Malware Detection in Portable Executable Files Using Machine Learning

by

Dasari Jayanth

2019BCS-016

A report submitted for Summer Project

Bachelor of Technology

in

CSE

ATAL BIHARI VAJPAYEE–

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT

GWALIOR - 474015, MADHYA PRADESH, INDIA

Report Certificate

I hereby certify that the work, which is being presented in the report, entitled **Malware Detection in Portable Executable Files Using Machine Learning**, for Summer Project in Computer Science Engineering and submitted to the institution is an authentic record of my/our own work carried out during the period May-2021 to August-2021 under the supervision of **Dr. Saumya Bhadauria.** I also cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

(Dasari Jayanth)

Date: **10/08/2021** Name and Signature of the candidate

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

(Dr. Saumya Bhadauria)

Date: **10/08/2021** Name and Signature of the Research Supervisor

Candidate's Declaration

I hereby certify that I have properly checked and verified all the items as prescribed in the check-list and ensure that my thesis is in the proper format as specified in the guideline for thesis preparation.
I declare that the work containing in this report is my own work. I understand that plagiarism is defined as any one or combination of the following:

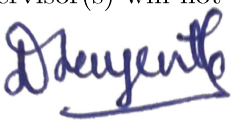
(1) To steal and pass off (the ideas or words of another) as one's own

(2) To use (another's production) without crediting the source

(3) To commit literary theft

(4) To present as new and original idea or product derived from an existing source.

I understand that plagiarism involves an intentional act by the plagiarist of using someone else's work/ideas completely/partially and claiming authorship/originality of the work/ideas. Verbatim copy as well as close resemblance to some else's work constitute plagiarism.
I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programmes, experiments, results, websites, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.
I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report/dissertation/thesis are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable. My faculty supervisor(s) will not be responsible for the same.

Signature: 

Name: Dasari Jayanth

Roll. No: 2019BCS-016

Date: 10/08/2021

Abstract

The number of Malicious files is increasing every day because of existing open-source malware and obfuscation techniques. So, it became essential to use machine learning methods in malware detection. This project aims to classify the unknown portable executable files as benign or malware with different machine learning methods by extracting the significant features from PE files using hybrid static malware analysis and training the various classifier models with the acquired feature set. By comparing the accuracies of these models, this project also aims to find the best-suited machine learning classifier model for malware detection for these static feature set characteristics.

Keywords: Malware Detection, Static Malware Analysis, Opcode N-grams, Byte N-grams, PE Header, PE Files, Machine Learning.

Acknowledgments

I am highly indebted to **Dr. Saumya Bhadauria**, and am obliged for giving me the autonomy of functioning and experimenting with ideas. I would like to take this opportunity to express my profound gratitude to her not only for her academic guidance but also for her personal interest in my project and constant support coupled with confidence boosting and motivating sessions which proved very fruitful and were instrumental in infusing self-assurance and trust within me. The nurturing and blossoming of the present work is mainly due to her valuable guidance, suggestions, astute judgment, constructive criticism and an eye for perfection. The mentor always answered myriad of the doubts with smiling graciousness and prodigious patience, never letting me feel that I am novices by always lending an ear to the views, appreciating and improving them and by giving a free hand in the project. It's only because of her overwhelming interest and helpful attitude, the present work has attained the stage it has.

Finally, I am grateful to my family and friends whose constant encouragement served to renew my spirit, refocus my attention and energy and helped me in carrying out this work.

(Dasari Jayanth)

Contents

Chapter

1 Introduction **1**

1.1 Context 1

1.2 Problem/Motivation 1

1.2.1 Problem Statement 1

1.2.2 Motivation 2

1.3 Objectives 3

1.4 Research work flow 3

2 Literature review **4**

2.1 Malware and its types 4

2.1.1 What is Malware 4

2.1.2 Types of Malware 4

2.2 Malware Analysis 5

2.2.1 Static Malware Analysis 6

2.2.2 Dynamic Malware Analysis 7

2.2.3 Hybrid Malware Analysis 7

2.3 Malware Detection Techniques 8

2.4 PE File Format 9

2.5 Key related research 11

2.6 Problem formulation 13

3 Methodology **14**

3.1 Proposed hypothesis 14

3.2 Mechanism/Algorithm 14

3.3 Analytical validation 15

4 Experiments and results **16**

4.1 Extracting PE Header information 16

4.2 Byte Files 17

4.2.1 Feature Extraction and Feature selection from Byte files 17

4.2.2 Extracting byte bi-grams 18

4.3 Asm Files 19

4.3.1 Feature Extraction and Feature Selection from Asm files 19

4.3.2 Extracting opcode n-grams(n=2,3) 20

4.3.3 Image Feature Extraction from asm Files 20

4.4 Results 21

5 Discussions and conclusion **22**

5.1 Salient Features of this Project 22

5.2 Limitations 22

5.3 Future scope 23

Bibliography **24**

Figures

Figure

2.1 Malware Analysis types and it's challenges 6

2.2 PE File Format 9

4.1 PE Header Extracted Sample 16

4.2 Byte File Sample 17

4.3 Byte vocab 18

4.4 ASM File Sample 19

4.5 ASM File features 19

4.6 ASM Image 20

4.7 Result 21

Chapter 1

Introduction

1.1 Context

The internship work is a part of Malware Detection in Portable Executable Files using Machine Learning project at Indian Institute of Information Technology and Management, Gwalior. This project aims to train a classifier model using static malware analysis to classify unknown PE files into malicious or benign files.

1.2 Problem/Motivation

1.2.1 Problem Statement

In the past decade, the malware industry has overgrown. Detecting malware attacks using traditional methods became difficult due to polymorphism, metamorphism, and soon. So, the demand for technologies to detect malware has grown abruptly. Effective and automated malware detection has become an essential requirement to guarantee system safety and user protection. The major part of protecting a device from a malware attack is identifying whether a given piece of file/software is malware.

1.2.2 Motivation

Attackers are targeting the Windows operating system as it is most widely used. The portable executable(PE) format is the file format for executables(.exe), DLLs, object code, and others used in 32-bit and 64-bit versions of the Windows operating system. So, PE files are used in this project.

Dynamic analysis may be accurate in analyzing the file, but it is expensive and prolonged as it requires execution and time to prepare the environment for malware analysis. So, it is vulnerable. Whereas static analysis is fast and safe as it executes the code before execution, so we can prevent the attack. In this project, static malware analysis will be used.

Machine learning is a data analytics tool used to perform specific tasks without explicit instructions effectively. It is difficult to detect malicious attacks with traditional methods due to its enormous growth and the number of attacks per day. Many malware programs are derived from another with just slight changes. So, it will be easy and efficient to use ML methods due to the vast data to verify. So, in this project, Machine learning is used in detecting malware.

There are many studies regarding both static and dynamic malware analysis and classification of malicious code. Most of the works have used only one type of feature construction methods like either byte n-gram or opcode sequence or PE headers. Due to the growth of malware in recent days, it became difficult to detect malware using one characteristic simply. By building a feature set combining more than one characteristic and extracting significant features, we can create a better feature set that gives greater accuracy. So, this project aims to build a feature set with more than one characteristic and train different classifiers using this feature set to improve accuracy in detecting malicious files.

1.3 Objectives

The main objective of this project is to create a malware detection model using hybrid malware static analysis on PE files. We train classifier models using extracted feature set to classify unknown PE files into malicious and benign files. The Feature set is created by extracting selected features from the byte n-gram sequences, opcode n-gram sequences, and PE header to achieve better accuracy and efficiency. It also covers various file types(.asm, .byte, .exe, .dll) . We train different classifier models like K-NN, Random Forest, Gradient Boosting, Logistic Regression, Decision Trees, Support Vector Machines(SVM) on this feature set and compare them to find which model gives better accuracy..

1.4 Research work flow

According to the research objectives, the report will describe the work flow as below:

Step 1 Data Collection: Downloading dataset from virusshare and kaggle.

Step 2 Data Analysis and Feature Extraction: Extracting Features from the downloaded files.

Step 3 Normalization of the features.

Step 4 Feature reduction: Selecting Best features from extracted features as there are so many features.

Step 5 Merging the features extracted.

Step 6 Dividing feature set into two sets train feature set and test feature set and training different classifier models and testing them.

Step 7 Model validation.

Chapter 2

Literature review

2.1 Malware and its types

2.1.1 What is Malware

Malware is malicious software designed or developed with evil intentions mainly to steal information, disrupt services or gain access. It can be code, scripts, executables, library files, or other forms of applications. Malware is a general term used to refer to hostile, intrusive, or annoying software.

2.1.2 Types of Malware

- **Backdoor:** It allows the attacker access by installing malicious code itself onto a computer. They usually let the attacker connect to the computer with little or no authentication and execute commands on the local system.

- **Botnet:** It allows an attacker access to the system similar to backdoor. All the computers infected with the same botnet receive the same instructions from a single command-and-control server.

- **Download:** Malicious code that only exists to download other malicious code. These are commonly installed by attackers when they first gain access to a system.

- **Information Stealing Malware:** Malware collects information from a victim's computer and usually sends it to the attacker. This malware is typically used to gain access to online accounts such as email or online banking. Examples are keyloggers, sniffers.

- **Launcher:** Malicious program used to launch other malicious programs to ensure stealth or greater access to a system.

- **Rootkit:** It is designed to conceal the existence of other code and are usually paired with other malware to allow remote access to the attackers.

- **Scareware:** Malware designed so that user infected with this malware is frightened into buying something. It has a user interface that makes it look like antivirus or other security programs.

- **Spam Sending Malware:** Malware infects a user's machine and then uses that machine to send spam.

- **Worms:** These are malicious codes that can copy themselves and infect additional computers.

2.2 Malware Analysis

Malware analysis is the art of dissecting malware to understand how it works, identify it, and defeat or eliminate it.

- The main purpose of malware analysis is to determine what happened precisely, to ensure that all infected machines and files are located.

- The goal of analyzing suspected malware is typically to determine what a particular suspect binary can do, how to detect it, and how to measure its damage.

There are two fundamental approaches to malware analysis. Static analysis involves examining malware without running it. Dynamic analysis involves running the malware.

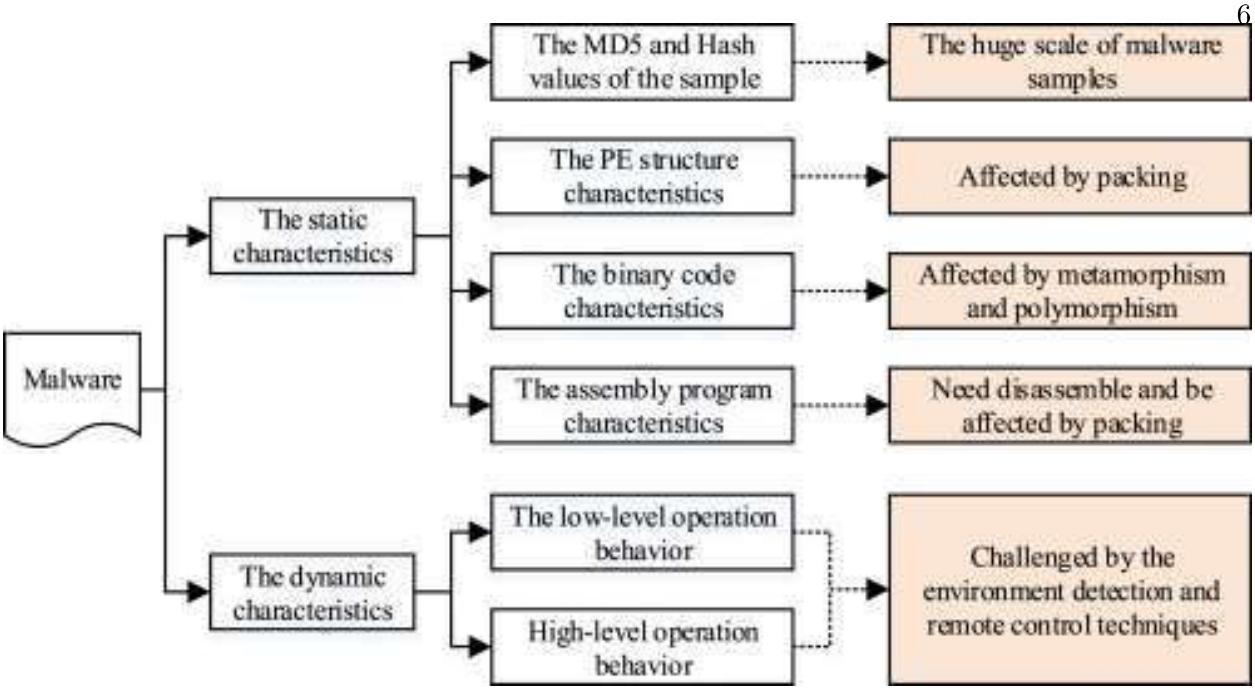


Figure 2.1: Malware Analysis types and it's challenges

2.2.1 Static Malware Analysis

Static malware is malware at rest. Static malware analysis is the process of extracting information from malware while it is not running. In static malware analysis(also called code analysis), the program is analyzed without executing it, and reverse engineering is performed using different tools like debugger, disassemble, decompile, etc. It is the easiest and least risky process. There is no risk of an infection occurring while analysis is taking place. N-grams byte sequences, Opcode sequences, API calls, PE header, control flow graphs are the different characteristics of static malware analysis.

An N-Gram is a contiguous sequence of n items from a given sample of text. For example: Given the input "this is Malware" using words as a unit, N-gram of size one produces: {this, is, Malware}. N-gram of size 2 produces: {this is, is Malware, this Malware}. N-grams are used in static heuristic analysis to extract the behavior of the samples where the assembly code or binary code is split using N-grams.

To characterize the contents of the n-gram, we move a fixed-length window over the extracted opcode sequence or byte sequence, where the n-gram is a subsequence of length n at each position. Opcode n-grams are extracted where each n-gram was composed of n consecutive opcode in the assembly representation of a given executable file. Byte n-grams are also built in a similar way using byte sequences. PE header represents a collection of metadata related to a Portable Executable file.

2.2.2 Dynamic Malware Analysis

Dynamic malware is malware in motion. Dynamic malware analysis is also called behavioral analysis. Dynamic malware analysis is the process of extracting information from malware while it is running. Unlike the limited view static analysis provides about malware, the dynamic analysis offers a more in-depth view of the malware's functions because the file is detected by monitoring its system interaction, behavior, and effect on the machine while executing it in the real environment (like sandboxes). Sandbox is an isolated environment, which is pre-built to record all actions of the process. This method has some drawbacks as not all malware executes its malicious activities right after being started. The main disadvantage of dynamic analysis is the detection of the simulated environment. Malware can often detect that they are being run in a simulated environment and change their behavior accordingly. Therefore, these methods are time-consuming in general. Malware sandbox, malware test environment, and dynamic analysis lab are different names given to a system where malware is executed for analysis.

2.2.3 Hybrid Malware Analysis

The hybrid malware analysis method is proposed to eliminate the limitations of static and dynamic malware analysis. A combination of both static and dynamic processes is called hybrid malware analysis. First, it analyses the signature specification for any malware code and then combines it with the other behavioral parameters to complete malware analysis.

2.3 Malware Detection Techniques

There are mainly two types signature-based and behavior-based detection. Signature-based detection maintains the data of signature and detects malware by comparing patterns against that data. A signature is a unique feature set that distinguishes executables like a fingerprint. Behavior-based detection(heuristic-based or anomaly-based) analyzes behavior and file characteristics to determine the malware.

2.4 PE File Format

A portable executable (PE) file is the most used file format due to the wide use of the Windows operating system. A PE file is a data structure that contains the information necessary for the Windows OS loader to manage the wrapped executable code. There are no mandatory constraints in many fields of PE files and contain many redundant fields and spaces, creating opportunities for malware propagation and malware attacks. A PE file contains the PE file header, section table, and section data. They have many valuable pieces of information for malware analysts, including imports, exports, time-date stamps, subsystems, sections, and resources. The following is the basic structure of a PE file.

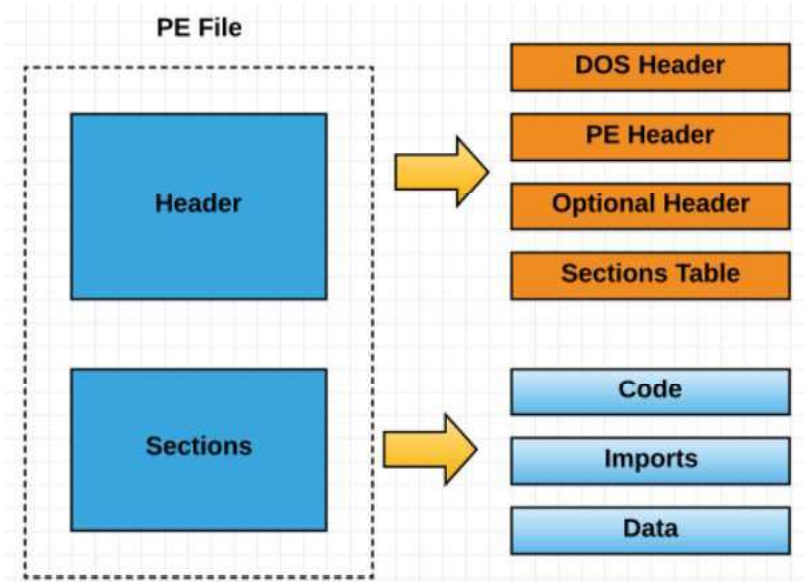


Figure 2.2: PE File Format

The header section contains information about the code and its application type, necessary library or kernel functions, and how much space is needed. In every PE file, DOS Header starts with the first 64 bytes. DOS can validate the executable and run it in the DOS stub mode. PE Header contains information, including the location and size of the code. PE sections include the main contents of the file.

The section segment contains the following components.

- The **.text section** contains the instructions executed by the central processing unit, and it is the only section that includes code.
- The **.rdata section** holds read-only data that is globally accessible within the program. Sometimes it contains a .idata and .edata section, which stores the export and import information. Its data is stored within the .rdata section if these do not exist as an independent.
- The **.data section** stores global data accessed throughout the program. Local data is not stored in any section of the PE file.
- The **.rsrc section** stores the resources that are used by the programs. Strings are generally stored in this section to serve the multi-language support.

2.5 Key related research

Walenstein [18] extracted PE header and body information base on static analysis. They evaluated the effect of information to distinguish benign and malicious files in their work. They extracted all PE header information, DLL names, and API function names called within those DLLs contained in a PE file.

Tina Rezaei et al.(2020) [14] proposed a model used in real-time malware detection systems. They created a model to detect malware with high speed and accuracy with only nine features with a significant difference in PE header and PE file structure.

Ajit Kumar et al. [10] proposed an ML-based model. They concentrated on creating an integrated feature set by combining raw features and derived features of PE files. The model will have low computation overhead and give better accuracy. They used Extra Trees Classifier for feature importance and trained various ML classification algorithms like decision tree, K-NN, random forest, and logistic regression and compared their results.

In 2009, Shabtai et al.[16] provided a taxonomy for malware detection using ML algorithms, some feature types, and feature selection techniques used in the literature. They focussed on the feature selection techniques like document frequency, gain ratio, and classification algorithms like Artificial Neural Networks, Bayesian Networks, Naïve Bayes, K-NN. They also experimented on how ensemble algorithms can be used to combine a set of classifiers.

In the model, Fuyong et al.(2017)[12] proposed, they selected k n-grams as features based on information gain. To classify malware and benign files, they took the average of the feature vector's attributes and classified them based on similarity in feature vectors of unknown samples and their average vectors.

Abou-Assaleh et al. [9] explored the idea of automatically detecting new malicious code based on byte N-gram analysis. After collecting N-grams from training data, a class profile is created with the normalized frequency of the most frequent N-grams. To achieve effective detection, they applied the CNG method based on the profiles of class representations.

In 2012 Shabtai et al.[17] proposed a framework for malware detection using features of opcode n-gram where n ranges from 1 to 6. They experimented to identify the best term representation(Term Frequency (TF) or Term Frequency-Inverse Document Frequency), to determine the n-gram size, and find the optimal K top n-grams and feature selection method. They evaluated the performance of various machine learning algorithms.

In 2010, Sami et al.[15] proposed a three-step framework to classify PE files based on API call usage. They extracted the list of imported API calls by analyzing PE files. They reduced the feature vector using the Clospan algorithm, and the Random Forest model is trained with the feature set.

In 2011, Eskandari and Hashemi [11] presented an approach to detect metamorphic malware through their Control Flow Graphs. They disassembled the PE files and applied a preprocessing algorithm to assembly files and API calls, and the vector representation is created from the resulting sparse graph. Their model achieved 97% accuracy using the Random Forest classifier.

Hassen and Chan (2017) [13] proposed a method to extract a vector representation of the linear time function call graph based on function clustering. They used the minhash hashing technique and improved the speed of measuring similarity between functions. Using function clustering from minhash signatures, they converted graph representation into a vector representation to compute similarity. Due to this, non-graph features can be used combinedly with graph features.

2.6 Problem formulation

In the past decade, the malware industry has overgrown. Detecting malware attacks using traditional methods became difficult due to polymorphism, metamorphism, and soon. The major part of protecting a device from a malware attack is identifying whether a given piece of file/software is malware. But, due to the growth of malware, detecting it with a single feature has become impossible, which most of the previous works did. So, in this project, we are doing hybrid static malware analysis, which includes byte n-grams, opcode n-grams, and PE headers as features for the training model to detect malware to improve efficiency.

Chapter 3

Methodology

3.1 Proposed hypothesis

In this project, we are doing hybrid static malware analysis. After downloading the datasets, we are going to extract the PE header features, byte n-grams features(n=1, 2) from byte files, opcode n-gram features (n=1, 2, 3) from asm files, and Image features (converted from asm files). After doing feature selection using random forest, ExtraTrees classifier, the hybrid feature set is formed by merging data. We will train different classifiers with the formed hybrid feature set and compare them to find the best model for classifying malware and benign files.

3.2 Mechanism/Algorithm

In this project, different machine learning algorithms are used.

- **Logistic regression** is a classification algorithm for categorical variables. It is most suitable when we need probabilistic results.
- **KNN Algorithm** is a method for classifying cases based on similarity to other cases.
- **Support Vector Machine** is a supervised algorithm that can classify cases by finding a separator.
- **Decision Tree Algorithm** The basic idea behind a decision tree is to map out all possible decision paths in tree form.

- **Random Forest** is a classification algorithm that consists of many decisions trees. It uses feature randomness and bagging for building each tree to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

- **Gradient Boosting** is a powerful ensemble machine learning algorithm that uses decision trees. The idea behind "gradient boosting" is to take a weak hypothesis or weak learning algorithm and make a series of tweaks to it that will improve the strength of the hypothesis. It produces ensemble of weak prediction models, typically decision trees.

- **Extra Trees classifier** uses a meta estimator which fits randomized decision trees (extra-trees) on various sub samples of the dataset. To improve the accuracy and control over-fitting it uses averaging.

For feature selection we used random forest and Extra Trees classifier. Random forest chooses the optimum split while Extra Trees chooses it randomly. However, once the split points are selected, the two algorithms choose the best one between all the subset of features. Extra Trees adds randomization but still has optimization.

3.3 Analytical validation

If a single characteristic feature like PE header is used to train the model, then the data we are testing should contain information PE header only. If the file is a byte file without proper header information, then it is useless. So, in this project, we are merging different characteristics and creating a hybrid model. We are integrating byte n-grams, opcode n-grams, PE header, Image features. So, even if information regarding one characteristic is vague, we can detect the file as malicious or not without much trouble with other characteristics.

Chapter 4

Experiments and results

4.1 Extracting PE Header information

After Extracting PE header information from the files the output will be a dicitonary containing the values of the header as given below.

{'AddressOfEntryPoint': 12803, 'BaseOfCode': 4096, 'BaseOfData': 61440, 'Characteristics': 258, 'CheckSum': 130103, 'DllCharacteristics': 33088, 'ExportNb': 0, 'FileAlignment': 512, 'ImageBase': 4194304, 'ImportsNb': 84, 'ImportsNbDLL': 2, 'ImportsNbOrdinal': 0, 'LoadConfigurationSize': 72, 'LoaderFlags': 0, 'Machine': 332, 'MajorImageVersion': 0, 'MajorLinkerVersion': 10, 'MajorOperatingSystemVersion': 5, 'MajorSubsystemVersion': 5, 'MinorImageVersion': 0, 'MinorLinkerVersion': 0, 'MinorOperatingSystemVersion': 1, 'MinorSubsystemVersion': 1, 'NumberOfRvaAndSizes': 16, 'ResourcesMaxEntropy': 5.31799550037944, 'ResourcesMaxSize': 9640, 'ResourcesMeanEntropy': 4.6933645770178005, 'ResourcesMeanSize': 2235.777777777778, 'ResourcesMinEntropy': 2.786195502566973, 'ResourcesMinSize': 104, 'ResourcesNb': 9, 'SectionAlignment': 4096, 'SectionMaxVirtualsize': 54298, 'SectionsMaxEntropy': 6.445974634168234, 'SectionsMaxRawsize': 54784, 'SectionsMeanEntropy': 4.748755211336212, 'SectionsMeanRawsize': 19148.8, 'SectionsMeanVirtualsize': 20810.8, 'SectionsMinEntropy': 2.120428447675006, 'SectionsMinRawsize': 4096, 'SectionsMinVirtualsize': 3814, 'SectionsNb': 5, 'SizeOfCode': 54784, 'SizeOfHeaders': 1024, 'SizeOfHeapCommit': 4096, 'SizeOfHeapReserve': 1048576, 'SizeOfImage': 118784, 'SizeOfInitializedData': 40960, 'SizeOfOptionalHeader': 224, 'SizeOfStackCommit': 4096, 'SizeOfStackReserve': 1048576, 'SizeOfUninitializedData': 0, 'Subsystem': 3, 'VersionInformationSize': 0}

Figure 4.1: PE Header Extracted Sample

There are a lot of header features. So, after doing normalization, we did feature selection using the

Extra Trees Classifier and extracted the most significant features. We got 14 important features. [DllCharacteristics, Characteristics, Machine, VersionInformationSize, SectionsMaxEntropy, Subsystem, ImageBase, MajorSubsystemVersion, ResourcesMinEntropy, SizeOfOptionalHeader, ResourcesMaxEntropy, SectionsMinEntropy, MajorOperatingSystemVersion, SectionsMeanEntropy.]

We Extracted these 14 features from all files after extracting PE headers.

4.2 Byte Files

4.2.1 Feature Extraction and Feature selection from Byte files

After downloading data from kaggle, sample of the byte file looks as shown below.

```
00401000 56 8D 44 24 08 50 8B F1 E8 94 B7 01 00 C7 06 74
00401010 BE 54 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020 C7 01 74 BE 54 00 E9 9E B8 01 00 CC CC CC CC CC
00401030 56 8B F1 C7 06 74 BE 54 00 E8 8B B8 01 00 F6 44
00401040 24 08 01 74 09 56 E8 E4 BA 01 00 83 C4 04 8B C6
00401050 5E C2 04 00 CC CC CC CC CC CC CC CC CC CC CC CC
00401060 8B 44 24 08 C3 CC CC CC CC CC CC CC CC CC CC CC
00401070 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
00401080 8B 44 24 08 66 8B 08 8B 54 24 04 66 89 0A C3 CC
00401090 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010A0 8B 44 24 04 8D 50 02 66 8B 08 83 C0 02 66 85 C9
004010B0 75 F5 2B C2 D1 F8 C3 CC CC CC CC CC CC CC CC CC
004010C0 8B 44 24 08 8A 08 8B 54 24 04 88 0A C3 CC CC CC
004010D0 8B 44 24 04 8D 50 01 8A 08 40 84 C9 75 F9 2B C2
004010E0 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
```

Figure 4.2: Byte File Sample

The raw data contains the hexadecimal representation of the file's binary content, without the PE header. After adding byte file size as feature we created a byte vocab as shown below and extracted byte uni-gram and byte bi-gram features.

```
byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,
29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,5
6,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,
84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b
1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,
df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

Figure 4.3: Byte vocab

4.2.2 Extracting byte bi-grams

Byte bi-gram features look like ['00 00', '00 01', '00 02', '00 03', '00 04', '00 05', '00 06', '00 07', '00 08', '00 09', '00 0a', '00 0b',.....].

Here we got 66049 byte bi-gram features alone. So, after doing normalization, we performed feature selection using random forest and selected top 200 features in byte bi-gram. Now, we have byte file features and top 200 byte bi-gram features including byte files size as a feature.

4.3 Asm Files

4.3.1 Feature Extraction and Feature Selection from Asm files

Sample of the asm file looks as shown below.

```
.text:00401170                                          locret_401170:                          ; CODE XREF: .text:00455FC1□j
.text:00401170 C3                                                       retn
.text:00401170                                          ; ---------------------------------------------------------------------------
.text:00401171 CC CC CC CC CC CC CC CC CC CC CC CC CC CC                 align 10h
.text:00401180
.text:00401180                                          unknown_libname_2:                      ; CODE XREF: .text:00455A3A□j
.text:00401180 8B C1                                                    mov     eax, ecx        ; Microsoft VisualC 2-11/net runtime
.text:00401182 C7 00 00 00 00 00                                        mov     dword ptr [eax], 0
.text:00401188 C3                                                       retn
.text:00401188                                          ; ---------------------------------------------------------------------------
.text:00401189 CC CC CC CC CC CC CC                                      align 10h
.text:00401190
.text:00401190                                          loc_401190:                             ; CODE XREF: .text:00455C6F□j
.text:00401190 8B 44 24 04                                              mov     eax, [esp+4]
.text:00401194 8B 10                                                    mov     edx, [eax]
.text:00401196 89 11                                                    mov     [ecx], edx
.text:00401198 C2 04 00                                                 retn    4
.text:00401198                                          ; ---------------------------------------------------------------------------
.text:0040119B CC CC CC CC CC                                            align 10h
.text:004011A0
.text:004011A0                                          loc_4011A0:                             ; CODE XREF: .text:00455B75□j
.text:004011A0 8B 44 24 10                                              mov     eax, [esp+10h]
.text:004011A4 8D 0C 00                                                 lea     ecx, [eax+eax]
.text:004011A7 89 4C 24 10                                              mov     [esp+10h], ecx
.text:004011AB 8B 44 24 08                                              mov     eax, [esp+8]
.text:004011AF 8D 0C 00                                                 lea     ecx, [eax+eax]
.text:004011B2 89 4C 24 08                                              mov     [esp+8], ecx
.text:004011B6 E9 7F B9 01 00                                           jmp     _memcpy_s
.text:004011B6                                          ; ---------------------------------------------------------------------------
.text:004011BB CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC+          db 15h dup(0CCh)
.text:004011D0                                          ; ---------------------------------------------------------------------------
.text:004011D0
.text:004011D0                                          loc_4011D0:                             ; CODE XREF: .text:00455E6D□j
.text:004011D0 8B 44 24 10                                              mov     eax, [esp+10h]
.text:004011D4 8D 0C 00                                                 lea     ecx, [eax+eax]
.text:004011D7 89 4C 24 10                                              mov     [esp+10h], ecx
.text:004011DB 8B 44 24 08                                              mov     eax, [esp+8]
.text:004011DF 8D 0C 00                                                 lea     ecx, [eax+eax]
.text:004011E2 89 4C 24 08                                              mov     [esp-8], ecx
.text:004011E6 E9 CC B9 01 00                                           jmp     _memmove_s
.text:004011E6                                          ; ---------------------------------------------------------------------------
.text:004011EB CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC+          align 20h
```

Figure 4.4: ASM File Sample

From the asm files we extracted the following features and also added asm file size as a feature.

```
prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']
 opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp',
            'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
keywords = ['.dll','std::',':dword']
registers= ['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
```

Figure 4.5: ASM File features

4.3.2 Extracting opcode n-grams(n=2,3)

Now, we extracted opcode bi-gram and opcode tri-gram features using opcode list as vocab. Opcode bi-gram features look like ['jmp jmp', 'jmp mov', 'jmp retf', 'jmp push', 'jmp pop', ,.....'movzx jz', 'movzx rtn', 'movzx lea', 'movzx movzx']
Opcode Tri-gram features look like ['jmp jmp jmp', 'jmp jmp mov', 'jmp jmp retf', 'jmp jmp push', 'jmp jmp pop', ...]

we got lot of features so after doing normalization, we did feature selection using random forest and selected top 100 features from each.Now, we have asm file features and top 100 opcode bi-gram features and top 100 opcode tri-gram features.

4.3.3 Image Feature Extraction from asm Files

We converted asm files into png format and extracted first 200 image pixels.

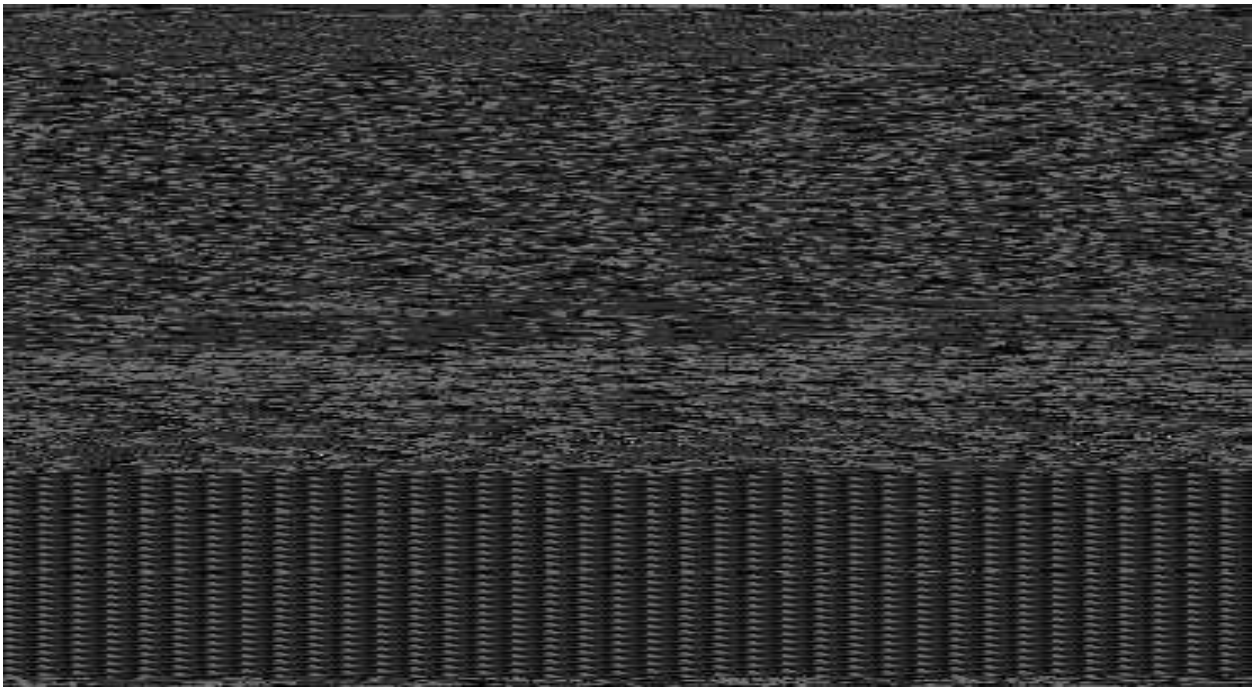


Figure 4.6: ASM Image

Now we have asm file features and top 100 opcode bi-gram features and top 100 opcode tri-gram features and 200 Image pixels.

4.4 Results

The total dataset contains 43003 files in which 23000 are malware and 20003 are benign files. Finally, we have 14 PE header features + Byte file features(unigram+byte file size) + 200 Byte Bi-gram features + Asm fie features(opcode uni-gram+ Asm file size + keywords,registers,prefixes) + 100 Opcode Bi-gram features + 100 Opcode Tri-gram features + 200 Image Pixels.

After merging these features and creating train and test sets, we trained these models(in the below table) got the results as shown in the below table.

Model Comparision			
Model	Jaccard Index	F1-score	Accuracy
Random Forest	0.992031	0.995699	0.995116
knn	0.985392	0.992096	0.993721
Decision Tree	0.978090	0.988022	0.992093
Support vector Machine	0.596193	0.561446	0.630391
Gradient Boost Classification	0.982899	0.990698	0.990698

Figure 4.7: Result

From the above, Random Forest is the best suited model and has high accuracy than other models. It got high accuracy of 99.5116%. This work demonstrates that Random Forest is the best algorithm for detecting malicious programs.

Chapter 5

Discussions and conclusion

5.1 Salient Features of this Project

- We created a feature set with significant features from hybrid static malware analysis of PE files.
- We can detect malicious files more accurately and fast from this model before executing the file, as the static analysis does not require execution.
- Even though files do not contain one particular type of characteristic, we can classify it based on the other characteristics.

5.2 Limitations

- This model fails to predict if the data provided doesn't contain the features and match the format the model is trained with.
- Even though this is fast, it can't give depth-in view of malware characteristics. So, it may not detect complex malware types.

5.3 Future scope

- We can include even more static characteristics(like control flow graphs) and perform feature selection using more different methods(like chi-square distribution etc).
- We can include dynamic characteristics along with static characteristics to detect even more complex malware types.
- We can create a website and host it on web for real time analysis of files on the cloud.

Bibliography

[1] https://docs.microsoft.com/en-us/windows/win32/debug/pe-format.

[2] https://learning.oreilly.com/library/view/mastering-machine-learning.

[3] https://link.springer.com/chapter/10.1007/978-3-030-62223-7_10.

[4] https://link.springer.com/chapter/10.1007/978-3-319-73951-9_2.

[5] https://virusshare.com/.

[6] https://www.ijert.org/research/malware-and-malware-detection-techniques-a-survey-ijertv2is120163.pdf.

[7] https://www.kaggle.com/c/malware-classification/data.

[8] https://www.virustotal.com/.

[9] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In Proceedings of the 28th Annual International Computer Software and Applications Conference. IEEE, 2004.

[10] K.S.Kuppusamy Ajit Kumar and G. Aghila. 'a learning model to detect maliciousness of portable executable using integrated feature set.'. Journal of King Saud University - Computer and Information Sciences, 31(ue 2):252–265, 2019.

[11] M. Eskandari and S. Hashemi. 'metamorphic malware detection using control flow graph mining.'. 06 International Journal of Computer Science and Network Security, 11(12), 2011.

[12] Z. Fuyong and Z. Tiezhu. Malware detection and classification based on n-grams attribute similarity. In 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC, volume 1, page 793–796, 2017.

[13] M. Hassen and P.K. Chan. Scalable function call graph-based malware classification. In CODASPY 17 Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, ACM, page 239–248, New York, NY, USA, 2017.

[14] Tina Rezaei and Ali Hamze. An efficient approach for malware detection using pe header specifications. In 2020 6th International Conference on Web Research (ICWR).

[15] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze. 'malware detection based on mining api calls.'. In Proceedings of the 2010 ACM Symposium on Applied Computing. SAC 10, ACM, page 1020–1025, New York, NY, USA, 2010.

[16] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer. 'detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey.'. Inf. Sec. Tech. Rep, 14(1):16–29, 2009.

[17] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, Y. Elovici, and Feb. 'detecting unknown malicious code by applying classification techniques on opcode patterns.'. Security Informatics, 1(1), 2012.

[18] Andrew Walenstein, Daniel J. Hefner, and Jeffery Wichers. 'header information in malware families and impact on automated classifiers.'. In 5th Int Conf on Malicious and Unwanted Software Malware 2010, page 15 – 22. Nancy, France, 2010.