and the malware detection method using visual descriptors were proposed. The effect of the RGB images in different dimensions created via the memory dump data on the classification success, the effect of the used global descriptors on the classification success, and the success of different machine learning algorithms in the malware detection classification were compared. The study proposes a solution for detecting the malware PE file and finding the class of the related malware.

1.3. Aim of the Thesis

With the development of the information systems, the number of malware released by attackers is increasing day by day, and this threatens the information systems in a serious way. It is seen that there are lots of studies conducted in the literature in order to protect the information systems that we frequently use in our daily lives from the threat of

malware, and most of these studies are inadequate for the detection of the advanced malware released today. In the study performed within the scope of this thesis, memory data were dumped and visualized in different dimensions. Afterward, the GIST and HOG global descriptors, which are among the visual descriptors, were utilized, and the success of the method was measured with different machine learning algorithms.

In this direction, the aims and objectives of the thesis study are as follows.

- Giving information about malware, the examination of their possible damages to the information systems.
- Giving information about the techniques that make the detection of malware difficult.
- Giving information about the malware detection studies in the literature, examining the methods and models included in these studies.
- Examining the advantages of the memory analysis method, one of the malware detection methods, compared to other methods
- Detecting the malware by using the memory dump data and classifying the malware
- Comparing the success of the image dimensions in malware detection by converting the memory dump data into images in different dimensions
- Measuring the performance of the GIST and HOG global descriptors which are used for feature extraction on the created images
- Observing the success of different machine learning algorithms in classification for malware detection
- Comparing the proposed model with the available solutions.

1.4. Thesis Structure

The thesis study basically consists of 8 main sections. In the first section, the definition, features and difficulties of malware detection, which is one of the main elements of the thesis, is addressed, and the significance of the memory-based malware detection method is introduced.

The second section is divided into three main headings. Under the first heading, the definition of malware is stated in detail and information is given about the types of malware, the available malware detection methods are introduced, and the difficulties of malware detection and the sandbox definition and types used in malware detection are touched upon. Under the second sub-heading, computer vision is introduced in the general terms. Under the third sub-heading, information is given about the definition of machine learning and the 3 different machine learning methods in the literature.

In the third section, previous studies in the literature are mentioned, and the systems developed in this field are stated as well. The studies that constituted the basis of our study are also discussed in this section.

In the fourth section, the GIST and HOG global descriptors are explained after information is given about the visual descriptors used in our study. Then, information is given about the process structure and the memory dump process is approached in detail; moreover, the popular machine learning algorithms of the literature are mentioned. Finally, in this section, information is given about the tools we used in our study.

In the fifth section, the architecture of our study is shown and the 4 steps that formed our study are addressed step by step in detail.

In the sixth section, after the features of the system and the dataset employed in our study is explained in detail, the evaluation criteria are discussed, and lastly, in this section, the results we obtained in our study are stated.

In the seventh section, the results we obtained in our study are interpreted, and the results are compared to one another.

In the eighth section, the last section, the results of the thesis study are evaluated, and future studies are stated.

2. BACKGROUND

In this section, information will be given about the malware which threatens the information systems, computer vision, and machine learning.

2.1. Malware

In this section, after the information is given about malware, the types of malware will be mentioned; after the difficulties of malware detection are discussed, the sandbox environment which is frequently used in malware detection studies will be touched upon.

2.1.1. Definition of Malware and Types

The use of computer systems has played a crucial role in sustaining our daily life. Computer systems have been developed for many years, and for this reason, the use of the computer systems in banking, defense, communication, education, entertainment sectors and dozens of sectors is becoming prevalent. Access to information has become easier with the increasing use of the Internet. As a result, malware and the tools that automatically create these malware are spread quite rapidly. This spread of computer systems also attracts the attention of many malicious users. Malicious users may lead to damages such as earning illegally, capturing the information of the victim, gaining unauthorized access on the system and making the system unusable. In the past, countries used a vast variety of tools when they wanted to fight, but today, it is seen that cyber attacks have joined these tools as a new one. Therefore, cybersecurity leads the study areas with increasing significance today, and more than 50 countries have officially released their cybersecurity strategy plans [7]. The USA has considered any cyber attack as a cause of war since 2011.

The malware that try to violate the computer systems' security policies - confidentiality, integrity, availability - which are also known as the CIA triad [8], cause large-scale companies to lose dozens of billions of dollars every year. According to the report published in 2019 by Accenture, which renders strategy, consultancy, digital, technology, and operation services, an economy of about 5.3 trillion $ has been directly or indirectly under the threat of cyber attack between 2019 and 2023 [9]. The number of malware is increasing every passing day. According to the report of the AV-Test Institute, more than 1 billion new malware appeared until 2019 [3], and the number of released malware has been increasing severely year by year. According to statistics, 70-80% of the malware are

transmitted to the systems from popular websites [10]. The purpose of information security is to protect individuals and companies against possible threats, and therefore, solutions such as anti-virus, firewall, and IDS against the threat of malware have been widespread.

The historical development of malware can be reviewed in 4 generations.

First-generation malware can be attributed to the period between 1987 and 1995; the malware of this generation are spread via floppy disks and files, and DOS viruses are observed in general.

Second-generation malware can be attributed to the period between 1995 and 2000; multimedia materials such as audios, pictures and videos were popular between these years and malware were spread into the Office programs such as Word and Excel using those materials via the use of macros.

Third-generation malware can be attributed to the period between 1999 and 2002. In this period, the popularization of the internet browser and e-mail was benefited from. Malware is spread into the target system via e-mails.

Fourth-generation malware is the period from 2001 until the present. This generation began with the Code Red worm. The malware that appeared in this generation caused severe damage to computer systems. Malicious types such as keyloggers emerged in this generation.

Malware is classified according to the system they will damage and the security policies they will violate. Peter Denning classified the malware in 1988 for the first time [11]. The malware classification by type is as follows:

- **Virus:** When viruses, which are considered among the oldest malware, enter the computer systems, they copy and integrate them into other applications. The ability to reproduce themselves is one of the main characteristics of viruses. They are triggered by the actions which require user interaction such as downloading files from e-mails, flash drives and unknown sources. Virus infection can be understood from slowing processes, deleted files, and programs functioning involuntarily [19].
- **Worms**: Worms in the computer memory need access to the Internet to spread and generally use internet protocols such as smtp, http, and ftp. It is difficult to

detect them since their sizes can be very small, and attackers can use the worms to create a backdoor.

- **Trojan horses:** Trojans, which are usually integrated into useful software, are also activated by the operation of the victim; trojan horses will look like a normal file, however, will damage the system when it is run. Trojan horses cannot reproduce themselves as worms and viruses do [20].
- **Spyware:** Spyware, which does not need to spread themselves after infecting the target system once, secretly collects the operations executed on the target system by the victim and information.
- **Backdoor:** It is a type of malware which helps with using the target system remotely, without the identification procedures. It emerged to easily re-access the system which had been accessed by the attackers before.
- **Spam:** Spams, which most users often encounter today, are a malicious type that keeps the users busy with advertisements and product promotion or for malicious purposes.
- **Keyloggers:** Keyloggers, which are among the most effective information acquisition systems, are used to obtain information by using keyboard gestures. The reason why banks or various platforms ask for password entry with the virtual keyboard is to take precautions against keyloggers.
- **Rootkit:** It is a type of malware which aims at corrupting the system by installing themselves as a core module of the operating system. Their detection by the user is very difficult because they can hide themselves for a very long time [18]. Rootkits, which delete system attack logs and can hide themselves from the process list, are the malware that enables the creation of a backdoor in the system.
- **Ransomware:** They are spread into the system via e-mails, trojan horses etc. After it is run, it encrypts the files in the system and makes them unusable. The purpose of making the system unusable is to request a ransom from the victim, and even if the victim pays the ransom, he is mostly unable to access his old data.

2.1.2. Difficulties Of Malware Detection

Attackers encrypt malicious code fragments to prevent the malware they have developed from being detected by information security applications, and these malware are separated into 4 classes [53]. Whereas the first examples of these malware are seen in encrypted malware, oligomorphic malware have been developed to find a solution for the deficiencies of the encryption method, polymorphic and then metamorphic malware, which are quite difficult for security applications such as antivirus to detect, have been developed in the following years.

Encrypted malware are divided into two sections; the first section has a fixed decrypter, and the second section of the code contains codes that damage the system, and the second section is decrypted by using the first section. In this type of malware, the decrypter is seen to be the same in every generation, and therefore, they can be detected easily; some anti-virus programs detect encrypted malware only by examining the patterns in the first section.

On the other hand, oligomorphic malware do not have a fixed decrypter, unlike encrypted malware, but instead, they use a decrypter key structure and selects a decrypter from this key set in each new generation. However, as the number of keys in the key set is limited, it is possible to guess the decrypter in the new generations.

Polymorphic malware have unlimited decoder keys. Thanks to their multi-layered and multi-password structure, they're protected against brute-force attacks. The encrypted functional section is decrypted with the decrypter and becomes active in the memory. In polymorphic malware, when the encrypted code fragment is decrypted, the same code block works in every generation, for this reason, it is possible to detect it with the memory-based signature detection method [54].

In metamorphic malware, 20% of the code is encrypted and the remaining 80% is comprised of code generation and conversion tools such as parser, sequencer, distributor, compressor and compiler. In every new generation, the code encryption is changed and features which do not change the functionality of the code but the structure of the code are added; for example, adding non-functioning code fragments into the code or changing the location of the code blocks. Due to this structure of metamorphic malware, signature-based malware detection methods are completely inadequate, however, they can be

detected via the memory analysis [54]. You can see the structure of metamorphic malware in Figure 2.1.

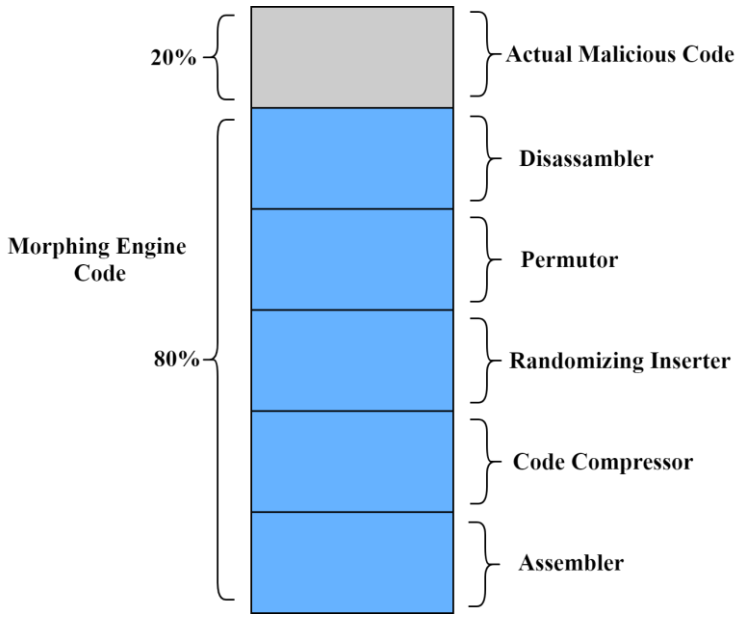


Figure 2.1. Metamorphic Structure

Attackers do not want the malware they have developed to be analyzed and, if analyzed, they do not want to be detected as malware as a result of the analysis. The methods used by malware authors for evading analysis and avoiding detection are stated below.

Kernel: Increasingly, malware authors are crafting attacks to inject malicious code into the operating system (OS) kernels, where it is essentially invisible to many security systems. With this method, which is frequently used by rootkit developers, the rootkit presence can be hidden via the malicious code injected into the OS kernel. While it may not be surprising that most signature-based detection technologies miss kernel-based malware, many behavior-based malware tracking tools, such as some sandboxes, cannot detect kernel-based malware using traditional hooking mechanisms.

Hardware: It is the method among those applied for evading analysis, which requires the highest skills and knowledge and makes use of the hardware components. Hardware trojans have drawn a lot of attention in the defense industry. In September 2007, Israeli jets bombed Syrian territory and the Syrian radars were remotely disabled via hardware trojans before this attack [4].

Detecting analysis environment: Environments such as sandbox and virtual machine are often used to analyze the malware behavior. In order to prevent malware analysis, methods such as sandbox detection are being developed by attackers. Using this method,

malware change their behavior as a legitimate PE file and hide malicious activities when malware are executed in an environment such as sandbox and virtual machine. Malware analysis can be evaded with this change in malware behavior. The developed malware can measure the CPU temperature, which cannot be measured in a VM environment, and decide whether the analysis environment is operated considering criteria such as the free space and total memory space of the hard disk, CPU core number and User interactions in the system.

Code injection: With the code injection technique, the malicious process can allocate the memory area of another process and inject the code into this area via WinAPI. Hence, it is possible to hide malicious activities by using another process [52]. To give a few examples for the code injection technique: (1) DLL injection - the malicious process allocates memory space in the memory area of the legitimate process and installs the DLL file precompiled with the VirtualAllocEx(), CreateRemoteThread(), SetWindowsHookEx() functions; thus, it works as if it is a part of the legitimate process. (2) Process Hollowing - This method is a little different from the DLL injection method. The malicious code that uses this method first runs a valid system process, then stops that process, and then replaces its codes with the original codes of that process, and finally, it is settled in the system by running the code it has stopped.

Fileless Malware: Even though this concept is perceived as a new term, it is known by the security industry for years. At the start of this year, more than 140 Fileless Malware attacks have been made, including banks, telecoms, and state institutions. As it is described, Fileless Malware is a type of malware that doesn't use any files in the process. Fileless malware shows that no files have been left in the local hard disk of the computers and frees security and judicial tools. It exists in the memory of your computer system and no antivirus programs directly examine the memory; for this reason, it is one of the easiest ways for attackers to go into your computer and steal all your data.

Today, modern malware use methods like encryption, AET, packers, and obfuscation for avoiding analysis.

Encryption: Most malware have a self-encryption mechanism to make their detection by security tools difficult. The encrypter and decrypter exist in the malware; the decryption mechanism is activated when the software is run and enables the decryption of the payload. A new version is formed by changing the encryption with the inclusion of the

same harmful codes in the version where Payload is opened, therefore, it is quite difficult to detect them.

Advanced Evasion Technique (AET): The AET, which started to be known in the cybersecurity world in 2010, is a new evasion method composed of the combination of approximately 200 evasion methods known. Technically, millions of evasion methods can be created via the AET. In this respect, it challenges the security solutions like IDS and firewall.

Packers: Binary packer is used to decrease the size of the original binary data, to complicate reverse engineering, and to encrypt the original data before running the PE file. Attackers use binary packers to make the analysis of their software difficult. In the general structure of the packers, uncompressed code and compressed payload are compressed together. Malware authors use packers in different complexities. Packing malware does not turn into any OS call (call-back, system call, etc.) during the extraction. You can see the original PE file layout and then the packed PE file layout that emerges after the process of the packer and also the unpacked version of this packed file in memory in Figure 2.3.

Obfuscation: They aim at making their detection difficult for security tools by subjecting a code that is readable by people to a conversion and making it unreadable for other people. Briefly, the readability of the original code following decompilation is considerably declined. Due to its structure, obfuscation decreases the success of the static analysis method substantially [17]. The de-obfuscation of these codes has been a subject of research for years since they rely on methods such as code encryption and code protection. Obfuscation methods are often used in the development of polymorphic and metamorphic malware. Figure 2.2 gives information about the general structure of obfuscation, and the source code has turned into a different structure from the original source code following the obfuscation process.

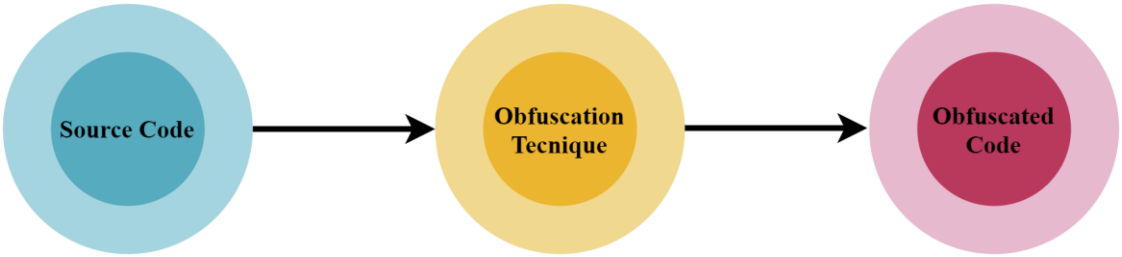


Figure 2.2. Overview Of Obfuscation

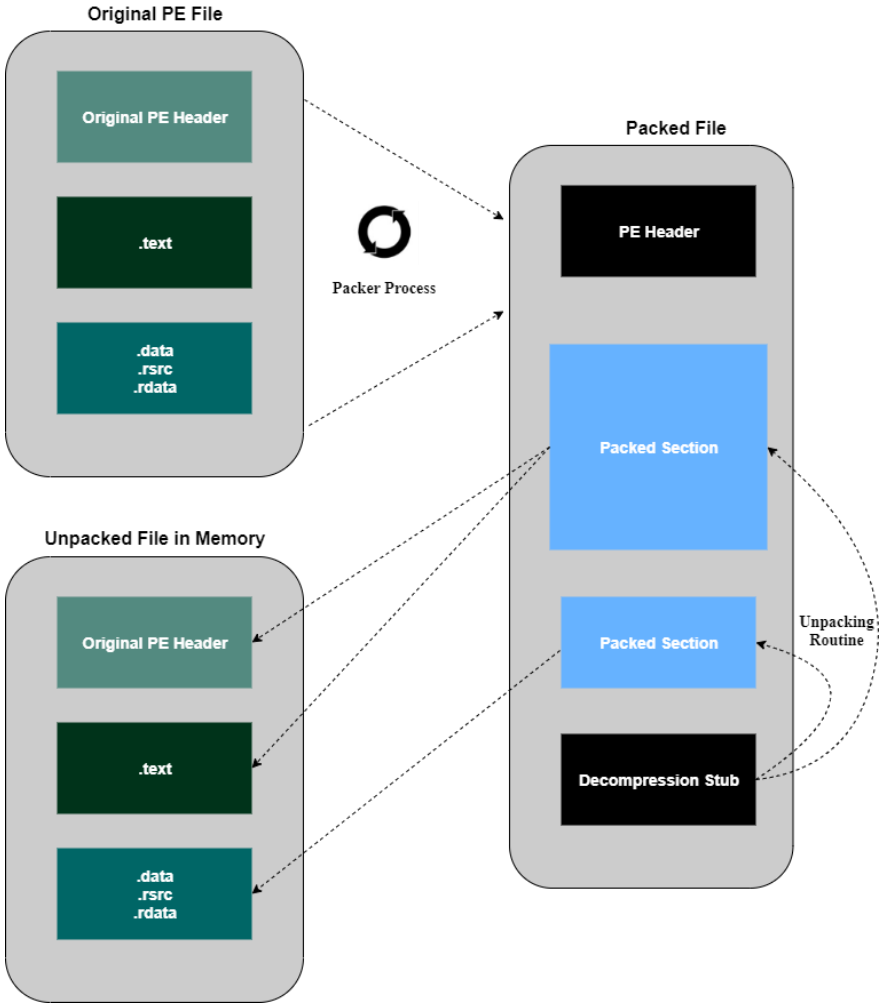


Figure 2.3. Packing process

Some of the obfuscation methods are listed below:

Register reassignment: It refers to the replacement of the registers used in the assembly code by another register without changing the behavior of the program. As can be seen in Table 2.1, the EAX, EBX and EDX registers in the original code fragment were

respectively assigned to the EBX, EDX, EAX registers, and the obfuscated code fragment in Table 2.1 was obtained.

Table 2.1. Register Reassignment

Original Code	**Obfuscated Code**
.model small	.model small
.code	.code
Mov ESI, EAX	Mov ESI, EBX
Mov AL, BYTE PTR DS:[EAX]	Mov BL, BYTE PTR DS:[EBX]
Test AL, AL	Test BL, BL
Je SHORT Test.00401236	Je SHORT Test.00401236
Push EBX	Push EDX
Pop DWORD PTR DS:[40F868]	Pop DWORD PTR DS:[40F868]
Rcr EBX, CL	Rcr EDX, CL
Bswap EBX	Bswap EDX
Push Test.00401238	Push Test.00401238
Pop EBX	Pop EDX
Mov DWORD PTR DS:[EBX], EAX	Mov DWORD PTR DS:[EDX], EBX
Inc EBX	Inc EDX
Bsr EAX, EDX	Bsr EBX, EAX
End	End

Dead Code Insertion: There are unnecessary instructions as the principle of the operation of the computer architecture (NOP). Thanks to these instructions, registers can be synchronized, but attackers can change the byte sequences through these instructions. You can see the example of Dead Code Insertion in Table 2.2 and the obfuscated code fragment was obtained by adding NOP instructions to the original code fragment, which does not change the behavior of the program.

Table 2.2. Dead Code Insertion

Original Code	**Obfuscated Code**
.model small .code Mov ESI, EAX Mov AL, BYTE PTR DS:[EAX] Test AL, AL Je SHORT Test.00401236 Push EBX Pop DWORD PTR DS:[40F868] Rcr EBX, CL Bswap EBX Push Test.00401238 Pop EBX Mov DWORD PTR DS:[EBX], EAX Inc EBX Bsr EAX, EDX End	.model small .code Mov ESI, EAX Mov AL, BYTE PTR DS:[EAX] Test AL, AL Je SHORT Test.00401236 Push EBX Pop DWORD PTR DS:[40F868] Nop Rcr EBX, CL Bswap EBX Push Test.00401238 Pop EBX Mov DWORD PTR DS:[EBX], EAX Nop Inc EBX Bsr EAX, EDX End

Code Integration: This method was first found by Zmist, and it is considered as an advanced code obfuscation method. In the execution of this method, legitimate software is divided into small manageable parts, and the malicious code is added to these parts after the code is rearranged [56].

Subroutine Reordering: This method is based on the displacement of the subroutes responsible for the operation of a certain task. If n different subroutes are defined, n! changes can be made in software.

2.1.3. Malware Detection Techniques

Researchers utilize 3 different methods for malware detection [55]. You can see these 3 methods in Figure 2.4.

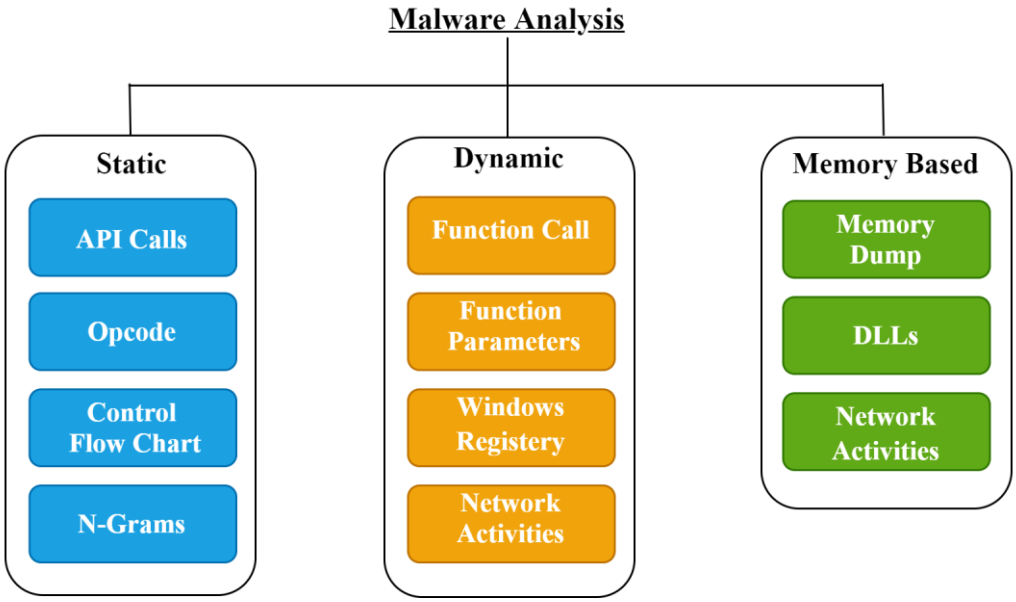


Figure 2.4. Malware Analysis Methods

Static analysis, which is among these methods, is based on the analysis of the software without running it. Attention is paid to Windows API calls, string signatures, control flow graphs, opcode and byte sequences in the analysis of the software on the hard disk. Tools such as IDA Pro have been developed to analyze the malware prepared by the attackers. The .dll files on the Windows operating system can be used over Windows API; there are various functions and data structures in these .dll files. With WinAPI calls, access to network services, graphical interface, basic and advanced services can be enabled. The behavior of the program can be evaluated through the examination of these calls, and they can be an important feature in malware detection. The strings, which are included in the software, can give information about the intent of the attacker. Control Flow Graphs inform us about the flow of the software. The opcodes that constitute the

first part of the machine code give information about which instruction the CPU will run. As the PE file is analyzed without being run in the static analysis, it is inexpensive. On the other hand, operations such as unpacking and decompression need to be performed for feature extraction from the PE file.

Initially, signature-based malware detection was suggested, but these methods can be evaded very easily [28]. In the signature-based malware detection method, primarily, the malware is examined by an expert team; when it is discovered that the software is malware, the signature of this software is found and saved in the database. When software with this signature is seen at a later time, it is directly evaluated and reported as malware. Therefore, the false positive (FP) rate is very low in signature-based malware detection [27]. The signature-based malware detection method is considered as a static analysis method as it can analyze the software by considering the byte sequence without running the software [51]. In Figure 2.5, you can find the general procedure of the signature-based malware detection method.

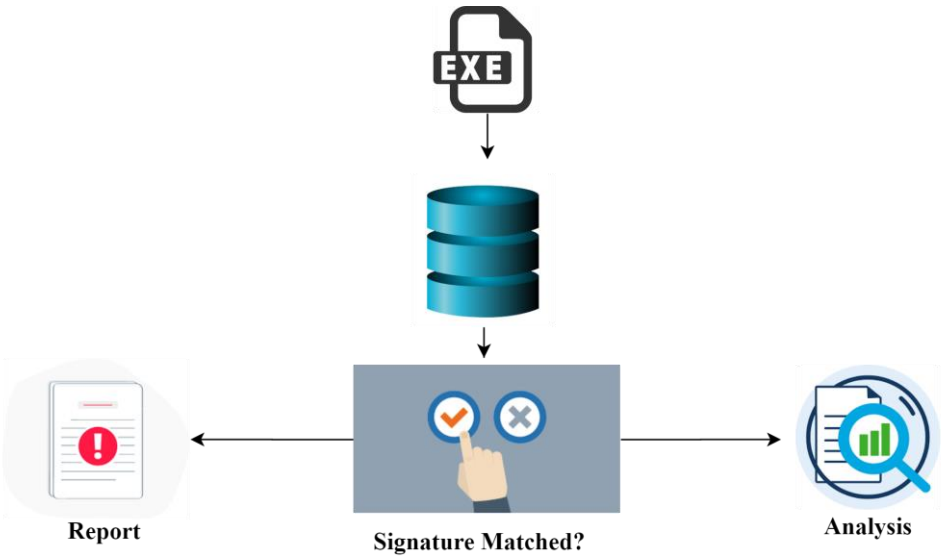


Figure 2.5. Signature-based malware detection general flow

Since the byte sequence of the PE file is very important in this method, any change in the byte sequence may result in dysfunction of this method. Attackers can change the byte sequence in different ways, as they know that the byte sequence plays an important role in this method. For example, they make use of the obfuscation methods explained in Section 2.1.2.

Many tools are used in the static analysis. These tools are briefly explained below.

BinText: It can extract texts in any kind of file or find plain text and unicode. It can show both the file offset and the memory offset of each string found. It can also extract the registry keys.

Resource Hacker: It is often used in the static malware analysis since it is a free application and it can extract resources from Windows binary files.

UPX: Developed with C++ Language, UPX can decompress in a number of executables formats. As it is extremely quick and free, it is frequently used in the static malware analysis.

IDA Pro: IDA Pro, which is a disassembler, can convert executables files to assembly instruction. IDA Pro, a quite popular tool, supports x86, x64, PowerPC, PowerPC-64, ARM64, etc. processor families.

Dynamic analysis, which is one of the malware detection methods and **dynamic** analysis detection methods, was included in the literature as behavioral analysis as well. In dynamic analysis, it is essential to run the software and perform its behavioral analysis, and software is executed in controlled environments in this method, for example, sandbox, virtual machine, etc. Dynamic analysis gets better results in malware detection compared to static analysis; however, it consumes too many resources and malware detection takes a long time. Besides, dynamic analysis can only be performed when the required conditions are fulfilled, otherwise, the code may not be executed, and unanalyzed [52]. Dynamic analysis can be used even in the obfuscated malware detection. In dynamic analysis, dozens of features such as function calls and function parameters, ethernet activities, Windows registry records and changes in the file system are taken into consideration. Some of the tools used for dynamic analysis are stated below.

Process Explorer: It is one of the sysinternals tools developed by Microsoft. Thanks to Process Explorer, the .dll used by the processes as well as the files opened/used by the process can be seen.

Process Monitor: By means of the Process Monitor tool, which is one of the sysinternals tools developed by Microsoft, file system, registry, and process/thread activities can be monitored in real-time. It has many advanced features besides the version with the combination of the features of the FileMon and RegMon tools.

TCPView: With TCPView, which is one of the sysinternals tools developed by Microsoft, all the TCP and UDP port connections used and opened by the processes can be displayed.

TDIMon: With TDIMon, which is a program that can monitor TCP and UDP activities in the system, network packets can be captured in real-time to be analyzed later.

Wireshark: Thanks to Wireshark, which is a program frequently used by network professionals and researchers worldwide, network traffic can be easily monitored through a GUI. Ethernet packets can be captured to be analyzed later. It is also completely free and open source.

Memory analysis method has increasingly been popular in recent years. When the data on the hard disk are complicated via the obfuscation method, the static analysis method becomes inadequate and can be easily evaded. However, since the instructions in the memory will work directly on the CPU, they exist in the memory without any encryption. For this reason, analysis over these data increases the performance in malware detection activities. Memory analysis, contrary to static analysis, does not require any unpacking or decompilation procedure before the analysis. Additionally, memory analysis can be done independently from the operating system whereas static and dynamic analysis are performed depending on the operating system [6]. Special hardware have been developed for memory acquisition and analysis to be utilized in this valuable information in the memory [52]. Until the system is shut down, there is valuable information in the memory such as DLLs, active process list, network activities, register keys, and operation system. Besides, procedures such as unpacking and decompression, which are frequently used in the static analysis method, are not required.

There are various tools to dump an application running in the memory. DumpIt, FastDump and WinDbg can be given as examples for these tools; moreover, and a desired process can be dumped over the task manager in the Windows operating system. Thanks to the Vmmap tool, the PID (process ID) of a running process as well as the byte sequences on its virtual memory can be seen.

2.1.4. Sandbox

There is a need for running software in order to review the behavior of malicious file, network activities and function calls for the purpose of malware analysis, however, it is not known what the possible damages of the malware in the system will be as soon as the malware is executed. Therefore, suspicious PE files are run in a test environment such as a virtual machine, sandbox, and emulator, and this environment makes it difficult or impossible to damage the user data. The analysis on virtual machine, sandbox, emulator, etc are considered as live analysis [52]. Sandbox, which is one of these test environments, has many different types.

As can be seen in Figure 2.6, the sandbox environment analyzes the suspicious file by being isolated from the secure space.

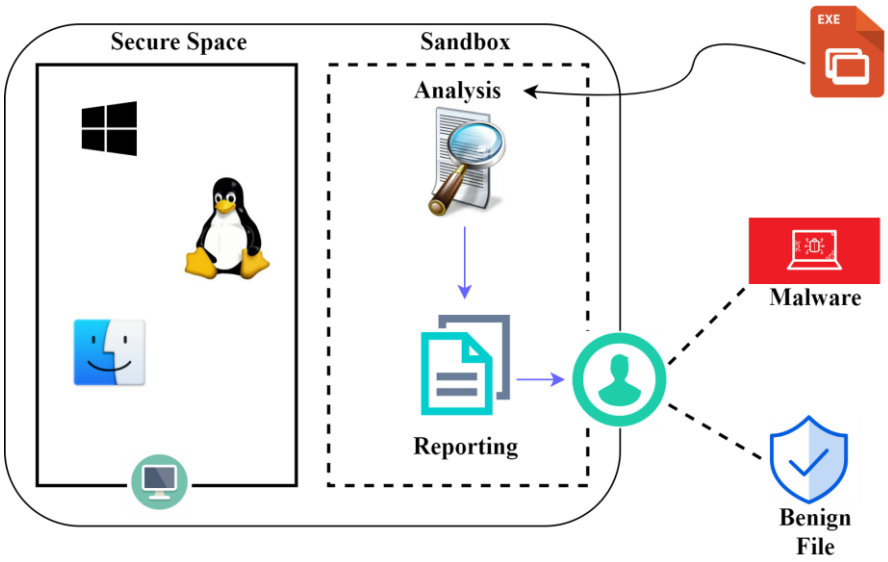


Figure 2.6. Sandbox overview

Anubis: Developed by International Secure Lab, Anubis is used for the analysis of unknown files and URLs [58]. Android APK files can be examined with the executable files specific to the Windows operating system. Windows API calls, function calls and machine calls can be traced [21].

CWSandbox: Thanks to CWSandbox, the files created or modified by the malware, changes made in windows registry, .dll downloads by the malware and network activities can be reviewed [57]. It is allowed to create reports in more than one format.

Norman Sandbox: Norman Sandbox provides the user with a controlled malware analysis environment, and therefore, no damage occurs on the host system during the

tests. According to the analysis result, API logs, changes in the file system, network service details, process and windows information can be monitored. The core component of Norman Sandbox is compatible with Windows functions such as Winsock, Kernel, and MPR, and also supports the Ethernet protocols such as HTTP, FTP, SMTP, DNS, IRC and P2P [58].

Cuckoo Sandbox: Cuckoo Sandbox, which is one of 100% open-source malware analysis tools can analyze the PE files, office documents, pdf files and e-mails in the Windows, Linux, macOS, and Android virtualized environments. Thanks to Cuckoo Sandbox, API calls and network communication encrypted with SSL/TLS can be examined. Besides, it provides an opportunity for an advanced memory analysis.

Windows Sandbox: It establishes a virtual Windows 10 environment via the Windows Sandbox Virtualization Technology that is released for Windows operating systems with the Windows 10 May update (1903). Suspicious files are run in this virtual environment. Windows Sandbox behaves like a recently installed Windows whenever it is run; therefore, all the files are deleted when sandbox is deactivated. Moreover, it uses features such as virtual GPU, smart memory management and integrated kernel scheduler.

2.2. Computer Vision Background

Computer vision is a discipline that reveals significant and useful information by detecting an image in a computer environment. This field is characterized by the methods developed similarly to the biological detection and differentiation of an object. It is necessary to obtain significant information in the methods established by the modeling and interpretation of an image. It is important to explain the concepts related to this field for the comprehensibility of the subject. Therefore, brief definitions of some terms are included in this section. As in other fields, the first procedure to apply for an image is feature extraction in this field, too. For this reason, it is necessary to determine the image characteristics, which are defined as features [59]. The identification process, which includes algorithms that will allow for displaying an image in a numeric order according to its pixel information, is called feature extraction.

Feature extraction is performed by using image descriptors in the field of computer vision. These can be classified in different forms in the literature; however, some concepts are used for classification. The first classification is color-based and enables the extraction of the image with appropriate features. The aim of the classification based on correlogram,

histogram and color moment is to assist the color-based classification [60]. Whereas correlogram is a concept frequently used in image matching, color moment is expressed as a concept obtained from mean and standard deviation calculations. The color histogram is a graph that shows the color frequency in the image. In the texture-based classification, which is the second classification, image descriptors are used. It enables finding the relationships in the image according to the similar patterns in the image.

The numeric display of the images is stated as feature vector. Other concepts that need to be explained for the studies in the field of computer vision give information about the image. The first concept, edge, is a set of linear points defined as one-dimensional to differentiate two images. It is formed by combining points with the highest values after the gradient values in the image are calculated. The second concept is to find the interest points, which is the highest gradient value [59]. There are certain algorithms which determine the interest points, also called keypoints, to find the vertex and edges in the image. The interest points are found based on methods such as maxima region and blob detection.

After the points depicting the images are found, the frequency of the points should be specified. Therefore, the histogram is used; it is a graph that presents the color distribution in the image. Since there is a Y value corresponding to each X value in this graph, it is similar to a column graph.

2.3. Machine Learning Background

Machine learning, which has been quite popular, and we have heard of in recent years, was actually a sub-branch in the field of artificial intelligence in computer science in 1959. Usage areas of machine learning are very broad, and we frequently use its applications in our daily life. It has applications such as face detection, document classifying, virtual assistant technologies, spam detection, and malware detection. In machine learning, sample data are introduced to the model, and then, a model is created via mathematical and statistical methods. Afterward, some inferences can be made by using this model.

Today, as a result of the incredible development and popularization of computer systems, it is becoming more and more difficult to control the systems. Data acquisition, evaluation of the acquired data by humans and finding the results take a very long time. Against this problem, results will be obtained with minimum human effort thanks to machine learning.

Machine learning is also used frequently by companies known worldwide. For instance, news source of Facebook is personalized specifically to every member.

There are different methods of machine learning according to the dataset to be used and the study to be conducted. As can be seen in Figure 2.7, these are supervised, unsupervised and reinforcement, which will be explained in detail under Sections 2.3.1, 2.3.2 and 2.3.3.

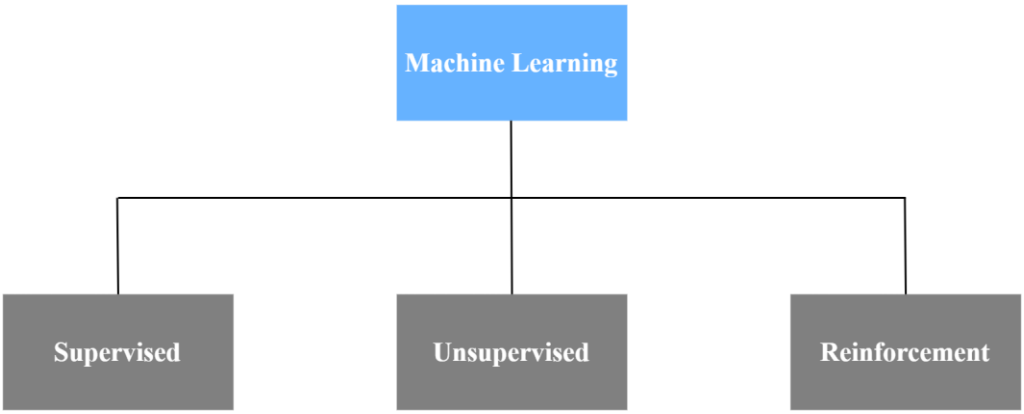
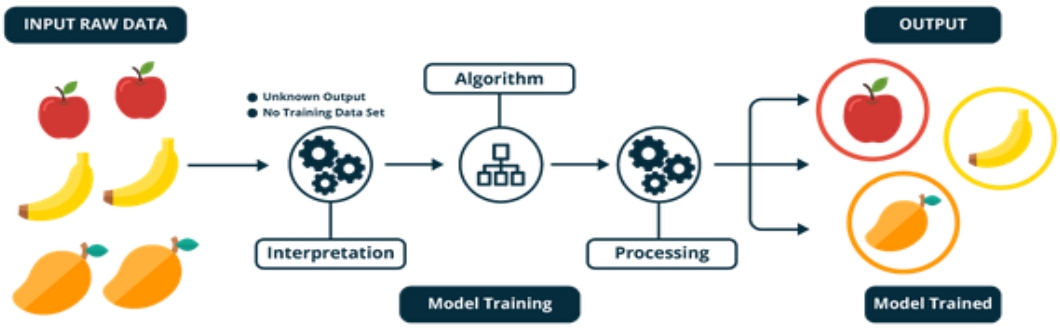


Figure 2.7. Types of Machine Learning

2.3.1. Supervised learning

In this learning method, the used dataset is labeled, and the model is trained over these data. There are certain inputs in the used dataset, and there are outputs for these inputs. In parallel with this information, the model learns with mathematical and statistical methods and is expected to make predictions when new inputs are received.

The most used supervised learning algorithms for malware classification are Naive Bayes, SVM, Decision Tree (DT), J48, Hidden Markov Models and Random Forest (RF) [16]. Besides, supervised learning is also used frequently in dimension reduction and feature extraction. Supervised learning is reviewed in 2 basic groups. These are classification and regression. The main difference between classification and regression is that a numeric value is obtained as a result of regression, however, category assignment is carried out as a result of classification. As can be seen in Figure 2.8, the labeled data are given to the algorithm and learning is done, and then, classification is performed according to the model created.



Fiqure 2.8. Supervised Learning [16]

2.3.2. Unsupervised learning

Contrary to supervised learning, unsupervised learning is machine learning technique used for predicting an unknown structure over unlabeled data. In this learning process, input data are given to the system, but desired outputs are not given. Unsupervised method works with the logic of clustering the data, which have close values to each other, within themselves by working on the untrained data and finding connections between the data. Unsupervised learning is rather used to solve problems such as clustering, dimension reduction, and association rule mining.

2.3.3. Reinforcement learning

This learning method is a little different when compared to the supervised and unsupervised methods. This learning method carries out its procedure in line with feedback. The important thing in reinforcement learning is to plan the sequence of correct movements in order to reach the target. During the learning period, the algorithm aims at reaching the target and receives a positive signal when it reaches the target; it receives a negative signal when it cannot reach the target. Whereas the algorithm tries to repeat the positive signals, it does not repeat the negative signals and carries on the learning process with its experience in positive signals. There is no condition like stopping learning, the algorithm keeps learning.

3. RELATED WORK

When we have a look at the studies on malware detection, it is seen that the static malware analysis method gained significance, and the signature-based approach was used in the early static malware analysis [28] but this approach cannot detect malware variants. In the proposed malware detection studies, machine learning and deep learning gained importance, and in a study, Sewak et.al [16] compared RF, which is a machine learning method for malware detection, to the Deep Neural Network (DNN) algorithms. Unique opcodes of every sample in the dataset used in the study on the Linux operating system were found and unique opcodes were mapped to an integer. This research result has shown that RF outperforms the DNN architecture. The used dataset has maybe affected the result of DNN classification, and as a result, deep learning approaches always provide better accuracy when compared to the machine learning approaches. One of the most important differences between machine learning and deep learning is that no required feature extraction in the deep learning method. In a study, Li Chen [29] vectorized the binary files via the static malware analysis method, and afterward, converted the vectors into images. In his study, it was expressed that the training period of Deep Neural Network (DNN) algorithm was shortened thanks to the Transfer Learning Method, which is commonly used in the field of computer vision.

The study of Dai et al. [24] was based on the memory dump since the memory dump data would provide sufficient information for malware detection. Malware was executed on Cuckoo sandbox, and the malware run in the sandbox environment obtained the memory dump data by using the sysinternal tool called Procdump, which was developed by Microsoft. The obtained memory dump data were converted to grayscale images; in this conversion process, the width was set to 2048 or 4096 according to the size of the dump data, and the height varies according to the file size. The images extracted the texture features through HOG. HOG feature vectors with fixed lengths were classified with Multi-Layer Perception (MLP), RF and k-NN classifier ($k = 3$, $k = 5$) and the highest accuracy value, 95.2%, was obtained with MLP. A number of researches conducted for malware analysis in recent years make use of hardware features. Hardware Performance Counter (HPC) can be given as an example for hardware features. The results of this research were compared to the study conducted with HPC [30], and it was seen that using the memory dump data gave a higher result; it has been concluded that the HPC method does not describe the malware behavior completely.

In their study, Halim, Abdullah, and Ariffin [31] used the Drebin Dataset for training and validation, and the dataset contains the features of 129.013 malware from a total of 179 families. These are 8 features: hardware component, requested permissions, application components, filtered intents, restricted API calls, used permissions, suspicious API calls, and network addresses. BOW was used for feature extraction, however, the BOW approach leads to information loss and coarse indexing since it deletes the spatial and sequence information in malware patterns. To overcome this problem, it was suggested to combine Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN). In the study, 5 different Neural Network models such as Multilayer Perception (MLP), CNN, LSTM, CNN-LSTM, and LSTM-CNN were used, and according to the results, LSTM-CNN outperformed with 98.53% accuracy whereas CNN obtained the lowest accuracy value with 87.91%. The outperformance of the LSTM-CNN combined neural network model showed that the hybrid schema neural networks were more accurate.

In their study [32], Dai et.al proposed a dynamic malware analysis method on a multi-feature basis, because the available dynamic malware detection methods are not sufficient against evasion attacks. Trough Cuckoo Sandbox, API call sequence, memory dump data and hardware performance counter (HPC) features were extracted. The word2vec method was used to convert the API sequence into a feature vector. Memory dump data were converted into grayscale images in the png file format, and bicubic interpolation was employed for grayscale images since the memory dump data can be in different sizes, thus, grayscale images with consistent sizes were created. The last feature used in the study was the hardware low-level and the Model-Specific Registers (MSR) of the CPU were examined. As a result of the study, the accuracy was achieved by 96.9%. According to the results obtained, it was observed that the multi-feature method was effective enough to combat the evasion of malware.

The first malware detection methods were completely software-based and primitive, and hardware-assisted methods have been proposed for malware detection in recent researches. In their study, Xu et al. [33] proposed a hardware-assisted malware detection method. Their study is based on the fact that the malware leaves fingerprints in the memory when it is run. In the study, the patterns of the system calls in the memory were used for kernel rootkit detection, and a detection rate of 99% was achieved.

The popularity of Memory forensics is increasing for operating systems such as Windows, and this approach has also started to be used in the area of mobile devices. Since the number of smartphones and tablets are rapidly increasing today, malware detection studies have also gained significance for mobile devices; accordingly, Tam and Edwards [34] proposed a malware detection method for mobile devices. In their study, the malware was run on the emulator with Android 4.4 installation, and memory snapshots were taken by using the loaded kernel module called Lime. The received memory dump data were analyzed via Volatility Framework, and the libraries used by the processes, file system violations, process tree and strings were utilized through Volatility Framework. In the study, a small dataset was used, and therefore, authors suffered from the dataset size.

Schmidt et.al [35] carried out static malware analysis in the Symbian operating system by using the function calls. After the function calls had been collected via the IDA Pro tool, only the names of the functions were taken into account, in other words, no attention was paid to the argument and parameters. In the dataset, 254 function calls were obtained from 33 malware, and 3366 function calls were acquired from 49 benign files. Moreover, approximately 70% of the malware in this dataset make 50 function calls. Additionally, the centroid machine learning method was also proposed in the study. This method was found to obtain higher accuracy and precision compared to the machine learning methods such as Naïve Bayes and Binary SVM.

In their research, Kapratwar et.al [15] carried out malware detection studies for Android OS, compared the performances of the machine learning algorithms in their study, performed dynamic analysis according to the system call and conducted studies according to the static analysis method under the permissions in AndroidManifest.xml. In the studies, the dataset containing 103 malware and 97 benign files were used, and experiments were executed by using various machine learning algorithms such as 100-tree RF, 10-tree RF, Naive Bayes and Basic logistic, system permissions and system call features for the comparison of the machine learning algorithms. As there are 135 system permissions in the Android Operating System and a significant part of them are not used by the APKs in the dataset, these permissions were reduced to 87 and kept as binary vectors. The best result was obtained from the 100-tree RF algorithm. In the experiment, where the system calls were reviewed by using the system calls, the AUC was obtained as 0.884, and this result showed that using the system calls alone was inadequate in

malware detection. In the experiment conducted according to the system access permissions in AndroidManifest.xml, the AUC result was acquired as 0.972, and access permissions resulted in high accuracy for malware detection released for Android OS.

In their study, Banin et.al [25] proposed a malware detection method using the memory access operations. The dataset used in their study was comprised of the PE32 files received from VirusShare repository [36], however, the idle status is observed after a part of the malware in the dataset are run, because no graphical user interface(GUI) is available. Therefore, only the files containing a GUI were filtered and they were used in the study. One of the main assumptions in the study is that similar opcodes will lead to almost the same memory access operations when they are used with similar parameters. Basic memory access operations such as read and write were analyzed by using the dynamic binary instrumentation tool called Intel Pin in order to monitor the memory activities. Memory access operations (memtrace) were classified with the machine learning methods 'Naive Bayes, Bayesion network, J48, k-NN, Artificial Neural Network (ANN) and SVM' by using n-gram in different sizes, and the proposed method functions very fast.

After the information systems became widespread, there has been a serious increase in the number of security breaches. Memory forensic, which is frequently used for security breaches, refers to the examination of the volatile RAM of the device. In the data within the RAM, there is information such as the running processes, kernel modules, file information, source codes and registry keys. In their study, Petrik et.al [6] proposed an OS-independent malware detection method. Whereas the MalRec dataset [37] was used for malware in the study, they created the dataset for benign software by themselves; in the studies performed in Windows 7 operating system, the snapshot of each sample was taken and then the pre-processing procedure was initiated, and in this procedure, null bytes were deleted and the dump file was compressed with the MemScrimper tool [38]. 43 features were extracted by applying plethora of statistical, numerical methods, and images were generated in the png format in the dimensions of 1000x1000, 2500x2500 and 5000x5000. Models were created through machine learning and CNN. The features used in the study were handled independently of the OS domain-knowledge. However, this method suffers from real-time detection because the pre-processing procedure of the sample in each dataset takes about 30-45 seconds.

The use of cloud computing is increasing day by day, which makes the virtual machines (VMs) the target of cyber-attacks. Wang et.al [39] designed and implemented an automatic detection system to detect the kernel rootkits in VMs from private cloud. Kernel rootkits are difficult to detect compared to other types of malware. They used the memory forensic method in their study; feature extraction is also very effective in malware detection in the memory forensic method. Each sample was dumped on OpenStack, which is a popular cloud computing OS, with 10-minute intervals, because the memory changes as long as the process runs. Feature extraction was performed with the Volatility Framework on the acquired memory dump file, and consequently, features such as Orphan Threads and Driver Objects were obtained. The obtained features were vectorized and classified with machine learning, and the kernel rootkits were detected via the RF classifier with an accuracy of 98%.

Even though the Mac OS X operating system is generally considered as a safe operating system, according to the report published by McAfee [40], approximately 150.000 new MAC malwares were released for the Mac OS X operating system in 2018. In their study, Pojouh et al. [41] obtained an accuracy of 91% through classification via the supervised machine learning by using the systems calls. In the dataset used in their study, there were 152 malware and 450 benign files.

David and Netanyahu [42] proposed automatic malware signature generation and malware detection. In their study, log files containing API calls, API call parameters, registry entries, URL address, the used ports, etc were created, then the log files were converted to 5000-word strings with a fixed size via unigram (1-gram) extraction, which is a natural language processing (NLP) technique. Binary-bit strings were classified with Deep Belief Network (DBN) and an accuracy of 98.6% was achieved.

Shijo and Salim [43] suggested a method for malware detection by integrating static and dynamic malware approaches. The features used in the suggested method consist of both binary codes and dynamic behavior. The dataset collected from the VirusShare website was used; whereas the features created for static malware analysis were extracted from printable strings (PIS), API calls were collected on the Cuckoo sandbox for dynamic analysis, and feature vectors were formed by using 3-gram and 4-gram. The created features were classified with RF and SVM on the WEKA tool, and the highest accuracy was obtained as 95.88% by using the static PSI feature. The highest accuracy was achieved as 97.16% through the SVM algorithm by using the Dynamic API-call-gram

features. When the classification was performed via the integrated method, thanks to SVM, an accuracy of 98.71% was achieved, and the benefits of the integrated method were observed to be the superiority of static and dynamic malware analysis.

In their study, Santos et.al [44] utilized both static and dynamic features, and in this proposed new hybrid method, and static analysis features and opcodes were used and each opcode term frequency (tf) and frequency of occurrence were calculated. The PE files run in the Sandbox environment were turned into system calls, raised errors such as loading a DLL and operations log and vectorized. They were classified with the machine learning method, and the hybrid approach was found to provide higher accuracy compared to both the static and the dynamic methods.

Shellcode is a code fragment that gives command-line access on the target OS by exploiting the detected security vulnerability, therefore, it is maybe the most important part of the malware. For this purpose, in their study, Cheng et.al [45] focused on shellcode detection and paid attention the Nature API calls instead of the Win32 API calls because Nature APIs reflect the behavior of the core of a program in a better way. Feature extraction was performed for the API sequences in each log file via the 4-gram, and then classified with SVM. However, this study suffers from high false-negative rate, because the used dataset is too small.

Visualizing malware is one of the fields of study, the popularity of which has been increasing in recent years, and Shaid and Maarof [46] ran malware in the VM environment and collected user-level API calls in their study. The reason behind collecting user-level API calls rather than kernel-level API is that the user-level API calls are collected faster and the kernel-level API calls vaguely reflect the behavior of the malware. Each API call is sorted by the level of maliciousness, for instance, the DeleteFileA call is considered malicious and the API calls were mapped to a color. In this mapping process, the hot-to-cold color map was used, in other words, the API calls with low levels of maliciousness were mapped to cold colors while the API calls with high levels of maliciousness were mapped to hot colors. By this way, the malware was visualized behaviorally, and information can be obtained about to malware even by looking with bare eyes.

In their study, Alazab et.al [47] proposed a static analysis malware detection method using a fully automatic system and the API calls. The PE files were unpacked with the PEiD

tool and then collected API calls for various compilers via the IDA Pro tool. Feature extraction was carried out for the API calls by using [1-5] gram and then classified with SVM. Accuracy obtained with 1-gram was found higher compared to other n-gram methods. This study suffers from the unavailability of unpacking tools to use in the study.

When today's information systems are examined, it is seen that the use of NTFS file system is common; for example, Windows 7 and Windows 10 OS use the NTFS file system. Windows OS is an operating system with a large number of users around the world, and it has been the target of the attackers as it uses NTFS. In their study, Alazab et.al [48] conducted a static analysis with NTFS images. They duplicated the NTFS disk image with the tool called dcfldd to ensure that no change occurred during the analysis. The metadata in the image were extracted and the hidden data were revealed. Thus, the hidden data in malware were identified.

4. METHODS AND TOOLS

With the development of the information systems, attackers develop malware for many purposes, especially illegal earning. In order to detect the developed malware, researchers propose several methods. When the first methods proposed for malware detection are viewed, the signature-based method comes into prominence, however, this approach cannot detect sophisticated malware which are developed today. Therefore, in this thesis, the memory analysis approach, which is very successful in detecting obfuscated malware. Feature extraction and image downscaling were performed by means of computer vision techniques, and the machine learning method was also benefited from. This section consists of 3 sub-headings, and in Section 4.1, visual descriptors, the used GIST and HOG global visual descriptors and Lanczos interpolation are stated. In Section 4.2, the process layout, PE file format, and memory dump process are mentioned. In Section 4.3 different machine learning algorithms used in the study are discussed.

4.1. Visual Descriptors

Today, studies on images go through a number of operations thanks to the developing technology. It is known that these operations develop every passing day and different methods appear. It is essential to extract the most appropriate features from the images in the field of computer vision. Therefore, visual descriptors are used, which enables obtaining the features such as color and shape comprising the image [61]. In this respect, the features that define the image are extracted. After this process, the related problem can be solved with classification and clustering algorithms. Different techniques have been developed in this field, and they are still being developed. In this thesis, visual descriptors were used to extract the features of the visualized malware memory dump data. In the literature, there are different types of visual descriptors used for categorization, and groups are formed on the basis of local, global and biologically inspired methods. Among the descriptors, GIST and HOG were used in the study.

4.1.1. GIST

Solutions for storing a very big data set comprised of images are very expensive, therefore, the characteristics of the images can be represented low dimensionally. Oliva and Torralba [62] proposed the GIST global descriptor and attracted attention especially with their intention in using scene categorization. In the study, they anticipated that each scene could be expressed with a similar spatial structure. In their study, perceptual dimensions (roughness, expansion, openness, naturalness, ruggedness) were used for the representation of the spatial structure of a scene.

In the GIST descriptor, the input image is subjected to a 3-step process; it is padded, whitened and normalized. After this process, the GIST global descriptor, which can work for an image with any size divides the image into 4x4 blocks and then a Gabor filter is applied to each block. The Gabor filter is one of the methods frequently used in image analyses; the edges on the image are found via the filter, and it is often used for plate, fingerprint, iris recognition, etc. This filter is based on the Gaussian formulization in equation-(1).

$$g(x,y) = \cos(2\pi \frac{x'}{\lambda} + \phi)\exp(\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}) \quad (1)$$

4.1.2. HOG

The HOG global descriptor was proposed in the study conducted by Dalal and Triggs [63] in order to detect the pedestrians in stationary images. The feature of this descriptor is to recognize the images with edge orientation histograms. This algorithm is comprised of 3 basic stages. It is used on images for object detection, and the image is divided into cells and defined. In the definition process, the following procedures are applied:

1. brightness is calculated,
2. gradient vectors are calculated in consideration with the horizontal and vertical directions, local histograms are calculated,
3. block normalization is performed, and the vectors obtained from all blocks are summed.

As explained above, the HOG global descriptor works by dividing the image into cells, and the cell size was selected as 32 pixels in this study.

The gradient calculation mentioned for this descriptor forms the basis of the algorithm. Gradient calculation process means obtaining vertical and horizontal edges via two-dimensional filters called Sobel filter. Afterward, histograms are drawn as per the direction and size information of the vertical and horizontal edges. This direction information is calculated for each pixel in a rectangular form. The set of directions holding the calculated direction information for each pixel is also stated as histogram, and generally consists of the parts which take values between unsigned and signed ranges. In the third step, normalization takes place; it is done to enhance the performance against the possibility of contrast and illumination. Because rectangular modeling is employed in the algorithm, normalization is performed with blocks with mxm sized sliding windows. Equations used in these operations are stated in equation- (2), (3), (4), (5).

$$G_x = G * D_x \tag{2}$$

$$G_y = G * D_y \tag{3}$$

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{4}$$

$$\theta = arctan\frac{G_x}{G_y} \tag{5}$$

4.2. Lanczos Interpolation

Interpolation is the process of findings the unknown intermediate value from the known values or the values in the function [64]. The function which represents an f(x) function by using the discrete points expressed with x_0, x_1,.... x_n is called interpolation function, and functions such as polynomial, trigonometric functions and exponential functions.

There are many interpolation methods such as nearest neighbor, bicubic and bilinear interpolation that allow for finding the similarities between the images in the image processing area. Among these methods, the Lanczos method can make 3D calculations

according to horizontal, vertical and depth directions, and to do this, it uses a kernel filter called Lanczos [65]. This filter is formulated in equation(6) as L(x):

$$L(x) = \begin{cases} sinc(x)sinc\left(\frac{x}{a}\right) & if \ -a < x < a, \\ 0 & otherwise \end{cases} \quad (6)$$

The values resulting from this filter are called kernels, and their sum is expressed as the interpolation value for the desired pixel value.

4.3. Memory Layout

Nowadays, with the increase in malware, cybersecurity researchers propose a number of techniques to detect malware. Since these techniques are increasing, attackers release more robust malware to eliminate known and noticeable traces [25]. Attackers can easily obfuscate their malicious codes in disk, therefore, static analysis in malware detection does not give accomplished results. Security researchers need more robust techniques to detect malware. In recent years, some malware detection techniques using memory dump files have given meaningful results, and also when recent studies are examined, the following evaluation is seen: recent malware detection techniques tend to utilize hardware-based features such as memory. In this thesis, memory dump file is utilized, and this section introduces the PE file format and the memory dump file format.

After the PE files are compiled, they are kept in the hard disk, and when the PE files need to be run, the file which is available in the hard disk due to the Von Neumann Architecture is conveyed to the memory. After the PE files that are kept in the hard disk are run, they are named as process, and some information is kept in the memory. The PE file format is named as process after being run and obtains a different file format in the memory. The modern OS can run more than one process at the same time, even though the physical memory may not be able to keep the information of this many processes. In order to address this problem, the swap area is used. The pages that need to be kept in the memory are kept in the swap area, and the swap area exists in the hard disk. Thus, this has paved the way for easily running more than one process. When we examined the Windows OS, it was seen that the pages belonging to a process were not placed sequentially in the memory, in other words, after the page of a process, there can be a page of another process [23]. However, when the operating systems such as VxWorks were reviewed, it was observed that all the pages belonging to a process were kept sequential. In this section,

the PE file format and the sections of a process in the memory will be explained, respectively.

The PE file format is the advanced version of MZ, which is an old 16-bit file format developed by Mark Zbikowski. Since the processor architectures change today, compliance problems have appeared regarding MZ, which is an old file format. In order to solve these problems, the PE file format was developed by Microsoft (Figure 4.1). We frequently use this file format today, and the file types such as *DLL*, *SYS* and *OBJ* also use the PE file format. There are data and tables necessary for running the file that are included in the PE file structure; these tables are significant for the operating system and the necessary DLL installations are carried out by means of these tables while running. The PE file format starts with DOS, which show that the file will run in the DOS system. The PE header is found and then the PE signature under this header is found; The PE signature is used to confirm that the file is a PE file, moreover, information such as the processor architecture the file complies with, the size of the code segment, the file type, such as system or DLL file, the file belongs to, initial addresses of the data and text segments under this header, and this information is quite important. The data directory in the PE file indicates the addresses of the directories, after which the section headers are available; here, the sections are put in order in the memory or disk according to the information in the PE header. These sections have permissions and these permissions give information about the section, e.g. execution, reading and writing, and antiviruses get more information about the file according to this information, and section headers show the initial address of the relevant section. Sections follow the section headers, and various information such as machine codes, static/global data, imported and exported function information and debug information in the sections [22].

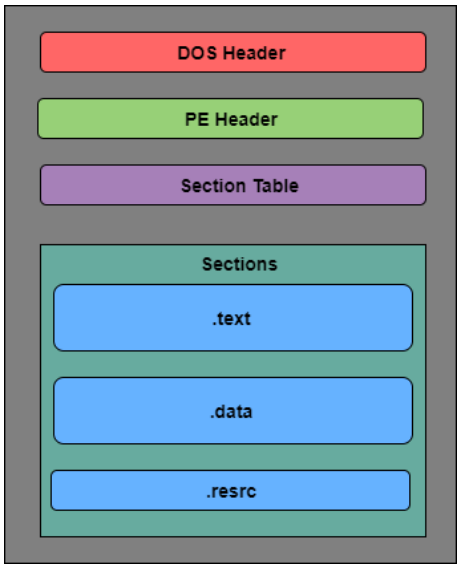


Figure 4.1. PE File Format

As mentioned earlier that the process structure kept in the memory is different from that of the PE file kept in the disk. The layouts of the processes in the memory can be stated as follows (Figure 4.2):

Dynamic Link Libraries (DLL): The DLL calls needed by the process are kept here [24].

Thread Stack: A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads. Each thread has its own register and can share the same memory with other threads. In this area, local variables, function return addresses, and function parameters are kept [24].

Process Heap: This dynamically allocates memory to a process during its run time. For example, data can be kept in the heap area with functions such as malloc used in C programming language. There is no size restriction like Stack, and reading and writing are slower in this area, because access is provided via the pointer.

Data Segment: In this segment, initiated variables, global variables and constant values are kept [24].

Text Segment: This segment exists only in the reading mode, no writing operation can be performed, because no modification can be made as a precaution against changing their own instructions. In this segment, there are code instructions, program counter (PC) value and the content of the process register [24].

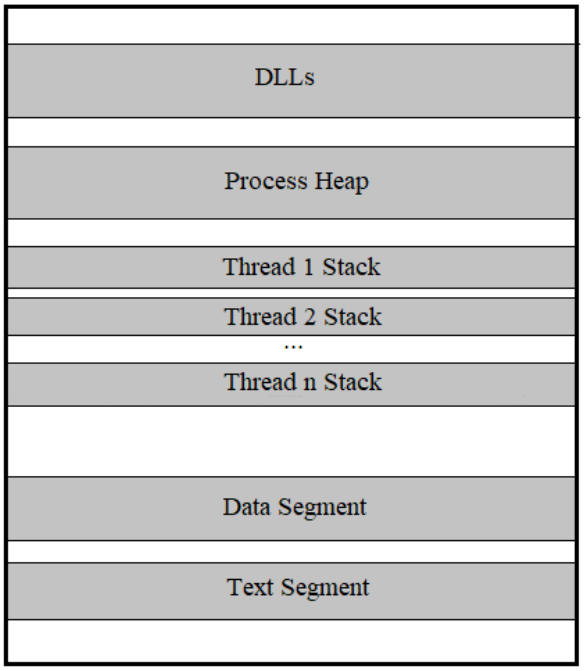


Figure 4.2. Process layout

The program addressed with different names such as the memory dump, core dump, and system dump is rather used for debugging and solving the errors, which is quite useful for program developers. The process can continue running after the dump procedure.

Unix-based operating systems use memory dump, generally, executable image-format; the ELF format is used in the modern Linux operating systems while the Mach-O format is used in the macOS operating system, and the extension of the created memory dump file changes according to the operating system. For example, the memory dump extension in the Windows operating system is **.dmp** whereas, it is **vmcore** in the modern Unix-based operating systems. Via Windows API, different segments of the process, the memory dump procedure of which is stated above, can be read or all the areas covered by a process mentioned above in the memory can also be read.

4.4. Machine Learning Methods

ML is a model that can make predictions on a specific dataset by using mathematical and statistical operations. It enables developing a model that can receive data from a data set as input data and predicting new data as output data. In this section, the machine learning algorithms, which do malware classification in the modeling section, are explained in

detail. Information is given about these algorithms because SVM, RF, XGB, J48, and SMO used in the model.

4.4.1. SVM

SVM, which is a frequently used and a very popular classifying algorithm, is a supervised machine learning algorithm. The SVM algorithm, which was proposed by Vladimir Vapnik, has many application areas such as face recognition, character definition, sentiment analysis and classification of images. The SVM algorithm based on the statistic and structural risk optimization has the following advantages.

- It is quite effective in high dimensional spaces.
- It is easy to apply.
- It is effective when the number of dimensions is higher than the number of samples.
- It can be used for solving many problems with small changes.
- It works memory-efficiently.
- It can be applied to both linear and nonlinear data.

It is possible to do classification by drawing a line between two groups on a plane. SVM is a method that can make binary classification through optimized hyperplanes by maximizing the distance between two classes. As can be seen in Figure 4.3, the closest class elements of the optimized hyperplane are called support vectors [49]. Basically, the SVM algorithm decides on how to draw this line.

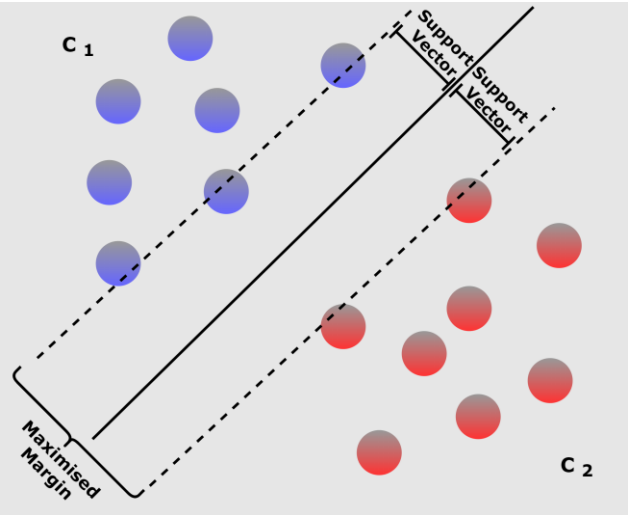


Figure 4.3. Support Vector Machines

The success of the SVM algorithm used for solving the classification problem in the machine learning literature depends on the selection of the linear and nonlinear SVM type suitable for the dataset. The linear SVM aims at finding the hyperplane that maximizes the distance between the vectors of different classes.

4.4.2. Random Forest

Random forest is a multi-class classification algorithm relying on the supervised learning method presented to the literature by Leo Brieman [66]. Random Forest algorithm is quite popular in machine learning applications since it provides good results both in regression problems and classification problems without a need for hyperparameter adjustment.

Even though random forest was initially developed for problems such as weather forecast and object recognition, its usage area has expanded today. The algorithm is based on the decision tree structure for solving these problems. More than one decision tree is randomly used in the algorithm, and there is a root node in the random forest algorithm. While the random forest (RF) algorithm is applied, some steps must be followed. In the first step, the data set is separated into train and test. In the second step, as per the number of variables designated at the beginning of the algorithm, branching is started to gain the most information. In the third step, branching is carried on until the classes are separated homogeneously. The operations are repeated until the randomly designated number of trees is obtained. The most appropriate model is discovered by summing the error rate of each tree.

Random forest has a lot of different aspects than decision trees. In random forest, randomly sampled vectors, which are produced from each of these trees, are created. Afterward, the class with the least error rate is selected. In the algorithm, there are two parameters containing the number of trees and the number of variables used in each node. Besides, there is no need for pruning in this algorithm, unlike the decision tree. This procedure was developed to prevent overfitting, however, random selection has a higher effect on performance. Random selection is done via the bootstrap method.

4.4.3. XGBoost

Xgboost was developed by Chen and Guestrin [67] as an alternative to the machine learning algorithms. It was aimed to make it faster and have higher performance compared to other algorithms. To run the algorithm, train and test data are determined

first. Then, the first weight assignment is carried out. The modeling step begins in this way. After the first iteration is completed, the weights that lead to misclassification are specified again. The accuracy of the model increases thanks to the continuous re-modeling. Because it is based on gradient boosting, it works fast, and decision tree is used in this algorithm [69]. Additionally, it is used for regression and classification problems. The first version of this algorithm is called Boosting, and it was developed over time, after which the XGBoost algorithm was created. For this reason, it is important to define the Boosting algorithm. The boosting algorithm was developed by Schapire to enhance the learning performance of the weak learning model [68]. It aims at training each weak predecessor in the model to follow one another. In other words, first of all, a model is developed, and then, the previous faulty outputs are taken into consideration while creating the new model. Modeling is done again with appropriate parameters. The most popular boosting method is Gradient Boosting, which includes Adaboost and XGBoost. Gradient Boosting corrects the errors of the previous model and adapts it to the new model. Unlike Adaboost, Gradient Boosting does not update the weights in iteration, however, works according to the function in equation-(7).

$$MSE = sum(y_i - y_{i_p})^2 \tag{7}$$

It has two methods, which are column sampling and subsample methods. Whereas the column sample method creates the tree with the samples randomly selected between 0 and 1, trees are created with the samples randomly selected from the sub-dataset without a certain interval in the subsampling method. The ability to obtain the best results in the algorithm thanks to parameter determination eliminated overfitting and underfitting.

4.4.4. J48

In the literature, there are many algorithms used in different fields, and among these algorithms, decision trees are frequently preferred in the classification problem. Decision trees are used for categorical and numerical classification and prediction because of their simple structure, being non-parametric and easy applicability for models. Decision trees were created by Quinlan through the development of the ID3 algorithm [70]. In this algorithm, a structure consisting of root and leaf nodes was formed. Each sample is indicated with a node. After preparing the tree structure by this way, a new sample is placed in an appropriate place beginning from the root. The last node determines the class of that data. In other words, there are 3 steps in this algorithm: (i) creation of the data set,

(ii) assignment of the data with the most distinctive feature as root, (iii) calculation of the information gain according to the new data. For the information gain step, the suitability of each node to the subset it is connected to is checked. It is necessary to remove the parts of a sample which do not have contribution for more successful trees that are far from noise without affecting the structure of the tree. This operation is called pruning in decision trees. The structure of a sample decision tree is shown in Figure 4.4.

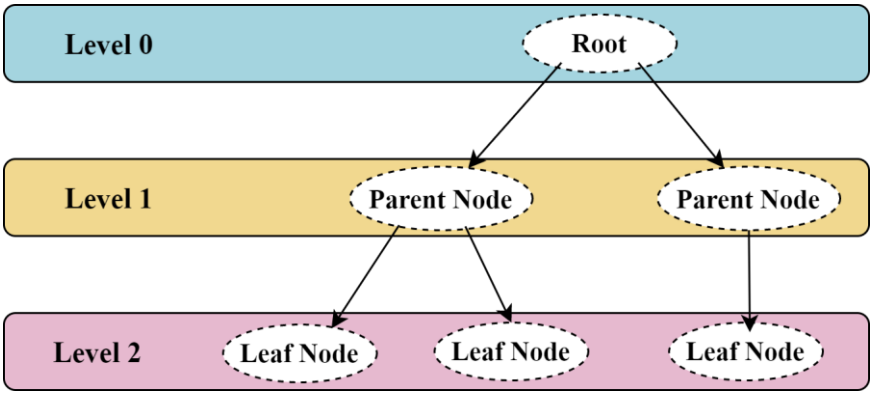


Figure 4.4. J48 Tree Classification

As the decision tree algorithms are based on information gain, different types have been developed to classify the data in compliance with the specifications given. Among them, the J48 algorithm divides the samples from the best place to maximize the knowledge gain. Mainly, it divides the tree according to the rules of "IF-THEN". It enhances the performance by pruning, which is called removing the insignificant data from the tree.

4.4.5. SMO

SMO (Sequential Minimal Optimization) was developed by John Platt. It is based on the support vector machine and preferred for optimization problems. To solve these problems, incomplete and categorical data in the dataset are also subjected to normalization and transformed into binary qualities [71]. Unlike the general SVM algorithm, this algorithm can solve more difficult classification problems in a faster way. Thanks to the learning algorithm called quadratic programming, there is no need for the stages of storing extra matrices and optimization. In addition, it solves as small optimization problems as possible in each learning step. The fact that it is comprised of two components as determination and optimization of the Lagrange multipliers shows

that it is more advantageous than SVM. This algorithm determines two Lagrange multipliers suitable for the problem for optimization. SVM can be updated in this way.

4.5. Tools

The study within the scope of this thesis consists of many phases and each phase is extremely important for study. In this section, the tools used in all the phases in the study are introduced and the version information of the tools used is also stated.

4.5.1. OpenCV

Open Source Computer Vision (OpenCV) is the image processing library which we often heard. It is frequently used in areas such as object detection, face recognition, and motion prediction, and it is also preferred by large companies such as Google, Yahoo, Intel, and IBM. The OpenCV library can be used with programming languages such as C/C ++, Python, Java and Matlab. In this study, OpenCV 4.1.2.30 in Python 3.6 was used.

4.5.2. Python

Python, which was developed by Guido Van Rossum in the early ‘90s, is a dynamic object-oriented, modular and high-level script language. Python works on independent platform; therefore, it works on many OSs such as Windows, Linux, Mac OS X, BSD, and Android. When compared to programming languages such as C/C ++, Python does not need compilation; in this respect, it is easier to develop it, it is easier to design it compared to other programming languages since it has available functions and data structure. As the Python language has a simple syntax, writing a program is easier and its readability by someone else increases. The popularity of Python has also increased with its widespread use in large IT companies. Companies such as Google, Dropbox, YouTube, Yahoo and Intel use the Python language for different purposes. In this study, Python 3.6 was used.

4.5.3. Scikit Learn

Scikit learn, which is one of the libraries frequently used in artificial intelligence studies, is free of charge and open source. It incorporates many basic methods such as linear regression, logistic regression, and decision tree. The fact that it incorporates basic methods and can easily have data analysis has caused it to be popular in the artificial intelligence field. In this thesis study, the 0.21.3 version of the scikit learn library was used.

4.5.4. Weka

Often mentioned in academic studies involving machine learning, Weka is an open-source program that was developed by Waikato University and in which requirements such as machine learning algorithms and data pre-processing are presented. It has a quite simple interface. It can read files in different file formats such as arff and csv. In this study, Weka 3.8 was used.

4.5.5. Bin2Png

It is possible to convert any binary file into an image. This image can be in various formats such as jpg, png, and jpeg, but the resulting image is in the RGB format. Since the dump files obtained in the study are binary, they can turn into images. For the conversion of the binary file into the png file, the "bin2png" script coded in the Python language on https://github.com/ESultanik/bin2png was used. Because this script was written in compliance with Python 2.7, it was adapted to Python 3.6 for more efficient use. Then, due to the large size of png images created using "bin2png", increasing difficulty in classifying them with machine learning and the need for comparing the results, the need for downscaling with minimum data loss emerged. Therefore, codes were added to the "bin2png" script for Lanczos interpolation.

4.5.6. Procdump

Thanks to this tool developed by Microsoft, we can monitor processes, and dump files can be created. While Procdump, which is a command-line application, was initially available only in the Microsoft Windows operating system, the Linux support was provided after November 2018. The Linux codes of the open-source Procdump tool can be accessed on https://github.com/Microsoft/ProcDump-for-Linux. Via the Procdump tool, dump files can be created in all the processes or the dump file can be created only in the target process. Also, creating dump files includes virtual address space [24]. In this study, Procdump v9.0 was used.

5. APPROACH

While detailed information is given in this section about the working principle of the system we recommended, the system consists of 4 phases (Figure 5.1). In Section 5.1, memory dumping of the files included in the dataset, in Section 5.2, the visualization of the memory dump data obtained, in Section 5.3, the feature extraction from the visualized images and in Section 5.4, the modeling and classification of the vectors obtained after the feature extraction process through 5 different algorithms are explained.

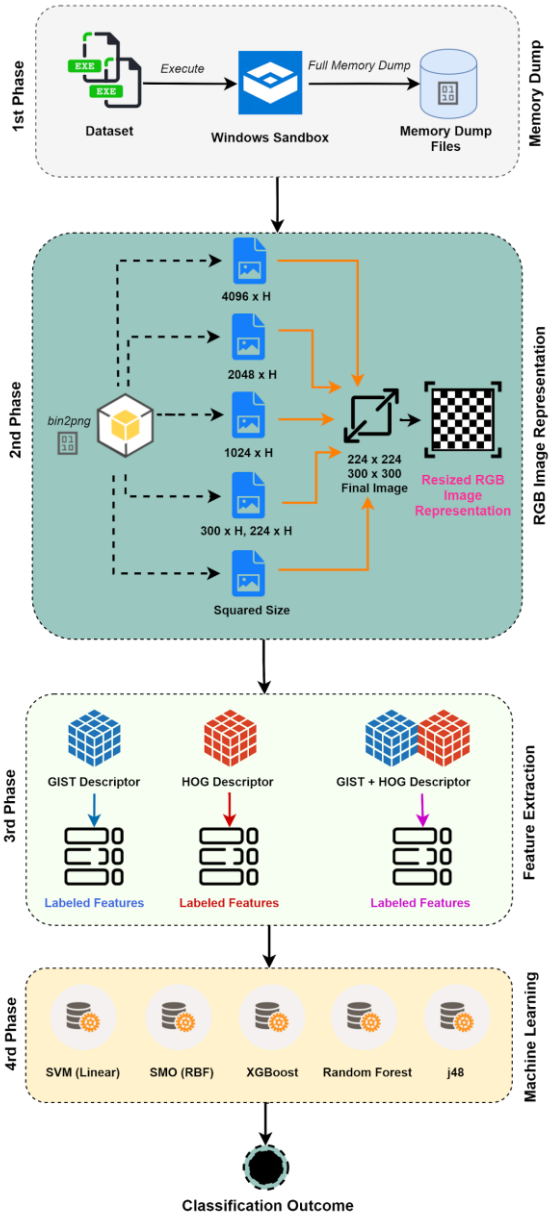


Figure 5.1. Model architecture [72]

5.1. Gathering Memory Data

When a PE file is executed, it leaves a trace in the memory. These data in the memory are alive until the power of the system is shut down. There is valuable information in the memory such as register keys, network activities, operating system, and DLLs, moreover, procedures such as unpacking and decompression in the static analysis method are not required. Memory dump data were used in our study due to being more robust to obfuscation than other malware analysis methods.

In order to get the memory dump of malicious PE files, it is necessary to run the PE files first. When malicious PE files are executed, they can damage the system in which they are run, therefore, these files are analyzed in the test environment. In our study, malicious PE files were executed in the Windows Sandbox environment, which was obtained with the May (1903) update of Windows 10. The PE files in the dataset we used were run in the Windows Sandbox environment and both the physical memory and virtual address space of the process were dumped via Procdump, which is a Windows Sysinternal tool as explained in Section 4.5.6. The Procdump tool was run with the arguments -ma -w. To explain the arguments, the memory of all the target process will be written on the dump file with the -ma argument and the Procdump tool will be waiting until the target process is run, through the -*w* argument. First, the Procdump tool was run, after 2500 ms, the PE file was run and dumping operation was carried out. The extension of the acquired dump files varies according to the operating system. While memory dump files with the .dmp extension appear in the Windows operating system, dump files with the vmcore extension are formed in the modern Unix operating systems. Since Windows Sandbox is used in the study, the obtained dump files have the extension of .dmp. The sizes of the dump files change according to several factors such as the DLL files used. The sizes of the dump files obtained in our study range between 10 MB and 100 MB. These memory dump files are called “Dumpware10” and they have been made accessible on https://web.cs.hacettepe.edu.tr/~selman/dumpware10/ to be used in future malware detection studies.

5.2. Image Representation

Visualization of the binary data has been used very frequently in recent studies. Thanks to the visualization of binary data, the computer vision methods can be used. In their study, Nataraj et al. [50] converted binary byte sequences into a gray-scale image, hence, they had a chance to employ computer vision and machine learning methods.

Each memory dump data of the processes we obtained in our study were visualized, so feature extraction operation could be carried out on the images by using the GIST and HOG global descriptors, which are among the computer vision methods. After the memory dump data explained in Section 5.1 were obtained, RGB images were acquired by running the "bin2png" script stated in Section 4.5.5. The image format was selected especially as png instead of different formats like jpg, jpeg, and bmp. Compression is done in jpg and jpeg image formats; however, some losses occur in the data. While this compression with losses is not preferred as it will not describe the memory dump accurately, no data loss occurs in the bmp image format, however, no compression can be done. Therefore, images with large sizes will be obtained while converting the memory dump files into images, and feature extraction over these images will be expensive. For this reason, the png image format was selected, because compression can be done without losing data. When the studies in the literature were reviewed, it was seen that the png image format was preferred [41,61,65].

Since we do not have detailed information about the dumped memory data, the dump data were formed with the square root method, in other words, images were created within the dimensions of a square; images were also formed with a width of 4096, 2048, 300, 224 x undetermined height. Since the byte number of the memory dump data is known in the square root method, the required pixel number is found in the RGB images, and then, the images are created with the same width and height. The total pixel number of the image to be created is desired to be the same with the required pixel number or to have the least difference. For instance, a 4x4 RGB image will be formed with the square root method for a 46-byte data. 1 pixel holds 3 bytes and a 4x4 RGB image can hold a 48-byte data (4x4x3), however, our data is 46 bytes, the remaining 2 bytes are considered as padding and filled with black color.

RGB images have been created with a width of 4096, 2048, 300 and 224 and undetermined height. The height of the images formed with a specific width is variable

because the sizes of the memory dump data are variable. Since the sizes of the images formed both with the square root method and a certain width are quite variable and they are largely due to their file sizes, the use of computer vision methods is restricted. Furthermore, it would extend the working duration of the machine learning algorithms, therefore, the images were downscaled to have the dimensions of 300x300 and 224x224 with the Lanczos interpolation method in the OpenCV library. As a result of this process, all the images had the same size.

Afterward this process, the images in Figure 5.2 were obtained after they were created in RGB with the 4096xUndetermined height method belonging to 3 different malware classes, the Lanczos interpolation was performed and downscaling to 300x300 was done.

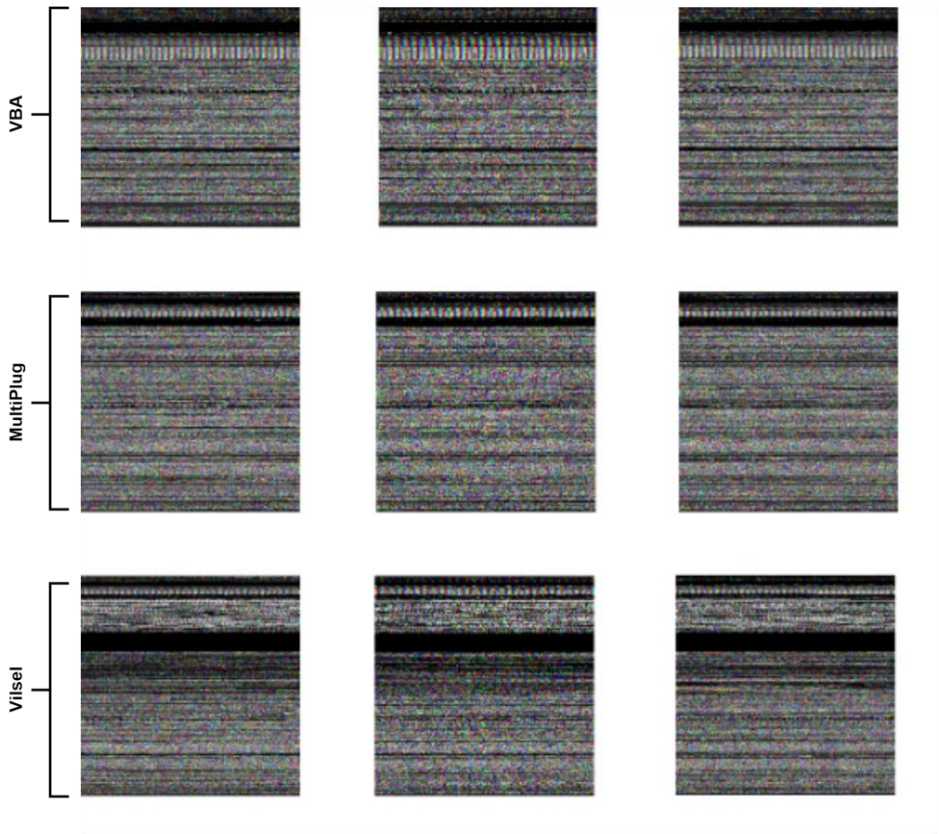


Figure 5.2. Visualization of memory dump in 3 different families

5.3. Image Feature Extraction

Images were created in the png format as RGB and the images were subjected to feature extraction and vectorized by using the GIST and HOG global descriptors explained in Section 4.1.1.1 and Section 4.1.1.2. The GIST and HOG global descriptors were written with Python programming language, and the Sklearn library was used. The cell size of the HOG global descriptor we used in our study was selected as 32 pixels, and feature extraction was performed by dividing the images into cells. All the RGB images formed with a fixed width of 4096, 2048, 300, 224 and square root images then downscaled to the fixed dimensions of 224x224 and 300x300 via Lanczos interpolation were vectorized in 3 different ways for GIST, HOG, and combined GIST and HOG.

The GIST and HOG vectors were combined by respectively using the vectors of the GIST global descriptor and the HOG global descriptor, and a new vector was obtained. These vectors were saved in the csv file. Since they consisted of a total of 6 different widths for each RGB image, they were downscaled to 2 different fixed sizes and then feature extraction was done as the GIST, HOG, and combined GIST and HOG. As a result, each PE file was vectorized in 27 different ways.

5.4. Classification via Machine Learning

Until this point, 4130 portable executable (PE) files in our dataset obtained from Comodo Inc. were run in the Windows Sandbox environment, and then, the memory dump data of the target processes were received with Procdump, which is a sysinternal tool developed by Microsoft. After this procedure, memory dump data were converted to RGB images in different sizes, and then, the images were reduced to the dimensions of 224x224 and 300x300 via Lanczos interpolation. For images with the dimensions of 224x224 and 300x300, feature extraction was done using the GIST and HOG global descriptors, and consequently, vectorization was performed in 27 different ways for each portable executable (PE) in the dataset and they were saved in the csv file.

3433 of the 4130 portable executable (PE) files in our dataset were kept for training purposes, so each of the 3433 files had a total of 27 different vectors in RGB. In this phase, a total of 27 different models were formed for each vector type of the RGB images. At this phase, SMO, Linear SVM, RF, j48 and XGB, which are machine learning algorithms, were used, and the Weka tool and developed with the Java programming language as an open-source and python scripts were utilized in the modeling. Many

machine learning algorithms are available in the Weka tool, therefore, in this study, the Weka tool was used for SMO, Linear SVM, RF, and j48 machine learning algorithms. However, python scripts were used as the XGBoost machine learning algorithm is not available in the Weka tool. The csv files where we saved the vectors during feature extraction can be read by the Weka tool. You can find the used machine learning methods and their corresponding algorithms in Table 5.1.

Table 5.1. The used machine learning models and corresponding algorithms

Machine Learning Models	Algorithms
SVM	SMO, Linear SVM
Decision Tree	Random Forest, J48, XGB

861 of 4130 portable executable (PE) files in the dataset were separated and assessed for validation. Vectorization was done in 27 different ways for each PE file, as described in Section 5.3. Classification was done with the help of the models that had been created in the machine learning phase. Following the classification process, each portable executable (PE) file in the dataset is assigned to one of 11 different classes in total, 10 of which are of the malware class and 1 is of the class with legitimate portable executable (PE) files tagged as 'Legitimate'. This procedure was carried out through the Weka tool and python scripts. Besides, the interpretation of the validation result was made according to the evaluation criteria.

6. EXPERIMENTS AND RESULTS

In previous sections, we touched upon the studies conducted on malware detection before, the functioning idea of this thesis study, the tools used and the main purpose of the study. In this section, the dataset, evaluation criteria and validation results will be discussed. In Section 6.1, the dataset which was used in the study and enabled us to create and verify a model, in Section 6.2, the evaluation criteria of the study and in Section 6.3, validation results will be stated. The modeling with the dataset and the verification of the results according to this model were accomplished on the system with Intel 3.6 GHz processor, 16 GB Ram, and 64-bit operating system.

6.1. Dataset

In the literature studies, it is seen that the preparation of the dataset and the structure of the test data are quite significant. Some restrictions according to the dataset seriously influence the results in the studies. When malware in datasets get into the hands of unwanted people, they pose a great threat, and therefore, it is very difficult to find malware dataset. When the studies conducted for malware detection are examined, it is seen that very small datasets are used. The total dataset used in this study was divided as 80% training and 20% test samples with 3433 training and 861 test samples, and it seems like a quite large dataset when compared to previous studies.

When datasets are free to use and can be shared by everyone, full public view, they are called open data; while close data cannot be accessed by everyone as a security precaution. While open data are free of charge or lead to minimum cost due to their availability and easy accessibility, close data result in a substantial financial burden. Close data were used in this study as the proliferation of the malware dataset mentioned above was not desired.

The dataset used in the study was supplied by Comodo Inc., and there are raw PE files comprised of 3433 training and 861 test samples. There are 11 classes in the dataset, and 10 of these classes include malware samples from different families whereas 1 class contains legitimate or benign PE samples. Both malware and benign PE files can be classified and the malware family can be identified with the benign PE file samples in the dataset. The PE files in the dataset were collected from the malware that emerged in 2017 and 2018, and the malware consists of the adware, trojan, worm and virus categories.

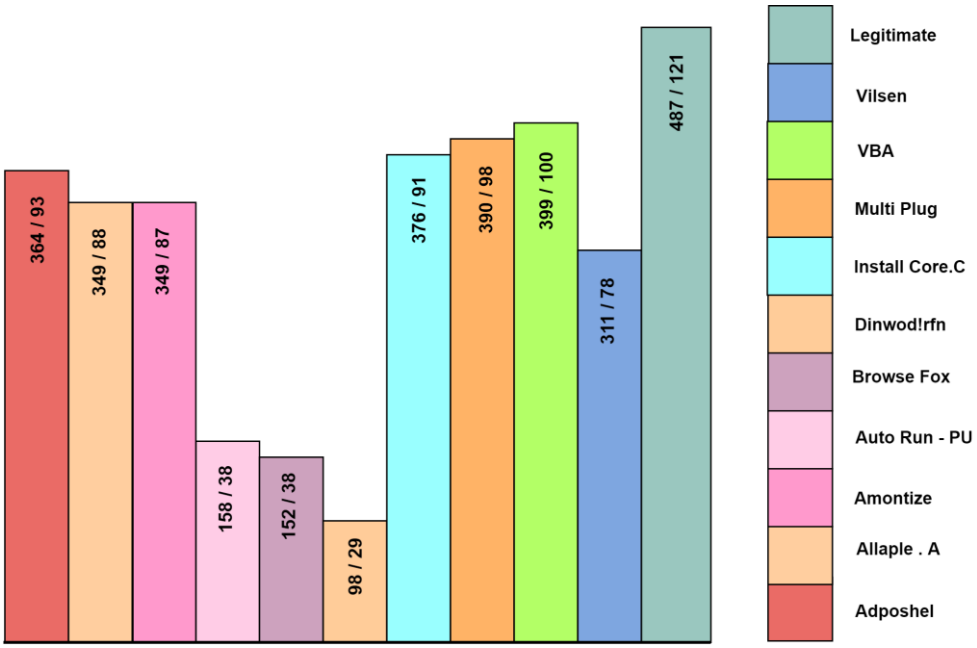


Figure 6.1. Distribution of Training/Test Samples

The numbers of the training and test samples of each class in the used dataset are seen in Figure 6.1, Table 6.1, where the categorical information of the malware is stated.

Table 6.1. The existing contents of dataset

Class Name	**Category**
Adposhel	Adware
Allaple.A	Worm
Amontize	Adware
AutoRun-PU	Worm
BrowseFox	Adware
Dinwod!rfn	Trojan
InstallCore.C	Adware
MultiPlug	Adware
VBA	Virus
Vilsel	Trojan
Legitimate	-

6.2. Evaluation Criteria

This section describes several classification metrics used by researchers. More than one evaluation criterion have been developed and used in order to measure the success rate in the classification models. Even though the criteria that include human intervention are quite popular among the improved evaluation criteria, it has been seen that they are very expensive and time-consuming compared to the evaluations made with computer intervention. One of the most common mistakes made in the classification models is the consideration of only one metric. For instance, most researchers only consider accuracy. Recall, precision, accuracy and F1 score can be given as examples for these metrics. All of these metrics were derived from 4 values in the confusion matrix presented in Table 6.1.

Confusion matrix: It is the shortest and clearest way to put forward the results of a classifier, and it is a measurement tool that gives information about the accuracy of the predictions [26]. This confusion matrix is used to describe the performance of the classification result with test data, the real values of which are known. You can see an example of this matrix in table 6.1, the size of the table is 2x2 and it can also be NxN.

True Positive (TP): It points at actually positive and positively predicted samples. Here, TP is found when we classify the malware file as malware.

False Positive (FP): It points at actually negative and positively predicted samples. FP is found when we classify the legitimate file as malware.

True Negative (TN): It points at actually negative and negatively predicted samples. Here, TN is found when we classify the legitimate file as legitimate.

False Negative (FN): It points at actually positive and negatively predicted samples. Here, FN is found when we classify the malware file as legitimate.

You can see the 4 values in the confusion matrix in Figure 6.2.

		Predicted Class	
		Malicious	Benign
Actual Class (Ground Truth)	Malicious	True Positive (TP)	False Negative(FN)
	Benign	False Positive (FP)	True Negative(TN)

Figure 6.2. Confusion Matrix

Recall or True Positive Rate (TPR): Recall, which is briefly signified with the letter r, is the ratio of correctly predicted malware files to those which are actually malware. Recall is calculated as in equation (8).

$$Recall(r) = \frac{TP}{(TP+FN)} \tag{8}$$

Accuracy: Accuracy, one of the easiest metrics to understand intuitionally, is the ratio of the correctly predicted malware and benign files to all the predictions. If we have high accuracy, the model we use can be considered the best, however, other metrics may need to be used if the FP and FN values are found quite different [31, 32]. Accuracy is calculated as in equation (9).

$$Accuracy(acc) = \frac{(TP+TN)}{(TP+FN+TN+FP)} \tag{9}$$

Precision or Positive Predictive Value (PPV): Precision, which is briefly signified with the letter p, is the ratio of correctly predicted malware files to the predicted malware files. Precision is calculated as in equation (10).

$$p = \frac{TP}{(TP+FP)} \tag{10}$$

F1-Score (F1): It is the harmonic average of precision and recall. For this reason, this metric considers the FP and FN results. Understanding F1-score is not as easy as accuracy, and it is more logical to use it instead of accuracy in the models with an irregular class distribution. If the FP and FN values are very different, it is necessary to have a look at both accuracy and F1 score [31, 32]. F1-score is calculated as in equation (11).

$$F1 = 2 * \frac{Precision*Recall}{(Precision+Recall)} \tag{11}$$

6.3. Results

In this section, the results of the experiments executed with the square root method and with the 224, 300, 2048, 4096 fixed-width method will be explained in detail.

First, the results regarding the images created with the square root method by using the GIST global descriptor are stated. Through Lanczos interpolation, the images that were created with the square root method were downscaled to the 224x224 fixed size. The images with the fixed 224x224 and 300x300 sizes were vectorized with the GIST global

descriptor, and the obtained vectors were classified with 5 different machine learning algorithms; the classification results are presented in Table 6.2 and Table 6.3.

In the experiment method described above, the results acquired by using the HOG global descriptor and GIST+HOG instead of the GIST global descriptor are demonstrated in Table 6.4 and Table 6.5.

Table 6.2. GIST results of the conversion to 224x224 via the square root method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	87,3403	0.017	0.882	0.873	0.874
SMO(RBF)		**89,7793**	**0.012**	**0.903**	**0.898**	**0.899**
SVM(Linear)		86,4111	0.016	0.872	0.864	0.866
XGB		86,8757	0.013	0.873	0.868	0.868
J48		72,3577	0.031	0.739	0.724	0.728

Table 6.3. GIST results of the conversion to 300x300 via the square root method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	300x300	88,0372	0.016	0.885	0.880	0.878
SMO(RBF)		**91,7538**	**0.009**	**0.917**	**0.918**	**0.916**
SVM(Linear)		88,7342	0.012	0.887	0.887	0.885
XGB		88,1532	0.012	0.887	0.881	0.880
J48		71,1963	0.033	0.724	0.712	0.716

Table 6.4. HOG results of the images created with the square root method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1 Score
RF	224x224	86,1789	0.019	0.878	0.862	0.860
SMO(RBF)		**87,8049**	**0.015**	**0.882**	**0.878**	**0.879**
SVM(Linear)		83,6237	0.020	0.843	0.836	0.838
XGB		85,8304	0.014	0.865	0.858	0.857
J48		68,0604	0.033	0.707	0.681	0.689
RF	300x300	88,8502	0.014	0.896	0.889	0.886
SMO(RBF)		**89,7993**	**0.012**	**0.901**	**0.898**	**0.898**
SVM(Linear)		85,1336	0.017	0.857	0.851	0.853
XGB		87,2241	0.013	0.876	0.872	0.870
J48		66,899	0.037	0.683	0.669	0.674

Table 6.5. GIST+HOG results of the images created with the square root method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	89.4309	0.015	0.903	0.894	0.895
SMO(RBF)		**93.4959**	**0.008**	**0.936**	**0.935**	**0.935**
SVM(Linear)		90.2439	0.012	0.903	0.902	0.903
XGB		89.7793	0.005	0.913	0.901	0.902
J48		73.5192	0.029	0.741	0.735	0.736
RF	300x300	91.1731	0.011	0.915	0.912	0.911
SMO(RBF)		**95.3542**	**0.005**	**0.954**	**0.954**	**0.953**
SVM(Linear)		92.2184	0.008	0.922	0.922	0.922
XGB		92.1022	0.008	0.924	0.921	0.921
J48		73.8676	0.028	0.759	0.739	0.738

As can be seen in Table 6.2, the highest accuracy was found as 89.7793% using the SMO (RBF) machine learning algorithm. The F1-score acquired with the SMO (RBF) algorithm is 0.899. The highest second accuracy obtained was calculated as 87.3403% with the RF machine learning algorithm. The lowest accuracy was found to be 72.3577% with the J48 algorithm. In Table 6.3, the highest result was achieved as 91.7538% with the SMO (RBF) algorithm. According to the results, it was observed that the images created as 300x300 gave higher accuracy than the images created as 224x224.

The results of the experiment conducted with the HOG global descriptor are given in Table 6.4. According to the results, the highest accuracy was obtained in the 300x300 image size as 89.7993% using the SMO (RBF) machine learning algorithm. The F1-score acquired using the SMO (RBF) algorithm is 0.898. In the images created as both 300x300 and 224x224, the lowest accuracy was calculated with the J48 algorithm. According to the results, the accuracy obtained from the images created as 300x300 was higher than the accuracy obtained from the images created as 224x224.

The results of the experiment executed with the GIST+HOG global descriptor to obtain higher accuracy are stated in Table 6.5. In the experiments performed with the 300x300 image size, the SMO(RBF) machine learning algorithm set forth the highest result, and this rate is 95.3542%, and the F1-score obtained with the same algorithm is 95.3542%.

Combining and using the GIST and HOG global descriptors helped with getting higher accuracy than using only the GIST or the HOG descriptor. Moreover, the accuracy and F1-score obtained in the 300x300 image size were found to be higher than the accuracy and F1-score obtained in the 224x224 image size. Finally, the SMO(RBF) machine learning algorithm had a better performance than other algorithms used.

Table 6.6. GIST results of the images created with the 224 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	86.0627	0.017	0.867	0.861	0.857
SMO(RBF)		**87.4564**	**0.014**	**0.880**	**0.875**	**0.875**
SVM(Linear)		85.3659	0.016	0.858	0.854	0.853
XGB		86.1236	0.016	0.868	0.862	0.864
J48		72.7062	0.029	0.731	0.727	0.726

Table 6.7. HOG results of the images created with the 224 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	86.8757	0.017	0.874	0.869	0.866
SMO(RBF)		**90.8408**	**0.010**	**0.910**	**0.909**	**0.909**
SVM(Linear)		85.7143	0.015	0.862	0.857	0.859
XGB		87.1456	0.014	0.872	0.873	0.872
J48		67.7125	0.036	0.687	0.677	0.679

Table 6.8. GIST+HOG results of the images created with the 224 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	89.8955	0.012	0.903	0.899	0.897
SMO(RBF)		**94.5422**	**0.006**	**0.946**	**0.945**	**0.945**
SVM(Linear)		92.1014	0.009	0.923	0.921	0.921
XGB		90.5614	0.010	0.905	0.904	0.905
J48		73.2869	0.028	0.741	0.733	0.735

Images created with the 224 fixed size were converted to 224x224 and 300x300 via Lanczos interpolation, and then, feature extraction was done with GIST, HOG, and GIST+HOG, and thus, the images were vectorized. The vectors were classified with 5 different machine learning algorithms. The results are given in Table 6.6, Table 6.7 and Table 6.8.

The results obtained with the GIST global descriptor are presented in Table 6.6, and according to the results, the highest accuracy was obtained using SMO (RBF). Furthermore, it was seen that 300x300 gave better results than 224x224 in the images downscaled via Lanczos interpolation.

In Table 6.7 and Table 6.8, the results obtained respectively with the HOG global descriptor and the GIST+HOG global descriptor are presented. It was observed that using the GIST+HOG global descriptor was more successful than using only the GIST or HOG global descriptor. The highest accuracy was achieved using the SMO (RBF) machine learning algorithm.

Table 6.9. GIST results of the images created with the 300 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	86.627	0.017	0.867	0.861	0.857
SMO(RBF)		**89.869**	**0.009**	**0.899**	**0.899**	**0.898**
SVM(Linear)		88.966	0.012	0.889	0.890	0.888
XGB		88.385	0.016	0.889	0.88	0.887
J48		72.762	0.029	0.731	0.727	0.726
RF	300x300	89.082	0.013	0.892	0.891	0.890
SMO(RBF)		**91.405**	**0.010**	**0.915**	**0.914**	**0.914**
SVM(Linear)		88.037	0.014	0.879	0.880	0.879
XGB		89.091	0.013	0.898	0.890	0.882
J48		74.099	0.027	0.756	0.741	0.746

Table 6.10. HOG results of the images created with the 300 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	84.9013	0.019	0.851	0.849	0.842
SMO(RBF)		**87.0116**	**0.011**	**0.870**	**0.871**	**0.870**
SVM(Linear)		83.0627	0.015	0.833	0.831	0.831
XGB		85.8302	0.016	0.0862	0.858	0.859
J48		65.6214	0.036	0.700	0.656	0.672
RF	300x300	86.0627	0.018	0.867	0.861	0.856
SMO(RBF)		**89.3148**	**0.012**	**0.894**	**0.893**	**0.892**
SVM(Linear)		85.4826	0.016	0.861	0.855	0.856
XGB		87.2128	0.014	0.870	0.871	0.870
J48		70.8479	0.031	0.726	0.708	0.712

Table 6.11. GIST+HOG results of the images created with the 300 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	89.7793	0.013	0.905	0.898	0.898
SMO(RBF)		**92.1022**	**0.009**	**0.921**	**0.921**	**0.920**
SVM(Linear)		91.7538	0.009	0.919	0.918	0.918
XGB		91.5214	0.004	0.924	0.921	0.923
J48		75.6098	0.027	0.764	0.756	0.759
RF	300x300	90.5923	0.012	0.908	0.906	0.904
SMO(RBF)		**93.1415**	**0.008**	**0.932**	**0.931**	**0.931**
SVM(Linear)		90.2439	0.010	0.904	0.902	0.902
XGB		91.8947	0.009	0.918	0.906	0.910
J48		76.4228	0.025	0.781	0.764	0.770

The images created with the 300 fixed size were converted to 224x224 and 300x300 with the Lanczos interpolation method. Afterward, the same procedures were applied, and the results are given in Table 6.9, Table 6.10 and Table 6.11.

According to the results, using the GIST+HOG global descriptor shows better performance than using only the GIST or HOG global descriptor. Additionally, it was observed that the size of 300x300 gave the highest result. When the GIST+HOG global descriptor was used in 300x300 images with SMO(RBF), 93.14% accuracy was achieved.

Table 6.12. GIST results of the images created with the 2048 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	87.1637	0.014	0.870	0.872	0.871
SMO(RBF)		**90.1063**	**0.011**	**0.904**	**0.901**	**0.902**
SVM(Linear)		87.7503	0.013	0.877	0.878	0.876
XGB		89.6348	0.005	0.897	0.897	0.897
J48		75.6184	0.025	0.764	0.756	0.758
RF	300x300	89.7527	0.013	0.904	0.898	0.897
SMO(RBF)		**92.3439**	**0.009**	**0.925**	**0.923**	**0.923**
SVM(Linear)		88.9282	0.013	0.894	0.889	0.887
XGB		89.2815	0.005	0.892	0.901	0.892
J48		78.2097	0.025	0.793	0.782	0.784

Table 6.13. HOG results of the images created with the 2048 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	85.6302	0.018	0.869	0.856	0.850
SMO(RBF)		**91.2194**	**0.010**	**0.912**	**0.912**	**0.912**
SVM(Linear)		87.8681	0.014	0.881	0.879	0.879
XGB		89.6348	0.005	0.898	0.896	0.896
J48		69.0224	0.032	0.703	0.690	0.695
RF	300x300	86.9528	0.016	0.880	0.869	0.868
SMO(RBF)		**91.2839**	**0.010**	**0.913**	**0.913**	**0.912**
SVM(Linear)		87.9859	0.013	0.880	0.880	0.880
XGB		88.1036	0.006	0.882	0.884	0.883
J48		74.9117	0.028	0.758	0.749	0.749

Table 6.14. GIST+HOG results of the images created with the 2048 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	91.0483	0.011	0.913	0.910	0.910
SMO(RBF)		**94.6973**	**0.008**	**0.949**	**0.947**	**0.947**
SVM(Linear)		92.1084	0.008	0.921	0.921	0.921
XGB		90.8127	0.004	0.912	0.913	0.912
J48		74.2049	0.028	0.760	0.742	0.746
RF	300x300	91.9906	0.010	0.923	0.920	0.919
SMO(RBF)		**95.2886**	**0.005**	**0.953**	**0.953**	**0.953**
SVM(Linear)		91.7551	0.008	0.917	0.918	0.917
XGB		91.4016	0.004	0.912	0.914	0.913
J48		79.9764	0.022	0.805	0.800	0.800

According to Table 6.12, Table 6.13 and Table 6.14, the images created with the 2048 fixed size were converted to 224x224 and 300x300 through the Lanczos interpolation method. 300x300 gave the highest result. Besides, the highest accuracy was acquired with the GIST+HOG, GIST, and HOG global descriptors, respectively. When the GIST+HOG global descriptor was used, the highest accuracy was obtained as 95.2886% using the SMO (RBF) machine learning algorithm in the 300x300 image size. F1 score was found as SMO (RBF) using the same algorithm.

Table 6.15. GIST results of the images created with the 4096 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	92.8955	0.014	0.929	0.929	0.928
SMO(RBF)		**94.3089**	**0.007**	**0.944**	**0.943**	**0.943**
SVM(Linear)		92.2184	0.009	0.922	0.922	0.922
XGB		90.4761	0.009	0.907	0.904	0.904
J48		77.3519	0.026	0.783	0.774	0.775
RF	300x300	93.1475	0.09	0.931	0.931	0.932
SMO(RBF)		**94.6574**	**0.006**	**0.948**	**0.947**	**0.947**
SVM(Linear)		92.9112	0.08	0.928	0.929	0.928
XGB		92.9182	0.007	0.930	0.929	0.929
J48		79.5587	0.023	0.804	0.797	0.797

Table 6.16. HOG results of the images created with the 4096 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	87.921	0.016	0.886	0.879	0.876
SMO(RBF)		**91.5215**	**0.010**	**0.917**	**0.915**	**0.915**
SVM(Linear)		88.8502	0.013	0.891	0.889	0.889
XGB		88.8501	0.011	0.892	0.888	0.886
J48		66.3182	0.039	0.681	0.663	0.667
RF	300x300	88.6179	0.015	0.892	0.886	0.884
SMO(RBF)		**92.6829**	**0.008**	**0.927**	**0.927**	**0.926**
SVM(Linear)		88.8502	0.012	0.888	0.889	0.888
XGB		90.2439	0.009	0.904	0.902	0.901
J48		70.0887	0.717	0.708	0.710	0.710

Table 6.17. GIST+HOG results of the images created with the 4096 fixed-size method

Algorithm	Image Size	Accuracy	FPR	Precision	Recall	F1-Score
RF	224x224	91.4053	0.011	0.920	0.914	0.914
SMO(RBF)		**96.1672**	**0.004**	**0.962**	**0.962**	**0.961**
SVM(Linear)		93.7282	0.007	0.937	0.937	0.937
XGB		91.8699	0.004	0.922	0.925	0.924
J48		78.8618	0.021	0.801	0.789	0.793
RF	300x300	92.9152	0.009	0.933	0.929	0.929
SMO(RBF)		**96.3995**	**0.004**	**0.965**	**0.964**	**0.964**
SVM(Linear)		93.2636	0.007	0.932	0.933	0.932
XGB		93.4959	0.006	0.936	0.934	0.934
J48		80.3717	0.022	0.813	0.804	0.806

The images created with the 4096 fixed size were converted to 224x224 and 300x300 with the Lanczos interpolation method. Afterward, the same procedures were applied, and the results are given in Table 6.15, Table 6.16 and Table 6.17. According to the results, the accuracy increased when the GIST+HOG global descriptor was used. On the other hand, it was seen that 300x300 images were more successful than 224x224 images in respect of performance. While the highest accuracy was obtained using the SMO (RBF) algorithm in the 300x300 image size, the lowest accuracy was acquired with the J48 algorithm.

Images were created via the square root method and the 224, 300, 2048, 4096 fixed-width method, the results of which are discussed in this section. According to the results, the highest accuracy was obtained from the images created with the 4096 fixed image size, and then the highest result was acquired from the images created with the 2048 fixed image size.

The highest accuracy was achieved with SMO machine learning algorithm in all image configurations. The highest accuracy achieved in all configurations with the SMO algorithm are shown in Figure 6.3 and Figure 6.4.

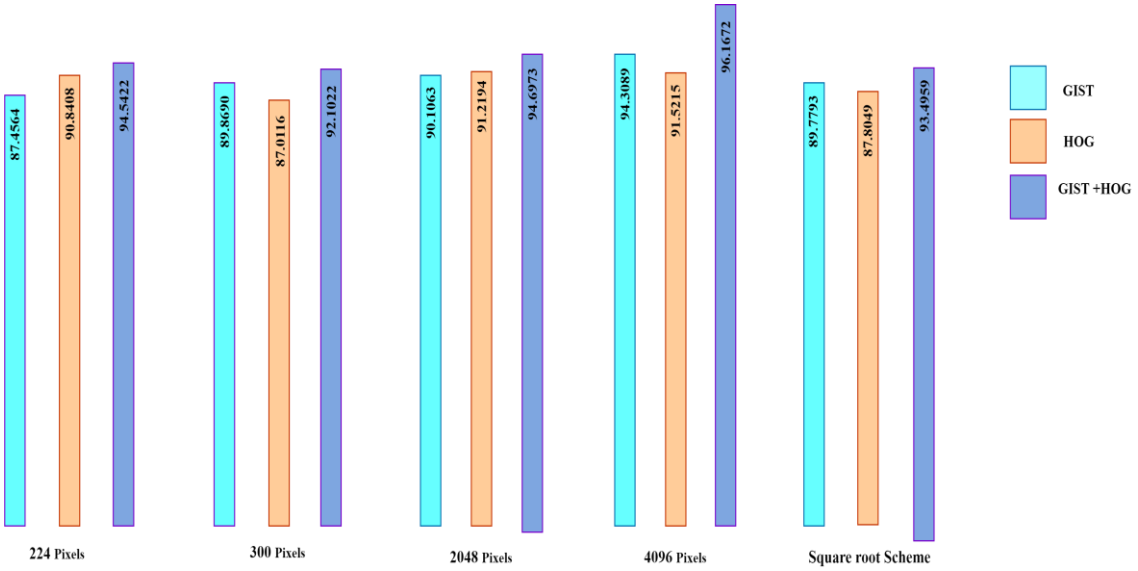


Figure 6.3. Highest Achieved Accuracy in 224x224 Image Size

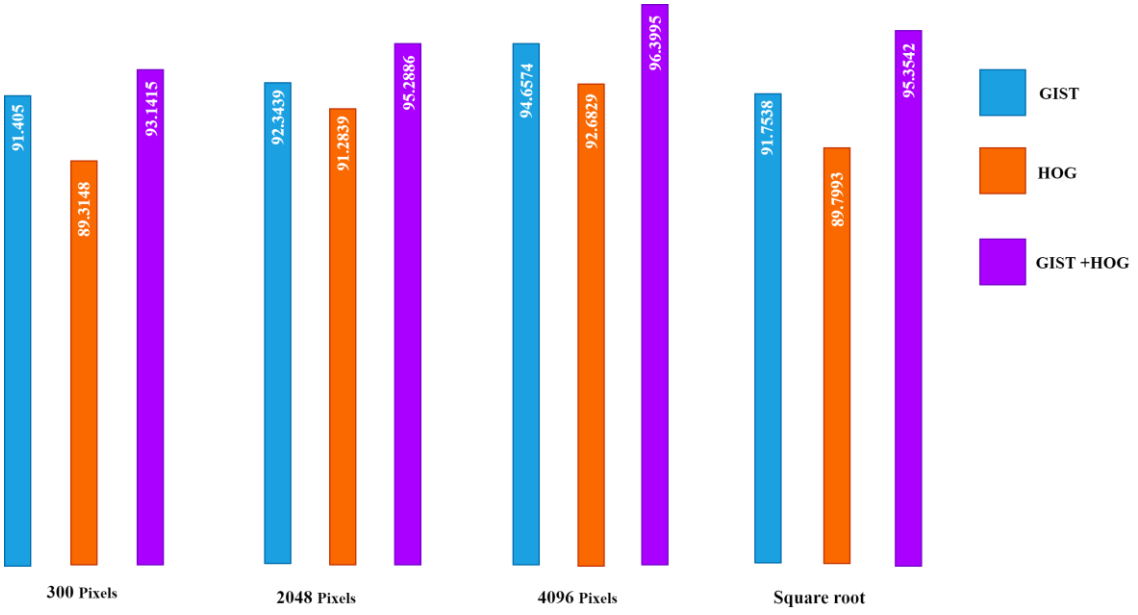


Figure 6.4. Highest Achieved Accuracy in 300x300 Image Size

Moreover, confusion matrix of our best configuration(4096 initial image size with SMO machine learning algorithm) is shown in Table 6.18.

Table 6.18. Confusion matrix of best achieved accuracy configuration

Dinwod	AutoRun	Legitimate	Allaple	BrowseFox	Amontize	VBA	Vilsel	Multiplug	InstallCore	Adposhel	
23	0	5	0	0	1	0	0	0	0	0	Dinwod
0	35	0	0	3	0	0	0	0	0	0	AutoRun
1	3	133	0	0	0	1	0	0	1	2	Legitimate
0	0	0	88	0	0	0	0	0	0	0	Allaple
0	1	0	0	37	0	0	0	0	0	0	BrowseFox
0	1	0	0	0	85	0	0	1	0	0	Amontize
0	0	1	0	0	0	99	0	0	0	0	VBA
0	0	1	0	0	0	0	76	0	0	1	Vilsel
0	2	3	0	1	1	0	0	91	0	0	MultiPlug
0	0	10	0	0	0	0	0	0	90	0	InstallCore
0	0	0	0	0	0	0	0	0	0	93	Adposhel

7. DISCUSSION

Within the framework of this thesis study, a memory analysis-based malware detection method has been developed and presented. PE files were executed on Windows Sandbox and memory dump operation of the target process was carried out. Memory dump data were visualized in 5 different sizes, 4096, 2048, 300, 224 and square root, without any loss of data. The visualized images were downscaled to 2 different sizes as 300x300 and 224x224 in order to shorten the training duration in the machine learning stage and because the sizes were quite relative. Feature extraction was done with the GIST and HOG global descriptors over the created images. Following this process, they were classified with 5 different machine learning algorithms.

Investigation of the effect of creating images in different sizes on accuracy was one of the purposes of this study. According to the results, it was seen that image creation with a fixed width increased the performance in malware detection. Since the patterns are not lost and more patterns are included in the images created with the 4096 fixed width, they contain more information. This can be understood from the fact that it has higher accuracy than other image sizes. It was observed that accuracy and F1-score decreased since patterns disappeared as the fixed-width size diminished. When an image is created with the square root method, the image size changes according to the size of the memory dump file. Image creation according to the size of the file is inadequate compared to image creation with a fixed width such as 4096 in terms of performance. The reason is the disappearance of the malicious code patterns when the image is created according to the size of the memory data. Additionally, when the image is created with both the square root method and the fixed width, downscaling is applied for 2 different sizes: 300x300 and 224x224. The accuracy of 300x300 images is higher than that of 224x224 images in malware detection. Since a 300x300 image contains more pixels, there is more data, therefore, more code patterns are possessed.

For feature extraction over the images, the GIST, HOG and GIST+HOG global descriptors were used, and it was observed how they affected accuracy. According to the results, it was seen that combination of the GIST and HOG global descriptors gave the highest accuracy, and the advantages of both the GIST and the HOG descriptors are benefited from this method. On the other hand, the GIST descriptor was more successful than the HOG descriptor due to the Gabor filter is used.

Another purpose of the study is to examine the success rate of different machine learning algorithms in malware detection. The machine learning algorithms addressed in the study were grouped as "Sequential Minimal Optimization (RBF), Support Vector Machine (Linear), Random Forest, XGBoost and J48". It was observed that the Sequential Minimal Optimization(RBF) algorithm was the algorithm with the highest accuracy, independently of the image size. The lowest accuracy was obtained from the J48 machine learning algorithm for all the image sizes and all the global descriptors that were used. With this study, promising results were achieved for malware detection studies.

8. CONCLUSION AND FUTURE WORK

In the first section of this study, we mentioned that previous studies conducted on malware detection has been inadequate and new approaches come into prominence. Together with new studies, malware authors implement some methods on the malware they develop in order to make malware detection difficult. The malware detection methods applied against these methods result in partial success.

There are 3 different methods such as static, dynamic and memory analysis in the literature for malware detection, and it has been seen in the literature that memory dump data are used against methods such as code obfuscation and packing, which make it difficult to detect malware.

According to the results, it was seen that accuracy increased when an image was created with a fixed size, the width increased and the number of the data in the image increased. Using the GIST and HOG global descriptors together was more successful than using only the GIST or HOG global descriptor. It was observed that SMO (RBF), one of the machine learning algorithms used, had higher performance than other algorithms.

After the memory dump data are converted to images, they will be modeled and classified with CNN in order to take this study further, and it is aimed to compare the classification success to the method we have proposed. Moreover, the proposed method will be repeated by using another dataset containing more PE files and more malware classes than the dataset we used in our study. In the proposed method, memory dump data were converted to images with different sizes and the prediction accuracy of image sizes in malware detection were compared, which is one of the purposes of this thesis. Accordingly, the prediction accuracy of the created images with different sizes will be compared by creating them in more different sizes. Furthermore, in the current study, the images were only created in RGB; they will be created and compared as grayscale images in the future study. Finally, success performances will be compared via the Bicubic interpolation method in addition to the Lanczos interpolation method we used in our study.

REFERENCES

[1] M. Egele, T. Scholte, E. Kirda, and C. Kruegel.. A survey on automated dynamic malware-analysis techniquesand tools. *ACM Comput. Surv*. 44, 2, 1–42, **2012**.

[2] G., Canbek, Klavye Dinleme ve Önleme Sistemleri Analiz, Tasarım ve Geliştirme, Master Thesis, *Gazi University, Institute of Science*, **2005**.

[3] Malware, https://www.av-test.org/en/statistics/malware/ (**26 January 2020**).

[4] O., Lysne, The Huawei and Snowden Questions: Can Electronic Equipment from Untrusted Vendors be Verified? Can an Untrusted Vendor Build Trust Into Electronic Equipment? (Vol. 4). Springer, **2018**.

[5] D., Loebenberger, Evolution! From Creeper to Storm, https://cosec.bit.uni-bonn.de/fileadmin/user_upload/teaching/07ws/malware/evolution_report.pdf (**28 January 2020**).

[6] R., Petrik, B., Arik, & J. M., Smith, POSTER: Towards Architecture and OS-Independent Malware Detection via Memory Forensics, **2018**.

[7] A., Klimburg, (Ed.). National cyber security framework manual. *NATO Cooperative Cyber Defense Center of Excellence*, **2012**.

[8] S., Spyridon, and D. Coss. The CIA strikes back: redefining confidentiality, integrity and availability in security, *Journal of Information System Security* 10.3, **2014**.

[9] The Cost Of Cybercrime, https://www.accenture.com/_acnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf#zoom=50 (**21 January 2020**).

[10] R., Rehman, G. C., Hazarika, & G., Chetia, Malware Threats and Mitigation Strategies: A Survey, *Journal of Theoretical and Applied Information Technology*, Vol. 29 No.2, **2013**.

[11] P., Vinod, R., Jaipur, V., Laxmi, & M., Gaur, Survey on Malware Detection Methods, *In Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*, pp-74-79, **2009**.

[12] S., Das, A. E., Dubrovsky, I., Korsunsky, A., Dhablania, & J. E. Gmuender, Just in time memory analysis for malware detection, *U.S. Patent Application No 15/783*,793, **2019**.

[13] K., Tam, A., Feizollah, N. B., Anuar, R., Salleh, & L., Cavallaro, The Evolution of Android Malware and Android Analysis Techniques, *ACM Computing Surveys*, 49(4), **2017**.

[14] Smartphone Market Share, http://www.idc.com/prodserv/smartphone-os-marketshare.jsp (**20 January 2020**).

[15] A., Kapratwar, F., Di Troia, & M. Stamp, Static and Dynamic Analysis of Android Malware, *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy 2017-January(January)*, 653–62, **2017**.

[16] M., Sewak, S. K., Sahay, & H., Rathore, Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection, *In 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 293-296, IEEE, **2018**.

[17] B., Gu, Y., Fang, P., Jia, L., Liu, L., Zhang, & M., Wang, A new static detection method of malicious document based on wavelet package analysis, *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), Adelaide, SA*, pp. 333–336, **2015**.

[18] J., Bickford, R., O'Hare, A., Baliga, V., Ganapathy, & L., Iftode, Rootkits on smart phones: attacks, implications and opportunities. *In Proceedings of the eleventh workshop on mobile computing systems & applications*, 49-54, 2010.

[19] A. Pektaş, Behavior based malicious software detection and classification, Master Thesis, Galatasaray University Institute of Science and Technology, Istanbul, **2012**.

[20] J., Aycock, Computer viruses and malware (Vol. 22). *Springer Science & Business Media*, **2006**.

[21] J. W., Oh, Recent Java exploitation trends and malware, *Presentation at Black Hat Las Vegas*, **2012**.

[22] Detailed Overview and Internals of PE File, https://nagareshwar.securityxploded.com/2013/10/15/overview-and-internals-of-pe-file/ (**23 January 2020**).

[23] Korkin, Igor, Ivan Nesterov. "Applying memory forensics to rootkit detection.", **2015**

[24] Y., Dai, H., Li, Y., Qian, & X. Lu, A malware classification method based on memory dump grayscale image. Digital Investigation, 27, 30-37, **2018**.

[25] S., Banin, Shalaginov, A., & Franke, K. Memory access patterns for malware detection, **2016**.

[26] D. S., Berman, A. L., Buczak, J. S., Chavis, & C. L., Corbett, A survey of deep learning methods for cyber security. *Information*, 10(4), 122, **2019**.

[27] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, A survey on heuristic malware detection techniques, *in IKT 2013 - 2013 5th Conference on Information and Knowledge Technology*, 113–120, **2013**.

[28] A., Sharma and S. K. Sahay. Evolution and detection of polymorphic and metamorphic malwares: A survey. *International Journal of Computer Applications*, 90(2):7–11, **2014**.

[29] L., Chen, Deep transfer learning for static malware classification. *arXiv preprint arXiv:1812.07606*, **2018**. [32]..

[30] J., Demme, M., Maycock, J., Schmitz, A., Tang, A., Waksman, S., Sethumadhavan, S., Stolfo, On the feasibility of online malware detection with performance counters, *In: International Symposium on Computer Architecture*, 559-570, 2013.

[31] M. A., Halim, A., Abdullah, & K. A. Z., Ariffin, Recurrent Neural Network for Malware Detection. *Int. J. Advance Soft Compu. Appl*, 11(1), **2019**.

[32] Y., Dai, H., Li, Y., Qian, R., Yang, & M., Zheng, SMASH: A Malware Detection Method Based on Multi-Feature Ensemble Learning. *IEEE Access*, 7, 112588-112597, **2019**.

[33] Z., Xu, S., Ray, P., Subramanyan, & S., Malik, Malware detection using machine learning based analysis of virtual memory access patterns. *In Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 169-174, IEEE, **2017**.

[34] K., Tam, N., Edwards, & L., Cavallaro, Detecting Android malware using memory image forensics. *In Engineering Secure Software and Systems (ESSoS) Doctoral Symposium*, **2015**.

[35] A. D., Schmidt, J. H., Clausen, A., Camtepe, & S., Albayrak, Detecting symbian os malware through static function call analysis. *In 2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, 15-22, IEEE, **2009**.

[36] Virusshare.com, http://virusshare.com/ (**26 January 2020**).

[37] G., Severi, T., Leek, & B., Dolan-Gavitt, Malrec: compact full-trace malware recording for retrospective deep analysis. *In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 3-23, Springer, Cham. **2018**.

[38] M., Brengel, & C., Rossow, Mem Scrimper: Time-and Space-Efficient Storage of Malware Sandbox Memory Dumps. *In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 24-45, Springer, Cham, **2018**.

[39] X., Wang, J., Zhang, A., Zhang, & J. Ren,. TKRD: Trusted kernel rootkit detection for cybersecurity of VMs based on machine learning and memory forensic analysis. *Mathematical Biosciences and Engineering*, 16(4), 2650-2667, **2019**.

[40] McAfee Labs Threats Report, https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf (**28 January 2020**).

[41] H. H., Pajouh, A., Dehghantanha, R., Khayami, & K. K. R., Choo, Intelligent OS X malware threat detection with code inspection. *Journal of Computer Virology and Hacking Techniques*, 14(3), 213-223, **2018**.

[42] O. E., David, & N. S. Netanyahu. Deepsign: Deep learning for automatic malware signature generation and classification. *In 2015 International Joint Conference on Neural Networks (IJCNN)*,1-8, IEEE, 2015.

[43] P. V., Shijo, & A., Salim Integrated static and dynamic analysis for malware detection. *Procedia Computer Science*, 46, 804-811, **2015**.

[44] I., Santos, J., Devesa, F., Brezo, J., Nieves, & P. G., Bringas, Opem: A static-dynamic approach for machine-learning-based malware detection. *In International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*, 271-280, Springer, Berlin, Heidelberg, **2013**.

[45] Y., Cheng, W., Fan, W., Huang, & J., An, A shellcode detection method based on full native api sequence and support vector machine. *In IOP Conference Series: Materials Science and Engineering*, Vol. 242, No. 1, p. 012124, IOP Publishing, 2017.

[46] S. Z. M., Shaid, & M. A., Maarof. Malware behavior image for malware variant identification. *In 2014 International Symposium on Biometrics and Security Technologies (ISBAST)*, 238-243, IEEE, 2014.

[47] M., Alazab, R., Layton, S., Venkataraman, & P., Watters, Malware detection based on structural and behavioural features of api calls, **2010**.

[48] M., Alazab, S., Venkatraman, & P., Watters, Digital forensic techniques for static analysis of NTFS images. *In Proceedings of ICIT2009, Fourth International Conference on Information Technology, IEEE Xplore*, **2009**.

[49] H., Erdal. Contribution of machine learning methods to the construction industry: *Prediction of compressive strength. Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, 21(3):109-114, **2015**.

[50] L., Nataraj, S., Karthikeyan, G., Jacob, & B. S., Manjunath, Malware images: visualization and automatic classification, *in Proceedings of the 8th international symposium on visualization for cyber security*, 1-7, **2011**.

[51] A., Souri, & R., Hosseini, A state-of-the-art survey of malware detection approaches using data mining techniques, *Human-centric Computing and Information Sciences*, vol. 8, no. 1, **2018**.

[52] O., Or-Meir, N., Nissim, Y., Elovici, & L., Rokach, Dynamic malware analysis in the modern era—A state of the art survey. *ACM Computing Surveys (CSUR)*, 52(5), 1-48, **2019**.

[53] I., You, & K., Yim, Malware obfuscation techniques: A brief survey. *In 2010 International conference on broadband, wireless computing, communication and applications*, 297-300, IEEE, **2010**.

[54] Polymorphic & Metamorphic Malware, https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf (**27 January 2020**).

[55] R., Sihwail, K., Omar, K. A., Zainol Ariffin, & S., Al Afghani, Malware detection approach based on artifacts in memory image and dynamic analysis. Applied Sciences, 9(18), 3680, **2019**.

[56] W., Wong, & M., Stamp, Hunting for metamorphic engines, *J. Comput. Virol.*, vol. 2, no. 3, pp. 211–229, 2006.

[57] C., Willems, T., Holz, & F., Freiling, Toward automated dynamic malware analysis using cwsandbox, *IEEE Security & Privacy*, 5.2: 32-39, **2007**.

[58] Chakraborty, Soumen., Malware Attack and Malware Analysis : A Research., 5(3): 268–72, **2019**.

[59] S., Krig, Interest point detector and feature descriptor survey. *In Computer vision metrics*, 187-246, Springer, Cham, **2016**.

[60] Y. Ren, Indexing and Searching for Similarities of Images with Structural Descriptors via Graph-cuttings Methods, *Doctoral dissertation*, **2014**.

[61] P., Dalka, D., Ellwart, G., Szwoch, K., Lisowski, P., Szczuko, & A., Czyżewski, Selection of visual descriptors for the purpose of multi-camera object re-identification. *In Feature Selection for Data and Pattern Recognition*, 263-303, Springer, Berlin, Heidelberg, **2015**.

[62] A. Oliva, and A. Torralba, Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3), 145-175, **2001**.

[63] N., Dalal, B., Triggs, Histograms of oriented gradients for human detection. *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, USA, **2005**.

[64] T., Moraes, P., Amorim, J., Silva, & H., Pedrini, 3D Lanczos Interpolation for Medical Volumes, *15th International Symposium on Computer Methods in Biomechanics and Biomedical Engineering*, **2018**.

[65] M. P. S., Parsania, & P. V., Virparia, A comparative analysis of image interpolation algorithms, *International Journal of Advanced Research in Computer and Communication Engineering*, 5.1, 29-34, **2016**.

[66] L., Breiman, Random forests, machine learning, *Kluwer Academic Publishers*, 45(1), 5-32, **2001**.

[67] T., Chen, & C., Guestrin, Xgboost: A scalable tree boosting system. *In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785-794, ACM, **2016**.

[68] R. E., Schapire, A brief introduction to boosting. *In Ijcai*, Vol. 99, 1401-1406, **1999**.

[69] J., Friedman, T., Hastie, & R. Tibshirani, Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors), *The annals of statistics*, 28(2), 337-407, **2000**.

[70] J. R., Quinlan, Improved Use of Continuous Attributes in C4.5., *Journal of artificial intelligence research*, 14, **2013**.

[71] J., Platt, Sequential minimal optimization: A fast algorithm for training support vector machines, **1998**.

[72] Tahillioglu E., Bozkir A. S., Aydos M., and Kara I., Makine Öğrenimi ve Bilgisayarlı Görü Yardımıyla Bellek Üzerinde Kötücül Yazılım Tespiti, International Conference on Access to Recent Advances in Engineering and Digitalization, ARACONF 2020, pp.151, **2020**.