# Towards Improved Malware Detection using Multilevel Ensemble Supervised Learning

A Thesis

*submitted in partial fulfillment of the requirements for the award of degree of*

**Master of Engineering**

in

**Information Security**

by

**Vidhi**

**801533015**

*Under the supervision of*

**Dr. Niyati Baliyan**

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA 147004

July 2017

# CERTIFICATE

I hereby certify that the work which is being presented in the thesis titled, **Towards Improved Malware Detection using Multilevel Ensemble Supervised Learning**, in partial fulfillment of the requirements for the award of the degree of Master of Engineering in Information Security submitted in **Computer Science and Engineering Department** of Thapar University, Patiala is an authentic record of my own work carried out under the supervision of **Dr. Niyati Baliyan** and refers other researchers' work which is duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree or diploma to this or any other University.

Vidhi

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

(Dr. Niyati Baliyan)
Assistant professor
CSED

# ACKNOWLEDGEMENT

First, I would like to express my deep gratitude towards my supervisor **Dr. Niyati Baliyan** for her invaluable advice and encouragement at every step of my Master's program. Without her unfailing support and belief in me, this would not have been possible. Her contribution to this thesis goes well beyond her role as an academic supervisor and includes constant support on a personal level, without which this journey might have become very difficult, and for this, I am truly thankful. She is a great mentor for my life as well.

I would like to acknowledge Dr. Maninder Singh, Head, and Dr. Shreelekha Pandey, P.G. Coordinator, Computer Science and Engineering Department, Thapar University for the motivation and the inspiration in the completion of this thesis.

I will be failing in my duty if I do not express my gratitude towards Dr. S.S. Bhatia, Dean of Academic Affairs, Thapar University for making provisions of infrastructure such as library facilities, computer labs equipped with Internet facilities, immensely useful for the learners to equip themselves with the latest knowledgde in the field. I want to express my sincere thanks to Dr. Prashant Singh Rana, Assistant Professor, Thapar University for his support and motivation.

Last but not least, I would like to thank my parents for their love and encouragement, without their blessings none of this would have been possible. I would also like to thank my close friends for their constant support.

Vidhi

# ABSTRACT

Malware is a computer program or a piece of software that is designed to penetrate and detriment computers without the owner's permission. There are different malware types such as viruses, rootkits, keyloggers, worms, trojans, spyware, ransomware, backdoors, logic bomb, etc. Volume, variant, and speed of propagation of malware are increasing every year. Antivirus companies are receiving thousands of malware on the daily basis, so detection of malware is a complex and time-consuming task.

Traditional signature based and anomaly based malware detection techniques are still in use. However, the signature based detection system fails for new unknown malware. In case of anomaly based detection, if the malicious activity behaves like a normal activity, the detection treats it as a normal one. Today's attackers are using various obfuscation techniques which has become a great challenge for the detectors to detect the malicious content with the traditional malware detection techniques.

In this research, multilevel ensemble classification approach is introduced to detect malware using the concept of API Calls usage frequency in a portable executable format to find accuracy, sensitivity, specificity, misclassification rate, Kappa, precision, false positive rate and false negative rate. The results show that the proposed multilevel ensemble approach can classify malware with 94.67% accuracy and 4.79% False Positive Rate.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| API | Application Program Interface |
| DLL | Dynamic Link Library |
| DT | Decision Tree |
| FN | False Negative |
| FNR | False Negative Rate |
| FP | False Positive |
| FPR | False Positive Rate |
| kNN | k-Nearest Neighbor |
| LDA | Linear Discriminant Analysis |
| NB | Naïve Bayes |
| NN | Neural Network |
| OEP | Original Entry Point |
| RF | Random Forest |
| SVM | Support Vector Machine |
| TN | True Negative |
| TNR | True Negative Rate |
| TP | True Positive |
| TPR | True Positive Rate |

# Chapter 1

# Introduction

Nowadays, one of the main dangers to computer system's security is the malware, a program that installs in the system without system user's knowledge in order to perform some unwanted actions. The malware interrupts computer operations and gathers private information, besides other undesirable behavior.

This chapter provides an overview of the concept of malware, various types of malware and malware detection techniques. It also briefly describes the research motivation for malware classification and the organization of the rest of the thesis.

## 1.1 Malware: Definition

The current decade of Information Technology and Data Science is suffering from threats of malware, i.e., malicious software. Malware is a code or software installed on a system without the owner being aware of it with a purpose to steal personal data from the system. This, in turn, threatens the computer's security, it is dangerous because computers are widely used in the domain of education, communication, medicines, banking, entertainment, etc. The malware can reach into the systems in many manners; the most frequent way is to download from the Web. Once the malware is able to enter the systems, based upon the functioning of malware, they infect the system. Sometimes, the malware does not completely infect the system, rather, they affect the system's performance or create an overload of processes. Malware is designed to accomplish some targets such as gathering sensitive information, accessing the personal computer system and altering data.

## 1.2 Malware Types

Malware are categorized based on the function performed by malware methods. The characteristics that distinguish malware categories are: self-replication; parasitic nature and population growth. The self-replication of malware is increased by generating a replica of self. The overall increase in the number of malware instances, due to self-replication property of malware is called population growth. Malware that do not self-replicate have zero population growth, although, malware having zero population rate may self-replicate. Parasitic malware are dependent malware, they are dependent on other executable files to exist. The various types of malware are:

### 1.2.1 Virus

A computer virus is a collection of lines of code, it adds itself to other applications. A virus needs a program or software called *host* in order to damage the system. A virus is dangerous because it can consume the entire system memory or even halt the system. For example, to execute in a computer system, a virus can add itself to some software or file (e.g. an audio file). Opening that file could activate the virus that may, for example, duplicate itself, degrade the system's performance or disable malware detectors enabled on the computer system.

### 1.2.2 Worm

Worm is a self-replicating software that runs malicious code, irrespective of another program. A major difference in a worm from a virus is that the former does not require human intervention for damaging the system.

### 1.2.3 Trojan Horse

Trojan Horse is a malignant code which appears to execute some useful activity by misleading users (e.g., play an audio file), while performing some unauthorized action (e.g., crashing the system, modifying or deleting the data, spreading malware across the network). Sometimes trojans are designed to be more irritating

than performing harmful activities (like playing annoying music, adding silly active desktop icons) or sometimes they can also cause some deliberate damage to the system (deleting the files).

### 1.2.4 Rootkit

Rootkit is a malicious computer software, which is deliberated to enable access to a system or its applications that would not otherwise be allowed (for example, to an unauthorized user) and basically designed to hide the malicious content of the code from the virus removal programs such as antivirus and firewalls. Rootkits enable root access to the computer which means they run at the lowest level of a system.

### 1.2.5 Spyware

Spyware is a software that aims to gather user's activity information without his/her knowledge and may also send that data to another entity without his/her permission. Spyware is tracking users' movements on the Internet and often serving up pop-up ads to users.

### 1.2.6 Keylogger

Keylogger is a serious form of spyware that can record keystrokes, read cookies and copy the files of a system to gain private information. A keylogger can record any information typed using a keyboard without the permission of the user.

### 1.2.7 Logic Bomb

Logic bomb is a malicious code piece that is deliberately added to an OS or software, in order to perform a malicious activity when certain constraints are met. Logic bomb performs various actions like deleting or modifying data, formatting a hard drive and crashing the system.

### 1.2.8 Backdoor

Backdoor is a means of accessing a system without going through the normal access routines. Virus or even sometimes the legitimate program can install a backdoor.

### 1.2.9 Adware

Adware is similar to spyware as they both gather information about the victim movements and display unwanted advertisements. Adware is mainly marketing-focused and pops up unwanted advertisements or redirects a victim's search request to certain advertising web sites.

### 1.2.10 Ransomware

Ransomware is a kind of malware that stop users from accessing their system, demands a ransom in exchange for getting the access of the system. E-mail attachments, malicious programs, and compromised websites are the mainly used to spread the ransomware.

### 1.2.11 Encrypted Malware

Encrypted malware comprises encrypted body and decryption code. The encrypted body and a key is XORed with the intention of making it tough to be discovered. The encrypted malware uses a different key per infection to make it unique and hide its signature so that it cannot be detected. The encrypted malware may be detected by examining its decryptor since the decryption is always same [1].

### 1.2.12 Polymorphic Malware

Polymorphic malware changes its original code with each iteration. In polymorphic malware, thousands of decryptors may be introduced by changing code in the next variant of the malware. Polymorphic malware consists of two parts; the code decryptor and the body of the malware. Whenever the malicious code executes, the mutation engine generates a novel decryptor and combines with it to make a novel form of malware. Polymorphic malware is constructed by employing obfuscation

procedures (inserting dead code, register reassignment, reordering subroutines, code transposition) [1].

### 1.2.13 Metamorphic Malware

Metamorphic malware changes its code to an equivalent one with each iteration. It is a type of polymorphic malware, where a new instance of the malware is created. In metamorphic malware, several obfuscation techniques (shrinking, expansion, permutation etc.) are used to reprogram itself into a newly transformed malicious code, i.e., similar to the original code. The metamorphic malware code is mutated while it transmitted into the network because of its metamorphic nature and it makes signature based detection approach ineffective.

The encrypted, polymorphic and metamorphic malware are most difficult to identify because they use various obfuscation and encryption techniques to hide their existence.

## 1.3 Malware Detection Approaches

Detection is the procedure of finding whether a code is actually benign or it has some malicious content. As malware has various kinds, behaviors and risk levels, the same malware detection techniques cannot be used for all malware. It is infeasible to have only one malware detection approach to efficiently handle all malware. Thus, security researchers are using two popular malware detection techniques for malware analysis. One technique is *anomaly based detection* and the other is *signature based detection* [2].

### 1.3.1 Signature-based Malware Detection Approach

It is sometimes also called misuse detection, since it maintains the database of known software. It is a technique that detects malicious software by comparing the signature against the database. Signature-based approach requires less amount of system resource to identify a malicious software. It has also been asserted that this approach can identify a familiar attack correctly [2]. To detect a malicious con-

tent in the code, the signature-based detection technique searches for a previously defined signature in the code.

The disadvantage of signature-based malware detection approach is the ineffective performance while dealing with unknown attacks since signatures are not provided for these attacks.

### 1.3.2 Anomaly-based Malware Detection Approach

Anomaly-based detection approach analyzes user's activity and the statistics of a process in usual conditions, and it verifies that the system is functioning in accordance with pre-established normal behavior. When an activity leads to dissimilar nature from the usual recognized behavior in a system, an attack can be diagnosed. This technique compares the present behavior of the activity with previous settled correct behavior because the efficiency of anomaly-based detection approach totally depends on the capability to recognize the malware patterns accurately, especially unknown malicious programs or software [2]. Anomaly-based detection approach works in two phases: Learning (Training) phase and Monitoring (Detection) phase. During learning phase, the malware detector learns the host machine's normal behavior. The anomaly-based detection approach detects the zero-day attack (a hole in a software which is unknown to its vendor and is exploited by the attacker).

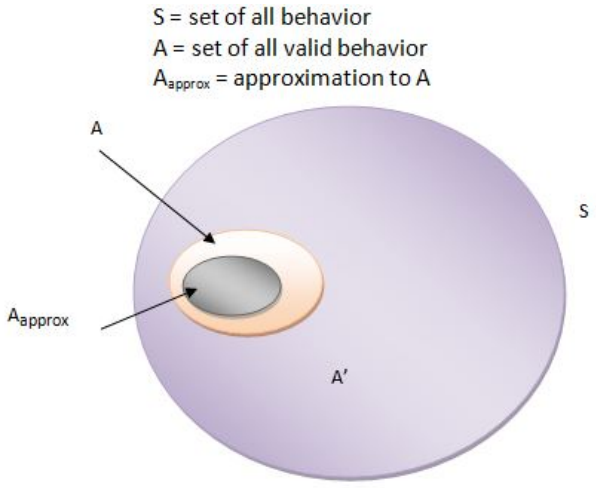



Figure 1.1: Behavior characterization in Anomaly-based Malware Detection

The disadvantage of anomaly-based detection approach is that it requires to update huge amount of information that characterized the system behavior in normal profile. If the malicious activity behaves like the normal activity then anomaly-based detection approach treats it as a normal activity. This is false negative. However, if the normal activity behaves abnormally it is alarming it as malicious thus, in this approach, false positive rate is high [3].

Figure 1.1 shows why anomaly-based detection approach is not sufficient for detecting the malware. As shown, $S$ represents the set of all the actions of a system, $A$ shows the set of all legitimate actions of a system which are derived from a set of non-conflicting requirements, and $A'$ shows all invalid behavior set.

Anomaly-based detection approach tries to approximate the implementation (as implementation covers relative requirements). $A_{approx}$ is an approximation of valid actions of a system that might be tagged as malignant under anomaly-based detection approach [4].

### 1.3.3 Specification-based Malware Detection Approach

This is a category of anomaly-based malware detection approach which uses a predetermined policy. The specification-based malware detection approach compares predetermined profiles (specifies what activity should be considered for a chain of events, particularly, the actions are allow, deny or log) which are specified by the vendors which are based on accepted definitions of benign activity against observed activity to discover deviations. It depends on vendor-created profiles which identify how a particular protocol must and must not be used.

### 1.3.4 Machine Learning-based Malware Detection Approach

This approach involves processing and learning a large number of examples to obtain useful, unpredictable, and previously undefined information, patterns, and trends from huge amount of data, which are saved in databases. This approach extracts features of various activities and then classifies them as normal or malicious. Machine learning techniques are used to group various types of malicious activities into profiles and then use these profiles to identify an attack whenever it arises.

There are three kinds of Machine learning methods: supervised, unsupervised and semisupervised.

- Supervised machine learning technique uses the training data to predict a hidden pattern. The training data is a combination of input variables and output classes or labels. Supervised machine learning models make predictions of the classes for the unseen new data. Various examples of supervised machine learning are classification, regression and attribute prioritization.

- Unsupervised machine learning technique is an attempt to recognize hidden data from given input data without introducing training data, i.e., combination of input variables and class labels. Various examples of unsupervised machine learning are clustering and associative rule mining.

- Semi-supervised machine learning technique is a combination of both labeled and unlabeled data, normally a large collection of unlabeled data with small volume of labeled data. Various examples of semi-supervised machine learning are graph-based methods and heuristic approaches.

## 1.4 Malware Analysis

Before creating the new malware's signatures, it is necessary to analyze the malicious activities and malware's capabilities can be detected either by inspecting the malware code or by running the malware code in a safe environment like virtual environment or in a sandbox [5]. Figure 1.2 [7] depicts the relation among the various types of malware detection techniques and analysis approaches. Both detection techniques can use any one of the three different malware analysis techniques, i.e., static, dynamic, or hybrid analysis technique.

### 1.4.1 Static analysis

It uses structural or syntactical properties of the process under inspection to obtain its maliciousness. A static analysis technique tries to detect malicious content without actually running the program [7]. In static analysis approach, the program

is decomposed by using several reverse engineering tools and techniques. Static analysis can be done through program analyzer, debugger, and disassembler.

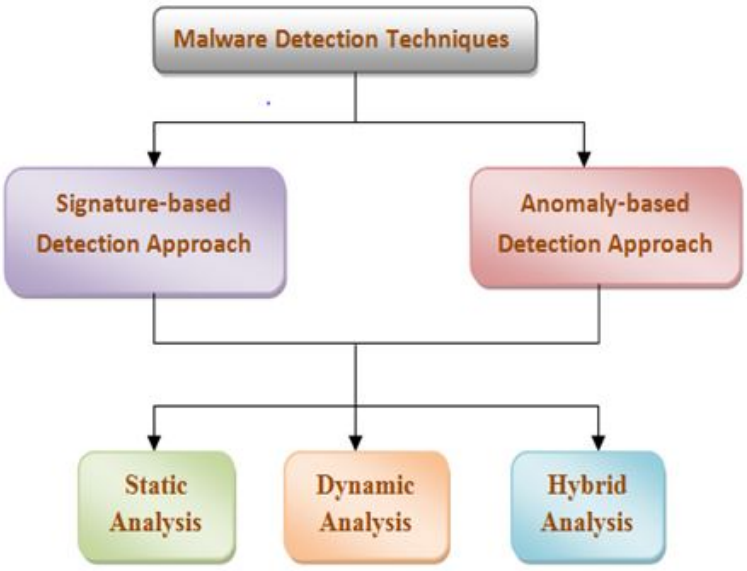



Figure 1.2: Classification of various malware analysis and detection approaches

### 1.4.2 Dynamic analysis

Dynamic analysis tries to detect malicious behavior of the program while the program is executing or after program execution. It can be done by monitoring system's function calls, tracking the data flow, analyzing function variables and tracing the instructions. Generally, a virtual machine is used for dynamic analysis; the abnormal application is generally executed in a virtual environment [7]. If the application behaves abnormally, it is defined as malicious.

### 1.4.3 Hybrid analysis

Hybrid analysis combines the above-mentioned two approaches. It first inspects the code, and then checks for the malicious signature. If malicious signature is present in the code, then it examines the behavior of the code by executing the code. Hence, hybrid malware analysis technique combines the advantages of both the dynamic analysis and static analysis techniques [4].

## 1.5 Research Motivation

Malware is malicious software installed on a system without the owner's cognizance with a purpose to damage or steal data from computer system. There are several kinds of malware such as Trojan horse, ransomware, Virus, Logic bomb, Adware, Worms, Backdoors etc. which can reach into the systems in various ways, and after entering into the systems, based upon their functioning they infect the system. The malware signature creation is very difficult task for antivirus vendor because of encrypted malware. In the era of malware detection approach zero-day malware (malware whose signature is not available and which is unknown to its vendor) increases one more challenge. Researchers have used various reverse engineering techniques to create malware signature but they are still facing problems because of the encryption and obfuscation techniques [1]-[3], [5]. The various obfuscation techniques are Instruction Substitution, Dead Code Insertion, Code Transportation and Register Renaming. The malware creator uses these obfuscation techniques to modify the code of the malware so that it would be hard for malware analyzer to understand the malicious intent.

There are various traditional malware detection approaches, i.e, signature based, specification based and anomaly based malware detection are used to detect malware. The signature based detection system (antivirus, and firewalls, etc. which try to compare the signatures of the malicious activity with the known signatures of the malware) fails to detect the new unknown malware. We need systems that do not have to check for updates of various signatures and are likely to detect unknown malicious programs and prevent the destruction from the attack. However, today, the attackers pack the malicious code in such a way that it is very difficult for the signature-based malware detectors to detect the malware. Therefore, it becomes a tough job for the signature-based malware detectors to identify the malware. One such method is to use the anomaly based malware detection method which analyzes the behavior of the activity and detects previously unknown malware. In anomaly-based malware detection approach, it raises the alarm signal even when the antivirus analyzes an effort to modify a file on the Web; however, there is a probability of false positives. In case of specification-based malware de-

tection system, a set of rules is designed to detect malicious behavior of an activity.

After studying the current problems of malware detection, we define a novel multi-level ensemble approach to find malware on the bases of the Application Program Interface (API) Calls. Moreover, we use supervised machine learning classifiers to classify the malware into six categories, namely, Adware, Backdoor, Benign, Trojan Horse, Virus and Worm.

## 1.6 Thesis Outline

Rest of the chapters in this thesis are organized as follows.

**Chapter 2**- This chapter briefly shows the literature review in the domain of Machine Learning for API Calls based Malware Detection.

**Chapter 3**- Here, the problem statement and objective for this research have been outlined.

**Chapter 4**- This chapter explains proposed framework for malware detection. It discusses the research methodology for malware detection including executable portable Selection, Preprocessing, Feature Extraction, Feature Selection and Classification.

**Chapter 5**- This chapter discusses performance evaluation of multilevel ensemble model, namely, Decision Tree (DT), Random Forest (RF), Linear Discriminant Analysis (LDA), Neural Network (NN), Support Vector Machine (SVM) and k-Nearest Neighbor (kNN). It briefly explains about performance evaluation parameters such as Accuracy, Kappa, Misclassification rate, Sensitivity, Specificity, Precision, False Positive Rate (FPR) and False Negative Rate (FNR)

**Chapter 6** This chapter presents the conclusion and contribution of this research work. It also mentions possible extensions to our research work in the future.

# Chapter 2

# Literature Review

For unknown malware detection and classification it is important to understand the malware features and various techniques used to classify them. In this chapter, the details about API Calls and survey of existing work related to malware classification techniques based on API Calls is provided.

## 2.1 Monitoring Behavior using API Calls

API is a component which offers application developers a simple interface for accessing the resources of a system. API Calls represent the interaction between the OS and a program. For capturing the API Calls of a program, debugging and hooking are the two methods. For example, the network API has functions to open and close sockets, and to send and receive the data using open sockets. Similar to the system calls which are used to observe the behavior of a program, API Calls made by a program are also used to observe the working of a program. API call is one of the most attractive ways that can reflect the behavior of a program like malware.

Paul and Kumar [8] presented a dynamic analysis of malware framework which represented the behavior artifacts which are discovered during the running of the malware because code obfuscation and packing approaches degrade the performance of static analysis. The agent.py (contain an agent folder which has a python script) is applied to the configuration of each guest OS (WindowsXPSP3) and the agent.py file must be copied to the startup folder of the guest OS so that malware

executes the analyzer which is contained in startup folder, whenever the malware is invoked.

Alazab *et al.* [9] implemented a method to analyze and classify the behavior of API function Calls automatically based on the malicious content which is hidden within any packed program. This approach uses four steps: malware unpacking, disassembling, API Calls extraction and API Call mapping and feature analyzing for implementing a fully automated system for analyzing the malware before actually executing it. This method identifies six main categories (copy/delete file, search file to infect, get file information, write/read file, change file attribute , move file) which are based on the suspicious behavior of API call features.

Choi *et al.* [10] proposed a static birthmark scheme which use Import Address Table notion, which is used to recognize Windows PE executables. After analyzing the Window executables the birthmark is obtained which is stored in feature dataset (used to match the features of various programs). The proposed method also provides software privacy or theft evidence even when code transformations have been applied to the copy by a malicious adversary.

Elhadi *et al.* [11] proposed API call graph based malware detection system. They compute the similarity using Longest Common Subsequence algorithm between the API Call graph samples and input samples using Delphi. The system has detection rate of 98% and FPR of 0%. It removes the complexity of data centric API call graph by selecting paths with the same edge label in the input API call graphs (query graph and data graph).

Qiao *et al.* [12] introduced CBM framework, which connects the various open source software tools to design an automated malware analysis system. In CBM, the API call sequences procured from dynamic analysis of malware are extracted by the Cuckoo to develop a representation method of malware behavior to encode the API call sequences to byte-based sequential data. New API hooks are appended to improve the Cuckoo's monitoring capability.

## 2.2 Machine Learning based Approaches

This approach extracts features of various activities and then classifies them as normal or malicious. Machine learning techniques are used to group various types of malicious activities into profiles and then use these profiles to identify an attack whenever it arises. There are three categories of Machine learning methods: supervised, unsupervised and semisupervised.

### 2.2.1 Supervised Machine Learning

Supervised techniques are implemented by constructing a classifier (trained by the samples). The samples are input variable (A) and a target variable (B) and the classifier learns the mapping function from the input variables to the target variable.

$$B = f(A) \tag{2.1}$$

The main intention is the approximation of the mapping function in a manner that whenever there is new input data (A), then output variable (B) can be predicted for that data. (Naïve Bayes) NB, RF, SVM and Maximum Entropy are most commonly used supervised techniques.

For example, suppose we want to predict whether an exe is malicious or not. Consider a set of data and its size, API Calls, Dynamic Link Libraries (DLLs), and MD5 values. From the MD5 values, we know whether its value is matched with the previous malicious MD5 values or not. From the dlls and API Calls we come to know what operations they perform. Therefore, the problem is merging all these inputs into a model that can predict whether a new exe has malware or not. Figure 2.1 [42] shows the steps of supervised machine learning process.

- **Supervised learning categories**

  *Classification:* Partitions the data into different categories or classes such that it can be recognized. Classification uses a trained set of previously labeled data to classify unknown data.

*Regression:* Maps the input space to the real value domain. For example, regression predicts the demand of an item by using its characterstics.

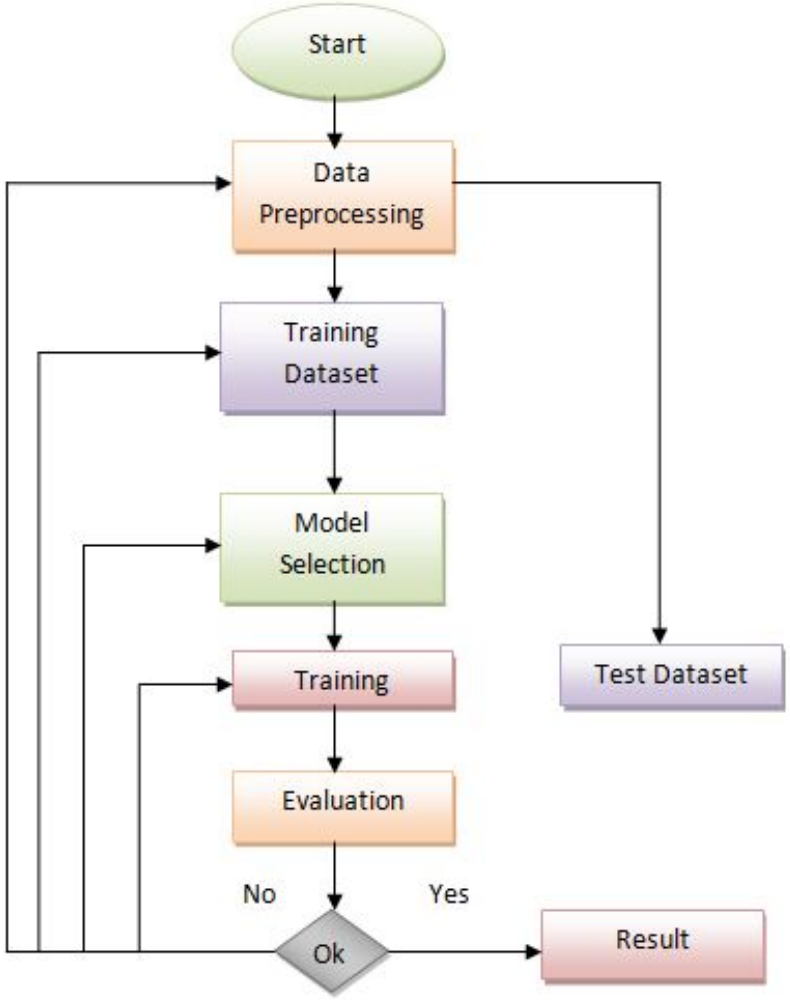



Figure 2.1: Supervised Machine Learning Process

Schultz *et al.* [14] used three types of features- dll used by the exes, dll fuction calls made by the exes and number of different function calls in each dll. Further, they implemented network level email filter that employed the NB, Multi-MB, RIPPER algorithms to discover malicious exes prior to them reaching the users through their mail. Their technique can either block the malware or wrap the malicious exe. They achieve 97.76% accuracy with the multi naïve Bayes method.

Sathyanarayan *et al.* [15] used the static analysis for the API extraction and presented a method for generating signature to identify unknown malware based on their API Calls. They created a single signature for an entire class of a malware

thus reduced the efforts to create signature for every malware. These API Calls' signature were compared with unknown malware's API Calls to identify whether the unknown malware belongs to a particular class or not. The limitation of this approach is that it could not work for packed malware. The obfuscation of malware could affect the accuracy of the system.

Wang *et al.* [16] presented virus identification technique on the basis of the API Calls sequence. Their technique designed Bayes algorithm for the identification and detection of apprehensive working based on function calls sequence by analyzing the common API Calls used by virus detection. The precision of the Bayes algorithm has been validated by using the training and testing samples and found to be 93.01%. They used the dataset of 714 malicious and normal programs.

Tian *et al.* [17] presented an effective classification technique for malware classification based upon the printable strings of the executables. They chose five algorithms, SVM, NB, DT, RF, IB1 to classify the malware and also used the boosting technique to determine if the results improved or not. They statically extracted the features with the IDA PRO after unpacking the malware and using k-fold cross validation. They achieved 97% overall accuracy.

Tian *et al.* [18] extracted the API Calls from both the malicious and benign files for modeling the various behavior patterns of these files and distinguish the malicious files from the benign files and also classify the malware family by using a binary SVM, RF, DT and IBK classification models and compared their accuracy with and without Adaboost classifier. They achieved 97.3% accuracy to classify the malicious files from the benign files. The malware family classification achieved 97.4% accuracy.

Salehi *et al.* [19] proposed to mine malware for detection of malware based on various API function Calls and arguments of these functions in order to identify malware and benign files. The method used two feature types: API Calls and a combination of API Calls with their arguments as features and analyzed these categories' effect on the classification process. In order to decrease the feature count, feature selection algorithm is used. The experiment showed 98.4% accuracy with RF, with 410 features in total and 826 malicious and 385 benign programs.

Natani and Vidyarthi [20] proposed a method to detect malware by mapping API Calls functions to malicious behaviors. The frequency of API Calls was exploited to label file as malicious or benign. The experimental results showed that the TPR at 1000 iterations using EnsembleAdaBoost was 92.31%, however, FPR was extremely high as compared to EnsembleBag. However, at 5000 iterations AdaBoostM1 generated similar result as EnsembleBag. The proposed method also works for rootkits which often use Native API functions. Experiments are carried out on 200 rootkits files and classified them. The results showed that bagging used in ensemble classifier gave better results as opposed to ensemble boosting. The proposed method has API collection, feature extraction and ensembling phases.

Alazab [21] presented a system where from the binary of a program the structural and behavioral features are automatically extracted. The extracted API Calls were used similarity based mining methods and machine learning methods to classify and profile the program files as either normal or malicious. The similarity analysis on large datasets were experimentally performed using the API call sequence. The frequency appearance of these API Calls resulted that this method is effective in profiling malware.

Kawaguchi and Omote [22] classified malware efficiently using initial behavior APIs which were based on 8 malware types (backdoor, send information, downloader, key logger, display add, send spam, copy itself and remote control) API functions (defined by referring the symantec security response) using SVM, C4.5, RF, NB, kNN by using activity logs of APIs of FFRI dataset [23] 2014 by dynamically analyzing them and creates the dataset using Cuckoo sandbox and achieved 83.4% accuracy with RF. The proposed method has three states: API extraction phase (extract APIs which are used by malware and creates a database of these APIs and discard the APIs which occur more than one time), learning phase (if a particular API occurs in a malware the value of the feature vector is 1, otherwise 0) and classification phase (classification of various types of malware are carried out).

Pirscoveanu *et al.* [23] proposed a novel approach for malware classification with the dynamic analysis of malware. This technique relies on features that capture the differences in behavior of 42000 malware samples of adware, trojan horse, po-

tentially unwanted programs and rootkit. The proposed method has three phases data generation, data extraction and classification (uses RF with 160 trees and generated Area Under Curve of 0.98 and 0.9 precision)

Shijo and Salim [24] presented an integrated static and dynamic analysis approach to improve the accuracy of detection of malware. Application signatures are used as unique identification feature for the classification. 7253 static features are extracted in integrated analysis and then classified these features using SVM classifier and RF classifier. The experimental results showed that for static analysis the accuracy is 95.8% and for dynamic analysis the accuracy is 97.1% and 98.7% in the combined analysis.

Fan *et al.* [25] utilized hooking methods to trace and monitor the samples behavior. The behavior records are used to train the classification model and then classify the malware from the benign using these trained models. The classification algorithm used are NB, J48 and SVM. Attribute selection technique is applied to reduce the features so that efficiency can be improved. Detection rate of J48 and NB are upto 95% and SVM shows the worst performance of upto 89% in terms of detection rate.

### 2.2.2 Unsupervised Machine Learning

Unsupervised learning has only input variables and no corresponding output variables. The main aim of unsupervised learning is to model the underlying distribution or structure in the data in order to learn more about the data. Here, unlike supervised learning there are no correct answers and there is no teacher. Algorithms are left on their own to present interesting patterns in the data.

Bayer *et al.* [26] proposed a novel approach that make clusters which are used to discover a partitioning from a set of malware so that their subset show equivalent behavior. The system starts with dynamic analysis of each malware sample which is enhanced with taint tracking and network analysis. The behavior profiles are extracted by abstracting the various system calls, their dependencies and the network activities into a generalized representation consisting of OS objects and OS operations. These profiles are the input for the clustering algorithm (requires less than quadric amount of computation). The proposed approach accurately

recognizes the malware that behaves in similar fashion.

Nakazato *et al.* [27] method focuses on the working of individual thread which are invoked by the original process. In the proposed method, malware samples are classified using clusters according to the malware's API Calls. The frequencies of each API call sequences are calculated and TD-IDF score is given to each API sequence which is used for visualizing the malware characteristics and then malware are classified according to these TD-IDF scores. The proposed method classified 2312 samples with different hash values into 93 clusters. The proposed approach is compared with name based clustering by using the precision and recall. 90% of samples are classified into 20 clusters.

Iwamoto and Wasaki [28] proposed an automatic malware classification methodology based on features of the source code. The samples are first unpacked using unpacker and then API Call sequences are extracted. The presence and absence of particular pair of consecutive API Calls functions in the API sequence graph are compared to identified malware features. Hierarchical cluster analysis which is based on extracted features is used to visualize the grouping of various samples with the same features. To find the degree of similarity among samples, Dice's coefficient is applied.

Kostakis [29] presented a distributed methodology that groups streams of huge graphs. The graph comparison measure is the GED and it is approximated using an adapted version of simulated annealing. Finally, after studying the problem of approximating the statistics of clusters for extracting cluster summaries when only part of all distances are available, experiments are carried out on the proposed algorithms and statistics are provided of a system instance that has clustered and contains over 0.8 million graphs.

Shuwei *et al.* [30] proposed clustering based malware detection method based on Shared Nearest Neighbour (SNN) and fixed length vectors are produced by taking the frequencies of system calls as input. To estimate the exact distance between the samples, Euclidean distance is calculated. SNN algorithm is combined with the DBSCAN (traditional density-based clustering algorithm). This makes it better application in the process of clustering of malware. The proposed method has

three phases: Feature Extraction (calculate frequencies of various system calls), Calculating similarity (using Euclidean distance, kNN and SNN) and Clustering (based on SNN density).

### 2.2.3 Semi-supervised Machine Learning

Semi-supervised learning is a category of supervised learning tasks and techniques that also make use of unlabeled data for training – typically a narrow amount of labeled data with a large amount of unlabeled data. These algorithms can perform well with very few labeled points and a lot of unlabeled points.

Santos *et al.* [31] pointed out that supervised learning requires a considerable number of labeled executables for benign and malicious data and develop a semi-supervised learning methodology for undiscovered malware detection. A semi-supervised algorithm Learning with Local and Global Consistency which learns from both labeled and unlabelled data and gives a solution with respect to the intrinsic structure displayed by both labeled and unlabelled instances is designed. N-gram distribution technique is used to represent executables. They also descover and evaluate the optimal number of labeled instances and affect of this parameter on the accuracy of the model. The major contribution of this research is to reduce the number of required labeled instances while maintaining high precision.

Bisio *et al.* [32] presented a semi-supervised framework, which relies on the use of biased regularization of SVMs as inductive bias. The presented framework has been tested on malware detection problems, where undefined mutations of malicious software pose a challenge. Experimental outcomes proved the efficiency of the semi-supervised framework compared to the SVM-only approach, detection accuracy is improved with the presented framework while false positives are reduced.

Mohaisen *et al.* [33] introduced AMAL, the first operational large scale malware analysis, clustering and classification system. AMAL comprises two subsystems, AutoMal and MaLabel. AutoMal executes the malware samples in virtual environments and gathers memory, file system, registry, and network artifacts, which are used for creating a rich set of features. As opposed to previous work, it merges signature-based and purely behavior-based techniques, thus producing extremely

representative features. Using the artifacts generated by AutoMal, MaLabel creates feature vectors and utilizes them to 1) perform binary classification of malware samples into distinctive, but generic families, and 2) cluster malware samples into various families. It provides high levels of recall, precision, and accuracy, for both clustering and classification.

# Chapter 3

# Problem Analysis

## 3.1 Problem Statement

Malware is malicious software installed on a system without the owner's information with the purpose to damage or steal data from computer system. There are various types of malware such as Virus, Worms, Trojan horse, Logic bomb, Adware, Rootkits, Backdoors etc. Antiviruses are attacked by thousands of malware everyday, moreover, the diagnosis of the malware is typical and takes a lot of time. The malware creator uses obfuscation techniques to modify the code of the malware so that it would be hard for malware analyzer to understand the malicious intent.

We can detect malware by various traditional malware detection approaches such as signature-based, anomaly-based and specification-based detection. The signature based detection system is unable to detect new unfamiliar malware. Additionally, we need systems that do not have to check for updates of various signatures and are likely to detect unknown malicious programs and prevent the destruction resulting from an attack. However, today, the attackers pack the malicious code in such a way that it is very difficult for the signature-based malware detectors to detect the malware. While using anomaly based detection, if the antivirus program identifies an attempt to modify a file or communication over Internet then it will raise an alarm, however still there is a chance of FPR. Moreover, in case of specification-based malware detection system a set of rules is designed to detect the malware. What features should be used to detect the malware and how these features should be represented so that malware can be classified accurately.

A comprehensive literature survey has been carried out (as described in chapter 2) and following research gaps have been identified:

1. Signature-based malware detection system fails to detect unknown malware.
2. Obfuscation techniques makes the malware detection process difficult.
3. Specification and Anomaly-based malware detection system require the malicious code to be executed.

This research proposes design and development of API Calls based predictor to bridge these gaps without executing the malware and thus inhibiting the infection of development environment.

### 3.1.1 Objectives

The main objective of this work is to analyze the API Calls of portable executable file, and unique API Calls which can be leveraged to find unknown malware. It is wholly based on Data Mining based malware detection technique which helps antivirus vendor to detect unknown and polymorphic malware. Moreover, it will reduce the time required to create a signature for malware. The following are prime objective of this work:

1. To explore and analyze different malware detection techniques.
2. To propose and develop API Calls' based malware detector which can be used to detect unknown and polymorphic malware.
3. To design an multilevel ensemble approach which is a combination of DT, RF, SVM, kNN, LDA and NN for detection and classification of malware.
4. To implement proposed method resulting in high accuracy and minimal the false positive rates.
5. To validate the performance of ensemble approach based on accuracy, misclassification rate, FPR, FNR, kappa, sensitivity, specificity and precision.

# Chapter 4

# Proposed Framework

This chapter demonstrates the design of proposed solution through *Towards Improved Malware Detection using Multilevel Ensemble Supervised Learning*. It briefly explains how the malware detection is carried out by using machine learning classification algorithms.

## 4.1 Framework for Malware Detection

This approach comprises Malware and Benign File Selection, Sample Disassembling, Extraction of API Calls of various dlls, Feature Selection and Classification. Model creation includes kNN, LDA, DT, NN, RF and SVM. Figure 4.1 shows architecture of malware detection system.

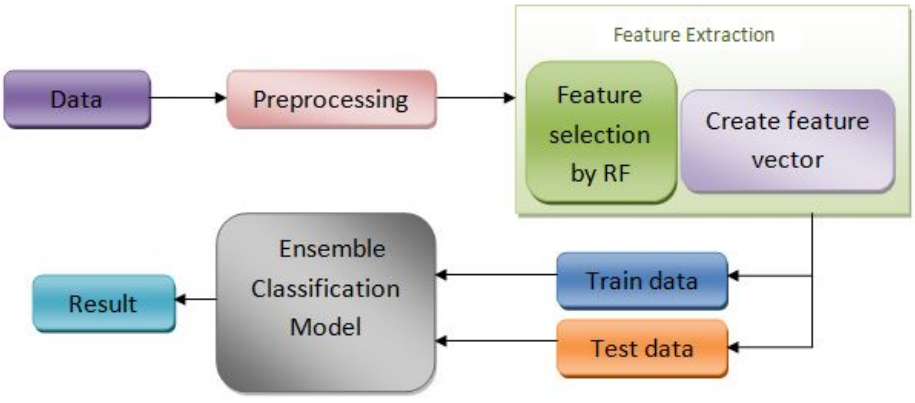



Figure 4.1: Architecture of malware detection system

### 4.1.1 Malware and Benign File Selection

For Malware File Selection, all the malicious files have been downloaded from malware website dasmalwerk [1]. It is a free malware analysis website. All downloaded malware are in portable executable file format. These malware can execute on Windows machine only. All the malicious files have been saved in the Window7 virtual machine. The malware feature extraction process has been done in virtual machine only. In this research, the major focus is on the 5 types of malware: worm, backdoor, virus, adware and trojan horse and benign files.

All the malware have been downloaded from the dasmalwerk site and the type has been tested using virustotal [2]. Virustotal website is owned by Google. It is a free online service that can be used to upload any file and check whether it is malicious or not. Virustotal website contains database of different antiviruses. When a file is uploaded in Virustotal website, all antiviruses scan that file and search for signature. If the antiviruses gets signature then file is displayed as malicious. After uploading file 0.exe, Virustotal website shows the output of all antiviruses as in Figure 4.2 and Figure 4.3.


Figure 4.2: Virustotal result for 0.exe

In Figure 4.3 after observing the output of all the antiviruses, it has been observed that it is a trojan horse. Type of every file is observed. For benign file, all the files

[1] http://www.dasmalwerk.eu
[2] www.virustotal.com

have been collected from the system32 directory of Windows operating system. All the benign files are in portable executable file format only.

| | | |
|---|---|---|
| Ad-Aware | Gen:Variant.Barys.501 | 20170613 |
| AegisLab | Troj.GameThief.W32.Magania.ensu!c | 20170613 |
| AhnLab-V3 | Dropper/Win32.OnlineGameHack.R3269 | 20170613 |
| ALYac | Gen:Variant.Barys.501 | 20170613 |
| Antiy-AVL | Trojan[GameThief]/Win32.Magania | 20170613 |
| Arcabit | Trojan.Barys.501 | 20170613 |
| Avast | Win32:Downloader-UAD [Trj] | 20170613 |
| AVG | Win32:Downloader-UAD [Trj] | 20170613 |
| Avira (no cloud) | TR/Spy.Gen | 20170613 |
| AVware | Trojan-Dropper.Win32.Farfli.e (v) | 20170613 |
| Baidu | Win32.Backdoor.DarkAngle.a | 20170613 |
| BitDefender | Gen:Variant.Barys.501 | 20170613 |

Figure 4.3: Virustotal antiviruses analysis result for 0.exe

### 4.1.2 Data Preprocessing

In data preprocessing, unpacking is done where it is identified whether Malware or benign file is packed or unpacked. If the file is found to be packed then different unpacking tools and techniques are used to unpack it. After this, the unpacked files are provided to feature extraction process which extracts the features of files. In case of unpacked files, the files are directly passed for feature extraction process. Section 4.1.2.1 explains unpacking process.

#### 4.1.2.1 Unpacking

In this process, it is checked whether the malware or benign file is packed or not. Below mentioned tools and techniques are used to detect whether the file is packed or unpacked.

- PEiD

  It contains the database of more than 470 signatures of packers [34]. Figure 4.4 shows packer detection using PEiD. By using PEiD, information is

obtained about Entry point address, Entry Point Section name, File Offset, Linker Info and Subsystem for particular executable file. Entry point means from where the actual execution of the program will starts. The subsystem indicates if the given file has Graphical User Interface or not. The Linker Info indicates that which version of linker is used while creating executable file. EP section implies the section name where the entry point lies.

When lab1.exe is opened in PEiD, it shows Entrypoint as 00005130, EP Section as snk1, Linker Info 6.0 and Subsystem as Win32 GUI. The lab1.exe is packed using PECompact 1.68 - 1.84 packer as shown in Figure 4.4.


Figure 4.4: Packer Detection Using PEiD

- OllyDbg Memory Map

  Most malware authors uses obfuscation techniques to hide the existence of malicious effect. They use different packers to compress the code and also use obfuscation methods like Dead Code Insertion and Instruction Substitution. OllyDbg [35] is an X86 debugger which mainly focuses on binary code analysis which is important when source code is not available. Mainly it has four windows namely text area, register window, hex dump section and stack section. Text area is an areawhere actually code in assembly language. Register window contain different registers such as EBX, EAX, EDX, ECX, ESI, EBP, EDI, EIP and EBP. These registers are used by the program so while debugging the value of register changes. Hence these changes can be seen in

the register section. The Hex Dump section contains code in hexadecimal format. The last is a Stack section which means stack is in predefined area of main memory. While the program is executing, the function parameter, return address and local variable is stored on the stack.

After clicking on M tab of OllyDbg, Memory Map for particular executable gets opened. Figure 4.5 shows Memory Map for an executable file lab1.exe. All exe files have mainly four sections- code, data, stack, and resource section. There are other sections such as read only data (rdata), export data (edata). The actual text of a whole program resides in code section. Data section contains global variables. The resource section contains all resources which are used by executables such as strings, icon, images, and menus. The idata section contains import function information and edata contains export function information.

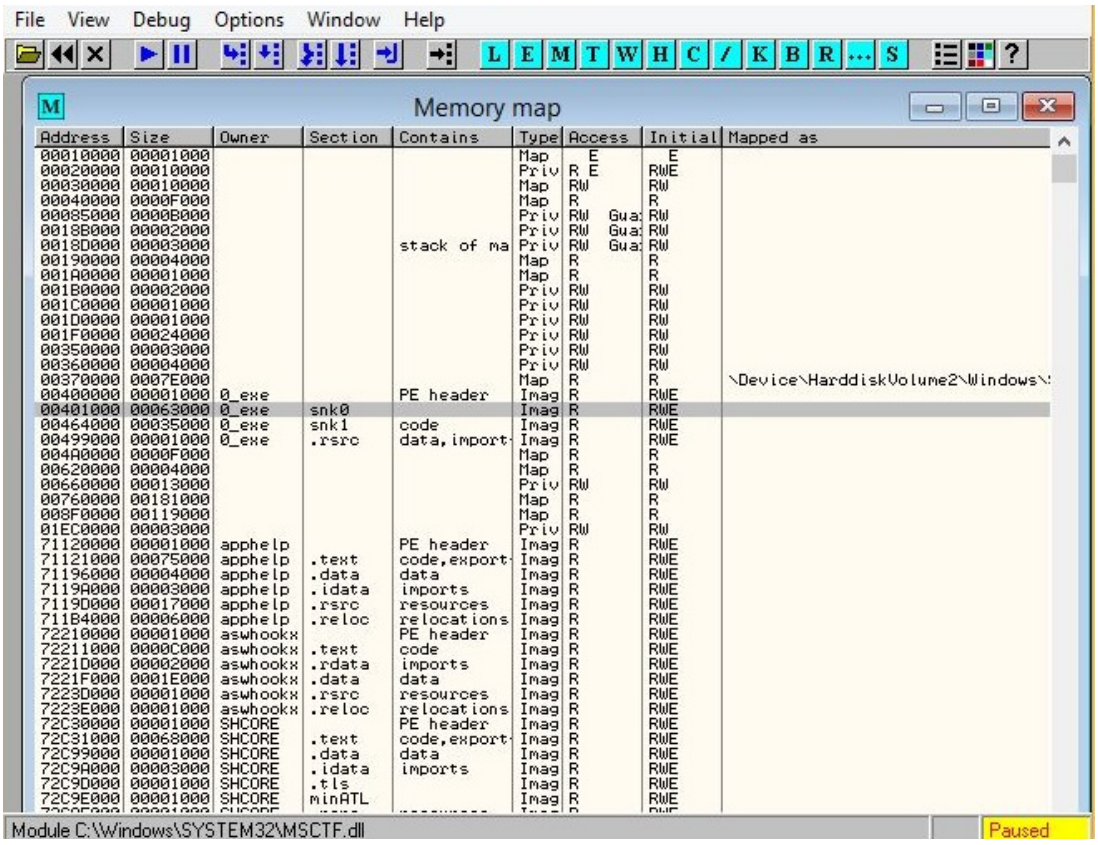



Figure 4.5: Ollydbg analysis to check packer

Name *Snk* has been observed in Memory Map for 0.exe in Figure 4.5 which indicates that lab1.exe is packed using SNK packer.

There is one more way to find out whether the exe is packed or not. If the size of Virtual Size (size of a program in main memory) of snk0 is more than SizeOfRawData (size of a program in hard disk) than the given executable is packed. In Figure 4.5 the Virtual Size of snk0 is 63000 bytes and and SizeOfRawData is 1000 bytes, hence it is indicates that lab1.exe file is packed.

- Unpacking

  The malware can be packed either manually or using tools like PE explorer, Exeinfo, PE Compact etc. There are two ways to perform Manual Unpacking:

  - Find the packing algorithm then write a code to execute it in reverse direction. By running the algorithm in reverse, program undoes every step of the packed program. Currently, there are different automation tools are available to perform the same process. However, this method is not useful because code written for unpacking the malware could be for a single packer.
  - Open the packed executable in debugger then click on the unpacking stub and perform necessary changes in PE header. This is more efficient approach.

  **Manual Unpacking Process:**

  Load the executable file in OllyDbg. Find the Original Entry Point (OEP). It is the first instruction before program packed. In manual unpacking, locating OEP is one of the difficult tasks. In OllyDbg, click on plug-in tab, select OllyDump, select Find OEP by section Hop. This program will come to halt after hitting the breakpoint just before Original Entry Point executes. As breakpoint gets hit, all the code has been unpacked into main memory and the original code is available for analysis.

  The last thing is to modify the PE header so that analysis tools read the code properly. Note down the original entry point. In OllyDump plug-in, select Dump Debugged Process to dump the executable from main memory to hard disk. If dump the exe after finding OEP and without performing any

changes into PE header the dump exe file will contain PE header of packed executable which is not same as PE header of unpacked program. Following changes need to be performed into PE header:

- The import table of exe must be restructured.
- The entry point in PE header must be changed to the OEP.

In case changes in PE header are not made by the user, then OllyDump will perform changes on its own [34].

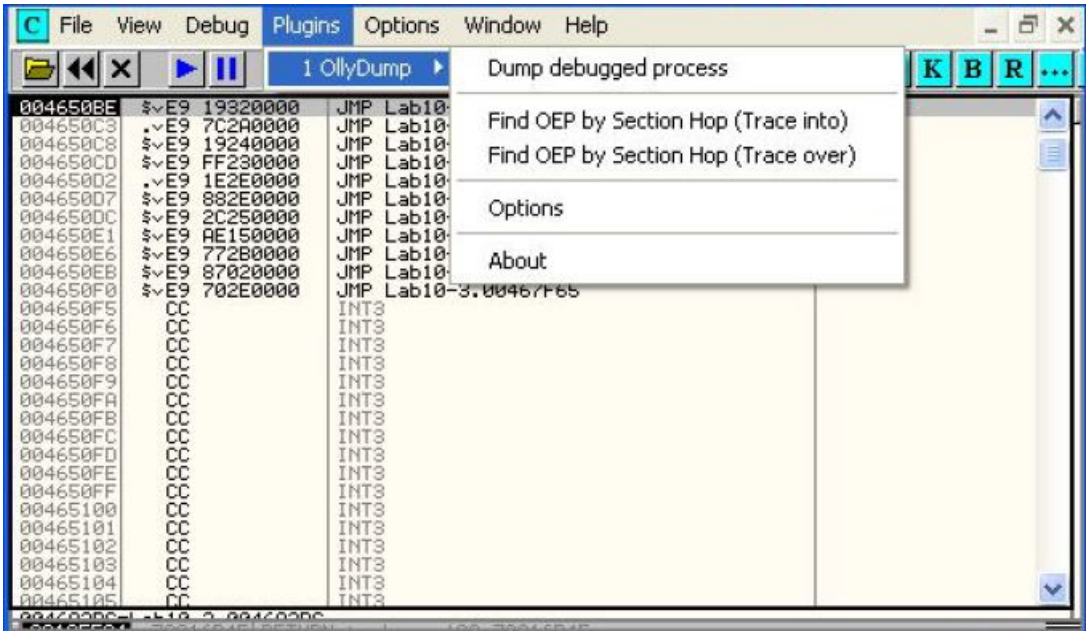



Figure 4.6: Olly Dump Plug-in Option

Figure 4.6 shows OllyDump plug-in option. The OllyDump Plug-in contains following option: Dump debugged process, Find OEP by Section Hop (Trace into), Find OEP by Section Hop (Trace over). The Dump debugged process means dumping the process from main memory to hard disk. Find OEP by Section Hop means (Trace into) means finding the OEP by using F7. Find OEP by Section Hop means (Trace Over) means finding OEP by using F8.

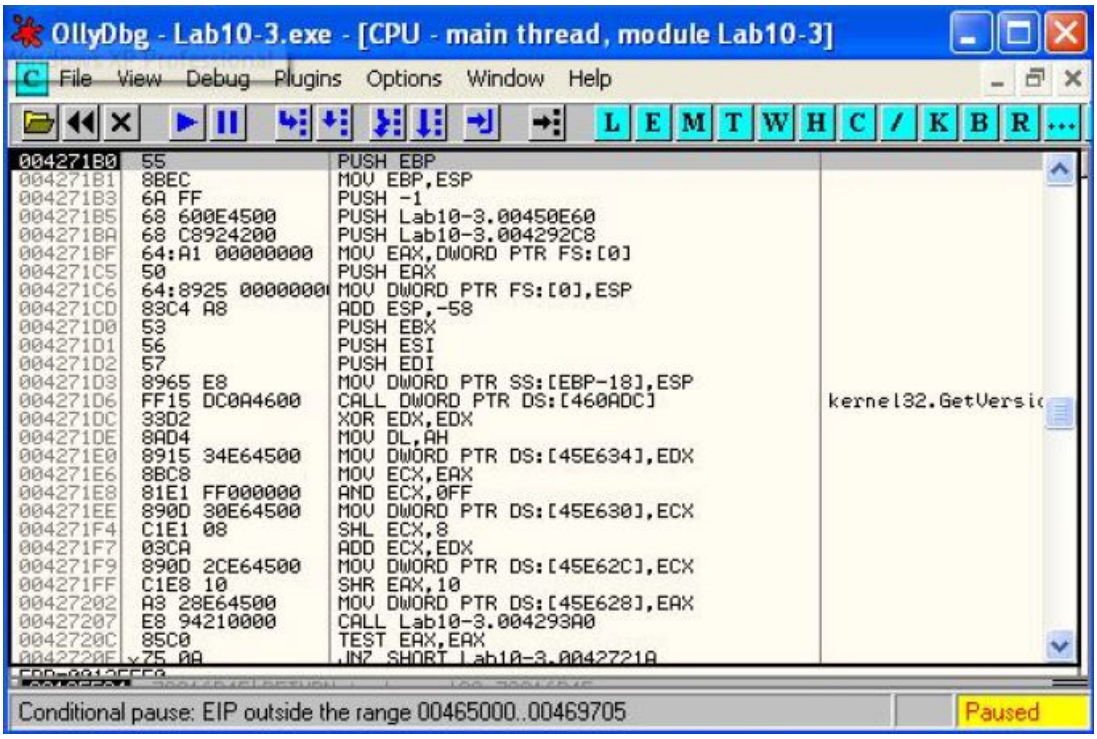



Figure 4.7: After changing the entry point

In Figure 4.6, Lab10-3.exe has been shown. Plug- in tab has been clicked to find OEP ,and OllyDump Plug-in has been selected. Select menu item to Find OEP by section Hop (Trace Over). After clicking on Find OEP by section hop then program will halt before entry point and new entry point is 0x004271B0 as shown in Figure 4.7.

After getting OEP, entry point in PE header of packed executable need to be modified. Moreover, import table is required to be reconstructed. On clicking on Dump Debugged process then new window gets opened as shown in Figure 4.8. In Figure 4.8, entry point is changed from 650BE to 271B0. Also Rebuilt import option is checked which means the OllyDump plug-in will automatically reconstruct import table for unpacked file Lab10-3.exe. Then the user needs to click on Dump button to save the unpacked program into hard disk after performing all necessary changes.

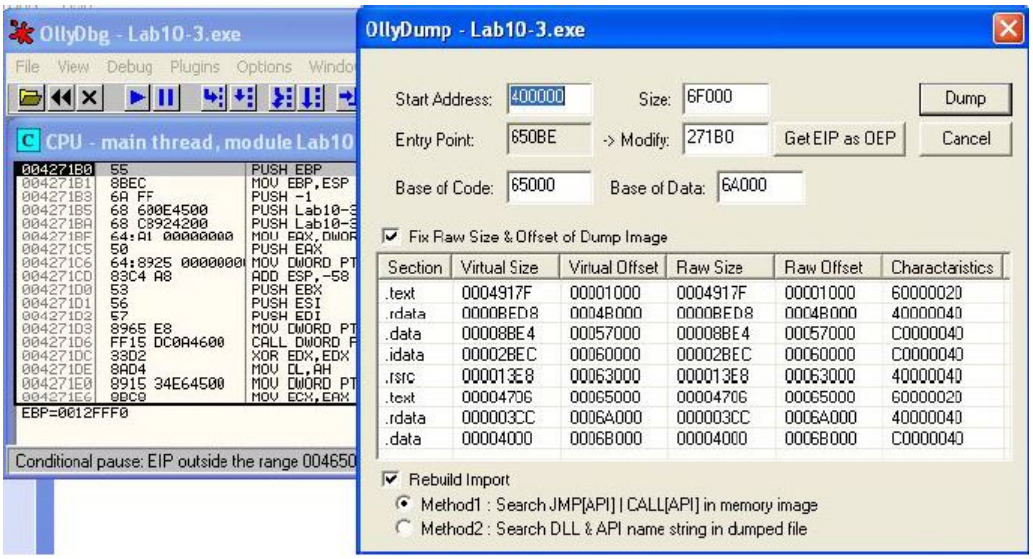



Figure 4.8: Dumping the exe file

### Automated Static Unpacking

Automated static unpacking program decrypts and/or decompress executable. It is the fastest method because it does not run the executable and finally, it converts the file into the original form. They are designed for specific packer. PE Explorer, a free program that works with exe and dll file contains several static unpacking plug-in. The default plug-in supports for PE Explorer are NSPack, UPack, and UPX. Unpacking of exe with PE Explorer is very simple.

When the exe file is opened in PE Explorer, the plug-in of unpacking perform checking, whether the given file is packed with NSPack, UPack and UPX. If it is found packed then automatically file will get unpacked after that the files needed to be saved with same or different name [34]. Figure 4.9 shows that LAB.exe has been packed with UPX packer, it gets detected with PE Explorer and has been automatically decompressed. In figure 4.9, text in green color shows that LAB.exe has been packed with UPX.

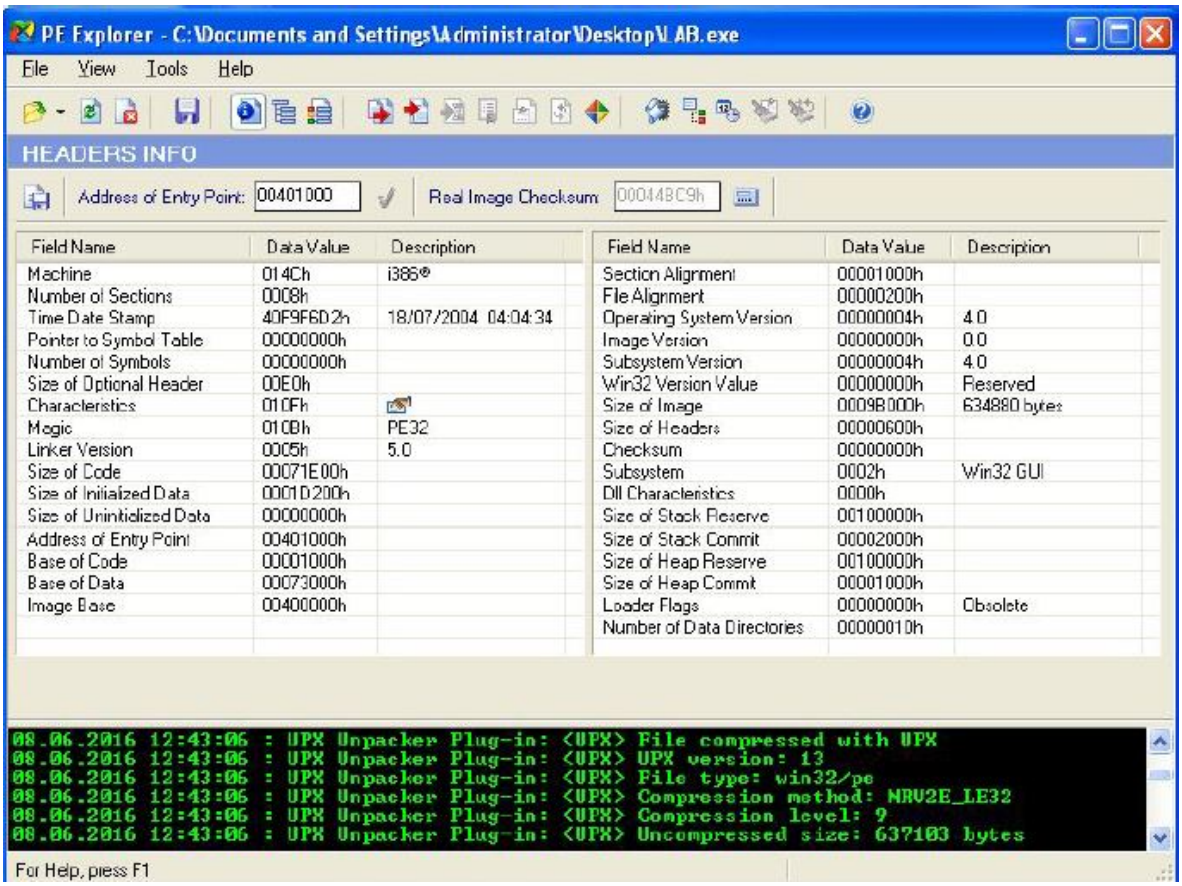



Figure 4.9: Automated Static Unpacking

### 4.1.3 Feature Extraction

Filtering files from API traces requires knowledge about API Calls. For extracting the features, API Calls have been recorded through IDA-Pro during static analysis. All API Calls have been retrived and regarded as a feature vector. The extracted APIs from the files have been presented into a database which will be used for feature vector generation. 180 API Calls from 7 most frequent dlls user32.dll, advapi32.dll, ws2_32.dll kernel32.dll, mscoree.dll, wininet.dll and ntdll.dll have been logged. The function of each dll is summarized in TABLE 4.1. The names of extracted features are listed in Appendix A.1.

Table 4.1: Summary of selected dlls

| Dll name | Function of Dll |
|---|---|
| kernel32 | Low-level operating system functions for resource handling and memory management |
| user32 | Windows management functions for message handling, timers, menus, and communications |
| advapi32 | Advanced API services library supporting numerous APIs including many security and Registry calls |
| ws2_32 | Provides TCP/IP networking functions and partial, broken compatibility with other network APIs |
| mscoree | Provides browser extension for Internet Xxplorer. |
| wininet | Enables applications to access standard Internet protocols, such as HTTP and FTP |
| ntdll | NT Layer dll that control NT system functions |

These APIs may create, open or delete registry keys and save and set values to registry keys; create a file, directory or process; search, move or delete a file; disturb global memory and virtual memory ; create a connection etc. For API call tracing, the files have been opened in IDA-PRO and copied the imports of every file into the notepad. Figure 4.10 shows the API Call extraction

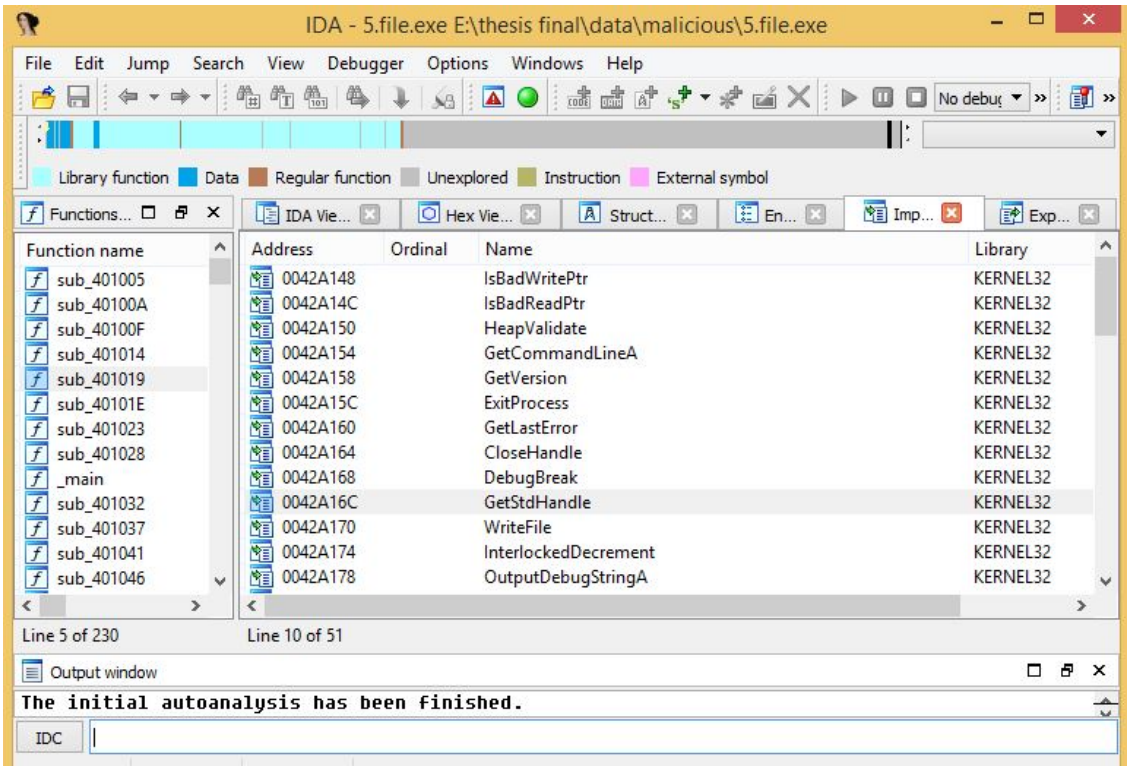



Figure 4.10: API Calls extraction using IDA-PRO

The API function Calls have been copied in the global list which has the API Calls of every input file. After copying the API function Calls of all 500 files, the unique API Calls have been found with python program as shown in Figure 4.11.

```
with open("E:\\c.txt",'r') as f:
    distinct_content=set(f.readlines())

to_file=""
for element in distinct_content:
    to_file=to_file+element
with open("E:\\b.txt",'w') as w:
    w.write(to_file)
```

Figure 4.11: Code Snippet to discover unique API Calls

After calculating the unique API Calls, the feature vector has been generated. For generating the feature vector, the API database used by all input samples and API used by each individual sample has been mapped. The presence or absence of unique API in each file has been checked. The feature vector is created as follow:

$$FEATUREVECTOR_{MALWARE_i} \begin{cases} 1, & \text{if } API_l \text{ is in } MALWARE_i \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

To create the feature vector, feature vector generation algorithm has been used which maps the API database used by all samples and API used by each individual sample. The following algorithm shows the feature vector generation steps in which the Local is an array which have API Calls used by each malware and featurevect is the unique API Dataset used by all the malware.

**Algorithm 1** Feature Vector Generation

**Input:** Local, Featurevect
**Output:** FeatureDB

```
1: for k = 0 to local.length do
2:     featurevect = list()
3:     while local do
4:         for l = 0 to local.length do
5:             while Featurevect.length do
6:                 if Local == featurevect then
7:                     FeatureDB_j = 1
8:                 else
9:                     FeatureDB_j = 0
10:                end if
11:            end while
12:        end for
13:    end while
14: end for
```

Algorithm feature vector generation calculates the frequencies of API Calls used by the malware. For example, Table 4.2 shows that $MALWARE_1$ has $API_2$ and $API_m$.

Table 4.2: Feature vector example

| File name | $API_1$ | $API_2$ | $API_3$ | .... | $API_m$ | $label$ |
|---|---|---|---|---|---|---|
| $MALWARE_1$ | 0 | 1 | 0 | .... | 1 | $label_1$ |
| $MALWARE_2$ | 0 | 1 | 1 | .... | 0 | $label_2$ |
| $MALWARE_3$ | 0 | 1 | 0 | .... | 1 | $label_3$ |
| .... | .... | .... | .... | .... | .... | .... |
| $MALWARE_n$ | 1 | 1 | 0 | .... | 0 | $label_i$ |

### 4.1.4 Feature Selection

Data can contain features that are highly correlated with each other. Many methods perform better if highly correlated features are removed. Random Forest has been used to determine the variable importance function to get the importance of each feature so that highly correlated features can be removed [36]. An example for such a measure in classification is the "Gini importance" available in random forest implementations. The Gini importance describes the improvement in the "Gini gain" splitting criterion. Figure 4.12 shows the variable importance.

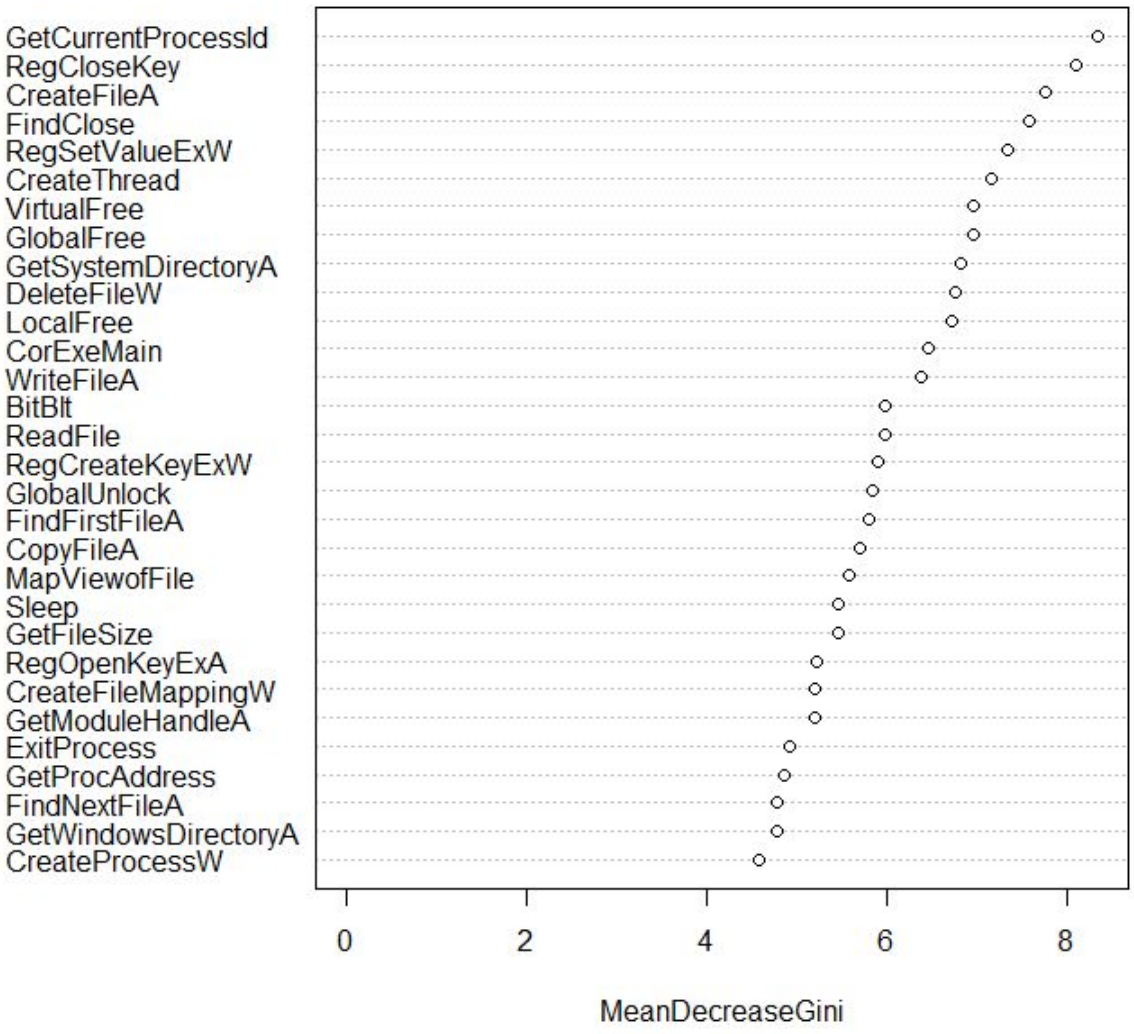



Figure 4.12: Variable Importance

45 features have been selected which are listed in Appendix A.2 and the dataset has been divided into training and testing components. The training set has 70% of the instances of the whole dataset and testing set has 30% of the instances. Training set has been used to train the classification models and then testing sets have been applied with decision label for validation.

### 4.1.5 Classification

Ensemble methods use various models and later merge them to obtain better output. Such methods generally produce more accurate solutions than an individual model would. Multilevel ensemble approach for classification has been proposed in this thesis by combining the kNN, LDA, DT, RF, NN, SVM models. Figure

4.13 shows the proposed multilevel ensemble classification model.

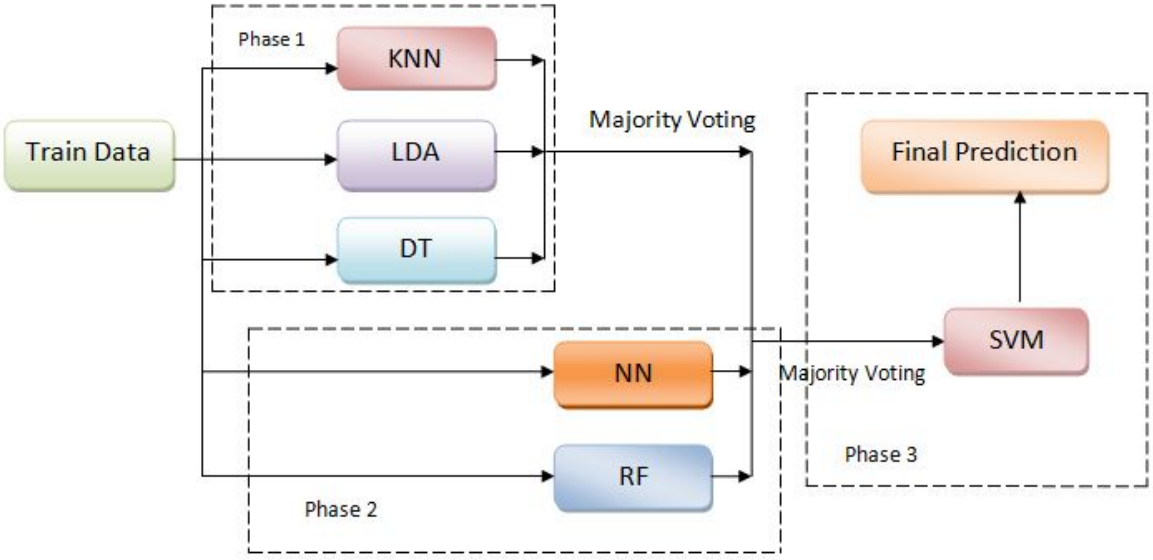



Figure 4.13: Multilevel Ensemble Classification Model

The classification models used are:

1. **k-Nearest Neighbor**

   kNN does not use the training data, i.e., training phase is very minimal. kNN keeps all the training data in case of lack of generalization. For testing, all the training data is needed. kNN makes decision based on the whole training data. In kNN, a label which has maximum confidence among $k$ data points nearest to object is assigned to each data point. The selection value is based on cross validation over a number of $k$ settings. Normally, a larger number of $k$ reduces the data noise effect on classification, while it can blur the difference among various classes. Thus, value of $k$ should be less than the square root of the total number of training data. For binary classification problems, the value of $k$ should be an odd number to avoid tied votes.

   For example, let us suppose we have 2 classes of positives and negatives and we want to classify the new object $x_q$. The object $x_q$ in Figure 4.14 [37] will be classified as negative for $k = 5$ because 3 of its nearest neighbors are

classified as negative.

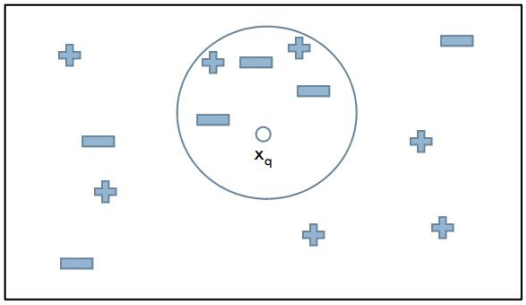



Figure 4.14: Example of kNN

2. **Linear Discriminant Analysis**

LDA is mainly used in the pre-processing step for pattern classification as a dimensionality reduction technique. The main aim of LDA is to project a feature space ( N-dimensional samples) onto a smaller subspace $X$ (where $X \leq N\text{-}1$) while maintaining the class-discriminatory information. For the cases in which within-class frequencies are unequal, LDA is used and the performance of these unequal frequencies are analyzed on randomly generated data [38].

3. **Decision Tree**

It is a classification tree that generates rules for the classification of a dataset. These trees are broken down a dataset into subsets. Decision trees do not require any domain knowledge and learning, the classification steps are fast and simple. Decision tree starts with root node that split into multiple branches. Here, each non leaf node represents testing condition that determines which branch to follow and leaf node contains decision. Each path from root node to leaf node produce a rule for a decision tree [39].

Every node of the decision tree is associated with a particular set of records R that is split by a specific test on a feature. For example, a split on a continuous attribute $A$ can be induced by the test $A \leq z$. The set of records $X$ is then partitioned in two subsets that leads to the left branch of the tree and the right one.

$$X_L = x \in X : x(A) \leq z \tag{4.2}$$

$$X_R = x \in X : x(A) > z \tag{4.3}$$

The divide step of the recursive algorithm to induce decision tree takes into account all possible splits for each feature and tries to find the best one according to a chosen quality measure (the splitting criterion). If dataset is induced on the following scheme

$$A_1,\ldots,A_m,C$$

where $A_j$ are attributes and $C$ is the target class, all candidates splits are generated and evaluated by the splitting criterion. Splits on continuous attributes and categorical ones are generated. The selection of the best split is usually carried out by impurity measures. The impurity of the parent node has to be decreased by the split. Let $(E_1,E_2,\ldots,E_k)$ be a split induced on the set of records $E$, a splitting criterion that makes use of the impurity measure $I(.)$ is:

$$\Delta = I(E) - \sum_{i=1}^{K} \frac{|E_i|}{|E|} I(E_i) \tag{4.4}$$

Decision tree assumes that only one variable is target and other variables are input. A decision tree for predicting survival on the Titanic [3] is shown in Figure 4.15. The feature questions are the nodes, and the answers "yes" or "no" are the branches in the tree to the child nodes.

[3] en.wikipedia.org/wiki/Decision_tree_learning

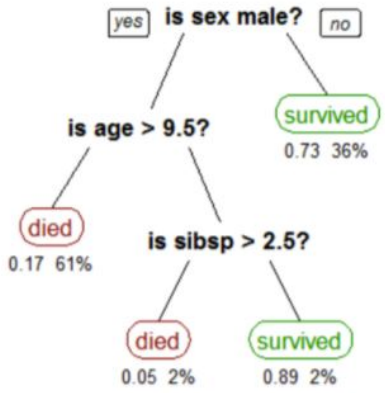



Figure 4.15: Example of Decision Tree

A decision tree showing The Titanic passengers survival ("sibsp" is the number of siblings or spouses aboard). The fvalues under the leaves show the probability of survival and the percentage of observations in the leaf.

4. **Random Forest**

Random forest is a supervised learning algorithm that performs both classification and regression. Random Forests construct multiple decision trees. A classification is given by all individual trees to predict the unknown data known as tree *votes* by classifying the bootstrap sample of input data. The output of random forest is decided by the classification having the maximum votes (across all DT in the forest).

Random input variable selection for a single decision tree is difficult. If a decision tree is constructed based on wrong input variable combination, the desired output cannot be achieved. However, in this each decision tree prediction depends on the random input variables which are sampled seperately with the equal distribution for the decision trees in a forest. The random forest mostly predict better results than an individual decision tree. The basic assumption of random forest is that "week learners" can be merged to build a "strong learner". Figure 4.16 shows an example of random forest [40].

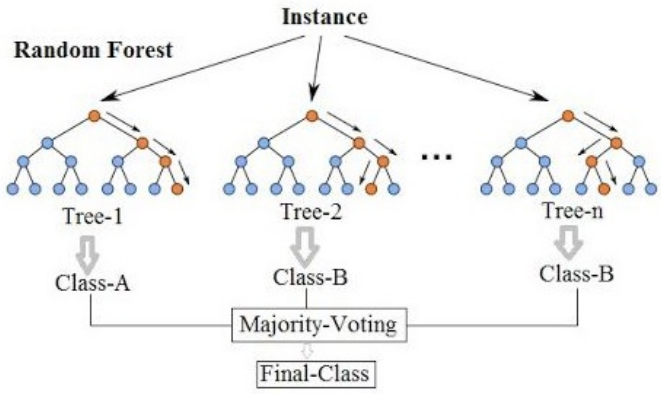



Figure 4.16: Example of Random Forest

Each tree is grown as follows:

(a) If the total number of cases in training set is $X$, from the original data, choose $X$ cases randomly but with the replacement. These samples will be training data.

(b) A number $n<N$ is specified for $N$ input variables such that at each node, $n$ inputs are randomly chosen out of the $N$ and to split the node from these $n$ variables the best split is used. The value of $n$ is consistent while the forest grows.

(c) Each tree is grown to the highest degree feasible.

5. **Neural Network**

Neural network is a set of connected inputs and outputs where a weight is associated with each connection. During the training stage, the neural network learns by adjusting the weights to predict the input variables class. Each input in neural network is a high dimensional vector. The neural network is arranged in a sequence of layers, where the input vector is projected to a "hidden layer." Each unit in the hidden layer is a weighted sum of the values in the first layer. This layer then projects to an output layer, which is where the intended result is obtained. Figure 4.17 [41] shows a neural network.

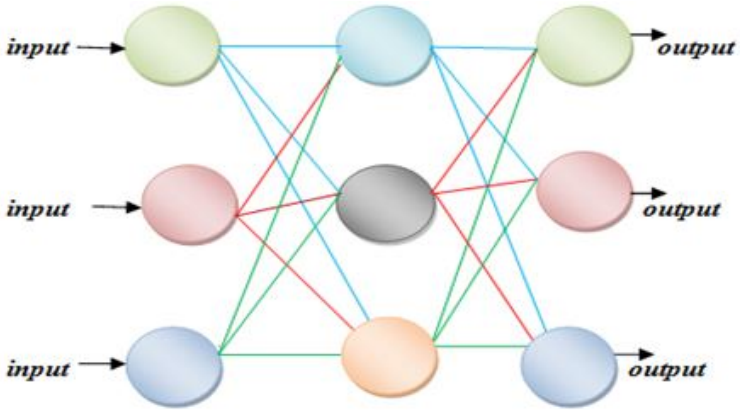



Figure 4.17: Example of Neural Network

6. **Support Vector Machine**

The basic idea behind the SVM is Decision planes that specify the boundaries of decision. A set of objects belonging to various class memberships are separated by decision planes[42]. SVMe tries to get the maximum separation between classes. Figure 4.18 (a) shows the linear SVMs example [42]. In this example, the objects either belong to RED class or to GREEN class. The decision boundaries are specified by the separating lines. On the right hand side of the boundary, all objects are GREEN and to the left hand side of boundary, all objects are RED. A new object (white circle) will be classified as GREEN if it lies to the right side of the boundary or classified as RED if it lies to the left side of the boundary.

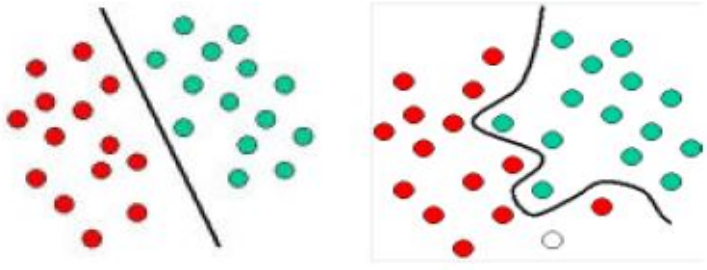


Figure 4.18: (a) Linear SVM example, (b) Example of hyperplane SVM

A classifier that partitions a set of objects into their respective domains with a line is called linear classifier and partitioning with a curve is known as hyperplane or nonlinear classifier [13]. An example of hyperplane classifier is shown in Figure 4.18 (b).

Figure 4.18 shows the basic concept behind SVM. In this figure, original objects are mapped by applying mathematical functions sets called kernels. This process of identifying various objects is called transformation or mapping. Figure 4.18 shows that the mapped objects are linearly separable. Thus, an optimal line has to be found rather than constructing a complex curve that can divide GREEN and RED objects.

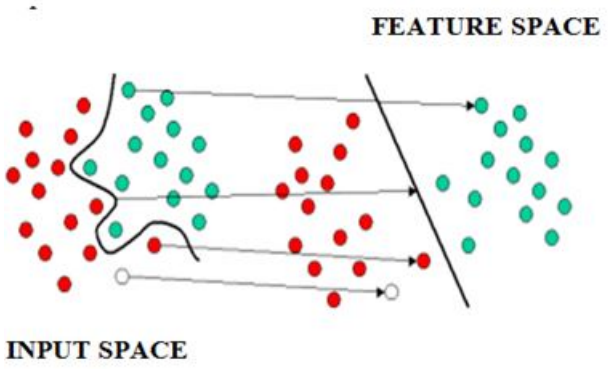



Figure 4.19: Object mapping in SVM

### Majority Voting

Consider $\chi$ a set of $N$ examples, $C$ a set of $Q$ classes and an algorithm set $S = \{A1, A2, AJ\}$ which contains the $J$ classifiers used for the voting. Each example $x \in \chi$ is assigned to have one of the $Q$ classes. Each classifier will have its prediction for each example. The final class assigned to each example is the class predicted by the majority of classifiers (gaining the majority votes) for this example. This can be formulated as follows. Let $c_l \in C$ denotes the class of an example $x$ predicted by a classifier $Al$, and let a counting function$F_k$ defined as:

$$F_k(cl) = \begin{cases} 1, & c_l = c_k \\ 0, & c_l \neq c_k \end{cases} \tag{4.5}$$

where $cl$ and $ck$ are the classes of $C$. The count of total votes for class $c_k$ can then be defined as:

$$T_k = \sum_{l=1}^{M} F_k(c_l) \tag{4.6}$$

The predicted class $c$ for an example $x$ using the algorithm set $S$ is defined to be a class that gains the majority vote as:

$$c = S(x) \tag{4.7}$$

$$where : S(x) = argmaxT_k k \in \{1, .....q\}$$

# Chapter 5

# Experimental Results and Discussion

The current work indends to discover unknown malware with the help of machine learning techniques. 500 malicious and benign files each containing API Calls approximately 100 to 700 in number have been analyzed. After testing files on virustotal, mostly antivirus identified files as virus, adware, backdoor, benign, trojan horse and worm. Hence, files have been classified into 6 categories. The primary goal of this research is API Calls usage and their occurring frequencies.

The most frequent 180 API Calls from 7 dlls have been found and extracted using IDA-Pro. After feature selection, 45 API Calls have been choosen and these 45 API Calls have been used as feature vector for classifiers.

The algoritms have been implemented in R [1] (environment for statistical computing and graphics), provides various statistical (nonlinear and linear modelling, time-series analysis, classical statistical tests, classification, clustering etc.). Table 5.1 describes the machine learning models that have been trained on the dataset with optimum tuning parameters.

[1] www.rstudio.com

Table 5.1: Machine Learning Models

| Model | Required Package | Tuning Parameter |
|---|---|---|
| kNN | class | k=5 |
| LDA | MASS | none |
| DT | rpart | none |
| RF | random forest | mtry=3, ntree=500 |
| NN | nnet | size=10 |
| SVM | kernlab | kernel="radial", gamma=0.1, cost=10 |

The proposed methodology works in three phases. All three phases are explained below:

**Phase 1:** kNN, LDA, and DT have been trained with 70% of the dataset and validated on the 30% of the dataset.

**Phase 2:** Majority voting scheme is applied on the outputs of phase1. If two or more classes have the same voting the predicted value of the model from phase 1 which has the highest accuracy is chosen as final prediction. In this phase neural networks and random forest are trained with 70% of the dataset and tested with 30% of dataset.

**Phase 3:** Majority voting scheme is applied on the outputs of phase 2 and the output predicted by applying majority voting on the outputs of phase 1 and the predicted output is applied to test the SVM model which gives the final predictions.

In case if two or more classes have the same votes, Influenced Majority Vote has been used which chose the best classifier's predictions.

## 5.1 Performance Evaluation Metrics

Evaluation metrics are the key to understanding how a classification model performs when applied to a test dataset. Accuracy, sensitivity, specificity, misclassification rate, kappa value, precision, FPR, and FNR have been measured to evaluate the performance of proposed approach.

1. **Accuracy**

   Total number of correctly classified observation divided by total number of

observations in entire dataset [43] is known as accuracy.

$$Accuracy = \frac{TP + TN}{P + N} \tag{5.1}$$

2. **Sensitivity**

   Also called the TPR or the recall measures the ratio of positives which are correctly recognized.

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.2}$$

   where TP+FN is total no. of values for a particular class

3. **Specificity**

   Also called the true negative rate (TNR) measures the ratio of negatives that are correctly recognized.

$$Specificity = \frac{TN}{TN + FP} \tag{5.3}$$

4. **Misclassification rate**

   Measure the proportion of instances that are wrongly classified.

$$Misclassificationrate = \frac{FP + FN}{T + N} \tag{5.4}$$

5. **Kappa**

   Measure of how well the classifier performed as compared to how well it would have performed simply by chance.

6. **Precision**

   Precision is defined as the fraction of correct predictions for a certain class.

$$Precision = \frac{TP}{TP + FP} \tag{5.5}$$

7. **FNR**

   Proportion of all negatives that are predicted as positive .

$$FNR = \frac{FN}{TP + FN} \tag{5.6}$$

8. **FPR**

Proportion of all positives that are predicted as negative

$$Precision = \frac{FP}{TN + FP} \tag{5.7}$$

Where

P: total number of positives.

N: total number of negatives.

TP: value which is predicted true.

FN: positives which are predicted as negatives.

FP: negatives which are predicted as positives.

TN: negative values which are predicted as negatives.

A comparison of performance evaluation metrics (measured in %) of different model has been given in Table 5.2.

Table 5.2: Comparison of performance evaluation metrics

| Model | Accuracy | Kappa | Miss Classification Rate | Sensitivity | Specificity | Precision | FNR | FPR |
|---|---|---|---|---|---|---|---|---|
| kNN | 73.33 | 67.59 | 26.67 | 70.04 | 94.54 | 69.72 | 6.46 | 29.96 |
| LDA | 76 | 70.69 | 24 | 78.62 | 95.22 | 74.17 | 4.78 | 21.38 |
| DT | 84 | 80.48 | 16 | 81.95 | 96.74 | 86.42 | 3.26 | 18.05 |
| NN | 92.67 | 89.98 | 7.33 | 91.27 | 98.46 | 94.03 | 1.54 | 8.73 |
| RF | 90.67 | 88.6 | 9.33 | 87.67 | 98.7 | 93.6 | 1.3 | 12.33 |
| SVM | 94 | 92.67 | 6 | 91.51 | 98.75 | 95.79 | 1.25 | 8.49 |
| Proposed method | 94.67 | 93.41 | 5.33 | 95.21 | 98.75 | 94.9 | 1.25 | 4.79 |

Figure 5.1 shows the graph of accuracy of each model. The highest accuracy is given by the proposed approach and least accuracy is given by kNN.

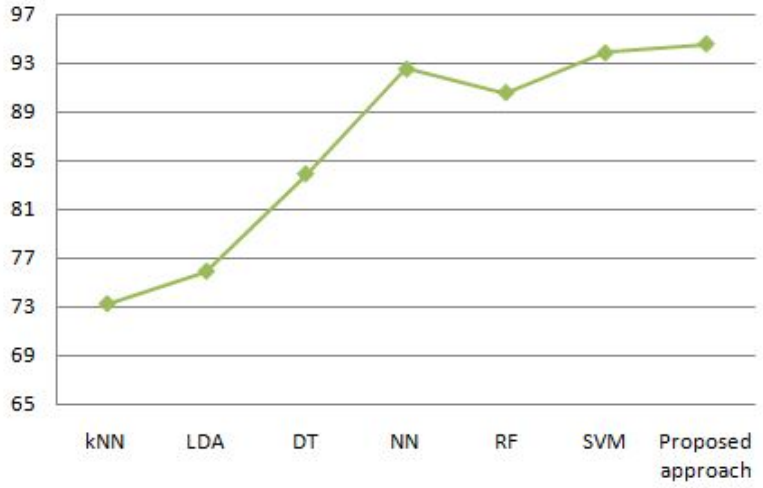



Figure 5.1: Performance evaluation with accuracy

Figure 5.2 shows the kappa value of various models.

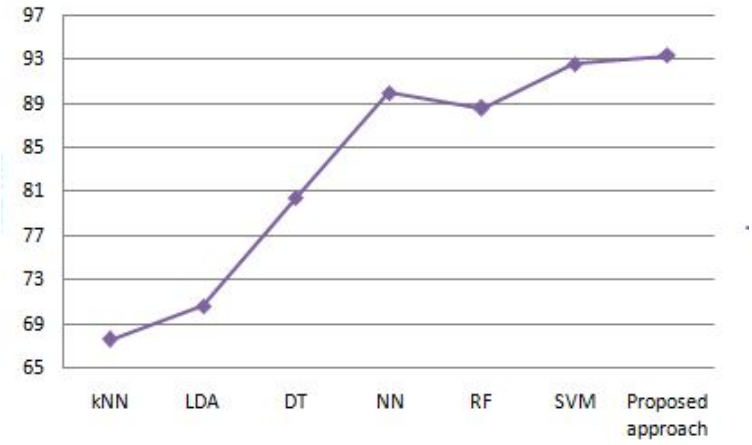



Figure 5.2: Performance evaluation with kappa value

Figure 5.3 shows the misclassification rate graph. The proposed approach achieved minimum misclassification rate.

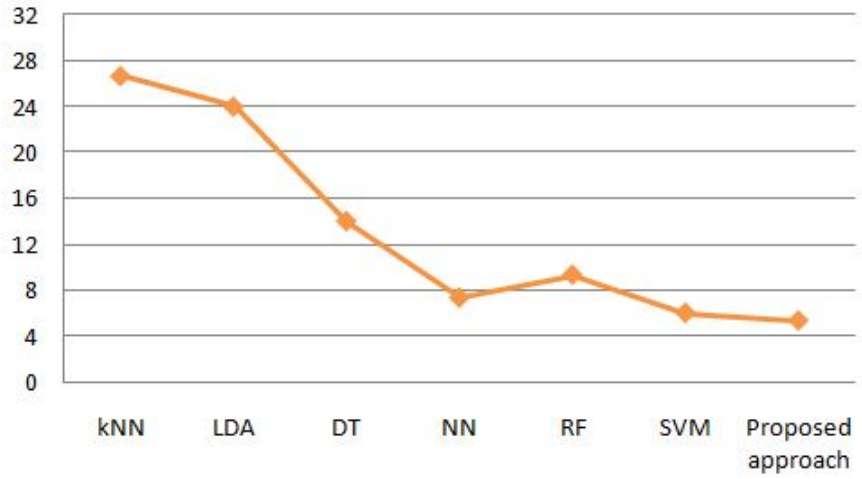



Figure 5.3: Performance evaluation with misclassification rate

Figure 5.4 shows the classification model vs. sensitivity graph. The highest sensitivity of 95.21% is given by the proposed approach and least sensitivity is given by kNN.

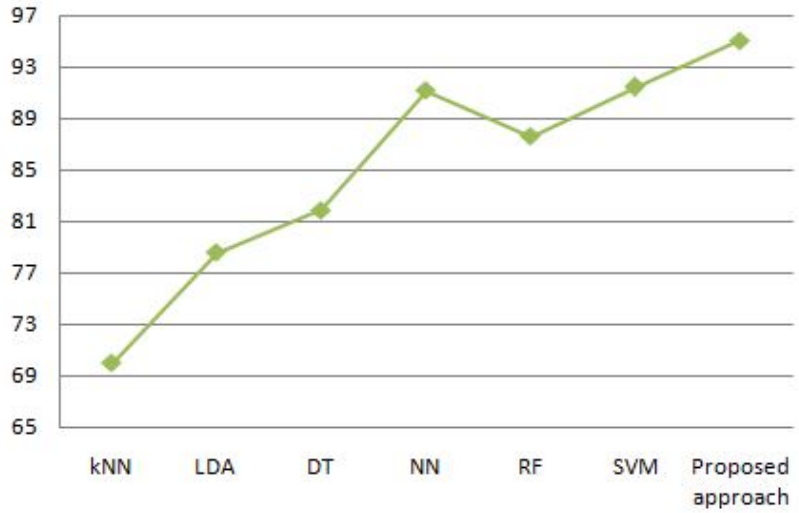



Figure 5.4: Performance evaluation with sensitivity

Figure 5.5 shows the graph of specificity of each model. The proposed approach achieved 98.75% apecificity and least specificity is given by kNN.

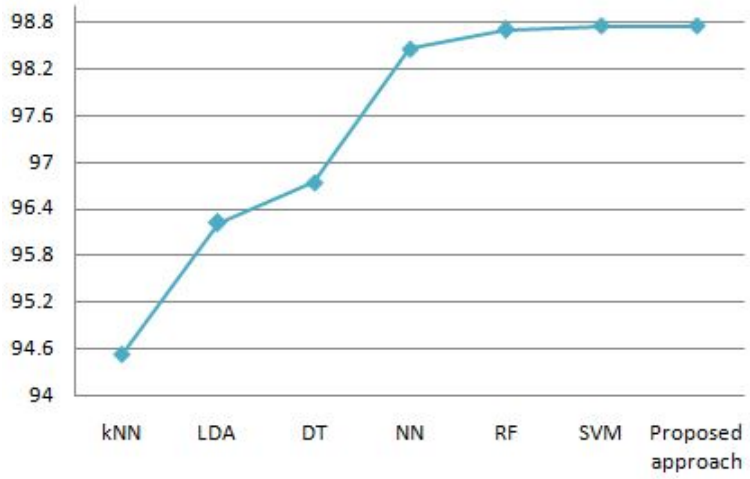



Figure 5.5: Performance evaluation with specificity

Figure 5.6 shows the graph of precision of each model. The precision of proposed approach is 94.9%.

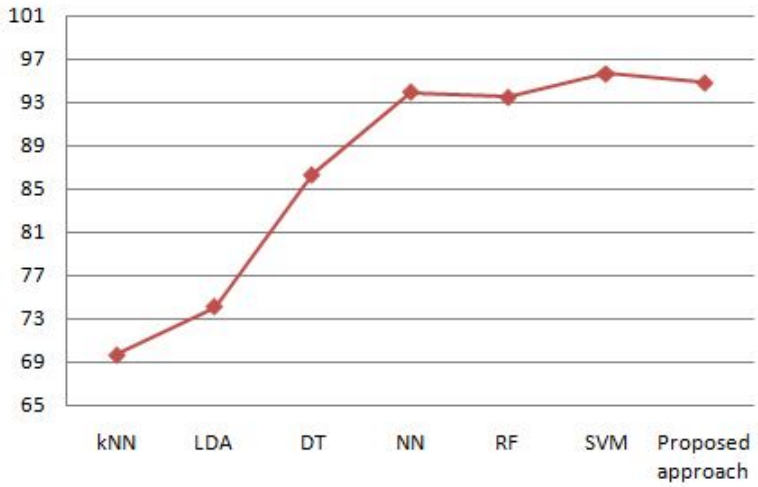



Figure 5.6: Performance evaluation with precision

Figure 5.7 shows the graph of FNR of each model. The proposed approach achieved the lowest FNR which is desirable.


Figure 5.7: Performance evaluation with FNR

Figure 5.8 shows the FPR of each model. The FPR of proposed approach is 4.79% which is low as desirable.

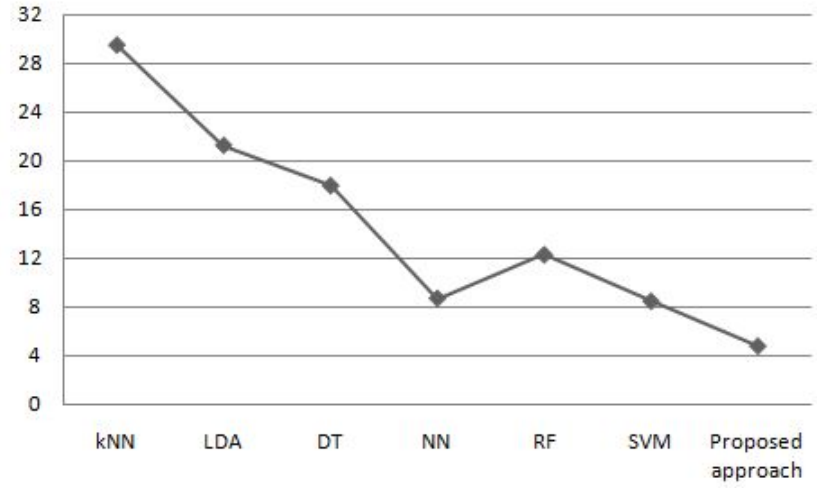



Figure 5.8: Performance evaluation with FPR

# Chapter 6

# Conclusion

## 6.1 Conclusion

Malware detection is an expending research domain owing to the increase of malware with each passing day. Signature based malware detection approach is facing problems for detecting unknown malware for example, zero day attack. Anomaly based malware detection approach treats a malicious activity as normal if it behaves like a normal activity. Hence, it is imperative to use traditional malware detection approaches alongside unconventional approaches to diagnose new malware. Machine learning is an appropriate technique to complement classical malware detection approaches. We acquired API Calls from 500 malicious and benign files and categorized them into 6 categories: adware, backdoor benign, trojan horse, virus and worm. We identified that, API Calls can be used to detect the unknown malware. We found 180 API Calls from 7 dlls and out of them 45 API Calls with their frequencies of occurrences are used as feature vector for classification. We constructed multilevel ensemble machine learning model with six supervised learning models namely SVM, DT, RF, LDA, kNN and NN and 94.67% accuracy and 4.79% FPR is achieved. The obtained results reveal a superior performance of the proposed procedures in identifying and distinguishing malicious files.

## 6.2 Research Contribution

In this research, malware dection system using multilevel ensemble supervised learning is presented. The major contribution of this research are

1. show how to use API Calls of portable executable format to detect and classify malware.

2. provide empirical validation of our approach with an extensive study of machine learning models for detecting and classifing malware.

3. our proposed approach also includes the multilevel ensemble classification to identify malware.

4. shows that proposed approach achieves high detection rate, even for unknown malware.

## 6.3 Future Work

In the future, we intend to input our dataset to additional machine learning approaches like unsupervised and semisupervised machine learning algorithms. We also plan to expand the dataset for malware classification. Furthermore, we plan to extend our method such that we focus on APIs sequence and try to classify malware into more categories.

# References


[1] A. Sharma and S. K. Sahay, "Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey," *International Journal of Computer Applications*, vol. 90, no. 2, pp. 7–11, Jan. 2014.

[2] H. D. Huang, C. S. Lee, M. H. Wang, and H. Y. Kao, "IT2FS-based ontology with soft-computing mechanism for malware behavior analysis," *Soft Computing*, vol. 18, no. 2, pp. 267–284, 2013.

[3] M. A. Jerlin and C. Jayakumar, "A Dynamic Malware Analysis for Windows Platform - A Survey," *Indian Journal of Science and Technology*, vol. 8, no. 27, 2015.

[4] N. Idika and AP. Mathur, AP. "A survey of malware detection techniques," *Purdue University*, 48, 2007.

[5] L. Nataraj, S. Karthikeyan, G. Jacob and B. S. Manjunath, "Malware images," *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec 11*, 2011.

[6] K. Mathur and S. Hiranwal S, "A survey on techniques in detection and analyzing malware execu-tables," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, pp. 422-428, 2013.

[7] P. Vinod, R. Jaipur, V. Laxmi, and M. Gaur. "Survey on malware detection methods." *In Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*, pp. 74-79, 2009.

[8] T. G. G. Paul and T. G. Kumar, "A Framework for Dynamic Malware Analysis Based on Behavior Artifacts," *Advances in Intelligent Systems and Computing Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications,* pp. 551–559, 2017.

[9] M. Alazab, S. Venkataraman, and P. Watters, "Towards Understanding Malware Behaviour by the Extraction of API Calls," *2010 Second Cybercrime and Trustworthy Computing Workshop,* 2010.

[10] J. Choi, Y. Han, S.-J. Cho, H. Yoo, J. Woo, M. Park, Y. Song, and L. Chung, "A Static Birthmark for MS Windows Applications Using Import Address Table," *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing,* 2013.

[11] A. A. Elhadi, M. A. Maarof, and B. Barry, "Improving the Detection of Malware Behaviour Using Simplified Data Dependent API Call Graph," *International Journal of Security and Its Applications,* vol. 7, no. 5, pp. 29–42, 2013.

[12] Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu, "CBM: Free, Automatic Malware Analysis Framework Using API Call Sequences," *Advances in Intelligent Systems and Computing Knowledge Engineering and Management,* pp. 225–236, 2013.

[13] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," vol. 3, no. 24, 2007.

[14] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," *Proceedings 2001 IEEE Symposium on Security and Privacy,* 2001.

[15] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar, "Signature Generation and Detection of Malware Families," *Information Security and Privacy Lecture Notes in Computer Science,* pp. 336–349, 2008

[16] C. Wang, J. Pang, R. Zhao, and X. Liu, "Using API Sequence and Bayes Algorithm to Detect Suspicious Behavior," *2009 International Conference on Communication Software and Networks,* 2009.

[17] R. Tian, L. Batten, R. Islam, and S. Versteeg, "An automated classification system based on the strings of trojan and virus families," *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, 2009.

[18] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," *2010 5th International Conference on Malicious and Unwanted Software,* 2010.

[19] Z. Salehi, M. Ghiasi, and A. Sami, "A miner for malware detection based on API function calls and their arguments," *The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012),* 2012.

[20] P. Natani and D. Vidyarthi, "Malware Detection Using API Function Frequency with Ensemble Based Classifier," Communications in Computer and Information Science Security in Computing and Communications, pp. 378–388, 2013.

[21] M. Alazab, "Profiling and classifying the behavior of malicious codes," *Journal of Systems and Software,* vol. 100, pp. 91–102, 2015.

[22] N. Kawaguchi and K. Omote, "Malware Function Classification Using APIs in Initial Behavior," *2015 10th Asia Joint Conference on Information Security,* 2015.

[23] R. S. Pirscoveanu, S. S. Hansen, T. M. T. Larsen, M. Stevanovic, J. M. Pedersen, and A. Czech, "Analysis of malware behavior: Type classification using machine learning," *2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA),* 2015.

[24] P. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection," *Procedia Computer Science,* vol. 46, pp. 804–811, 2015.

[25] C.I. Fan, H.W. Hsiao, C.H. Chou, and Y.F. Tseng, "Malware Detection Systems Based on API Log Data Mining," *2015 IEEE 39th Annual Computer Software and Applications Conference,* 2015.

[26] U. Bayer, P. M. Comparetti,C. Hlauschek, C. Kruegel, and E. Kirda, (2009, February). "Scalable, Behavior-Based Malware Clustering," *In NDSS,* Vol. 9, pp. 8-11, Feb. 2009.

[27] J. Nakazato, J. Song, M. Eto, D. Inoue, and K. Nakao, "A Novel Malware Clustering Method Using Frequency of Function Call Traces in Parallel Threads," *IEICE Transactions on Information and Systems, vol. E94-D,* no. 11, pp. 2150–2158, 2011.

[28] K. Iwamoto and K. Wasaki, "Malware classification based on extracted API sequences using static analysis," *Proceedings of the Asian Internet Engineeering Conference on - AINTEC 12,* 2012.

[29] O. Kostakis, "Classy: fast clustering streams of call-graphs," *Data Mining and Knowledge Discovery,* vol. 28, no. 5-6, pp. 1554–1585, Oct. 2014.

[30] W. Shuwei, W. Baosheng, Y. Tang, and Y. Bo, "Malware Clustering Based on SNN Density Using System Calls," *Cloud Computing and Security Lecture Notes in Computer Science,* pp. 181–191, 2015.

[31] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised Learning for Unknown Malware Detection," *Advances in Intelligent and Soft Computing International Symposium on Distributed Computing and Artificial Intelligence,* pp. 415–422, 2011.

[32] F. Bisio, P. Gastaldo, R. Zunino, and S. Decherchi, "Semi-supervised machine learning approach for unknown malicious software detection," *2014 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings,* 2014.

[33] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: High-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251–266, 2015.

[34] M. Sikorski and A. Honig, *Practical malware analysis: the hands-on guide to dissecting malicious software.* San Francisco: No Starch Press, 2012.

[35] O. Yuschuk, "Ollydbg," (2007).

[36] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, "Bias in random forest variable importance measures: Illustrations, sources and a solution," *BMC Bioinformatics,* 25-Jan-2007. [Online]. Available: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-25. [Accessed: 03-Jun-2017]

[37] S. Dua and X. Du, *Data mining and machine learning in cybersecurity.* Boca Raton, Fla: Auerbach, 2011.

[38] P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis, *Robust Data Mining.* New York, NY: Springer New York, 2013.

[39] "Random Forest Template for TIBCO Spotfire®," Random Forest Template for TIBCO Spotfire® — TIBCO Community. [Online]. Available: https://community.tibco.com/modules/random-forest-template-tibco-spotfirer. [Accessed: 08-Jun-2017].

[40] G. J. Williams, *Data mining with Rattle and R: the art of excavating data for knowledge discovery.* New York: Springer, 2013.

[41] "Artificial neural networks," *Wikipedia*, 25-May-2017. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_networks. [Accessed: 25-May-2017]

[42] T. Hill and P. Lewicki, *Statistics: methods and applications ; a comprehensive reference for science, industry, and data mining.* Tulsa, OK: StatSoft, 2006.

[43] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences,* vol. 231, pp. 64–82, 2013.

# List of Publications

1. Vidhi, N. Baliyan and R. K. Chahal, "A Review on Data Mining based Intrusion Detection Systems," in Scopus indexed *International Journal of Control Theory and Applications,* 2017, [Published].

2. Vidhi and N. Baliyan, "Malware Detection using Multilevel Ensemble Supervised Learning ," *Journal of Computer Information [communicated],* 2017.

# Video Link

https://youtu.be/mwi0ibjJXJk

# Appendix A

# List of Features

## A.1 List of Extracted Features

| | | | |
|---|---|---|---|
| SetFileTime | WriteFileA | FindFirstFileA | CreateFileMappingW |
| FindClose | MapViewofFile | FindClose | RegCreateKeyExW |
| FindNextFileA | IsTextUnicode | SetFocus | GetWindowsDirectoryA |
| Connect | GetParent | UpdateWindow | RegisterWindowMessageW |
| ExitProcess | CreateFileA | OpenFile | GetSystemDirectoryA |
| EndDialog | GetMessageW | LocalFree | UnMapViewofFile |
| CloseHandle | GetFileTime | GlobalFree | VirtualProtect |
| VirtualFree | SetEvent | RegCloseKey | RegOpenKeyExA |
| GetProcAddress | CharLowerW | CopyFileA | SetFileAttributesW |
| ShowWindow | ReadFile | DeleteFileW | SetFileAttributesW |
| GetFileSize | Accept | BitBlt | GetForegroundWindow |
| SetScrollPo | GetLastError | Sleep | RegQueryValueExW |
| CreateProcessW | MoveWindow | CreateThread | FileTimeToSystemTime |
| GlobalLock | GlobalUnlock | CorExeMain | GetFileInformationByHandle |
| SetWinEventHook | PostQuitMessage | wsprintfW | GetModuleHandleA |
| GetSystemMetrics | GetMenuState | MessageBeep | GetCurrentProcessId |
| RegisterClassExW | PeekMessageW | MessageBoxW | SetWindowPlacement |
| CharUpperW | freeaddrinfo | LoadIconW | WriteProcessMemory |
| ReleaseDC | GetDlgCtrlID | CheckMenuItem | SendMessageW |
| GetWindowLongW | lstrlenA | WinHelpW | GetWindowPlacement |
| AcceptEx | closesocket | DisconnectEx | DialogBoxParamW |

| | | | |
|---|---|---|---|
| lstrcpynW | CreateEventW | GetTickCount | GetCommandLineW |
| recv | lstrcmpW | MulDiv | TranslateAcceleratorW |
| htonf | FreeLibrary | LocalAlloc | QueryPerformanceCounter |
| GetAddrInfoEx | getservbyport | GetService | GetHostNameW |
| htonl | inet_ntoa | inet_ntoa | RegSetValueExW |
| listen | ntohl | ntohs | GetCommandLineW |
| recvfrom | RIONotify | select | SetAddrInfoEx |
| shutdown | TransmitFile | WSAAccept | WSAConnect |
| CoInitializeEE | UpdateError | MD4Final | CallFunctionShim |
| CoUninitializeCor | CloseCtrs | CorExitProcess | CorBindToRuntimeByCfg |
| FtpCommandA | FtpCommandW | FtpGetFileA | GopherOpenFileW |
| FtpRenameFileA | FtpPutFileW | NtCreateFile | HttpEndRequestW |
| HttpQueryInfoA | InternetAutodial | NtDeleteFile | InternetConnectW |
| HttpQueryInfoW | ShowCertificate | NtLockFile | UrlCacheValidate |
| DbgPrint | LdrAddRefDll | LdrLoadDll | LdrSetDllDirectory |
| GetProcessTimes | NtAddAtom | NtAllocateUuids | NtCancelTimer |
| NtCompactKeys | NtClose | NtClearEvent | HttpEndRequestA |
| MD5Final | NtCreateSection | NtCreateTimer | InternetCrackUrlA |
| NtFlushKey | NtLoadKey | NtLoadDriver | ShowSecurityInfo |
| NtOpenKey | NtOpenFile | NtOpenTimer | NtQueryEvent |
| NtQueryKey | NtQueryTimer | BackupRead | BackupWrite |
| CloseState | CloseStateLock | CopyContext | CreateMutexA |
| CtrlRoutine | DebugBreak | DecodePointer | DeleteFileTransactedW |
| ExitThread | FindAtomW | FreeResource | GetProcessIdOfThread |

## A.2 List of Selected Features

SetFileTime
FindFirstFileA
FindClose
FindClose
GetWindowsDirectoryA
SetFileAttributesW
GetSystemDirectoryA
SetFileAttributesW
CloseHandle
GlobalFree
VirtualFree
RegSetValueExW
GetProcAddress
CopyFileA
GetModuleHandleA
DeleteFileW
RegQueryValueExW
BitBlt
GetCurrentProcessId
Sleep
CreateProcessW
GetFileInformationByHandle
GlobalLock
CorExeMain
UnMapViewofFile
CreateFileMappingW
MapViewofFile
FindNextFileA
RegCreateKeyExW
CreateFileA OpenFile
cell8
WriteFileA
GetFileTime
VirtualProtect
RegOpenKeyExA
RegCloseKey
WriteProcessMemory
ExitProcess
ReadFile
LocalFree
Accept
Connect
GetLastError
GetFileSize
FileTimeToSystemTime
CreateThread
GlobalUnlock

# Appendix B

# Performance Evaluation Metrics

Table B.1: Sensitivity By Class

| Model | Adware | Backdoor | Benign | Trojan Horse | Virus | Worm |
|---|---|---|---|---|---|---|
| kNN | 76 | 40 | 53.84 | 70 | 79.17 | 100 |
| LDA | 77.27 | 81.25 | .8181 | 52.27 | 79.17 | 100 |
| DT | .8800 | 52 | 69.23 | 86.67 | 95.83 | 100 |
| NN | 96 | 88 | 76.92 | 96.66 | 95.83 | 96.96 |
| RF | 96 | 68 | 69.23 | 100 | 95.83 | 100 |
| SVM | 96 | 88 | 69.23 | 100 | 95.83 | 100 |
| Proposed model | 100 | 94.44 | 100 | 84.5 | 92.31 | 100 |

Table B.2: Specificity By Class

| Model | Adware | Backdoor | Benign | Trojan Horse | Virus | Worm |
|---|---|---|---|---|---|---|
| kNN | 93.6 | 94.4 | 94.89 | 88.33 | 96.03 | 100 |
| LDA | 93.75 | 91.04 | 97.12 | 93.40 | 96.03 | 100 |
| DT | 97.6 | 98.4 | 100 | 90.83 | 93.65 | 100 |
| NN | 99.2 | 99.2 | 100 | 94.21 | 98.42 | 100 |
| RF | 100 | 100 | 100 | 90.83 | 97.62 | 100 |
| SVM | 100 | 100 | 100 | 93.33 | 99.21 | 100 |
| Proposed model | 100 | 95.21 | 100 | 97.29 | 100 | 100 |

Table B.3: Precision By Class

| Model | Adware | Backdoor | Benign | Trojan Horse | Virus | Worm |
|---|---|---|---|---|---|---|
| kNN | 70.37 | 58.82 | 50 | 60 | 79.17 | 100 |
| LDA | 68 | 52 | 69.23 | 76.67 | 79.17 | 100 |
| DT | 88 | 86.6 | 100 | 70.27 | 74.19 | 100 |
| NN | 96 | 95.65 | 100 | 80.55 | 92 | 100 |
| RF | 100 | 100 | 100 | 73.17 | 88.46 | 100 |
| SVM | 100 | 100 | 100 | 78.95 | 95.83 | 100 |
| Proposed model | 100 | 77.27 | 100 | 92.11 | 100 | 100 |

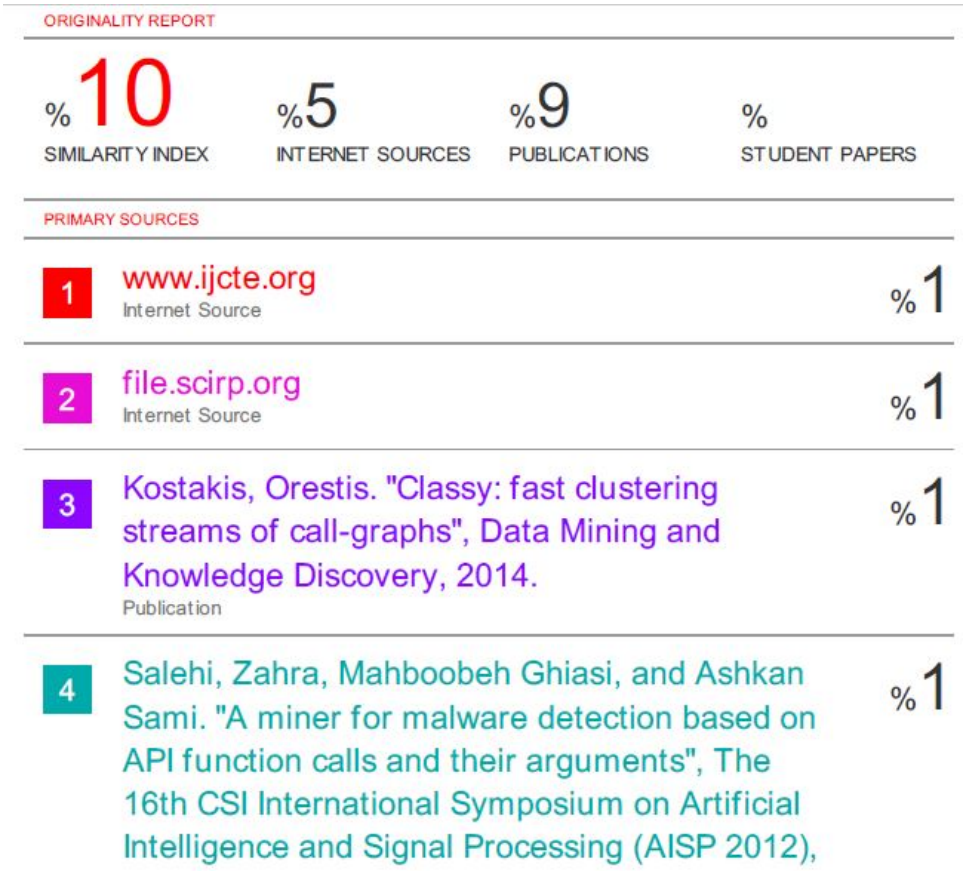
Table B.4: FNR By Class

| Model | Adware | Backdoor | Benign | Trojan Horse | Virus | Worm |
|---|---|---|---|---|---|---|
| kNN | 24 | 60 | 46.16 | 30 | 20.83 | 0 |
| LDA | 22.73 | 18.75 | 18.19 | 49.93 | 20.83 | 0 |
| DT | 12 | 48 | 30.77 | 13.33 | 4.17 | 0 |
| NN | 4 | 12 | 73.08 | 3.34 | 04.17 | 3.04 |
| RF | .06 | .32 | .3077 | 0 | .0417 | 0 |
| SVM | 4 | 12 | 30.77 | 0 | 4.17 | 0 |
| Proposed model | 0 | 5.56 | 0 | 15.50 | 7.69 | 0 |

Table B.5: FPR By Class

| Model | Adware | Backdoor | Benign | TrojanHorse | Virus | Worm |
|---|---|---|---|---|---|---|
| kNN | 6.4 | 5.6 | 5.11 | 11.67 | 3.9 | 0 |
| LDA | 6.25 | 8.96 | 2.88 | 9.34 | 3.97 | 0 |
| DT | 2.4 | 1.6 | 0 | 9.17 | 6.35 | 0 |
| NN | 0.8 | 0.8 | 0 | 5.79 | 1.58 | 0 |
| RF | 0 | 0 | 0 | 9.17 | 2.38 | 0 |
| SVM | 0 | 0 | 0 | 6.67 | 0.79 | 0 |
| Proposed approach | 0 | 4.79 | 0 | 2,71 | 0 | 0 |

# Plagiarism Report



ORIGINALITY REPORT

| %10 | %5 | %9 | % |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | www.ijcte.org<br>Internet Source | %1 |
| 2 | file.scirp.org<br>Internet Source | %1 |
| 3 | Kostakis, Orestis. "Classy: fast clustering streams of call-graphs", Data Mining and Knowledge Discovery, 2014.<br>Publication | %1 |
| 4 | Salehi, Zahra, Mahboobeh Ghiasi, and Ashkan Sami. "A miner for malware detection based on API function calls and their arguments", The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012), | %1 |