Islamic University of Technology (IUT)

Department of Computer Science and Engineering (CSE)

An Efficient Feature Extraction Method For Static Malware Analysis Using PE Header Files

Authors

Onamika Hossain - 180041102 ,

Sadia Tasnim Dhruba - 180041210 &

Fabiha Jalal - 180041220

Supervisor

Dr. Md Moniruzzaman

Assistant Professor, Department of CSE

A thesis submitted to the Department of CSE

in partial fulfillment of the requirements for the degree of B.Sc.

Engineering in CSE

Academic Year: 2021-22

May - 2023

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Onamika Hossain, Sadia Tasnim dhruba and Fabiha Jalal under the supervision of Dr. Md Moniruzzaman, Assistant Professor of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Onamika Hossain

Student ID - 180041102

Sadia Tasnim Dhruba

Student ID - 180041210

Fabiha Jalal

Student ID - 180041220

An Efficient Feature Extraction Method For Static Malware Analysis Using PE Header Files

Approved by:

Dr. Md Moniruzzaman

Thesis Supervisor,

Assistant Professor

Department of Computer Science and Engineering

Islamic University of Technology (IUT)

Imtiaj Ahmed Chowdhury

Lecturer

Department of Computer Science and Engineering

Islamic University of Technology (IUT)

Acknowledgement

We would like to express our grateful appreciation for **Assistant Professor Dr. Md Moniruzzaman** and **Imtiaj Ahmed Chowdhury** , lecturer of Department of Computer Science & Engineering, IUT for being our adviser and mentor. Their motivation, suggestions and insights for this research have been invaluable. Without their support and proper guidance this research would never have been possible. Their valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to them.

Abstract

Detecting malware is crucial for safeguarding various devices, ranging from personal computers to large-scale systems,because computer systems continue to face serious security concerns from an increasing number of malware occurrences. Static analysis offers the ability to extract multiple file characteristics across various categories of information, eliminating the expenses and risks associated with dynamic analysis. By leveraging PE header information in machine learning classifiers, an efficient feature extraction method can be developed to minimize the time required for feature extraction and therefore improve the analysis process. The objective is to enhance extraction time while maintaining a reasonable balance with other parameters, such as execution time, accuracy, and f measure.

Contents

1 Introduction

This chapter covers the fundamentals of malware, its varieties, notable historical attacks, and malware analysis classification.

1.1 Overview

Finding out a piece of malware's functioning and purpose is done through malware analysis.This procedure will show you what kind of malicious software has invaded your network, the harm it may do, and—most importantly—how to get rid of it. In this research we covered several malware analysis techniques, their approaches as well as malware detection techniques, android malware etc.

The whole paper can be summarized to these following works proposed by us

- We provide a fresh and effective feature extraction technique which can reduce the feature extraction time using PE header files in static malware analysis.
- We create and publicize our own dataset using the malware executable files from malware bazaar.abuse website.
- We implement it in such a way that it would not affect the existing parameters such as execution time etc.

1.2 Malware

The term "malware," which stands for "malicious software," is any software or program created with the intention of damaging, exploiting, or being able to access computer systems, networks, or user data without authorization. It is produced with malice on the part of people or organizations referred to as "cybercriminals."

1.2.1 Types of Malware

Malware can use a variety of attack techniques and manifest in many ways, including:

1. **Virus:** Viruses are self-replicating computer programs that affix to healthy files and propagate throughout a computer system when the infected files are run.

2. **Worms:** Worms are self-replicating programs, similar to viruses, but they can propagate without a host file. Instead, they take advantage of holes in computer networks and spread via network connections.

3. **Trojans:** Trojans are malicious programs that trick users into installing or running them by imitating legal software. Once within a system, they have the ability to build backdoors, steal confidential data, or grant unauthorised access to outside adversaries.

4. **Ransomware:** The files of the victim are encrypted by ransomware, making them unobtainable, and the victim is then asked to pay a ransom in exchange for a key for decryption. Usually, infected websites or malicious email attachments are how ransomware spreads.

5. **Spyware:** Without user knowledge or permission, spyware tracks and gathers data about their activity. It has the ability to monitor keystrokes, seize passwords, record surfing patterns, and send the stolen data to unauthorized parties.

6. **Adware:** Adware is created to display a lot of unwelcome and excessive advertising, frequently in the form of pop-up windows. Although not always malevolent, adware can harm system efficiency and jeopardize user privacy.

7. **Botnets:** Botnets are hacked computer networks that are frequently managed by a central Command-&-Control (C&C) server. This compromised machines, referred to as "bots" or "zombies," can be used to conduct coordinated attacks, send unsolicited emails, or mine cryptocurrencies.

Data loss, financial losses, identity theft, system breakdowns, and unauthorized access to private data are just a few of the negative repercussions that malware can have. Up-to-date antivirus software, safe browsing practices, avoiding downloading

files from dubious sources, and routine operating system and application updates are all necessary for malware protection.

1.2.2 Example of significant Malware Attacks

Malware analysis focuses on gaining an understanding of the malware's functionality from the data that has been extracted, how the malware got onto the computer & how to prevent future attacks that are similar to this one.Following are some significant malware attacks-

1. **Emotet:** For many years, Emotet has been a sophisticated financial Trojan. Virus-ridden documents and spam emails are the main methods of distribution. Emotet has the ability to download and install more malware onto affected PCs as well as steal sensitive data, including banking credentials.

2. **WannaCry:** In 2017, the ransomware outbreak known as WannaCry was widely publicized. It quickly moved across networks by taking advantage of a Windows operating system flaw. After infecting a machine, WannaCry encrypted the files on that system and demanded a Bitcoin payment to decrypt them.

3. **NotPetya:** In 2017, businesses and organizations became the target of another ransomware outbreak called NotPetya. It employed several methods to spread among networks and was largely propagated by a corrupt software update. Significant inconvenience and monetary damages were brought on by NotPetya.

4. **Mirai:** The virus attack known as Mirai specifically targeted Internet of Things (IoT) devices. By taking advantage of default or weak credentials, it attacked susceptible devices like routers and webcams. Mirai developed a botnet for the purpose of carrying out widespread distributed denial-of-service (DDoS) attacks.

5. **Stuxnet:** In 2010, researchers found the highly advanced worm Stuxnet. Systems utilized in nuclear reactors and other industrial control systems were

specifically targeted. PLCs were exploited by Stuxnet in order to destroy centrifuges used in uranium enrichment.

1.2.3 Types of Malware Analysis

Next three sections are the various types of malware analysis.Static analysis, dynamic analysis and lastly hybrid analysis. Each malware analysis method has benefits and drawbacks.

1.2.4 Static Analysis

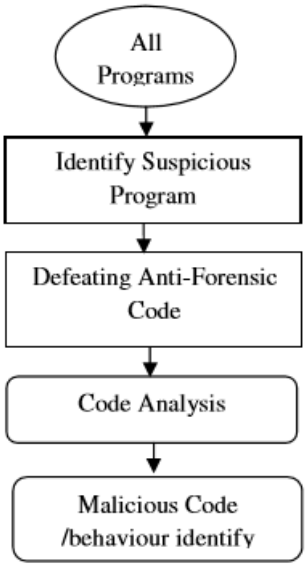


Figure 1: static analysis

Here **??** shows the steps in malware static analysis. In this figure, the suspicious programs have to go through anti-forensic code analysis along with malicious code behaviour analysis. For application analysis, it is necessary to look over all of the manifest file's information (which includes Meta data for things like the requested permission, services, broadcast receivers, content provider, activity, and SDK version), as well as to look at the byte code and extract all of the objects and methods.There are certain anti-forensic methods that make it difficult for analysts to locate questionable code. Defeating Anti-Forensic code is an intermediate step

in static analysis process as shown in Figu re1. The method for anti-forensics are:

- Code Obfuscation
- String Encapsulation
- Environment verification
- Self-define communication protocol
- Sensitive data access

1.2.5 Dynamic Analysis

Typically, we use dynamic analysis to examine how an application behaves. Here, we keep track of the file system and network activity as well as incoming SMS calls, the loading of extra Dex or native code during execution, and file system and network movement.

For analysing malware there are lots of framework are available now a days such as :

Taintdroid : Taintdroids is an Android solution that employs various granularity taint tracking.First, TaintDroid recognizes all data from sensitive sources as sensitive data and automatically taints that data. Taint droid records the labels of the data as it leaves the system or when it is transported via the network, identifying who transmitted the data.

DroidScope : DroidScope exports three-tiered APIs that correspond to the hardware, operating system, and Dalvik Virtual Machine layers of an Android device.

Profile Droid : There are four levels in the Profile Droid framework analysis portion.User interaction, operating system, and network are static specifications.The first and second parts of each of the four levels are monitoring and profiling.

1.2.6 Hybrid Analysis

Hybrid analysis refers to the fusion of static and dynamic analysis. Malware identification is made easier with the use of hybrid analysis, and its accuracy also rises.

Mobile Sandbox : Here, static analysis is performed on an APK file.Therefore, it must first scan the manifest.xml for suspicious code and run anti-virus, user permission, and manifest.xml scans. Additionally, an emulator is utilized to run the code and see how it behaves during dynamic analysis.

Andrubis :

In this method first static analysis should be done and then that result is used for dynamic analysis.In static analysis Andrubis is concentrated of android manifest.xml and bytecode and in dynamic analysis Stimulation, taint tracing,method tracing, system level analysis should be followed.

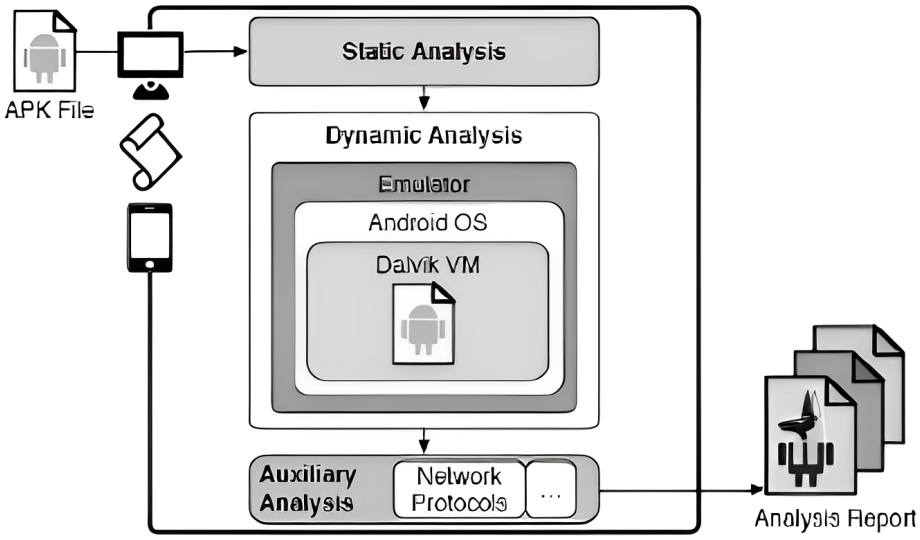


Figure 2: Andrubis

1.3 Portable Executable File

Portable Executable file headers are one of the key components in malware analysis, particularly in static analysis. The reason PE header files were chosen is that they are virtually universally present in Windows OS files and executables.

1.3.1 Portable Executables (PE)

What is PE :

The executable file format for Windows is called Portable Executable, or PE. This format is used by many binary files, including as core dumps, DLLs, and object code. It is merely a data structure that houses details about the file. It is divided into two sections: a pe header and a section.

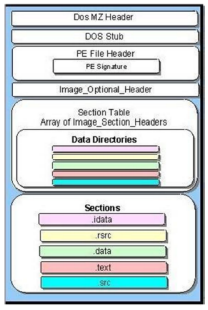


Figure 3: Sections of PE Header

PE headers : This contains all the data that the operating system needs to run the executable file. It includes details such as necessary libraries, an import table, code, and metadata, among other things. All of these are broken down into sections, some of which are listed below.

- **Mz-dos:** The supplied type is specified as being an executable file by Mz Dos.The magic number value is range from 0x54AD in all MS-DOS-acceptable exe files.
- **Dos stub**:Used to test compatibility, it does nothing more than report "program cannot run in dos."
- **PE file header:** he PE file header, which includes the file's md5 and sha256 signatures, machine information, the number of sections, and the size of the optional header.
- **Imageheader:**Information about the exe file, including entry points and components, is contained in this section.

- **Section Table:** Provides details on the loading of executable files into memory. Containers, virtual size sections, raw data size, a link to raw data, and section attributes.

- **data:** This part contains the initialized data that will be used by the application later on during execution, such as strings or constants.

- **Rdata:** Consists of a debug directory that houses several forms of file-related debug information, including its size, type, and location.

- **.idata:** Components that will be imported from DLLs are contained in the section idata that contains the import table.

- **.edata:** The information and functions that will be transferred to DLLS are also included in the export table for sections.

- **.rsrc:** Resources, including photos and other assets, are kept in this area (rsrc).

- **.bss:**The uninitialized data for the application is represented by this.

2 Literature Review

In this section first we discussed about the explored area (such as : malware detection, android malware, ML-DL based malware, dynamic and static analysis etc) so far and then our related works which contains our main working direction

2.1 Explored Areas

The concept of cyber security[1] has come to attention after the massive growth of data transfer online. The motive of Cyber security using ML/DL is not lightly constrained to ensuring safeness of the data but also to maintaining the integrity of the legislative society.

Over the years, numerous cyber crimes have been reported. Preventive measures to these have not been derived drastically. This study, particularly, makes a review on cyber crimes and their effects on the socio-economy structure. Moreover, probable technological threats due to the incapabilities of the derived preventives have been presented.A major portion of this literature concentrates on defining cyber security and terrorism. The motivation behind cyber terrorism and the trends of cyber security are marked particularly. Social media, a potential gateway to cyberbullying, has been reviewed. Several case studies, presented in this paper, analyses the incident and the after-effects. Solutions to cybercrime have been proposed based on the security flaws extracted from the case studies. However, the main motivation of this paper was to establish effective solutions.The solutions would be–firewall, data authentication, malware, strong encryption algorithms, deep learning-based algorithms to provide security etc.This paper opens up the scope of exploration is cyber security fields, its components and prevention of cyber terrorism to secure a much more protected cyberspace.But no analytical formulation or intensive studies have been incorporated in this paper.

This document emphasizes banking malware[2] and the anti-analysis trends in

detecting and classifying banking malwares. The frequency and complexity of banking malware has increased since the emergence of the Zeus malware kit in 2007. This analysis is presented in the manuscript, which also looks at patterns in the development of anti-analysis tools. It includes various malware families e.g. Zeus V2, Citadel, Vawtrak, Dridex, Rovnix, Neverquest2.In this paper, first the malware families , their behaviors are discussed. Then after analyzing these families the challenges are included in the latter part of the paper.Most Zeus samples exit by using the exitProcess API after failing the hardware locking check.To get past this anti-analysis measure, a solution was created.A programmed debugger was utilized to execute the malware.A breakpoint was established on the ExitProcess API following the loading of the malware sample. Automatic analysis tools are required in the Citadel for malware. Citadel does not establish a connection to its C2 server when it determines that it is executing in a VM or a sandbox.Instead, it tries to connect to a C2 name that was chosen at random.The analysis system will receive a false negative when the connection to the random C2 name fails.A malware analyst can conclude from this that the sample is no longer active and that no more research is necessary.Similarly the vawtrak and dridex malware also react differently to these analysis methods.But Dridex makes this kind of analysis harder, by implementing its own GetProcAddress. Therefore, this paper depicts the banking malware families and their analogy when applied analysis methods.

2.1.1 Mobile malware

The concept of selective adversarial learning[3] for mobile malware was proposed as it has also been demonstrated that overtraining with samples might cause the analysis to perform less well. That's why selective adversarial retraining .Adversarial samples are created by carefully introducing minor disruptions to valid inputs in order to deceive the classifier.Existing research chooses the adversarial sample in a randomized manner.In this document two approach are proposed-

- Depending on the malware's cluster center's proximity to the location.
- Using probability obtained by kernel-based earning (KBL), with a 6% in-

crease in detection In this paper, selective sampling has proven to be quite helpful in developing classifiers that can handle imbalance data.

In the malware detection field, it has yet to be used for adversarial retraining. This study fills that need.Nevertheless, hostile samples put some machine learning techniques, such as neural networks, at risk. Here the proposed methodologies are-

Androguard tool- for extracting static features

Monkey tool-for dynamic features

meu(x)=[meu0(x),meu1(x)] The first one is the probability of a sample benign, and the second one is the probability of malware. This equation produces an output that differs from the initial forecast and corresponds to the attacker's objective if the delta(x) perturbation is included.The main goal was to design the adversarial sample in such a way that the model would recognize it as benign. Crafting has its own set of limitations. They only permitted features to be added to feature vectors, for example. It's also possible to add features to the manifest.xml file without changing the code. Finally, no changes to dynamic features are permitted.The two approaches were then thoroughly discussed. Based on distance from cluster center.

The distance in this case is calculated using the euclidean distance. Malware samples are taken from the DREBIN collection. then use randomized and selective sampling to assess adversarial training. 98.3% accuracy was reached when a DNN was utilized with pure data and 10 fold stratification cross validation. The accuracy is highest when samples from 58 to 65 percent are used for retraining. With 65% adversarial samples, the KBL approach increases accuracy by 6% in comparison to random selection. This work measures and evaluates the impact of this on the effectiveness of DL models for hostile retraining.The disadvantages of this approach are Accuracy decreases when the data sample is greater than 65 percent, while accuracy is maximum between 58 and 65 percent.

The prime motive of [4],the goal of this study is to provide a real-time, useful dataset for malware identification. The research-based datasets that are currently available are ineffective because-

- They lack diverse categorization of malware apps
- The datasets are mostly emulated and are assumed to be different than the obtained datasets when malware is actually installed in the devices.

The authors have developed a new dataset namely CICAndMal2017 that compensates for the above-mentioned shortcomings. The second motive of this paper is to analyze and classify the benign and malware apps based on the network traffic. Finally, the proposed work is capable of malware binary detection, malware category classification, and malware family characterization with an 85% average accuracy and a recall of 88% for three ML classifiers RF, KNN, and DT used in this paper.Sample datasets for this research have been collected from different open sources. To categorize the malware into different categories, the authors have used different actions to trigger the malware. Thus the authors have extracted specific feature sets for organized classification.In the end, the authors have concluded that the network traffic feature is useful for detection but not good for classification if solely used.

2.1.2 Deep learning Based Detection

The paper[5] proposed to use a deep learning model to develop a classifier based on several characteristics of Android applications so that malicious Android apps may be appropriately distinguished from good Android apps.Here, a deep learning method is used to meet the critical requirement for malware detection using an Android malware characterisation and identification technique. There have been recent research efforts on static analysis, dynamic analysis, and also machine learning methods like SVM, which have been proven to be viable solutions in comparison to the default permission based approach, but DBN model can identify malware in a more accurate manner when compared to the SVM-based solution. Here, two types of features—API function calls and permissions—are employed for the weighted deep learning technique to malware detection.The dataset uses 1400 safe applications that were downloaded from Google Play.To recognize risky permission combinations in feature sets, nine grouping criteria are included.This

method employs just 237 characteristics and yields accuracy of above 90%. As a result, this strategy requires substantially less preprocessing time. Similar approaches can be observed in [6].

The paper[7] is suggested Maldozer is an autonomous malware detection tool that uses deep learning to classify data in a particular order. [8].MalDozer has great accuracy in identifying malware and correctly classifying it with its associated families on a given dataset. The F1-Score it receives, which ranges from 96% to 99%, is amazing and demonstrates how well it can identify malware samples. MalDozer also has a low false positive rate, which ranges from 0.06% to 2%, demonstrating its capacity to lessen the range of innocent applications that are mistakenly classified as malicious softwares. It takes a raw sequence of API method calls as input.MalDozer exhibits strong performance in both the tasks when applied to the same datasets. During the detection task, it obtains an impressive F1-score evaluation result ranging from 96% to 99%, indicating its high accuracy in identifying malware samples. Additionally, in the family attribution task, MalDozer maintains its effectiveness by correctly attributing Android malware to their actual families, achieving an F1-score between 96% and 98%. This showcases MalDozer's robust capabilities in both detecting malware and accurately classifying them into their respective families. Here they propose an automatic feature extraction technique during the training using method embedding, where the input is the raw sequence of API method calls, extracted from DEX assembly.The detection technique employed by the system is sample-based, which allows it to automatically recognize patterns during the training phase of newly encountered malware. This means that the system learns from a collection of malware samples and develops the ability to identify similar patterns or characteristics in previously unseen malware instances. By analyzing these patterns, the system can effectively detect and classify new malware samples without relying on predefined rules or signatures. This app pushes toward portable detection solutions.

The paper[9] introduces a framework for malware detection that makes use of permission and applies linear regression as a detection technique. For permission-

based Android malware detection, two classifiers are introduced and assessed using four different datasets. The importance of application permissions in strengthening Android security is emphasized by the study. To assess the rights asked by applications and determine whether or not they are malware, machine learning models are used. To achieve accurate findings, it is important to reduce the regression model's mean square error. The bagging method, which is frequently employed in ensemble learning, is the basis for the classification models in this study. The datasets used are those from AMD, M0Droid, Arslan's, and Lopez, and the results provided are the average of 10-fold cross-validation.The results show that classifiers built on linear regression typically produce good results. Additionally, data belonging to the same class in permission-based malware detection share a linear subspace and can be described by a linear equation. By giving the regression coefficients random values, the study investigates several regression models. Comparative analysis is done between the suggested rule-based classifiers and well-known classification algorithms as KNN, NB, SVM, and DT. Both strategies perform better than NB and KNN, with the key benefit being the ease of use and simplicity of the classifiers based on multiple linear regression models. The study also includes bagging techniques, although these do not produce fruitful outcomes or create sophisticated search methodologies like hybrid or heuristic techniques.

The Component Traversal approach, which they propose in this work[10], may automatically execute the code routines of each supplied Android application (app) to the fullest extent possible. They create weighted directed graphs based on the extracted Linux kernel system calls and then implement a deep learning framework based on the graph based features for the detection of recently discovered Android malware. To compare alternative malware detection strategies, a thorough experimental investigation using an actual sample collection from the Comodo Cloud Security Center is conducted. Promising trial outcomes show that their suggested approach outperforms current Android malware detection alternatives. A commercial Android anti-malware program has also been integrated with

the technology, Deep4MalDroid.They build the weighted directed graphs based on the extracted system calls and then employ a deep learning framework for the detection of recently discovered Android malware.To compare alternative malware detection strategies, a thorough experimental investigation using an actual [11] sample collection from the Comodo Cloud Security Center is conducted. Promising trial outcomes show that the suggested approach outperforms current Android malware detection alternatives. A commercial Android anti-malware program has also adopted the created Deep4MalDroid technology.

2.1.3 Malware classification

Understanding the malware patterns and the families of malware can help in detection. Due to this, Here [12] is the thorough assessment on the cutting-edge methods for detecting, identifying, and classifying Android malware families. the three components of the literature: the type of analysis, the features, and the methodology and approaches. Researchers can concentrate more on dangerous families by identifying malware families.Therefore, by recognizing the associated family and capturing its impact on users, detecting the risky categories can aid detection systems in discovering more malware.Describe a novel taxonomy that groups every connected work in familial classification according to the types of studies, features, and approaches that have been employed.

In analysis based malware, there are 3 types. static ,dynamic and hybrid and all of them have both advantages and disadvantages. then in technique based malware there are two types :model based and analysis based. both of them have some other types too and many researchers work according to those types. About feature there are two types static and dynamic which are feature based. there is some limitations and challenges for identifying malware families and also here discussed about the future direction according to Android malware familial detection, classification, and categorization.

2.1.4 Malware Visualization

This research[13] they highlight visualizing malware behavior and its potential benefit for malware classification. This research shows that malware behavior visualization can be used as a way to identify malware variants with high accuracy.The proposed techniques are-

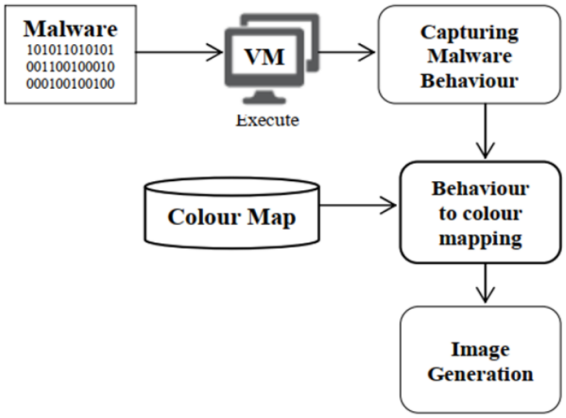


Figure 4: Process in malware behaviour visualization

To capture malware behavior first obtain the behavior data, then run API call monitoring through VM for all malware samples , then user mode API [13] reflects the exact nature of a particular malware. Then the color mapping from behavior is performed. 9 shows the process of malware behaviour visualization and its steps.

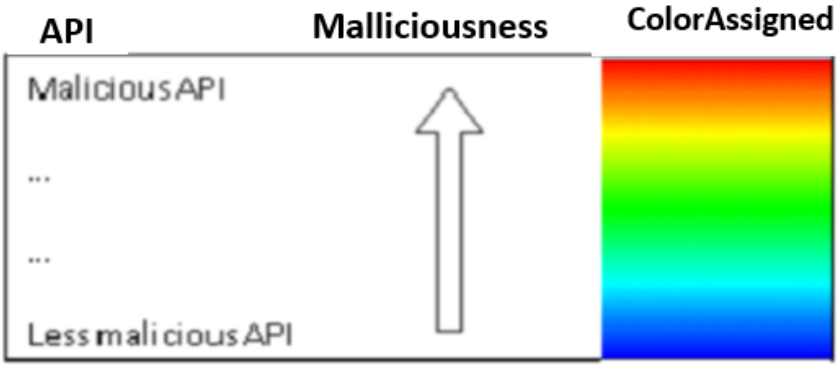


Figure 5: API to color mapping

Utilizing the RGB color model, the experiment's colors are depicted. Red (1,0,0), yellow (1,1,0), green (0,1,0), cyan (0,1,1), and blue (0,0,1) are the first four colors in the model. [13].Then the image is generated.After API color mapping,64 x 4 pixel image will be painted to the behavior image. The malware images are painted top-to-bottom. The observation of the experiment is-they notice that malware from the same family tends to have a recognizable pattern in them.There are also family with minor as well as major changes between behavior image of its samples.

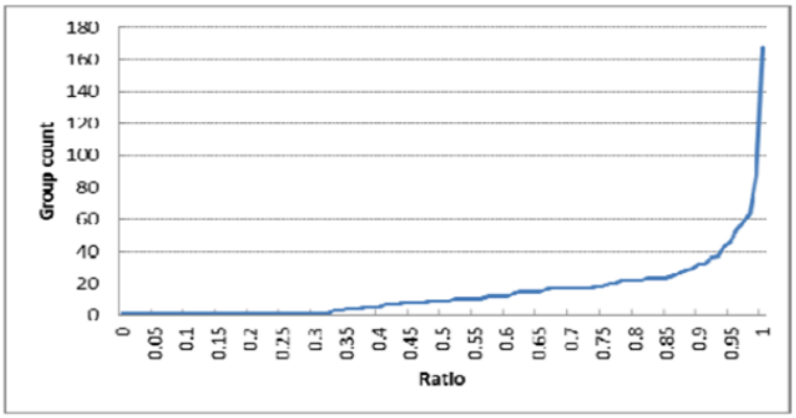


Figure 6: number of malware groups identified in similar groups

In the experiment the classify malware sample with behavior image.1,101 distinct samples from 12 distinct malware families were used. Images depicting how malware behaves will be compared to one another, and those with a high degree of resemblance are going to be placed together.

The accuracy of malware classification using malware behavior is 99.33%. That's why we use behavior images for identifying malware variants. But using behavior images we can not detect any malware.Besides, major and minor changes did not seem to hide the overall pattern and the similarity between variants of the family. In order to classify images of 25 prominent malware categories regardless of class balance, this paper's citation of awan2021image advocated spatially attention and CNN using deep learning. Performance was assessed using the following metrics: recall, specificity, then precision, and F1 score on the Malimg benchmarking

dataset, where our suggested model with class balance achieved 97.42%, 97.95%, 97.33%, 97.11%, and 97.32%. They offered a straightforward answer. Offer a state-of-the-art solution with cutting-edge methodology for image-based malware detection. They suggested a transfer learning-based architecture that would classify malware from various families using class weighting rather than class balancing methods.They produced malware binaries as images. The binaries were changed to 8 bit vectors for this.When converting vectors to grey scale, each pixel of the resulting image stands for an intensity.In the proposed architecture they have 3 parts: a) Transferred learning model based on Image Net (VGG 19). It performs better on malware dataset. b) CNN model enhanced by attention. Generated attention used dynamic spatial convolution. It is better for image processing and vision taste. c) Spatial attention generated by normalized vector and 2D convolution layers.

The paper[14] introduces MDMC, a deep learning-based markov image technique for byte-level malware classification. They took three actions. malware binaries were transformed into Markov images based on a byte transfer likelihood matrix. For Markov image classification, deep CNN is utilized. Two datasets, Drebin (97.364%) and Microsoft (average accuracy: 99.264%), are used in the experiments.For Markov image generation, They considered malware binary as byte stream which was represented by stochastic process.The transfer probabilities of each state were used to create a Markov transfer probability matrix. Using some formulas, probability is estimated using some frequency. Probability matrix is generated by some formulas and Markov image was generated using this matrix. In micro dataset, for clear display, they used experimental data for the first 60 epoch. In MDMC training accuracy converged faster. Testing accuracy is almost 98 to 100%. In Drebin dataset, testing accuracy is 94 to 98% for MDMC.

In this research[15] The authors of this study concentrated on malware analysis utilizing the static and 18 dynamic methods, which will enable us to assess damage, identify signs of infiltration, and estimate the level of sophistication of a malware intrusion. Analysis can be divided into two categories: basic and

advanced.Additional analyses, including the fundamental static analysis, are performed in advance static analysis. Additionally, advanced dynamic analysis performs the same functions as static analysis. After that, we learn about many tools for both fundamental and advanced static and dynamic analysis. Several computer applications were utilized for research purposes in this article to analyze malware and learn more about it. The malware QQQ.exe is the subject of tests employing both static analysis and advanced dynamic analysis techniques.We started by establishing whether the software was malware or not while evaluating the malware implementing a sophisticated static analysis approach, which includes basic static analysis alongside a few additional improvements. This method allowed us to identify the malware's generation date and time as well as the portable executable's headers. Based on these findings, combining advance static malware detection and extensive dynamic malware analysis may give a more fascinating and vivid picture of the characteristics of the malware QQQ.exe.[15]

2.1.5 Dynamic Analysis

The author of [16] includes different malware identification methods using dynamic analysis and pattern matching techniques for identification of malware samples.In order to discover malware variants and new types of malware families, this study proposes clustering of unknown malware samples using a two-layered Malware Variant Identification via Incremental Clustering (MVIIC) technique. Finally, this study presents a hybrid strategy, in which Yara scanning is used to eliminate existing malware, followed by clustering, which works in tandem to identify new malware variants. Malware identification is carried out at two stages in this study. The first layer rejects samples of well-known malware families using the previously created Yara rules. In the second layer, a brand-new incremental clustering approach is used. The optimal cluster distribution may be determined or approximated using incremental clustering, which can also identify tiny size clusters. When new data (unidentified malware samples) become available, this is crucial for malware clustering since such data provide distinct and tiny clusters.New

malware families and variations can then be found by randomly selecting samples from the clusters.

Algorithm 2 MVIIC Algorithm

```
Require: Malware data set
  Submit malware data set to Cuckoo sandbox
  while Sandbox processing samples do
    Wait
  end while
  Extract features from Cuckoo report
  Encode features to vector format
  Run incremental clustering program
  while Clustering in progress do
    Wait
  end while
  Read clustering metrics
```

Figure 7: MVIIC algorithm

According to this study, clustering can offer a higher level of efficiency than Yara rules and is resilient to subtle alterations brought about by malware strains.This study suggests a hybrid strategy that first uses Yara scanning to get rid of existing malware, then clusters working together to identify new malware variants. The outcomes are assessed using the F1 score and V-Measure clustering metrics.

The paper proposes[17] Dl-droid to detect malicious android applications by dynamic analysis which is performed on real devices. For dynamic feature it can achieve 97.8% detection rate and for both static and dynamic feature it can achieve upto 99.6% detection rate. It employs a state-based method to input creation. The authors used DL-Droid to examine the effectiveness of the stateful input generation technique using the state-of-the-art stateless (random-based) input generation as a baseline for comparison.

The paper proposes[18] an effective dynamic framework named EnDroid [**?**]is a powerful dynamic framework that enables extremely accurate detection of malware based on several kinds of dynamic behavior data.EnDroid employs a feature selection method to extract important behavior features while discarding unimportant features. When paired with program analysis approaches, ML methods can

instinctively infer behavior characteristics of applications. Malicious programs use a number of deformation technologies to circumvent static analysis, which puts those techniques to the test. In this work, the authors concentrate on runtime application monitoring and profiling for a variety of behavioral aspects, as well as the implementation of very efficient malware detection. The lowest level behaviors of the component are described via system calls. To extract crucial behavioral features from noisy, pointless, and redundant features, the chi-square feature selection technique is used.EnDroid has the ability to distinguish between harmful and useful applications.To test the efficacy of EnDroid, they use a variety of machine learning algorithms, and they discover that stacking based on already-used ensemble approaches yields the highest performance. In this study, they demonstrate the efficiency of EnDroid for identifying and categorizing Android malware families. They also contrast the effectiveness of EnDroid's detection with that of the cutting-edge dynamic analytic tool, Maline.[19]

DroidDeepCNN- employing a deep CNN to achieve malware classification from the raw opcode sequence of disassembled programs.[19]

Adagio- outlines a detection approach built on function call graphs, correlates these graphs with vector space, and employs graph kernels to capture structural relationships.

Revealdroid- lightweight analysis that can extract API-based features, such as the resolution of reflections calls and function calls generated by native binaries, is presented.

MKLDroid- use a graph kernel to extract structural and contextual data from application PRGs, and gives a malicious code address for interpreting malicious behavior.

These techniques use analysis of static code to derive features for malware detection based on machine learning. Yet, dynamic code loading needs runtime data, which is difficult to gather by purely static analysis.

It detects 96.56 percent of malware using stacking on the M1 dataset and gener-

ates 1.85 percent false positives. As some malicious actions cannot be generated in dynamic analysis environment, they discover that the majority of erroneous negatives are brought on by a lack of dynamic behavior features retrieved.Due to issues with missing essential resources, UI activities, or libraries, the dynamical detection platform MonkeyRunner + DroidBox may occasionally be unable to activate dangerous behaviors of malware.

According to the experimental findings, stacking produces the best categorization performance. EnDroid does not support network-based malware because it only accepts ip and port as network operating features. They are unable to put a stop to this.

Traditional malware detection techniques are no longer effective since modern malware hides from scanning engines by using obfuscation techniques (encryption, tokenization, data masking, etc.).De-Lady[20] is a malware detection framework for Android that is based on deep learning and dynamic analysis. It employs a durable obfuscation strategy.It makes advantage of the behavioral traits from a dynamic examination of a program running in an emulated setting. The deep learning model is trained using the logs that record an app's behavioral traits, and it is then evaluated.

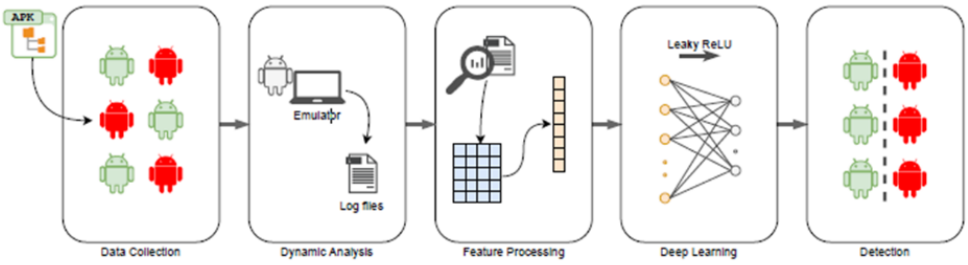


Figure 8: Architecture of the proposed approach

They first run the Android program Package (APK) file for the aforementioned program in an emulated environment before classifying it as malicious or benign. A feature vector representation of the program is produced from the parsed and

preprocessed logs produced by the emulator. The deep learning model is trained using this feature vector. The testing dataset is used to compare the trained model to for malware detection. Reconstructing RPC & IPC interactions and android-specific objects is done using the CopperDroid emulator.For creating user interaction to operate the emulator, monkey tools are employed. System call monitoring, binder analysis, network traffic capture, and composite behavior interaction were all clearly mentioned in the logs that were collected. Dynamic analysis logs are extracted in JSON format. The DNN is then fed standardized feature vectors.For all of the hidden layers, they utilized the Leaky ReLU activation mechanism. Leaky ReLU activation functions allow for a little positive slope on a negative argument while forwarding the positive portion of the argument.De-Lady is compared against 13,533 applications in the banking, gaming, media player, utilities, etc. categories.The rate of detection is 98.08%.

Method	Accuracy	F-measure	Error Rate
KNN	94.48	79.87	5.52
Naive Bayes	91.60	65.40	8.40
SVM (linear)	95.77	85.16	4.23
Decision Tree	97.09	90.26	2.91
Random Forest	97.01	89.6	2.99
XGBoost	96.98	89.53	3.02
De-LADY	98.08	98.84	1.92

Figure 9: Comparison of De-lady with popular machine lerning classifier

2.1.6 Static analysis

This document[21] emphasizes FamDroid.Using static analysis techniques, FamDroid is a learning-based categorization approach for Android malware. It can get a 99.12% F1-Score and properly categorize 98.92 % of malware samples into respective families.They experimented with the 5560 malicious Android applications in the Drebin Dataset.The writers are from Beijing University of Posts and

Telecommunications' School of Electronic Engineering and Beijing KeyLaboratory of Work Safety Intelligent Monitoring. According to McAfee's mobile threat report, there were 30 million mobile malware infections in the first quarter of 2019. The effect of malware detection can be improved by classifying Android malware into multiple known families. It can extract regular characteristics from malware actions that can be used to investigate unknown infections.They present FamDroid, an analysis-based learning-based Android malware family classification scheme.

This paper[22] explores the examination of PE programs using PEfile, a Python-based toolset. A flexible program called PEfile examines malware files in a simulated setting. Using PEfile, four separate datasets of malware packages are examined. The final product is produced using three separate algorithms, including:

1)Extraction algorithm (Feature Extraction),

2) Selection algorithm (Feature Selection)

3) Dataset Algorithm (Dataset Creation).

The selected features from each malware package are then compared and analyzed.In order to enhance the automatic identification and analysis of PE malware programs in malware samples, this article introduces the methodology and the PE-FILE reader module. The Extraction Algorithm was created to extract the feature set of PE malware and provide semantic signatures.The Selection Algorithm is a wrapper designed to remove features with uncertain performance effects and minimize feature dimension.The Dataset Algorithm was developed to store each PE sample's experimental information into a CSV file. After processing the feature extraction and feature selection tests, the researchers were able to develop a dictionary, apply it to various malware packages of directory samples, and generate datasets in CSV format.The researchers were able to evaluate and analyze several static properties from four separate malware packages in a virtual computer by using Portable Executable Reader Module (PEFile).

In this study[23],The study conducts tests using widely used datasets from prior work and examines the possibility of static analysis in detecting malware inside

the Android ecosystem. It presents an innovative static analysis-based approach to detecting Android malware. Static analysis is looking at the structure and code of Android applications without actually running them. Static analysis can offer helpful insights for spotting potential malware by examining several aspects including permissions, API calls, and code patterns.

The method for Android malware detection that is proposed in the research makes use of these static analysis approaches. It looks at the information that may be gleaned from static analysis and how these data can be used to train machine learning models or use rule-based methods to categorize programs as dangerous or benign. Each attribute utilized in the proposed strategy is assessed using various machine learning techniques to show how effective it is at identifying malware and to provide information to the digital investigators.Applications from the Play Store were fetched for the study in this article, which created the benign application dataset of the suggested technique.The suggested static analysis approach's characteristics are derived from both

(1)AndroidManifest.xml and

(2) the application source code files (Java files).

Then features are extracted from the apk file with apk tools. In order to determine which machine learning algorithm performs the best in terms of accuracy, the suggested static analysis technique was used with a variety of algorithms. . the performance can be evaluated by using the confusion matrix as this is a classification problem. Information gain of each feature is experimented for feature selection.According to the experimental result, the proposed approach's accuracy is calculated as high as 0.987. To comprehend the true objectives of API requests, source code analysis may be added to the suggested technique as future work.

The increasing frequency of cyberattacks has led to a growing demand for more efficient techniques in malware analysis and detection. To address this, researchers have explored the automation of these tasks using machine learning-based approaches, aiming to counter the escalating threat posed by sophisticated mal-

ware.In this particular study [24], two categories of malware detectors were implemented based on strings and specific PE header features. For each category, six different machine learning classifiers were employed. The experiment utilized a dataset consisting of 427 malicious PE files from the APT1 dataset and 989 benign PE files. The performance of these detectors was assessed by evaluating their detection precision and required execution time.The results revealed that the string feature set achieved a high level of accuracy in detecting malware within this dataset. Furthermore, the execution times of the detectors were found to be manageable, which is crucial for their application in sophisticated malware analysis scenarios. It is worth noting that the study focused on static analysis techniques and their effectiveness in detecting malware based on specific features. This research contributes to the ongoing efforts in developing efficient and accurate machine learning-based approaches for malware analysis and detection.

In this paper, a data mining-based approach[25] is suggested, wherein to increase malware detection accuracy and lower detection error rates, the PE file components table and its properties are used. It is quite challenging to find packed malware using its signature. The PEiD tool is employed in this article to ascertain whether the virus is packed. The malware will initially be unpacked if it has been packaged. All malicious and clean files have their PE header and section tables information extracted from them and put in the features database. Eight features are chosen from the PE header and section tables using the forward feature selection approach, which is how the performance of the classifier is assessed. The test files are evaluated using the classifiers DT, NN, ID3, NB, and SVM.The tests performed on 971 executable files showed that the DT classifier is 98.26% accurate. Anti-virus systems can use the classification results to increase their rate of malware detection.

The process of finding malware hidden inside PE files can be greatly aided by artificial intelligence (AI). This paper[26] uses artificial intelligence to investigate the characteristics of PE file headers as a means of identifying malware and evaluate the impact of these characteristics on the level of accuracy. Numerous PE

characteristics are used in the study. In order to determine the relative efficacy of two distinct algorithms, each has two possibilities. Testing is done with a predetermined control data set to determine relative performance. The degree of accuracy discovered after conducting numerous different groups of experiments is the criterion applied.Each study begins with a set of attributes, and then additional features are gradually introduced to investigate how these features affect accuracy. This was critical in demonstrating that not all characteristics increase accuracy. A large number of characteristics may not always increase accuracy, as some evidence suggested. It was demonstrated using graphs that accuracy will improve after the addition of a specific amount of features. Graphs also demonstrate that accuracy fluctuates as characteristics are added, improving occasionally while degrading at other times, indicating that not all features are helpful. Using a total of 29 features, more than 100 runs were completed. Decision Tree has the highest accuracy of 0.987 and 0.979.In Neural Networks-Multi-layer Perceptron (NN MLPC) and Decision Tree, respectively, the maximum accuracy was 0.987 and 0.979.They discovered that, generally speaking, more attributes linked to greater accuracy. There are indications, nevertheless, that increasing the number of characteristics may not always improve accuracy.

This research[27] suggests a malware detection technique that employs stacking and static file analysis to quickly and precisely identify new infections. Furthermore, malware can be discovered without actually running any malware by exploiting data from PE headers collected through static analysis. Because the extracted data were processed in different ways and used in machine learning models, the features of the pe packer utilized in the suggested study technique were most effective in experiments. Therefore, we decided to use pe packer data as the shape data for the stacking model. To identify with high accuracy, detection models are created based on additive techniques rather than single models. Findings: The suggested detection system can classify malicious or common files with speed and accuracy.Furthermore, tests revealed that the suggested approach surpasses currently used single model-based detection systems and has a 95.2% malware de-

tection rate. Therefore,this paper suggests a technique for learning, analyzing, and detecting malware files by using portable executable (PE) header feature values to the stacking approach after data pre-processing with the static analysis technique directed at malicious and benign files. 470 malicious files and 466 benign files were utilized to extract pe packer features for the input dataset. Extra Tree, Random Forest, Light XGBoost, and XGBoost were used in the stacking model's sub-model, and XGBoost was utilized in the meta learner. The Ubuntu 18.04 (64-bit) operating system was utilized for all trials, together with Anaconda3 (64-bit) and Python 3.6. A model to implement a highly accurate model was developed using the stacking method, which produced the final prediction value utilizing the prediction results of the single models. The study's findings revealed that the stacking model had a model accuracy of 96.71%, a classification accuracy of 94.6%, a detection rate of 95.2%, and a false-positive rate of 5.3%. The pace of the stacking model, one of the machine learning ensemble approaches, was considerably slower than that of the single models, even though it was confirmed to have a superior performance of malware detection than using simply the single model. In this research, a method for detecting malware using machine learning techniques is presented. In order to make malware detection more reliable, this study examines several malware detection techniques and how ML might be included. Malicious and good executable PE characteristics are used to train machine learning models. It has been shown that the model has a 99.4% prediction accuracy rate. This strategy would aid researchers in better comprehending and creating antivirus software that can detect even complicated infections.This study presents a real-time malware classification method utilizing PE files. We extracted various features from the dataset of 56 features using an extra-tree-classifier, and the top 13 features were then trained using six different machine learning methods, yielding a 99.4% accuracy rate. In the future, additional static and dynamic features may be combined to improve the model.

2.2 Related Works

2.2.1 MCFT-CNN

This paper[28] A new ResNet50 model called MCFT-CNN was introduced in the study, along with a traditional ResNet50 model, for both deep transfer learning and traditional learning approaches. The MalImg and Microsoft malicious software challenge datasets were used to train the models. The MCFT-CNN model is capable of identifying unknown malware samples without the need for feature engineering, which typically incurs additional computational burden on the classification or detection system, even when evading techniques are employed during malware development.
The primary contribution of this research in the field of malware classification can be summarized as follows:

- The study explores both traditional method of learning and also transfer learning approaches for training the models. The traditional learning approach involves training the ResNet50 model from scratch, while the transfer learning approach introduces a novel MCFT-CNN model that leverages knowledge from the dataset named ImageNet which is a pretrained dataset.

- The MCFT-CNN model demonstrates exceptional performance in accurately classifying malware into their respective families and efficiently identifying unknown malware samples. It outperforms other state-of-the-art deep learning algorithms, particularly when applied to larger volumes of malware image datasets.

- The paper comprehensively covers various aspects of malware classification and detection techniques using deep learning and image processing on malware image datasets. The malware binaries undergo preprocessing to convert them into grayscale images, and it is observed that images from the same malware family variants exhibit structural similarity.

- In the experimental setup, 75% of the dataset is utilized for training purposes, while the remaining 25% is used for testing the models' performance.
- The authors propose the MCFT-CNN model as a novel approach for classifying malware images into their respective families, employing transfer learning techniques.

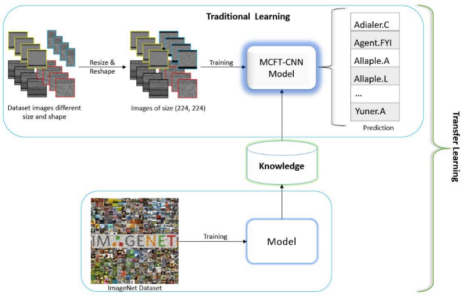


Figure 10: The conventional and transfer learning model architecture

The final layer of the ResNet50 architecture has been altered in this model by being swapped out for a completely connected dense layer. For categorization purposes, the output of this layer is subsequently routed via a SoftMax layer. The pre-trained weights of the ImageNet model, which have knowledge of low-level picture attributes, also contribute to the model's knowledge base.

The model obtains a precision of 99.05% with the Adam optimizer and 99.22% with the NAdam optimizer when trained using the conventional learning method. Notably, the model achieves a prediction time of 5.14 milliseconds for unknown malware samples, displaying outstanding performance.

Additionally, the model reaches accuracy of 99.18% using the Adam optimization approach and 99.10% using the NAdam optimizer when using the transfer learning strategy. The suggested approach classifies fresh malware samples in 5.14 milliseconds, which is comparable to the prediction times of the conventional learning model.

The shortcoming of the suggested model's CNN-based deep-learning approach is that uniform image size is used as the model's input. Future work should make use of the spatial pyramid pooling layer, that can accept images of any size as input.

The model's performance would undoubtedly increase if it took into account the full-size virus image.

2.2.2 Using PE Header Criteria to Detect Malware

About the work : This paper[29] is about Several ML classifiers that are recognize malware programs are built based on their header information and PE file structure.

Motive of the work : To develop a method that can be employed in real-time malware detection techniques that has high accuracy while only extracting a small amount of characteristics and quick model training.[30]

- **Why used ML algorithm :**
  - When using machine learning techniques,the training data can be examined for patterns, lessons can be drawn from them as well, and then data that resembles the training data can be found.
  - To create new versions, malware authors would typically merely alter a small portion of the code. Therefore, technologies based on machine learning have the capacity of identifying this infection.
- **Why used static features :**
  - Despite the possibility that dynamic features are more useful than static features, extracting them requires more effort and computer resources.
  - An instantaneous, quick malware analysis method currently a crucial and difficult work due to the growth in malware development.
  - Static feature extraction is more quicker and less expensive. Consequently, static features are utilized in this work.
- **Why used PE :** The PE file format, which contains structural data and enables the separation of harmful and benign programs, is the one that malware uses the most frequently. Although other file formats may occasionally

emerge, executable code-containing files are nearly exclusively in the PE file format.

Dataset : Here, a dataset of 2460 PE files was employed, including 1230 samples of malware and 1230 samples of benign code.1230 malware samples are randomly chosen from this collection. Additionally, benign samples are gathered from a typical Windows XP computer's "Program Files" and "System32" folders. ESET NODE32 antivirus was used to check all of the benign samples to make sure there were no dangerous files present.

Methodology:

- First, using a Microsoft document, the header section of PE files and its component along with structures are inspected.
- A particular set of features (nine features total) is extracted from each file in accordance with the PE header and its structure.
- Several ML models are trained using features that were extracted(Random Forest, SVM and KNN) to identify malware.
- K-fold Cross Validation is employed to assess the proposed approach.The following step is to utilize one part as test data and the remaining k-1 parts as training data. The end result is achieved by averaging the data from K rounds after repeating this operation for k times.

Result Analysis: Malware can be accurately detected by all three models. In all scenarios, random forest outperformed the other two distinct approaches, with an accuracy of 95.5%.

Limitations and Future work: Nine features is too much high amount for this accuracy.So if the accuracy would be high or feature number is less than 9 then this method would be more effective.So this can be the future work.

2.2.3 APT1 Dataset with String and PE Header Features

Cybersecurity experts may find it helpful to automate the malware detection tasks and work with Malware evaluation and identification methods based on machine

classifier	*Accuracy*	*Precision*	*Recall*	*F-measure*
Random Forest	**95.5**	**95.7**	95.4	**95.5**
SVM	94.4	93.5	**95.6**	94.5
KNN(K = 1)	94.0	94.6	93.5	94.0
KNN(K = 3)	93.7	94.5	92.8	93.6
KNN(K = 5)	93.7	94.6	92.8	93.6

Figure 11: Performance analysis using 9 PE header features

learning are needed to battle the continually growing and more complex malware files. Using strings and specific PE header features, respectively, they implemented two categories of malware detectors in this paper. For each category, they utilized a different set of six machine learning-derived classifiers. Each detector was tested using the same dataset, which contained 989 benign PE files and 427 benign and malicious PE files taken from the APT1 dataset. The effectiveness of several detectors was demonstrated, contrasted, and evaluated on the basis of detection precision and required execution time. The results of our experiment showed that the string-based feature set might provide extremely precise outcomes for this specific dataset. They also mentioned that these detectors' execution times appeared acceptable, which should be taken into account in sophisticated malware analysis software.

About the work : In this paper[24], The authors propose and implement six different machine learning classifiers, together with statically derived characteristics from two separate types of executables: string information & Portable Executable header data.The scikit-learn machine learning toolkit was utilized to develop these feature extraction techniques in Python.The productivity of twelve malware detection tools is then evaluated and compared in terms of detection precision and processing speed. The test results showed that the string feature analyzers outperformed the PE header-based detectors in terms of correctness.

Reason behind using Static analysis: Dynamic analysis has the noteworthy disadvantage of leaving some implementation routes undetected during the evaluation procedure, for example, if the malware is intended to run on specific occasions or when it senses a change in its operational environment. Dynamic malware

analysis is useless because of the cunning and cunning behaviors of malware. The use of dynamic analysis is subject to some limitations due to regulated network behavior and restricted network access. Additionally, Static analysis is significantly safer and less prone to host device risks since it can tell if an executable is malicious or not without actually running it.

Dataset : In this research they used the APT1 dataset. Thirteen X.509 encryption certificates and more than 40 malware types can be found in this dataset.The malware in this collection is considered to be exceptionally intelligent and dangerous, and it is characterized as an advanced persistent threat (APT).The authors used all 427 PE files which are malicious and 989 PE files which are benign from the APT dataset to carry out their research.

Working Principle : The working approach involved randomly dividing the dataset comprising APT1 malware and benign files into five groups, which were then cross-validated a total of five times. The string and PE header feature engineering are described below:

String features: Static analyses of malware can extract a variety of information from the APT1 dataset's files because they are not encrypted [14]. Finding the pertinent data in the binary, putting it in a Python lexicon, and using Sklearn's adaption of the hashing method to combine the retrieved strings into the feature vector were the steps required to extract these string features.A required minimum string size of 5 characters was used to extract strings into a Python format, after which they were hashed.

hasher = FeatureHasher (2000)

$hashed_features = hasher.transform([string_features])$

PE header features: The PE file structure, which is a Windows operating system standard, contains all the data required for an OS to load a file.They developed an extraction method using Python's pefile module by dividing the retrieved features into two main categories—Dense (50 features) and Sparse (17 features)—PE

header characteristics are converted into two arrays.

Dense Features: It refers to the primary features that are present in every executable file pe header.It demonstrates a high degree of complexity or contains a substantial amount of information. Certain dense features include header files such as-

- File
- Optional
- Section

Sparse Features: Sparse features of a PE header file refer to attributes or characteristics that are sparsely populated or have a low occurrence in the file.These features might not be present in every PE file and may vary depending on the specific file and its purpose.The examples are-

- Section names
- Imported symbols
- Warning strings

PE header features are extracted and then added to a pandas dataframe containing labels.Then they are ready to be fed into 6 hybrid detectors.

Experiment results : Setup: Six virtual processors, 100GB of RAM, and the 64-bit Ubuntu Linux guest OS were installed in a VirtualBox-based VM. The Dell Precision Tower 7910 machines used to host the VM were each outfitted with a single Intel Xeon E5-2620 v3 CPU (clocked at 2.40GHz, with six cores and twelve threads) and 128GB of RAM. Comparison of results between String features & PEheader features:

- Average feature extraction time of 6 detectors using -
  String: 66.20 seconds
  PE header: 1157.81 seconds 19 minutes

Accuracy Score	0.929
Precision	0.930
Recall	0.930
F1 Score	0.926
Model Execution Time	269.47 seconds

Figure 12: Results for SVM classifier using string features

- Support Vector Machine -

  String : Performance evaluation of the string feature SVM classifier-

  PE header : With a level of precision of 0.54, recall of 0.73, and a f1-score of 0.62, it obtained an accuracy score of 0.731. However, compared to the Logistic Regression, its model execution was longer.[24]

After evaluating and contrasting each detector utilizing both strings and PE header attributes on the identical APT1 malicious and benign dataset, the accuracy scores and total the time of execution of detectors constructed for using the string attributes and PE header features independently are presented in the following table:

Detector Type	*Accuracy: Strings*	*Accuracy: PE Header*	*Run Time: Strings*	*Run Time: PE Header*
SVM	0.929	0.731	335.67 s	1,297.05 s
LR	0.956	0.636	69.32 s	1,158.54 s
RF	0.977	0.897	72.90 s	1,158.90 s
XGB	0.979	0.912	275.38 s	1,158.84 s
LR/XGB	0.980	0.915	268.42 s	1,159.50 s
LR/RF/NB	0.973	0.898	104.40 s	1,158.71 s

Figure 13: performance assessment between string & PE Header features

Result Analysis : The APT1 dataset revealed that much more widely used terms by Windows-based OSes, such as names of parts of the hardware, were present in significantly more of the strings from the benign cases which appear to be windows OS executables[24]. This is why string features performed exceptionally well.Therefore, It was their hypothesis that, for this specific dataset at least, When the same sorts of classifiers were used, the string features produced greater accuracy ratings than the PE header features because they provided an increased

degree of difference among malicious and benign samples.

3 Proposed Method

3.1 Introduction

Our target is to build an efficient feature extraction method which will decrease the feature extraction time for PE header files. As mentioned in the previous paper,In comparison to those applying the chosen PE header features, the detectors by using the string features outperformed them in terms of evaluation parameters like accuracy, extraction time etc.This could occur because of this specific APT1 dataset. That's why we are aiming to develop this efficient feature extraction method using PE header files and we are going to test this method among three significant malware datasets. EMBER dataset having a training sample of 900k & Testing sample 200k. Also we have already started processing malware executable files from malware bazaar . abuse website. We create our own database and publicize it. Therefore, we want to improve the extraction time without causing much imbalance to the other parameters e.g. execution time, accuracy, f values etc.

3.2 Importance of decreasing Feature Extraction Time

We are emphasizing on decreasing the feature extraction time of malware static analysis using PE header file features. Feature extraction time is a very crucial part of malware analysis process. 1024 million malware detected in 2021. That means 33 malware in a Second. The malware attack frequencies also increased over the years. For example, in 2021 one ransomware attack occurred in every 11 second !

That's why in order to improve the execution time and therefore the run time we have to focus on improving the feature extraction time first.

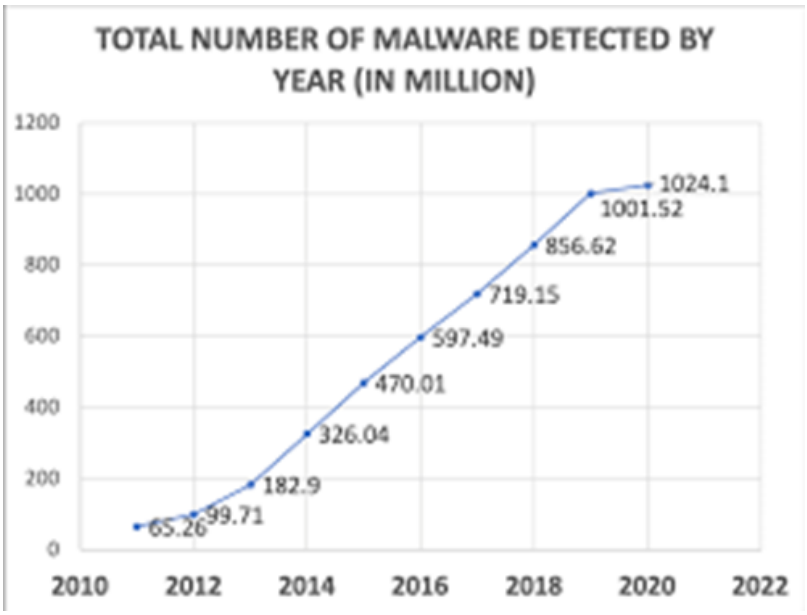


Figure 14: Number of malware detected over the years 2010-2020

3.3 Overview of implementation

3.3.1 Dataset Preparation

1. **Data Collection:** For implementing this, first we had to collect malware sample(executable files) from malware bazaar.

2. **Data Processing:** Then we kept those files into one file and upload that file.For finding exe and non exe file we check the file extension and print the file name along with it's extension.We used PEFILE module in python. Among the **Dense Features** optional header & file header features are processed.

    - **File Header:** The first part of a PE file is called the file header, and it comes first in the file. It includes crucial details about the PE file's general structure and features. The file header contains information such as the entry point, machine architecture, number of sections, and signature.

    - **Optional Header:** Within the Portable Executable file format, the optional header comes right after the File header. It offers more thor-

ough details on the PE file, as well as a number of optional features and setup options. Magic Number, Image Base:, Section Alignment, Data Directories, and other options are included in the optional header.

3. **Feature Engineering:** We then extracted optional header and file header features. Created dataframe from the list of features. Then we transformed the dataframe into CSV file.

4. **Labeling & Splitting Dataset:** We split the training and testing data into 80: 20 ratio.

5. **Data Analysis:** We trained and tested the dataset for different machine learning classifiers with and without Hyper-parameter tuning. The machine learning classifers are-

    - Random forest
    - K-nearest
    - Decision Tree
    - Support Vector Machine
    - Logistic Regression
    - Gradient Boosting

3.4 Dataset Information

- Number of Malware Executables : 289
- Total Number of features Extracted: 33
- Total number of optional header feature : 7
- Total number of File header feature : 26
- Size of the dataset : 290 rows * 34 columns

3.5 Prototyping

At first we have worked with 17 features selected from the feature list stated above.

	Filename	Machine	NumberOfSections	Characteristics	AddressOfEntryPoint	ImageBase	SectionAlignment	FileAlignment	Subsystem	DllCharacteristics
0	e94037a20e9c3c9b09811f71090503a5f9b70626bc54ca...	0	4194304	8192	512	3	34144	4	0	0
1	e46684d8719f3585105ec7f8a92b30af5ef89bd57a91c6...	388622	4194304	8192	512	2	34112	4	0	0
2	27ce6c9751973535b4462b9ae21d9278fcef0d7e46e195...	27600	4194304	4096	512	2	49472	10	0	10
3	cd9fd32450b4349d32975961ce18329da72df7885241cc...	17189	4194304	4096	512	2	32768	5	0	0
4	Mal_2.exe	745196	4194304	4096	512	2	33088	6	1	6
5	40feab4dd757a120d60e8bf1892464454bf157bc7fba4a...	27600	4194304	4096	512	2	49472	10	0	10
6	662b0600e1c0e866f21fe9a41f5836c66861909a725ab9...	33280	5368709120	4096	512	2	49504	10	0	10
7	e62e296cf606dcbf457563f5e2d3a989f318f8a6300b11...	17253	4194304	4096	512	2	33024	5	0	0
8	1cf2f9ca3cc70c2919ba016a0166a95c81e1a09e95aaa5...	17269	4194304	4096	512	2	33024	5	0	0
9	9bff71afadddb02956bd74c517b4de581885b0d6ff0077...	937630	4194304	8192	512	2	34112	4	0	0
10	479f98406130a68ba1c59a48f96f3c099994290b6ca66f...	27600	4194304	4096	512	2	49472	10	0	10
11	MAL_4.exe	17253	4194304	4096	512	2	33024	5	0	0
12	ce095951b07d0d736318adaa328b7e063b4bc4ecc1253c...	625674	4194304	8192	512	2	34112	4	0	0
13	e75715cec331e8848e56834721ee6c09f8b8093a74fc59...	19232	268435456	4096	512	2	1344	5	1	0
14	f00e204620c934d028745f4b3eb1ae6101bb1bba23d51c...	42488	4194304	4096	512	2	33024	5	1	0
15	97e993ff871f0d3c484ec6059bd63ec6273d82e420aa96...	17141	4194304	4096	512	2	32768	5	0	0
16	bace391f012bcae2f947e6d7e079df8bff56809cce50af...	745196	4194304	4096	512	2	33088	6	1	6
17	7254155d9388c0f53ce6f892c18f0516a1270c223f13e1...	17077	4194304	4096	512	2	32768	5	0	0
18	cc7636560da3a88deaa58de31e3c9bf91b6dae8d07312e...	102222	4194304	8192	512	2	34112	4	0	0
19	227b396c6dceeb7107850a0fd635299670d01e91fef3aa...	27434	4194304	4096	512	2	33088	6	0	0
20	d80323dd840c86cc64285f6353ccc84080b4c242df320b...	570686	4194304	8192	512	2	34112	4	0	0

MajorOperatingSystemVersion	MinorOperatingSystemVersion	MajorImageVersion	MinorImageVersion	SizeOfImage	SizeOfHeaders
0	1302528	512	34404	2	34
0	409600	512	332	3	258
0	1331200	1024	332	5	258
0	41627648	1024	332	4	258
0	929792	1024	332	10	33167
0	1331200	1024	332	5	258
0	2285568	1024	34404	6	34
0	790528	1024	332	4	258
0	663552	1024	332	4	258
0	1024000	512	332	3	258
0	1331200	1024	332	5	258
0	790528	1024	332	4	258
0	647168	512	332	3	258
0	88820	1024	332	6	258
0	4317184	1024	332	3	259
0	41500672	1024	332	4	258
0	884736	1024	332	10	33167
0	794624	1024	332	4	258
0	114688	512	332	2	258
0	1175552	1536	332	5	258
0	598016	512	332	3	258

Figure 15: Representation of dataset with 17 features

After that we extracted more feature from malware executable files and worked with 34 features.

	Filename	Machine	NumberOfSections	TimeDateStamp	PointerToSymbolTable	NumberOfSymbols	SizeOfOptionalHeader	Characteristics	Maj
0	e94037a20e9c3c9b09811f71090503a5f9b70626bc54ca...	34404	2	1681578150	0	0	240	34	
1	e46684d8719f3585105ec7f8a92b30af5ef89bd57a91c6...	332	3	1684246832	0	0	224	258	
2	27ce6c9751973535b4462b9ae21d9278fcef0d7e46e195...	332	5	1468633330	0	0	224	258	
3	cd9fd32450b4349d32975961ce18329da72df7885241cc...	332	4	1650659699	0	0	224	258	
4	Mal_2.exe	332	10	1649952623	0	0	224	33167	
5	40feab4dd757a120d60e8bf1892464454bf157bc7fba4a...	332	5	1468633330	0	0	224	258	
6	662b0600e1c0e866f21fe9a41f5836c66861909a725ab9...	34404	6	2921055480	0	0	240	34	
7	e62e296cf606dcbf457563f5e2d3a989f318f8a6300b11...	332	4	1663571011	0	0	224	258	
8	1cf2f9ca3cc70c2919ba016a0166a95c81e1a09e95aaa5...	332	4	1637656058	0	0	224	258	
9	9bff71afadddb02956bd74c517b4de581885b0d6ff0077...	332	3	1681542146	0	0	224	258	
10	479f98406130a68ba1c59a48f96f3c099994290b6ca66f...	332	5	1468633330	0	0	224	258	
11	MAL_4.exe	332	4	1663571011	0	0	224	258	
12	ce095951b07d0d736318adaa328b7e063b4bc4ecc1253c...	332	3	2269685076	0	0	224	258	
13	e75715cec331e8848e56834721ee6c09f8b8093a74fc59...	332	6	1520187015	0	0	224	258	
14	f00e204620c934d028745f4b3eb1ae6101bb1bba23d51c...	332	3	1637602881	0	0	224	259	
15	97e993ff871f0d3c484ec6059bd63ec6273d82e420aa96...	332	4	1640284562	0	0	224	258	
16	bace391f012bcae2f947e6d7e079df8bff56809cce50af...	332	10	1649952623	0	0	224	33167	
17	7254155d9388c0f53ce6f892c18f0516a1270c223f13e1...	332	4	1658415671	0	0	224	258	

MinorLinkerVersion	...	SizeOfHeaders	CheckSum	Subsystem	DllCharacteristics	SizeOfStackReserve	SizeOfStackCommit	SizeOfHeapReserve	SizeOfHeapCommit	LoaderFlags	NumberOfRvaAndSizes
0	...	512	1321309	3	34144	1048576	4096	1048576	4096	0	16
0	...	512	0	2	34112	1048576	4096	1048576	4096	0	16
0	...	1024	1368633	2	49472	262144	8192	1048576	4096	0	16
0	...	1024	376787	2	32768	1048576	4096	1048576	4096	0	16
25	...	1024	5154991	2	33088	1048576	16384	1048576	4096	0	16
0	...	1024	1327823	2	49472	262144	8192	1048576	4096	0	16
20	...	1024	2285655	2	49504	524288	8192	1048576	4096	0	16
0	...	1024	349667	2	33024	1048576	4096	1048576	4096	0	16
0	...	1024	255062	2	33024	1048576	4096	1048576	4096	0	16
0	...	512	0	2	34112	1048576	4096	1048576	4096	0	16
0	...	1024	1354366	2	49472	262144	8192	1048576	4096	0	16
0	...	1024	349667	2	33024	1048576	4096	1048576	4096	0	16
0	...	512	0	2	34112	1048576	4096	1048576	4096	0	16
0	...	1024	73975	2	1344	1048576	4096	1048576	4096	0	16
0	...	1024	508220	2	33024	1048576	4096	1048576	4096	0	16
0	...	1024	247962	2	32768	1048576	4096	1048576	4096	0	16
25	...	1024	0	2	33088	1048576	16384	1048576	4096	0	16
0	...	1024	407975	2	32768	1048576	4096	1048576	4096	0	16
0	...	512	0	2	34112	1048576	4096	1048576	4096	0	16
32	...	1536	0	2	33088	1048576	4096	1048576	4096	0	16
0	...	512	0	2	34112	1048576	4096	1048576	4096	0	16

Figure 16: Representation of dataset with 34 features

3.6 Solution Approach

In order to improve the feature extraction time , we have to pay great attention to feature selection process.

- First we checked whether the file executable or not.We excluded the .rar,.elt files and took only .exe file into consideration.
- Only taking dense features- file header feature and optional header features significantly improved the feature extraction time because these header fields are present in almost every malware file and easily recognizable to extract.
- For the feature extraction time, timer function is set explicitly around the extraction code to determine the accurate time. Then using more number of features its again evaluated to check for any improvement.
- When including the section header features , the feature extraction time was 35.23 seconds.
- For less number of feature we calculate the feature extraction time 15.98 seconds.
- Then we trained the model as well as testing with different classifier such as Random Forest, K Nearest Neighbor, Decision Tree, Support Vector Machine, Logistic Regression, Gradient Boosting Classifier etc.[31]
- We evaluated these classifier by their accuracy,F1 Score,Precision Score and Recall Score[32] . After that we improve the testing for the individual classifier.
- To improve the values, we performed hyper-parameter tuning on these dataset. It helped improve the accuracy and other evaluation parameters.
- Then in the same way we calculate the accuracy for more number of features. And in this case run time 17.23 seconds. that is less then less number of features. Again we run the model with training and testing. And later with hyper-parameter tuning the results are improved.

4 Result Analysis

Here are the numeric results of accuracy, f1 score, precision score & recall score [32] of different Machine learning classifiers [31] using less number of feature(17 features). Here we deliberately excluded the classifiers' result which produced ambiguous and sparse values. **Feature Extraction time required:** 15.98 seconds.

Classifier name	Accuracy	F1 Score	Precision Score	Recall Score
Random Forest	0.9	0.47	0.5	0.45
K Nearest Neighbor	0.9	0.47	0.5	0.45
Decision Tree	0.8	0.45	0.5	0.45
Gradient Boosting	0.8	0.44445	0.5	0.4

Table 1: After training and testing of individual classifier for less number of features

We can also visualize it in the graph given below -

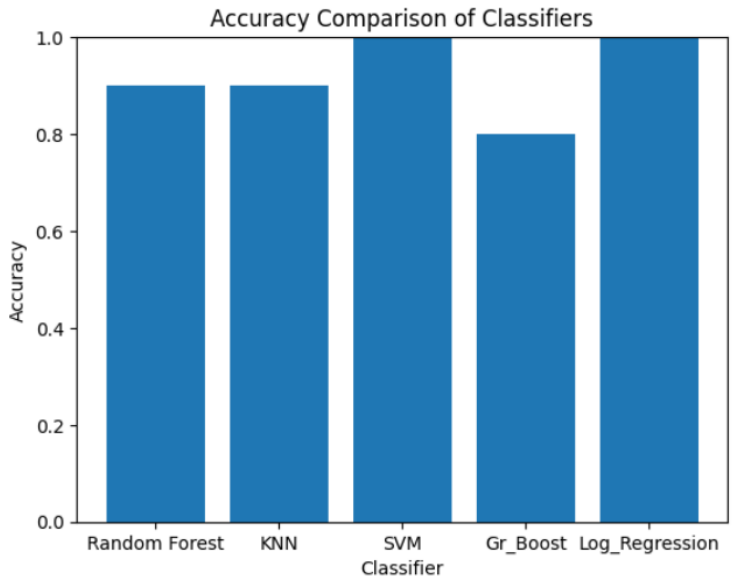


Figure 17: Result analysis of different classifiers using less number of features.

Hyper-parameter tuning using GrideSearchCV : After improve testing , the result we got is given below :

Classifier name	Accuracy	F1 Score	Precision Score	Recall Score
Random Forest	0.9	0.947	1.0	0.9
K Nearest Neighbor	0.9	0.947	1.0	0.9
Decision Tree	0.8	0.889	1.0	0.800
Gradient Boosting	0.8	0.945	1.0	0.9

Table 2: After improvement of individual classifier for less number of features

We can also see it in the graph given below -

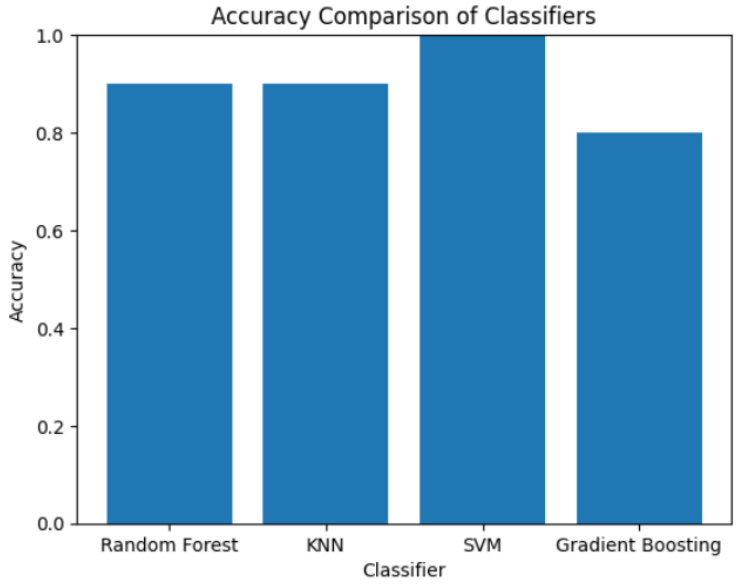


Figure 18: Visualization of improvement of Evaluation parameters for less number of features

Now for more number of features(approximately 34 features) the feature extraction time is 17.23 seconds. The training and testing of individual classifier for more number of feature is given below-

Classifier name	Accuracy	F1 Score	Precision Score	Recall Score
Random Forest	0.7	0.208	0.194	0.21875
K Nearest Neighbor	0.6	0.1875	0.1875	0.1875
Decision Tree	0.7	0.175	0.175	0.175
Gradient Boosting	0.8	0.235	0.222	0.25

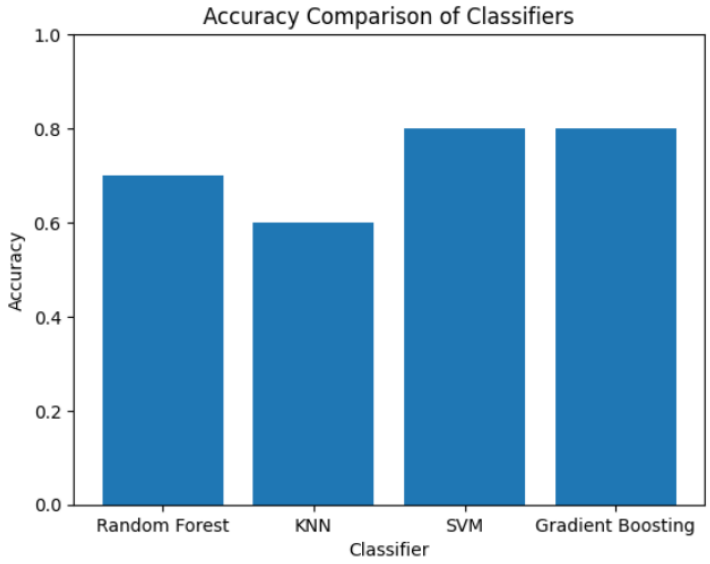
Table 3: After training and testing of individual classifier for more number of features(33 features)

After **hyper-parameter tuning using GridSearchCV** we achieved improved results.

Classifier name	Accuracy	F1 Score	Precision Score	Recall Score
Random Forest	0.7	0.658	0.622	0.7
K Nearest Neighbor	0.6	0.6	0.6	0.6
Decision Tree	0.7	0.7	0.7	0.7
Support Vector Machine	0.8	0.640	0.8	0.711
Logistic Regression	0.6	0.685	0.8	0.6
Gradient Boosting	0.8	0.711	0.8	0.753

Table 4: After improving of individual classifier for more number of features

The visualization of improvement after hyper-parameter tuning for more number of feature(34 features) is given below-



4.1 Performance Analysis

1. Our proposed methodology and dataset outperforms the APT1 dataset and feature extraction method by all evaluation criteria. The accuracy, run time is far better than that of APT1 dataset.

2. The main characteristics of the PE header are the file header and optional header components. So considering and selecting these features improved the performance.

3. The feature extraction time difference when using PE header features in APT1 dataset vs using PE Header features in our dataset is huge .
   In APT1 dataset: 1154.81 seconds or 19 minutes
   In our dataset: 17.23 seconds

5 Conclusion

To reduce feature extraction time, we created a dataset and extracted PE Header features. The current work did not significantly reduce extraction time, but there are several future directions to explore. The correlation between dense and sparse features should be explored more. Paralleling feature extraction and optimizing algorithms can boost efficiency. Future work can speed up feature extraction, which is crucial for data analysis pipeline efficiency and timely decision-making.

5.1 Future work

1. Explore alternative feature extraction methods and algorithms to reduce extraction time. This could involve researching and implementing cutting-edge feature extraction methods or novel sparse data methods.

2. Use multi-core processors or distributed computing systems to parallelize feature extraction. Processing multiple features simultaneously could speed up feature extraction.

3. Analyze and optimize feature extraction algorithms. This could involve identifying algorithm bottlenecks or computational inefficiencies and devising ways to overcome them, such as algorithmic modifications or algorithm-specific optimizations.

4. Reduce feature dimensionality while maintaining discriminative capability by using techniques such Principal Component Analysis, or PCA, or t-SNE which stands for t-distributed Stochastic Neighbor Embedding . It takes less time to extract features while using a lower-dimensional space for features.

5. Due to the core function and method of APT1 dataset not being publicly available , we were unable to compare the feature extraction time using the same methodology or same set of executables. So implementing and improving the pefile module from core will be a significant research contribution in this field.

6. Lastly, combining the dense and sparse features and efficient feature selection will lead to efficient feature extraction and therefore improve the whole static malware analysis process.

References

[1] Rohit Kalakuntla, Anvesh Babu Vanamala, and Ranjith Reddy Kolipyaka. Cyber security. *HOLISTICA – Journal of Business and Public Administration*, 10(2):115–128, 2019.

[2] Paul Black and Joseph Opacki. Anti-analysis trends in banking malware. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 1–7. IEEE, 2016.

[3] Mahbub Khoda, Tasadduq Imam, Joarder Kamruzzaman, Iqbal Gondal, and Ashfaqur Rahman. Selective adversarial learning for mobile malware. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 272–279. IEEE, 2019.

[4] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*, pages 1–7. IEEE, 2018.

[5] Wenjia Li, Zi Wang, Juecong Cai, and Sihua Cheng. An android malware detection approach using weight-adjusted deep learning. In *2018 international conference on computing, networking and communications (ICNC)*, pages 437–441. IEEE, 2018.

[6] Caio C. Moreira, Davi C. Moreira, and Claudomiro de S. de Sales Jr. Improving ransomware detection based on portable executable header using xception convolutional neural network. *Computers Security*, 130:103265, 2023.

[7] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. Maldozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, 24:S48–S59, 2018.

[8] ElMouatez Billah Karbab. *Resilient and Scalable Android Malware Fingerprinting and Detection*. PhD thesis, Concordia University, 2020.

[9] Durmuş Özkan Şahın, Sedat Akleylek, and Erdal Kiliç. Linregdroid: Detection of android malware using multiple linear regression models-based classifiers. *IEEE Access*, 10:14246–14259, 2022.

[10] Shifu Hou, Aaron Saas, Lifei Chen, and Yanfang Ye. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, pages 104–111. IEEE, 2016.

[11] Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *2017 European intelligence and security informatics conference (EISIC)*, pages 99–106. IEEE, 2017.

[12] Fahad Alswaina and Khaled Elleithy. Android malware family classification and analysis: Current status and future directions. *Electronics*, 9(6):942, 2020.

[13] Syed Zainudeen Mohd Shaid and Mohd Aizaini Maarof. Malware behavior image for malware variant identification. In *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*, pages 238–243. IEEE, 2014.

[14] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. *Computers & Security*, 92:101740, 2020.

[15] Saurabh Chaudhary. Advance malware analysis using static and dynamic methodology. Technical report, EasyChair, 2020.

[16] Paul Black, Iqbal Gondal, Adil Bagirov, and Md Moniruzzaman. Malware variant identification using incremental clustering. *Electronics*, 10(14):1628, 2021.

[17] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer. Dl-droid: Deep learning based android malware detection using real devices. *Computers & Security*, 89:101663, 2020.

[18] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6:30996–31011, 2018.

[19] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6:30996–31011, 2018.

[20] Vikas Sihag, Manu Vardhan, Pradeep Singh, Gaurav Choudhary, and Seiil Son. De-lady: Deep learning based android malware detection using dynamic features. *J. Internet Serv. Inf. Secur.*, 11(2):34–45, 2021.

[21] Liang Zhao, Jiayang Wang, Ye Chen, Fan Wu, Yuan'an Liu, et al. Famdroid: learning-based android malware family classification using static analysis. *arXiv preprint arXiv:2101.03965*, 2021.

[22] Rico S Santos and Enrique D Festijo. Generating features of windows portable executable files for static analysis using portable executable reader module (pefile). In *2021 4th International Conference of Computer and Informatics Engineering (IC2IE)*, pages 283–288. IEEE, 2021.

[23] Abdullah Talha Kabakus. What static analysis can utmost offer for android malware detection. *Information Technology and Control*, 48(2):235–249, 2019.

[24] Neil Balram, George Hsieh, and Christian McFall. Static malware analysis using machine learning algorithms on apt1 dataset with string and pe header features. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 90–95. IEEE, 2019.

[25] Nahid Maleki and Hamid Rastegari. An improved method for packed malware detection using pe header and section table information. *International Journal of Computer Network & Information Security*, 11(9), 2019.

[26] Hasan H Al-Khshali, Muhammad Ilyas, and Osman N Ucan. Effect of pe file header features on accuracy. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1115–1120. IEEE, 2020.

[27] Chang Keun Yuk and Chang Jin Seo. Static analysis and machine learning-based malware detection system using pe header feature values. *International Journal of Innovative Research and Scientific Studies*, 5(4):281–288, 2022.

[28] Sushil Kumar et al. Mcft-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things. *Future Generation Computer Systems*, 125:334–351, 2021.

[29] Tina Rezaei and Ali Hamze. An efficient approach for malware detection using pe header specifications. In *2020 6th International Conference on Web Research (ICWR)*, pages 234–239. IEEE, 2020.

[30] Tina Rezaei and Ali Hamze. An efficient approach for malware detection using pe header specifications. In *2020 6th International Conference on Web Research (ICWR)*, pages 234–239, 2020.

[31] Kim-Kwang Raymond Choo and Ali Dehghantanha. *Handbook of Big Data Analytics and Forensics*. Springer, 2022.

[32] Saneeha Khalid and Faisal Bashir Hussain. Evaluating dynamic analysis features for android malware categorization. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pages 401–406, 2022.