ABSTRACT

Malicious code is an increasing problem around the world. Research indicates that malicious software has an adverse impact on vulnerable systems. The main defense technologies against malicious software are malware detectors. But it is almost impossible for security experts to detect malware by traditional or human-centered methods because new malwares are becoming more sophisticated. It is a big challenge for us to develop new strategies against new malwares. Therefore, it is imperative that we research and comprehend the successes and failures of anti-malware methodologies. Moreover, one of the challenges with digitally stored data is the provision of more efficient ways for detection of malware to ensure the security. In this dissertation, alternative malware detection approaches that improve speed, false alarm rates, model losses and resistant to various forms of attacks, together with a security scheme that provides the complementarity of malware detection techniques are proposed.

This dissertation gives a detail overview of the malware, and the detection techniques used by security experts, also discusses the role of machine learning in the field of malware detection. Further, the dissertation proposes a model of reverse engineering methodology for malware detection, where two distinct models are utilized to identify the obscure or new sort of malware. GoogleNet and ResNet models are researched and tried which belong to two different platforms i.e. ResNet belongs to Microsoft and GoogleNet is the intellectual property of Google. Two sorts of datasets are utilized for training and validation the models. One of the datasets was downloaded from Microsoft which is the combination of 10868 records and these records are binary records. These records are additionally isolated in nine diverse families. Second dataset is considerate dataset and it contains 3000 benign files. The said datasets were initially in the form of EXE files and were changed over into opcode, after that changed over into images.

The dissertation further aimed to propose an improved CNN model to detect malware in the high-speed networks. The tracking of network intrusion is an essential part of cyber security. Currently, the popular detection technology used the traditional machine learning algorithms to train the intrusion samples, to obtain the intrusion detection model. However, the traditional machine learning algorithms have the disadvantage of low detection rate. Deep learning is more advanced technology that automatically extracts features from samples. Since the accuracy of intrusion detection is not high in traditional machine learning technology, this dissertation proposes a network intrusion detection model based on CNN algorithm. Experimental results on KDD99 data sets demonstrate that the current proposal will enhance intrusion detection precision significantly.

To further improve the network security, this dissertation proposes a hybrid botnet detection technique based on two-stage detection method for P2P botnets. The first stage is based on port judgment, DNS query and data flow count in the session to filter non-P2P traffic; and the second stage is based on session characteristics to identify P2P botnet. The method is used on the bases of session feature to effectively reduce the data packets to be analyzed. Furthermore, Machine Learning algorithms are used to classify and identify the traffic.

Lastly, for comprehensive security of computer networks, this dissertation aimed to propose a botnet detection technique that is multi-layered model, and which is more efficient as compared to existing machine learning and other published models. It is difficult to identify Peer-to-peer botnets as compare to Hypertext Transfer Protocol (HTTP), Internet Relay Chat (IRC), and other types of botnets because P2P traffic has typical features of the centralization and distribution. To resolve the issues of P2P botnet identification, this dissertation proposes an effective multi-layer traffic classification method by applying deep learning optimization algorithms to network traffic features.

In general, this dissertation presents a compact multiple malware detection technique to realize the operation and vulnerabilities of malicious codes. The aforementioned efforts contribute to the development of cyber security strategies to ensure the security of digital infrastructures. This work presents methods on signature-based and anomaly-based malware detection using deep learning algorithms. Convolutional Neural Network is used for signature-based malware detection and decision trees for anomalies detection which effectively detects P2P botnets. Decision tree is applied for feature selection to pick up the most relevant features and ignore the irrelevant features in network traffic.

**Keywords:** Malware Detection, Intrusion Detection, Botnet Detection, Image Processing, Machine Learning, Anomaly Detection, Feature Extraction

Contents

[Chapter 1 Introduction 1](#_Toc3209632)

[1.1 Background 1](#_Toc3209633)

[1.2 Problem Statement 1](#_Toc3209634)

[1.2.1 Malware Analysis through reverse Engineering and feature extraction 1](#_Toc3209635)

[1.2.2 Intrusion Detection in Computer Networks 2](#_Toc3209636)

[1.2.3 Botnet Detection Based on Two-Stage Technique 2](#_Toc3209637)

[1.2.4 Botnet Detection Based on Multi-Layer Technique 2](#_Toc3209638)

[1.3 Motivation 2](#_Toc3209639)

[1.4 Aim and Objectives 2](#_Toc3209640)

[1.5 The contribution of the Dissertation 2](#_Toc3209641)

[1.6 Outline of the Dissertation 2](#_Toc3209642)

[1.7 Summary 2](#_Toc3209643)

[Chapter 2 Theoretical Background and Literature Review 2](#_Toc3209644)

[2.1 Overview of Malware 2](#_Toc3209645)

[2.1.1 What is Malware? 2](#_Toc3209646)

[2.1.2 What is Cleanware? 2](#_Toc3209647)

[2.2 Types of Malware 2](#_Toc3209648)

[2.2.1 Trojans or Remote Access Trojans (RAT) 2](#_Toc3209649)

[2.2.2 Worms and Viruses 2](#_Toc3209650)

[2.2.3 Rootkit 2](#_Toc3209651)

[2.2.4 Bootkit 2](#_Toc3209652)

[2.2.5 Botnets 2](#_Toc3209653)

[2.2.6 Backdoor 2](#_Toc3209654)

[2.2.7 Ransomware 2](#_Toc3209655)

[2.2.8 Downloader 2](#_Toc3209656)

[2.2.9 Reverse Command Shell 2](#_Toc3209657)

[2.2.10 Browser Hijacker 2](#_Toc3209658)

[2.2.11 Information Stealing malware 2](#_Toc3209659)

[2.2.12 Scareware 2](#_Toc3209660)

[2.2.13 Spamware 2](#_Toc3209661)

[2.2.14 Spyware 2](#_Toc3209662)

[2.2.15 Trackware 2](#_Toc3209663)

[2.2.16 Adware 2](#_Toc3209664)

[2.2.17 Potentially Unwanted Programs 2](#_Toc3209665)

[2.3 Malware Analysis and Detection Techniques 2](#_Toc3209666)

[2.3.1 Anomaly Based Detection 2](#_Toc3209667)

[2.3.1.3 Hybrid Detection (Anomaly Based) 2](#_Toc3209668)

[2.3.2 Signature Based Detection 2](#_Toc3209669)

[2.4 The Role of Artificial Intelligence in Malware Detection 2](#_Toc3209670)

[2.4.1 Malicious Code Detection using Machine and Deep Learning 2](#_Toc3209671)

[2.4.2 Intrusion Detection using Machine and Deep Learning 2](#_Toc3209672)

[2.4.3 Botnet Detection using Machine and Deep Learning 2](#_Toc3209673)

[2.5 Summary 2](#_Toc3209674)

[Chapter 3 Malware Detection Through Reverse Engineering and Feature Extraction 2](#_Toc3209675)

[3.1 Introduction 2](#_Toc3209676)

[3.2 Proposed Model 2](#_Toc3209677)

[3.3 Data Preparation 2](#_Toc3209678)

[3.3.1 Dataset 2](#_Toc3209679)

[3.3.2 Environmental Setup 2](#_Toc3209680)

[3.3.3 Data Preparation Technique 2](#_Toc3209681)

[3.3.4 Malware Detection by GoogleNet Model 2](#_Toc3209682)

[3.3.5 Malware Detection by ResNet Model 2](#_Toc3209683)

[3.4 Discussion on Implementation and Evaluation Results 2](#_Toc3209684)

[3.5 Summary 2](#_Toc3209685)

[Chapter 4 An Improved Convolutional Neural Network Model for Intrusion Detection 2](#_Toc3209686)

[4.1 Introduction 2](#_Toc3209687)

[4.2 Proposed Model Design 2](#_Toc3209688)

[4.2.1 The architecture of Convolutional Neural Network Model 2](#_Toc3209689)

[4.2.2 Intrusion Detection Model Framework 2](#_Toc3209690)

[4.3 Data Preparation, Experiments and analysis 2](#_Toc3209691)

[4.3.1 Dataset 2](#_Toc3209692)

[4.3.2 Data preprocessing and Attack Detection 2](#_Toc3209693)

[4.3.3 Setting Experimental Parameters 2](#_Toc3209694)

[4.3.4 Analysis of Experimental Results 2](#_Toc3209695)

[4.4 Summary 2](#_Toc3209696)

[Chapter 5 A Hybrid Technique to Detect Botnets based on P2P Traffic Similarity 2](#_Toc3209697)

[5.1 Introduction 2](#_Toc3209698)

[5.2 Approach 2](#_Toc3209699)

[5.3 Proposed Model 2](#_Toc3209700)

[5.3.1 First Stage of Traffic Classifier: 2](#_Toc3209701)

[5.3.2 Second Stage of Traffic Classification 2](#_Toc3209702)

[5.4 Experimental Results and Analysis 2](#_Toc3209703)

[5.4.1 Evaluating Metrics 2](#_Toc3209704)

[5.5 Dataset and Experimental setup 2](#_Toc3209705)

[5.6 Classifier Selection 2](#_Toc3209706)

[5.7 Summary 2](#_Toc3209707)

[Chapter 6 An Adaptive Multi-Layer Botnet Detection Technique 2](#_Toc3209708)

[6.1 Introduction 2](#_Toc3209709)

[6.2 Proposed Scheme 2](#_Toc3209710)

[6.2.1 Multi-Layer Detection Method 2](#_Toc3209711)

[6.2.2 Traffic Reduction 2](#_Toc3209712)

[6.2.3 P2P and Non-P2P Traffic Classification 2](#_Toc3209713)

[6.2.4 Feature Extraction and Feature Reduction 2](#_Toc3209714)

[6.2.5 P2P Traffic Classification 2](#_Toc3209715)

[6.3 Evaluating Metrics 2](#_Toc3209716)

[6.3.1 The accuracy of Detection Model 2](#_Toc3209717)

[6.3.2 False Alarm Rate 2](#_Toc3209718)

[6.4 Dataset and Experimental setup 2](#_Toc3209719)

[6.5 Detection Results and Discussion 2](#_Toc3209720)

[6.6 Comparison with other Classifiers 2](#_Toc3209721)

[6.7 Summary 2](#_Toc3209722)

[Chapter 7 Conclusion and Future Work 2](#_Toc3209723)

[7.1 Conclusion 2](#_Toc3209724)

[7.2 Future Work 2](#_Toc3209725)

[Acknowledgments 2](#_Toc3209726)

[References 2](#_Toc3209727)

[Research Results Achieved During the Study for Doctoral Degree 2](#_Toc3209728)

List of Figures

[Figure 1-1 A Systematic View of the Dissertation 2](#_Toc3209837)

[Figure 2-1 Malware Detection Techniques 2](#_Toc3209838)

[Figure 2-2 Characterizing the behaviors in Anomaly Based Detection 2](#_Toc3209839)

[Figure 2-3 Anomaly Based Detection Approach 2](#_Toc3209840)

[Figure 2-4 Signature Based Detection 2](#_Toc3209841)

[Figure 2-5 Signature Based Malware Detection Approach 2](#_Toc3209842)

[Figure 3-1 Architecture of Proposed Model i.e. from Data preparation to the prediction 2](#_Toc3209843)

[Figure 3-2 Implementation of Deep Learning 2](#_Toc3209844)

[Figure 3-3 Overview architecture of Preparation Dataset using opcode Environmental Settings 2](#_Toc3209845)

[Figure 3-4 Inception modules in GoogleNet Model 2](#_Toc3209846)

[Figure 3-5 ResNet Modules, the bottleneck residual blocks are used in ResNet deeper model 2](#_Toc3209847)

[Figure 3-6 ResNet model structure of the 34-layer plain network 2](#_Toc3209848)

[Figure 3-7 ResNet Model structure of the 34-layer residual network 2](#_Toc3209849)

[Figure 3-8 Typical structure of ResNet-50 2](#_Toc3209850)

[Figure 3-9 Model Accuracy and Model Loss of ResNet 18 2](#_Toc3209851)

[Figure 3-10 Model Accuracy and Model Loss of ResNet 34 2](#_Toc3209852)

[Figure 3-11 Model Accuracy and Model Loss of ResNet 50 2](#_Toc3209853)

[Figure 3-12 Model Accuracy and Model Loss of ResNet 101 2](#_Toc3209854)

[Figure 3-13 Model Accuracy and Model Loss of ResNet 152 2](#_Toc3209855)

[Figure 3-14 Execution Time of Different Models 2](#_Toc3209856)

[Figure 3-15 Comparison of the Training and Testing accuracy of Different Models 2](#_Toc3209857)

[Figure 4-1 Convolutional neural network classical structure diagram 2](#_Toc3209858)

[Figure 4-2 Proposed Model for Intrusion Detection 2](#_Toc3209859)

[Figure 4-3 Hyperplane of the maximum margin between two classes 2](#_Toc3209860)

[Figure 4-4 Increasing the accuracy of test results by increasing the number of epochs 2](#_Toc3209861)

[Figure 4-5 Comparison of Improved CNN and other models 2](#_Toc3209862)

[Figure 5-1 Architecture of the proposed method 2](#_Toc3209863)

[Figure 5-2 First Stage Traffic Classifier 2](#_Toc3209864)

[Table 5-1 Common applications and their corresponding ports 2](#_Toc3209865)

[Figure 5-3 K-Fold Cross Validation 2](#_Toc3209866)

[Figure 5-4 Wrapper Method for Features Selection 2](#_Toc3209867)

[Figure 5-5 Comparison of three machine learning classifiers on P2P botnet detection 2](#_Toc3209868)

[Figure 6-1 Difference between centralized and decentralized P2P botnets. 2](#_Toc3209869)

[Figure 6-2 Overlapping qualities among three types of traffic. 2](#_Toc3209870)

[Figure 6-3 Architecture of the proposed method 2](#_Toc3209871)

[Figure 6-4 Flow chart for network traffic reduction 2](#_Toc3209872)

[Figure 6-5 First Stage Traffic Classifier 2](#_Toc3209873)

[Figure 6-6 Flow chart for Feature Reduction 2](#_Toc3209874)

[Figure 6-7 Final Layer of Traffic Classifier 2](#_Toc3209875)

[Figure 6-8 Network traffic capture with Wireshark 2](#_Toc3209876)

[Figure 6-9 Network traffic capture with Wireshark 2](#_Toc3209877)

[Figure 6-10 Benign Network Traffic Capture 2](#_Toc3209878)

[Figure 6-11 Neris Botnet Capture 2](#_Toc3209879)

[Figure 6-12 RBot Botnet Capture 2](#_Toc3209880)

[Figure 6-13 Donbot Botnet Capture 2](#_Toc3209881)

[Figure 6-14 Sogou Botnet Capture 2](#_Toc3209882)

[Figure 6-15 Nerris-2 Botnet Capture 2](#_Toc3209883)

[Figure 6-16 Fast-Flux Botnet Capture 2](#_Toc3209884)

[Figure 6-17 Bot-2 Botnet Capture 2](#_Toc3209885)

[Figure 6-18 Bot-3 Botnet Capture 2](#_Toc3209886)

[Figure 6-19 Web traffic identification under different thresholds during the training process 2](#_Toc3209887)

[Figure 6-20 Comparison with other machine learning classifiers on P2P botnet detection 2](#_Toc3209888)

List of Tables

[Table 2-1 Advantages and Limitations of Anomaly-Based Malware Detection 2](#_Toc3209889)

[Table 2-2 Advantages and Limitations of Signature-Based Malware Detection 2](#_Toc3209890)

[Table 3-1 Data set for Microsoft Malware Classification 2](#_Toc3209891)

[Table 3-2 Comparison Results of GoogleNet and ResNet Models 2](#_Toc3209892)

[Table 4-1 Distribution of KDD99 datasets 2](#_Toc3209893)

[Table 4-2 Metrics for Prediction 2](#_Toc3209894)

[Table 4-3 Comparison of SVM, DBN, and Improved CNN models 2](#_Toc3209895)

[Table 6-1 Common applications and their corresponding ports 2](#_Toc3209896)

[Table 6-2 Session characteristics 2](#_Toc3209897)

[Table 6-3 Packet counts of the different captured scenarios 2](#_Toc3209898)

[Table 6-4 Comparison with other published work and other machine learning classifiers 2](#_Toc3209899)

Chapter 1 Introduction

1.1 Background

Today, we live in the digital world and the major portion of the internet user understands that our personal data is far more resilient than before [1]. There are many new revelations about ID frauds and data theft, with the consequences felt by the huge number of customers, and through banks and financial institutions are actively planning to safeguard themselves by enhanced safety measures, we can also play a vital role in this battle. Cybersecurity is not just about companies and governments. Our computers, tablets and cell phones are likely to reveal personal data that cybercriminals, e.g., suspicious IP addresses, names and birth dates would like to have. If a scammer gets access to my account details, he or she can quickly send a text or email to someone I know, using my name to scroll on a malicious link. In a globalized world, we have an obligation to defend our systems and the individuals with whom we communicate, and everything begins with a knowledge of information security.

1.2 Problem Statement

Our primary goal is to systematically study the evolution of characteristics of malware and gain the representative conclusions and insights for the security community. This dissertation analyzes the different types of malware, comprehensively and proposes four analyses and techniques for malware detection that are elaborately described below.

1.2.1 Malware Analysis through reverse Engineering and feature extraction

One of the significant challenges in the realm of security threats is malicious software which is also referred to as malware. The main focus of malware is, to gather personal information without the attention of users and to disturb the computer operations which makes problems for users. A lot of malicious software types are introduced by attackers, i.e. Virus, worm, rootkit, trojan horse, backdoor, spyware, adware, and so on. Annual antivirus reports show that lots of new malicious software are developed each day [2]. All these new malwares are becoming more robust that conventional detection methods can no longer detect them.

Malware classified in different families has multiple characteristics or features. Many authors used machine learning models such as Regression, K-nearest-neighbor, Random Forest, etc. The main disadvantage of using machine learning is, features extraction is manual. [3] provided an analysis of various machine learning strategies for malware detection that had recently been proposed by different researchers. Unlike Machine Learning, the manual steps of extracting features are skipped in deep learning. For instance, we can feed images and videos directly to the deep learning model, that can recognize objects in images and videos. In other words, the deep learning model is more intelligent than machine learning model. This dissertation aimed to use convolutional neural networks due to its reliability, and it can be adhered to the entire image and we can presume they are best to use for feature extraction. Recently Convolutional Neural Networks [4] is the new approach to detect malware by using image-based similarity technique. Its automated analogy helps experts visually classify prevalent chunks of code or precise blocks of instructions within a sample. In this work, we used three different datasets and compared the accuracy. Secondly, we used different techniques to prepare datasets for training and testing purposes. we trained and tested the CNN model for better understanding of the malware behavior.

1.2.2 Intrusion Detection in Computer Networks

The detection of network intruders is an essential component of network security. Currently, the popular detection technology used the traditional machine learning algorithms to train the intrusion samples, to obtain the intrusion detection model. However, these algorithms have the disadvantage of low detection rate [5]. Deep learning is a more advanced technology that automatically extracts features from samples. Since the accuracy of intrusion detection is not high in traditional machine learning technology, this study proposes a network intrusion detection technique based on convolutional neural network algorithm. The model can automatically extract the effective features of intrusion samples so that the intrusion samples can be accurately classified.

1.2.3 Botnet Detection Based on Two-Stage Technique

Research works on botnets among our surveyed literature focuses mainly on designing systems to detect command and control (C&C) botnets, where many bot-infected machines are controlled and coordinated by few entities to carry out malicious activities [6]. Those systems need to learn decision boundaries between human and bot activities; therefore ML-based classifiers are at the core of those systems and are often trained by labeled data in supervised learning environments. Clustering is mostly used in natural language processing (NLP), to build a large-scale scheme to identify bot queries [7]. In botnet detection literature, two core assumptions are widely shared:

i. Botnet protocols are mostly C&C.

ii. Botnet behaviors are different and distinguishable from the legitimate human user, e.g., human behaviors are more complex [8].

Other stronger assumptions include that the bots and humans interact with different server groups, and features are independent which are generated by bots and humans, from different messages. Classification techniques, e.g., Weighted Least Square, Binary Classifier and hypothesis testing, are common system components [9]. Attempts have been made to abstract state machine models of the network to simulate real-world network traffic and create honeypots. The ground reality is often heuristic, labeled by human experts, or a combination is used, for example, the game masters visual inspections serve as ground truth to detect bots in online games [10]. In retrospect, the evolution of botnet detection is clear from earlier and more straightforward uses of classification techniques such as clustering and NB, the research focus has been expanded from the last step of classification to the important preceding step of constructing suitable metrics, that measures and distinguishes bot-based and human-based activities [7][8].

1.2.4 Botnet Detection Based on Multi-Layer Technique

In recent years, the botnets have been the most common threats to network security since it exploits multiple malicious codes like a worm, Trojans, Rootkit, etc. The botnets have been used to carry phishing links, to perform attacks and provide malicious services on the internet. It is challenging to identify Peer-to-peer botnets as compare to Internet Relay Chat (IRC), Hypertext Transfer Protocol (HTTP) and other types of botnets because P2P traffic has typical features of the centralization and distribution. To resolve the issues of P2P botnet identification, an effective is propose multi-stage traffic classification method by applying machine learning classifiers on features of network traffic. This work presents a method based on decision trees which effectively detects P2P botnets. Decision tree is applied for feature selection to pick up the most relevant features and ignore the irrelevant features. At the first stage of the proposed model, filtered non-P2P packages to reduce the amount of network traffic through well-known ports, DNS query, and flow counting. At the final stage, we successfully detected P2P botnets using decision tree Classifier by extracting network communication features.

1.3 Motivation

Malware is a malicious coded system designed to attack computer systems. Malware is developing in the enormous number consistently. It carries worms, adware, spyware, viruses, Trojan horses, etc. This malware is intended to steal sensitive information or acquire root rights and benefits. It's an epidemic around the world and Studies have suggested that the effect of malicious software is worsening every day, especially in the financial sector [1][2]. A security breach can influence almost everything that communicates on the Internet or is linked to a computer or other intelligent electronic device. Including:

 Internet communication such as email, telephone and text messages

 Transport systems, such as air traffic control, air navigation, road traffic

 Financial networks such as personal bank accounts, personal loans, and salaries, mortgages, etc.

 Medical systems, such as medical records and automated equipment

 Education systems, such as grades, report cards, and research information

Therefore, understanding various kinds of malware, their negative impact and the countermeasure methodologies is significant and relevant. Security problems in automated networks around us can afflict us with malware, so we must safeguard our devices from malicious software assaults using various detection methodologies. This research is therefore focused on different malware detection methodologies.

1.4 Aim and Objectives

This research seeks to provide computationally efficient and robust alternative security schemes based on hybrid approaches involving malware detection, intrusion detection, and botnet detection to ensure security from different kind of threats. Specifically, it aims at:

 Improving the speed and robustness of the CNN model for malware detection. This is achieved by decoding the malicious software using opcode technique and analyzing with improved CNN model.

 Designing an intrusion detection scheme that has high accuracy, is robust against different kinds of attacks and produces better recovery outcomes.

 Designing a two-stage botnet detection model which can effectively classify the network traffic.

 Providing comprehensive security that ensures that the network is protected against threats. This is achieved using a multi-layer botnet detection scheme.

1.5 The contribution of the Dissertation

Novel hybrid security approaches are proposed in this dissertation against different types of malware. Several techniques have been combined to provide efficient algorithms for security against different types of malware.

More importantly, two very efficient algorithms: one that eliminates the limitations of recently proposed malware detection algorithms; and another that introduces a novel approach for botnet detection have been proposed. A summary of the main contributions of this dissertation is given below:

 The Convolutional Neural Networks for detection of malware, based on image similarity has been proposed which is further described in chapter 3. The gray scale image and opcode sequence accuracy is achieved further the model is successfully analyzed and detected an unknown or new type of malware. An opcode sequence algorithm is built which takes less time to train the classifier. Better results are achieved in terms of training/testing accuracy and speed of detection which is further described in chapter 3.

 This dissertation studies the network intrusion detection based on convolutional neural networks (CNN) and combines the convolution and pooling operations to extract the feature relationships between the data in better ways. This technique solves the failures and problems of traditional machine learning models. The deep-seated mining of the relationship between data features and the better understanding of the relationships between features than general neural networks, further described in chapter 4.

 The dissertation further describes the methods proposed in chapter 5 and 6 to detect bot-bling traffic in different phases. The focus of this method is on non-P2P traffic filtering and the extraction of the characteristics of the session.

 In Chapter 6, this research presents a multi-layer approach to classify network traffic (P2P botnet traffic and non-P2P traffic) and identify botnets by applying machine learning classifier. The second stages such as port filtering, DNS query, and flow counting. The proposed technique of this study covers the limitations of single stage botnet detection, e.g., class imbalance. The accuracy of our model is 98.7 % because the threshold of false alarm rate was reduced to 3. The accuracy was improved up to 99% by considering the factor if benign files also send out search requests consistently so benign file may be reported as botnets. Additionally, it was observed the accuracy might be improved by increasing the epochs of deep learning algorithms at the expense more execution cost. To validate the performance of our proposed technique, we done the experiments on diverse datasets and the results are compared with five machine learning algorithms implemented for botnet detection.

1.6 Outline of the Dissertation

Following this introductory chapter, the rest of the dissertation is organized as follows also shown in Figure 1-1:

In chapter 2, theoretical background concepts about malware and types of malware are presented. It begins with overviews of malware concepts, the theory of malware types, detection and prevention techniques in the literature review. The chapter continues to review some key published works on malware detection, intrusion detection, and botnet detection. In addition, the evaluation metrics are detailed in this chapter with the aim of avoiding repetition of the same discussion throughout all subsequent chapters.

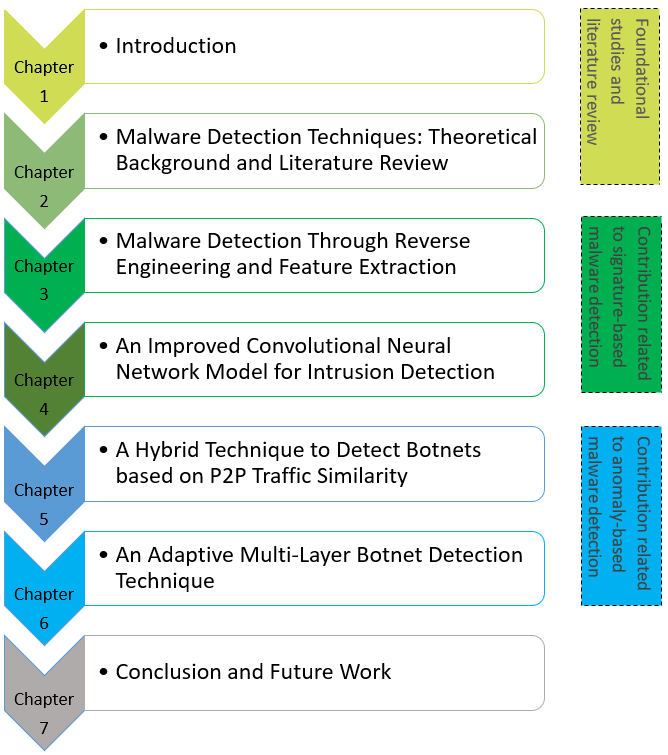


Figure 1-1 A Systematic View of the Dissertation

In chapter 3, a model is proposed to detect malicious code based on the opcode reverse engineering technique. Overviews of the ResNet model and GoogleNet models are given. Experimentation and performance evaluation are then given in this chapter.

In chapter 4, an efficient model is proposed which successfully detects intrusions in computer networks. An improved CNN model is explained first. The details of the algorithm are presented, followed by experiments and performance assessment.

Chapter 5 presents a botnet detection technique based on a two-stage strategy. The classification systems used for the scheme are first presented, followed by detail descriptions, experimentation and performance evaluation of the algorithm.

Chapter 6 presents a botnet detection technique based on multi-layer strategy. The layers of the botnet detection scheme are first discussed in detail, followed by a brief description of the experimentation and discussion of results.

Finally, in chapter 7, the concluding remarks of the dissertation are given. This is followed by some potential directions for work to be done in the future.

1.7 Summary

Malicious software’s are the most significant security threats to our private data. Collecting information and studying on the discovered malware on the Internet is undoubtedly important. Moreover, mitigation and detection strategies based on the observations gained from computational modeling are even more important. This chapter briefly discussed the significance of this dissertation in terms of background, problem statement, motivation towards this project, aim, and objectives of the project and contribution of the project. This chapter also presented the outlining of all the upcoming chapters in this dissertation.

Chapter 2 Theoretical Background and Literature Review

Reviewing the related research is an essential step for systematic analysis. Accordingly, this chapter describes a detailed literature review, related to content-based image retrieval systems, low-level feature extraction, and integration; image segmentation approaches, object analysis and scene understanding, and applications of contemporary integration of CBIR as below.

2.1 Overview of Malware

2.1.1 What is Malware?

According to Khan et al., [11] malware is a malicious program that is intended and engineered to gain access to a management system without the approval of the administrator. It enables an intruder to fulfill his manipulative and criminal activity intentions. The malware can paralyze or destabilize the procedure of the system, enabling intruders to access top secret and sensitive data and to spy on private and personal computers. There are millions of malware discovered currently, and the list is expanding day by day [1][2]. Some of them are discussed in detail in section 2.2 of this chapter.

2.1.2 What is Cleanware?

Cleanware is a software program that is not deemed to be malevolent. To make sure that an unidentified file is not malevolent, it is important to distinguish malicious software from Cleanware. Examples might be:

• System software’s that coordinate with the hardware system and provides an environment to work with the hardware system.

• Open annexes from reputable sources in e- mails or text messages.

• Add the external disks or Flash drives on our system from legitimate companies.

• Application software or Legal computer software that belongs to established distributors or companies like Microsoft, Norton, Google, Yahoo et.

2.2 Types of Malware

Cybercriminals explicitly design malicious software to be maneuverable so it can remain on the targeted system for a prolonged period of times without the user's consent or knowledge. Malware generally portrays themselves as Cleanware, but the impacts of such malware often harm users, they are disastrous for businesses. If malicious software is continued to spread over a data network, it can cause massive harm and disturbance, requiring additional organizational rescue operations.

It is a symbolic sign that something is terrible to run into the phrase that begins with mal "malware". The term malware is generally seen by most scientists as a deceleration of two phrases "malicious software". By purposeful renovation, the word has bad overtones, but the actual malware evolutionary taxonomy seems to be less clear. This is because, on our computer journey, we can experience a maximum range of malware, with new forms and classifications of the threats taking shape as the world is moving towards a digital era.

Recently a new study in 2017 [12] discovered that malicious software for portable devices such as laptops, smartphones and netbooks is growing and sometimes even pre-installed by the companies. Let's see how intruders configure and activate such provocations and some notorious and harmful examples of these malicious software types, what are the different types of malicious software’s and how are they categorized?

2.2.1 Trojans or Remote Access Trojans (RAT)

A Remote Access Trojan (RAT) is the form of malicious software that gives an intruder backdoor access and enables a contaminated host to be monitored and controlled via a back channel. RAT’s are frequently transmitted via open source software and sent through e-mails as attachments [13].

2.2.2 Worms and Viruses

Malicious code in the form of virus aimed at copying and infecting more systems. Viruses do not generally change other programs. On the other hand, worms, often look for a specific system requirement and change it when they find it. The SCADA systems targeting Stuxnet are the biggest outbreak [14][15].

2.2.3 Rootkit

Rootkits are the form of malicious software’s which are built to conceal other software and is usually linked to other malicious software, such as a keylogger. This allows the intruder to keep remote access and make it hard for researchers to detect the code [16].

2.2.4 Bootkit

Bootkit is a type of rootkit. It can be easily understood by its name that it is concealed in the boot sector and after its infection, it is difficult for anti-malware and virus scanners to detect these kinds of codes [17].

2.2.5 Botnets

A backdoor-like software, with the distinction that the harmed computer systems create a network of hackers receiving instructions from a server known as a command & control server. The botnets have been used to carry phishing links, to perform attacks and provide malicious services on the internet [18]. It is difficult to identify Peer-to-peer botnets as compare to HTTP, IRC and other types of botnets because P2P traffic has typical features of the centralization and distribution.

2.2.6 Backdoor

Software that automatically installs itself on the systems which make "a door" to link up intruders to the systems. With little or no identity verification, these kind of software’s creator, achieve and execute code on the system [19][20].

2.2.7 Ransomware

Ransomware is among the most popular malicious software to keep running on all operating systems. The main objective of any ransomware is to scare a contaminated user to purchase something from the attacker [21]. It usually has an interface with material information to make disbursements. The ransomware cautions the consumer that crypto protocols on their private data system contain malware and that payment with cryptocurrencies is the only way of getting rid of it. After getting payments which is also called extortion of ransom, they will provide key to the user to open their encrypted system as a return if it does nothing more than confiscate money from people or demolish computer systems.

2.2.8 Downloader

A downloader is a software encoded in webpages, system engineering, desktops etc., that have certain malicious software to download [22].

2.2.9 Reverse Command Shell

The intruders have full access to the host which has initially been damaged with reverse shell malware or provides the attacker unauthorized approval to interact with an infected system. Their interfaces work on the contaminated network as a backdoor. The way a reverse shell operates is that it allows the intruder to execute commands and type them as the attacker is local. For packaged reverse shells, Windows cmd.exe and Netcat are widely used. Such mechanisms are used to conceal the contaminated information system of the user, which gives the time to implement commands on the contaminated host [23].

2.2.10 Browser Hijacker

Browser hijacking is a malicious activity in which malware is engineered and configured to change the home page of a website e.g., search engines. They are always installed by open source software’s and intended to target the more newbie user who does not regard them as malevolent [24]. They are malevolent because spyware or adware mostly have access to the privacy of an online user.

2.2.11 Information Stealing malware

Information-stealing malware is also known as password stealers, keyloggers, or sniffers, retrieve data and send it to another location. These kinds of malware can also be termed and labeled as riskware’s when used by an authorized person in appropriate behavior and status. Furthermore, if the program is abused by an intruder, the safety of the user or system may be affected [25]. Keylogging software, for example, is often used to track users. These kinds of attacks are the most popular in banking systems online.

2.2.12 Scareware

Scareware works like ransomware which usually tries to scare the contaminated person to purchase something from the attacker [26]. It generally comes as an email attachment with blackmailing text that the user is at risk and should remove another risk. Many victims will buy the software’s to delete the viruses and risk in response to these emails or text message.

2.2.13 Spamware

Spamming is also known as phishing and these are malicious software’s that are part of the botnet dominated by a command and control server that operates as a decentralized spamming network. Spamware normally spreads other malicious software or gives computer resources by harming another system with malevolent activity. The ISPs sometimes take defensive measures against this malware by deactivating the Internet service of the offender or by flagging it as spam [27].

2.2.14 Spyware

Spywares collect the information on browsing activities of the user or favorite software’s. The collected data about the user are generally sold out to other companies or businesses [28].

2.2.15 Trackware

Trackware provides the tools for the third-party companies to identify a user or a management system, typically with a digital signature. Trackware is quite commonly used to monitor cookies [27].

2.2.16 Adware

Adware transmits malware contents via web-browser, desktop computer or smartphone app. An alternative title for this type of malware advertising malicious software under the flagship of established companies [29].

2.2.17 Potentially Unwanted Programs

PUP is also known as Potentially Unwanted Software (PUS), Potentially Unwanted Web Application (PUWA, Popups) and Potentially Unwanted Application (PUA) [30]. It is normally software that operates and seems to have a suspicious behavior with unnecessary and unwanted services and functions but does not meet the standards for malicious software, and these qualities make it difficult to analyze and classify PUPs, which are deemed useful for some people, but malevolent for other people. A PUP can affect profitability, security, and privacy sometimes, but it can also place undesired pressure on the system's resources.

#### Unintended impact on productivity:

 Modification to the user settings.

 Ineffectiveness.

 The programme, which points to unneeded interruptions, missed opportunities or reduced productivity, and acts in unpredicted, undesirable and unapproved actions.

 Operators of the impacted systems must often carry out time-consuming maintenance and disinfecting procedures.

#### Unwanted stress on the device's resources:

 Extra stress by using resources unusually e.g., processor, memory, HDD and printers etc

 Using Higher Bandwidth.

#### Compromises security:

 Advertising and defenseless to unanticipated, unpopular and unfounded applications.

#### Compromises privacy:

 Unnecessary disclosure of personal information, as well as fragile software’s, to unidentified or unauthorized parties.

2.3 Malware Analysis and Detection Techniques

In the last decade, data mining strategies have been focused on malware detection systems. As advancement grows, the fight between security modulators and malicious software intellectuals continues. The suggested methods are inadequate while the genetic and complicated evolution of malicious software is rapidly changing and is, therefore, more difficult to identify. The following sections provide a concerted and comprehensive study of the threat detection methods of malicious code using data mining strategies. It also categorizes intrusion detection methodologies into two major categories, such methodologies are based on signatures and anomaly detection. Figure 2-1 shows a correlation between different sorts of detection methods. One of three methods can be used in each detection technique: static, dynamic or hybrid. The strategy based on an anomaly or signature is defined by how the method collects metadata about malware. In order to identify the maliciousness of a program, structural characteristics or syntax of the program is used by static analysis (static) / process (dynamic) under inspection. For instance, a static method detection of signature-based can only use syntaxial information (e.g., byte sequence) to decide malevolence, while a dynamic method used the PUI's runtime information (e.g., architectures seen with the runtime stack). A static method usually tries to identify malicious code before the execution or running of the infected program. In contrast, a dynamic method tries to identify suspicious behavior during or after execution of the program. Hybrid methodologies merge the characteristics of two strategies, i.e., Static and dynamic metadata is used to identify malicious code.

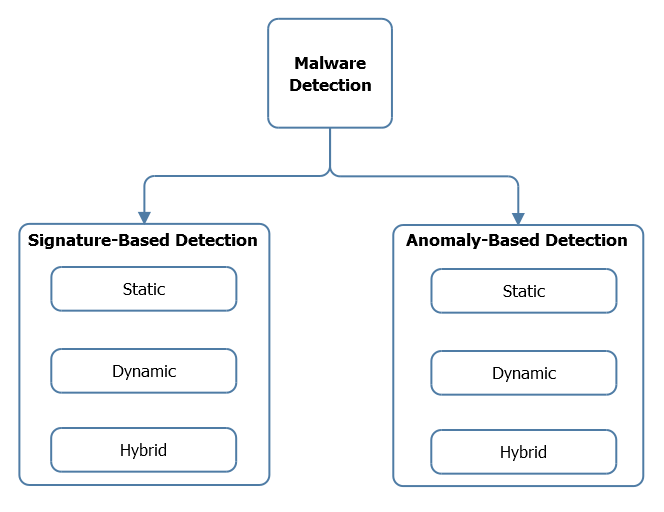


Figure 2-1 Malware Detection Techniques

2.3.1 Anomaly Based Detection

The detection of anomalies normally takes place in two phases: learning phase which is also known as the training phase and testing phase which is known as the detection stage. The detector tries to learn normal behavior during the training phase. The detector may learn the host's or PUI behavior or a mixture of both during the training phase. The ability to identify zero-day attacks is a major benefit of anomaly-based detection [31]. The two basic disadvantages of this method are its high false alarm rate and the robustness of evaluating which characteristics should be learned during the learning phase.

Figure 2-2 shows the detection systems based on anomaly detection alone is inadequate for the detection of malware. VB is the set of all valid systems behavior extracted from a set of non-contradictory requirements, where VA denotes all invalid behaviors set. VA is the approximation to VB, that means that requirements of the VA are approximated to VB.

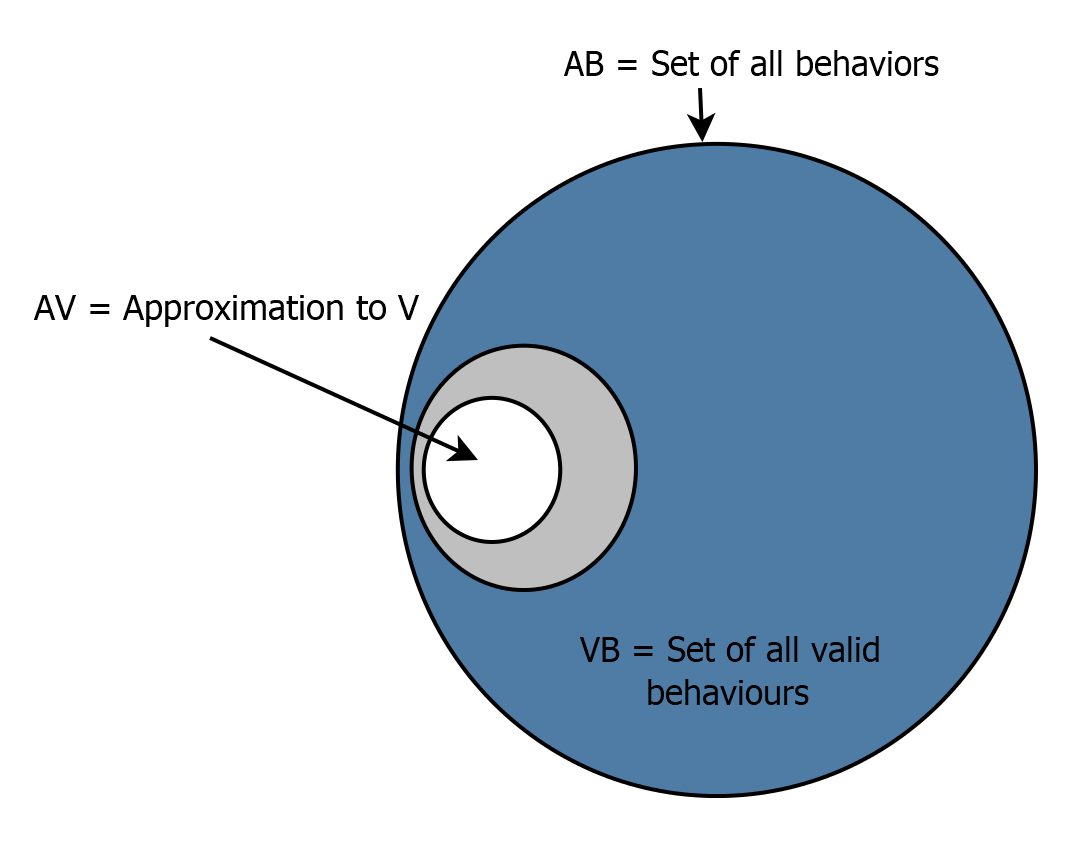


Figure 2-2 Characterizing the behaviors in Anomaly Based Detection

Figure 2-2 shows the estimation (approximation) of all legitimate behaviors performed by anomaly-based detection methods as set AV. For instance, if an anomaly during the learning phase is not seen, an exception shown during the testing stage could lead to an incorrect alarm i.e., it can lead us to the high false positive rate usually associated with threat detection methods based on anomalies. The ability to show newly discovered behavior during the detection is not zero. It is an open computer science major issue to develop better approaches to the appropriate response of a computer system.

Table 2-1 Advantages and Limitations of Anomaly-Based Malware Detection

|  |  |  |
| --- | --- | --- |
| **Advantages** | | **Limitations** |
| Detection of unconceived malicious software attacks |  | The complexity of storage for behavioral patterns |
| Dependency on the data flow detector |  | Complexity of time |
| Polymorphic malware detection | |  |

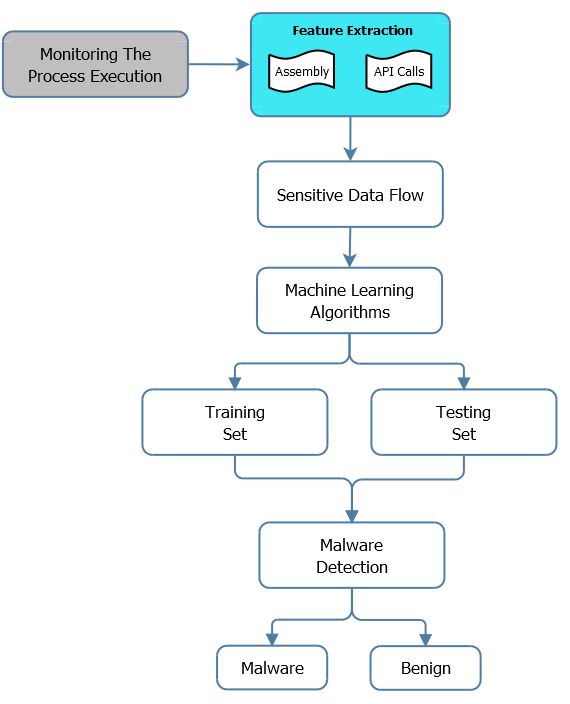


Figure 2-3 Anomaly Based Detection Approach

2.3.1.1 Static Detection (Anomaly Based)

Characteristics of the file framework of the inspected program are used to detect malware in static anomaly-based detection. A major benefit of static anomaly detection is that these can simulate malicious code without allowing the program carrying the malicious code to run on the host system.

Li et al. [32] described the evaluation of file-print (n-gram) as a way to identify malicious code. During the learning phase, a prototype or set of designs has extracted that attempts to classify the different file types based on their systemic (byte) dynamics in a system. Such models come from learning the types of files that the program chooses to manage. The author assumes that benevolent files for their various varieties have predictable frequent byte compositions. For example, benign pdf documents have a distinctive allocation of bytes that differ from doc or exe files. Any document under inspection, which is considered to differ "too much" from the paradigm or set of designs, is labeled as suspicious. Some other framework marks these suspicious documents for further evaluation or decides whether it is malevolent. The authors assume that more work must be done to ascertain the feasibility and efficacy of an assessment of 2 grams or 3 grams. The detection findings fluctuated when tested on multiple types of files.

2.3.1.2 Dynamic Detection (Anomaly Based)

In the detection of dynamic anomalies, data collected from the execution of the program is used to simulate malware. The detection stage scans the inspected program throughout its implementation and checks for contradictions with what has been discovered during the learning phase.

Wang and Stolfo [33] presented a mechanism “PAYL”, that computes the anticipated payload for each service (port) on a system. They developed a byte frequency distribution that enables each host's services to evolve a centroid model. During the training stage, the centroid model is calculated. The detector matches up upcoming payloads to the centroid model and measures the distance between the two Mahalanobis.

Lee and Stolfo [34] suggested the use of machine learning methods for intrusion detection, namely association regulations, and frequent episodes. The rules of association and regular episodes can allude as a set of rules collectively. Rule sets are formed for different safety-critical elements of the target host (e.g., the intensity with which a variation of system requests is invoked over a short time when the driest runs). These rules serve to understand what normal behavior for the programs of the host is.

Boldt and Carlson [35] presented the concept of software that invades privacy. The main categories of privacy-invasive software are adware and spyware. PIS is often acquired as part of the software for file sharing. To help to detect the PIS, Boldt, and Carlson used the Forensic Tool Kit (FTK). The basic approach is to create a PIS-free system, a "clean" system. Ad-Aware was the most common PIS cleanup tool, so the authors used static analytical techniques to evaluate Ad-Aware. Boldt and Carlson found that Ad-Aware generated false negatives and false positives by using their technique.

2.3.1.3 Hybrid Detection (Anomaly Based)

Wang et al. [36] proposed a technique to simulate a type of malicious code called "ghostware." The Ghostware is malicious software that tries to hide from the verifying utilities of the operating system. It is usually done by encrypting and altering the outcomes of these queries so that ghostware footprints cannot be discovered/identified via API queries. For instance, if a user executes an instruction to list the documents in the recent directory, "dir," the ghostware will eliminate almost all of its resources from the actual results transferred by the command "dir."

For the real-time detection tools, Farhat and Robert [37] proposed a technique but generally impossible to implement. The purpose of the suggested method was to track changes to the shielded data. This strategy is warned because it could not pinpoint the elimination of items from the catalog of shielded data. It may be clear why it is impossible to implement this model. The authors' suggested estimate still leads to unpleasant computing extra costs.

2.3.2 Signature Based Detection

Signature-based tracking tries to design malware's malevolent actions and detects malware by using this model. The compilation of these systems represents the expertise of signature detection. This malevolent behavior design is often called the signature.

Theoretically, any malicious software showing the malevolent actions stipulated by the signature should be identified by a signature. Signatures require a repository, like any information that resides in large amounts requiring storage. This storage database reflects all the knowledge of the vulnerability scanning mechanism based on the signature. The storage database is searched when the technique is trying to evaluate the PUI that whether the file includes a known signature. The main drawback of signature-based mechanisms is illustrated in **Figure 2-4**. Since the set of possible malevolent tendencies is infinite size, there are no established strategies for accurate representation through signatures (set of malevolent behaviors). In addition, a signature database is a fragile approach (Set of malevolent behaviors). Yet another disadvantage of signature-based mechanisms is that the development of signatures generally requires human intervention/expertise. This causes the human error to be introduced as well as tends to take much more time than if the growth of signatures was fully automated. Since some malicious code can spread very quickly, the capacity to develop a precise signature rapidly is becoming essential. There are automated signature architects [38], but there is more room for improvement in this area.

Some examples of signature-based detection methods try to take advantage of the fact that much malicious software is derived from preceding malicious software. Malicious code signatures should be constructed in a way that encapsulates the malevolent nature of the malware. The main reason why this signature construction framework is used is, to reduce the susceptibility to obstructions of malicious code scanners. Another important reason is that the number of malware signatures collected in the database is minimized. The storage might not be a problem at present, but might become a serious issue over time, as this will have a significant impact on the malicious code detector's time complexity.

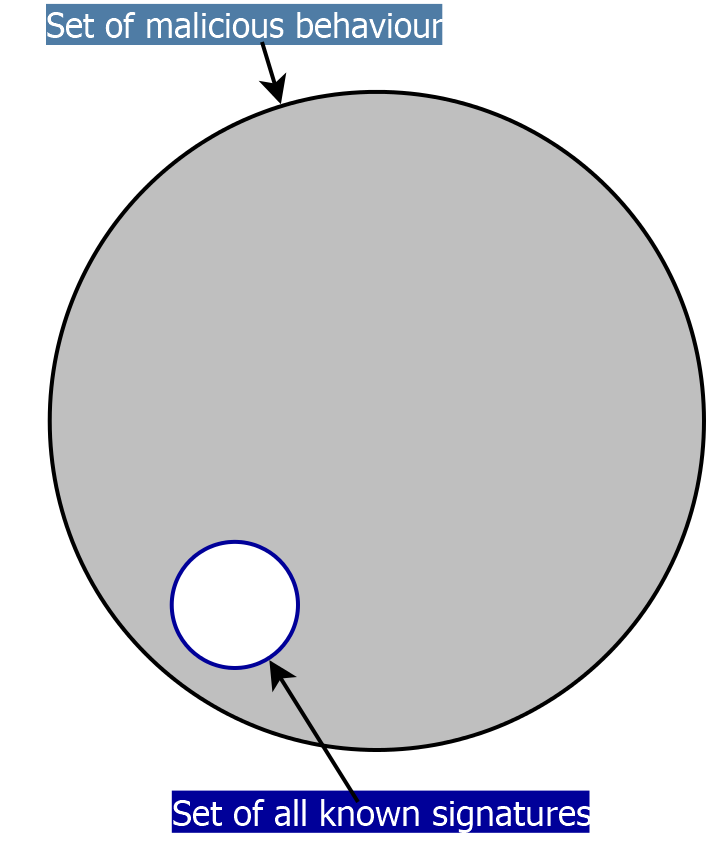


Figure 2-4 Signature Based Detection

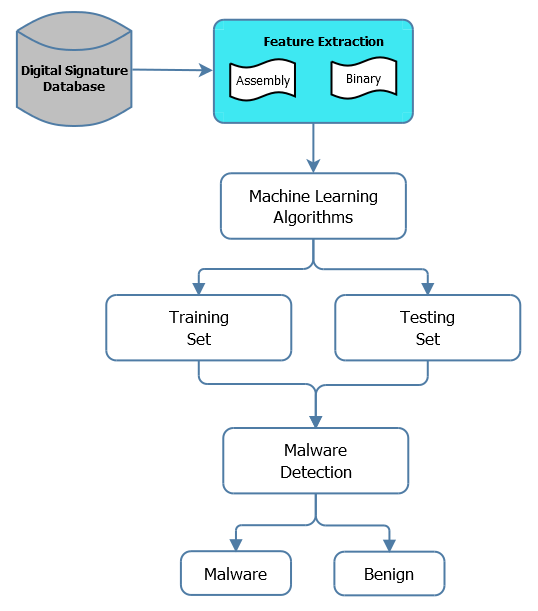


Figure 2-5 Signature Based Malware Detection Approach

2.3.2.1 Static Detection (Signature Based)

Static signature-based malware detection is defined by an evaluation of the code sequences that expose the program's maliciousness. The aim is to assess the software that reflects the program's behavior. The static assessment of this software gives an estimation of the executable's run-time behavior under evaluation. Typically, signatures are defined by code sequences.

Sung et al. [39] proposed a technique called “SAVE” (Static Analysis Vicious Executables). The sequence of Windows API calls gives the signature form for a given virus. A 32-bit number represents each API call. The most important 16 bits correlate to the API call module, while the least relevant 16 bits correlate to the situation of the API functions in the API functionality vector. The distance between known signatures and the sequence of API calls discovered in the monitored program is measured by Euclidean. The median of three resemblance functionalities shows the similarity between the API sequence of the PUI and the repository signatures. The PUI is marked as malicious if the difference is 10 percent or less.

A Framework represents malware signatures in Christodorescu et al.’s work [40]. Each template is three times as many commands, parameters and ceremonial symmetries as possible. Templates try to categorize the signature of a malicious code instance and yet retain the nature of the behavior of the malware. Christodorescu et al. also discovered that their algorithm did not contain false positives on 2,000 benign Windows programs.

Kumar and Spafford [41] proposed a general scanner that identified malware on the basis of standard matching expression. The sequence matching algorithm measures up all documented malware that matches this nibble value to see if the input stream sequence matches a documented virus or not. Since the proposed scanner was primarily designed for SunOS, the application of the proposed system was therefore not easily comparable to other malware detectors of various operating systems.

2.3.2.2 Dynamic Detection (Signature Based)

Dynamic signature-based detection is exemplified through the use of information collected only to determine its malevolence during the PUI execution. Dynamic detection based on signatures looks for behavioral patterns that would expose a program's real malicious intent.

Attacks were modeled in Ilgun et al.'s [42] work as a state transition diagram. Presuming that there's some oversight mechanism, this framework moves these data to a preprocessor that formats the data in a way that can be examined with a state transition diagram. The data were then compared to known installations in sort of the state transition diagrams.

Ellis et al. [43] proposed a dynamic signature-based detection technique for worm detection which was based on known malicious behaviors. The authors presented four behavioral signatures different from each other’s. Such signatures can be recognized by tracking the flow of data from a single node. This detection approach isn't quite as effective when used in a peer-to-peer system.

2.3.2.3 Hybrid Detection (Signature Based)

The hybrid detection strategy utilizes characteristics of both the static and dynamic to ascertain the PUI's maliciousness. Mori et al. [44] proposed a mobile self-encoding and tuples malware detection tool. According to Mori et al., [44] viruses do not show their patterns by encrypting themselves, the authors, therefore, developed a method and mechanism to tackle this problem.

Castaneda et al. [45] proposed a technique to capture malicious processes using the honeypot IDS. To capture the malicious process, the honeypot IDS uses a signature-based method. The technique tries to find malicious payload once the malicious process is captured. The technique does this by making copies of the executable files and continuing with an anti-worm payload and overwriting it. Four different anti-worm replication systems were assessed by Castaneda et al. via computation. The worm being used in the computation was version 2 of Code-Red I (CRIv2). Each computation was started with 360,000 hosts and these hosts were vulnerable.

Table 2-2 Advantages and Limitations of Signature-Based Malware Detection

|  |  |  |
| --- | --- | --- |
| **Advantages** | | **Limitations** |
| Quick identification |  | Replication of relevant data in the huge database |
| Easy to operate |  | Signature based detection fails to detect Polymorphic malicious code |
| Widely reachable | |  |
| Find detailed information about malicious code | |  |

2.4 The Role of Artificial Intelligence in Malware Detection

In recent times, the evaluation of vulnerability scanning methods has increased the use of big data and machine learning to identify exploitable software’s via data mining strategies [46][47]. Machine learning techniques can take concealed instances from a certain set of preparations containing both malicious code and harmless examples. Such basic examples can distinguish between malicious code and good code [48][49]. Malware is exceptionally insightful bullying for embedded systems and the world wide web [50]. As advancement in the technology grows, the fight between security modulators and malicious software creators continues. Malicious code is a program that enables your structure to achieve something an attacker needs [50]. The most commonly used replication of malware acquires a simple example of how to coordinate the identification of manipulative code. Malware programmers typically do not create new code without planning and preparation but revamp the obfuscated code with new models or mixing strategies [51]. With a huge number of malicious software situations every day, it has become increasingly vital to prepare innumerable fossils that exhibit analogous behavior. [52].

2.4.1 Malicious Code Detection using Machine and Deep Learning

For the classification and detection of unknown codes in their families, various machine learning methodologies are proposed, such as Naive Bayes, Support Vector Machine, Clustering etc. M. Damshenas et al. [53] proposed a malware detection technique for mobile devices. A server analyzer and a lightweight client agent are part of this technique. For each application, the server analyzer generates a signature. The envisaged method can generate formalized mobile malicious code signatures predicated on their behavior, and this is their main research contribution. It relates the signature generated and previously blacklisted malware signatures. The static analysis strategy was proposed by N. Milosevic et al., [54] in Android apps, and malevolent behavior was detected and analyzed within the software. They used machine learning strategy to identify malware families; this strategy is a known as signature-based detection. To evaluate their model, they used Support Vector Machine (SVM) and achieved the accuracy of 95.6%. Lee et al., [55] proposed clustering machine learning to address the issue of the worm's signature. They used the nearest neighbor technique.

A malware detection methodology was proposed by Siddiqui at al. [56]. By using data mining on file features, they used malware detection approaches. The file characteristics and the threat detection phases were classified into the evaluation using the parameter length instruction sequence. To classify the malware, they used the decision tree and random forest algorithm. Egele et al. [57] analyzed malicious behavior within the code. They developed binary obfuscation techniques that transform the binaries of malware into self-compressed files. They also developed a method which recognizes binary files that constrained reverse engineering uniquely. To categorize the malware, Nataraj et al. [58] used an image analysis technique. They transformed binary malicious code into gray-scale images. Nataraj et al.'s proposed solution represent executable binary files in bitmap images on a gray-scale. Kong et al. [59] built a model based on structural information to classify malware. Their model extracts the features of each malware sample for structural information using the function call graph. They have used a method of distance measurement which clusters the samples of malicious code belonged to their respective family and used a classifier which identifies malicious software in their families. Tian et al. [60] used a library from Weka [61] to categorize the malicious code of the Trojans using frequency length. The percentage of bytes is calculated in the executable files. The findings in their research show that the malicious code family is recognized by frequency function and can be combined with other malware classification system features.

For unknown malware detection, Santos et al., [62] used a semi-supervised learning technique. They used algorithm learning with Local and Global Consistency (LLGC) to reduce the number of instances required while counterfeiting high accuracy and also to determine the optimum number of marked instances that improve the accuracy of the model. Santos et al., [63] proposed another collective learning technique. It is also a semi-supervised learning form that introduces the optimization method for the partly labeled classification of data. Collective classification algorithms are used with a set of unlabeled and labeled instances to build different machine learning classifications. Zolkipli et al. [64] used software packages to examine malware behavior, such as HoneyClients, Amun. They had to use Cw-sandbox [65] and Anubis to evaluate the malware behavior of each sample. The malicious code was split into two families, i.e., Trojans and Worms. The biggest drawback of this study was, it cannot be customized.

Deep learnings might have been the remedy for the issue above. But again, before the model is learned, a pre-data processing step such as feature engineering is still required. In addition, the model's training dataset could not normally necessarily reflect real-world malware. For example, For instance, Hardy et al. [66] designed an anti-malware in which the Windows API survey creates a consequent ID, that is handled as an input to the deep learning model (e.g., Auto-Encoders stack), and after that, the model parameters were fine-tuned. Joshua and J. Berlin [67] designed a technique which extracts the features first, like semantic byte features, histogram features of string 2d and PE metadata, PE import features and then introduced to the deep neural network (DNN) for training. Yuan and Xue [68] proposed a static analysis approach to extract features like authorization required and sensitive API. A dynamic analysis was also used by them to extract features from 500 samples for approximately 200 features as input for the Deep Belief Network.

There is a correlation between the Android malicious apps execution logic and the calling of functions order. In relation to the above solutions, which apply DNN to malicious code assessment based on "exploitable attack" and "privileged escalation," another tier of anti-malware is predicated on byte-code or op-code analysis of n-grams. For instance, Abou-Assaleh et al. [69] and Reddy [70] calculated the n-grams using the binary byte-code approach and then evaluated the malicious code detection using the k nearest neighbor (KNN). Moskovitch et al. [71] proposed a method of performing reverse-engineering of the dataset first and then evaluate op-code for malware detection. Moreover, one more category was proposed by Nataraj [58] that depends on the transformation of malware into images, e.g., Nataraj first transformed binary byte-code into the gray-scale images and applied pattern recognition to the gray-scale images.

Although, all the above methodologies achieved a good level of efficiency in detection. But again, the overall number of malicious software has increased dramatically, as stated in the introduction. The size and weight of the sample used to train the model also has a major effect on the precision of the detection in the training phase even more and more error handling strategies are being discovered. In particular, we note that, despite the precision of detection of the n-gram strategy, the n-gram method required significant computing resources and time to manage the dynamic growth of the necessary design parameters [72]. However, CNN can tolerate explosive data growth if we have limited computing resources and time because the increasing number of parameters does not imply the growth of computing time and resources needed. Recently, Tong and Yan [73] proposes a deep-learning based malicious code detection method, in which the op-code sequences are embedded as one-hot vectors for the CNN inputs. But this method needs to dismantle the Android apps using reverse engineering methods to derive small source code from classes, which means that malware cannot be handled with encryption and obfuscation.

2.4.2 Intrusion Detection using Machine and Deep Learning

Intrusion detection technology is an important part of computer network security. The concept of intrusion detection was first proposed by James Nderson in 1980 [74]. The goal of intrusion detection is to correctly identify abnormal network behavior. The current popular intrusion detection method is to reduce the error rate by using different machine learning techniques. Aljawarneh et al. [75], constructed a set of human intrusion detection models by combining various machine learning algorithms, such as support vector machines, Bayesian classification, and decision trees. Pan et al. [76], proposed a hybrid machine learning technique combining Zhi-Mean and SVM to detect attacks. Shin et al. [77], used the bogey-means algorithm to calculate the similarity between data, and by adjusting the parameters. Azab et al. [78] proposed machine learning techniques for Zeus V1.x, Zeus V2.x and benign HTTP traffic detection in networks. Bamakan et al. [79], and Mohamad Tahir et al. [80] applied a neural network to detect intrusion in the network. Zhao [81], proposed the LSSVM model for network intrusion detection. Jha et al. [82], used hidden Markov models to study network intrusion detection. Horng et al. [83], applied the SVM method to IDS. Traditional machine learning methods are very effective in intrusion detection, but they also have limitations, because the traditional machine learning technology needs to artificially construct sample features. Its performance is dependent on its quality. In order to solve this problem, researchers have introduced deep learning techniques. Gao et al. [84], applied deep trust network in intrusion detection and achieved better results than other traditional machine learning methods. Raman [85], applied probabilistic neural networks to detection techniques. Peddabachigari [86], proposed a hybrid intrusion detection model based on deep learning and verified that the model is more efficient than traditional machine learning methods.

2.4.3 Botnet Detection using Machine and Deep Learning

Machine Learning algorithms have been widely used to classify internet traffic. Irrespective of the class imbalance problem, ML algorithm classifiers such as Decision Trees and Neural Networks, may produce a high accuracy but low byte accuracy. Zhang et al., [87], proposed two algorithms based on feature selection and extended the *wsu\_auc* selection to apply the best features practically. They achieved more than 94% accuracy with an average byte accuracy of over 80%. Chen et al. [88], has proposed a high- speed network detection method for botnets. In this PF RING, the problem was solved at a high packet drop rate and for extracting required fields from traffic information. The author used the random forest technique on the CTU dataset. They have acquired high precision, but the unconvincing aspect of this publication is, they used only offline data and no other data was collected online. Azab et al. [78] proposed machine learning techniques for *Zeus V1.x*, *Zeus V2.x* and *benign HTTP* traffic detection in networks.

Wang et al. [89] proposed a solution to detect malicious Android apps. The paper used mobile traffic in which every HTTP traffic was treated as a document and then the document was used for word segmentation based on N-gram generation to generate candidate features which effectively characterizes the particular flow of HTTP. The paper used the SVM machine learning algorithm for validation. The paper shows a high accuracy of 99.15% on static data while its detection rate was 54.81% on real environment testing.

Albanese et al. [90] proposed a graph-based botnet detection technique called MTD (Moving Target Defense). The paper claims that the limitation of the static solution has been addressed as MTD periodically alter the placement of detectors, but the said paper did not mention the traffic speed which is affected by the periodically altering the placement of detector.

Haddadi and Nur Z. [91] have investigated five different botnet detection techniques. Two of them were rule-based systems and other three of them were machine learning based techniques. The paper analyzed these approaches with different scenarios with 24 publicly available datasets.

Alazab [92] examines the evolution of malware by the nature of its activity and variants. The paper investigates malware implication on the computer industry and provided a framework using feature extraction from malicious code which reflects its behavior.

Alazab [93] proposed a four-step methodology to develop a fully automated system for suspicious behavior detection. The four-step method classified the behavior of API function calls based on the malicious intent hidden with the packed program.

Zeng and Shen [94], proposed a two-step distributed approach for storm botnet detection which includes a set of heuristics and first-step port numbers and an SVM classifier. The accuracy of their method was more than 95% with 8 - 12% of FP rate. This scheme works well with 0% FP rate and 8% false negative rate (FN) to detect storm botnets Host. According to Zhang et al., [95], the P2P client is first identified by extracting the statistical fingerprint of P2P communication, and the legal P2P network and the P2P botnet are further distinguished.

Detection based on host behavior [96][97], detects zombie programs by monitoring changes in the host process, file, network connection, and registry content in a controllable environment. The method cannot discover new and variant botnet programs. For example, an attacker could use such new detection and hiding techniques such as rootkits, anti-debugging to avoid such detection strategies.

The detection based on traffic behavior [98], is mainly used in the botnet C & C control phase because there is a difference between the flow of the C & C control phase and the normal network traffic in the flow characteristics and communication rules, including the average packet size, Periodic connections, and so on [99]. Therefore, you can combine machine learning, a neural network for real-time botnet monitoring [100]. The botnet detection method based on traffic behavior mainly analyzes the following two characteristics:

1) Connection failure rate, and, 2) Flow characteristics.

The streaming feature also includes the number of upstream and downstream packets, the size of the uplink and downlink transmission bytes, the average length of the uplink and downlink data packets, the maximum length, the average variance, the duration of the data stream, and the packets that have been loaded in one stream. The streaming feature selection method has a high detection rate because it does not depend on the botnet class to extract the common feature vector of the stream. Therefore, the detection strategy is widely concerned by experts and scholars in the field of traffic analysis. In high-speed, complex, and changing network environments, the main factors that determine the efficiency and accuracy of detection are the characteristics of the extraction and the classification strategy used. According to Zhang [95], the communication behavior between zombie hosts that join the same botnet is similar. Therefore, P2P botnet traffic identification can adopt the following scheme:

First, the traffic of the network is processed and analyzed, and the characteristics are extracted. Then, the clustering algorithm was proposed to cluster the data extracted from the previous stage. Finally, the P2P botnet i.e., the traffic follows which cluster? The scheme is to set the threshold to improve the detection accuracy, without the use of existing botnet data stream for training. However, if there is only one zombie host in the current network, or if no traffic from different zombie hosts is found in the captured packets, this method will not have much effect.

The three DT algorithms REPTree, Carriage, and C4.5 were analyzed in the study [101][102], with C4.5 with the lowest performance because its algorithm is easily under pruning, and the overfitting algorithm is more severe than REPTree. Encrypted P2P or unknown traffic can be classified by the statistics-based method, but it is not highly accurate, and cannot identify untrained P2P traffic correctly.

2.5 Summary

We presented a series of discussion on the techniques, examples, problems in the area of malware detection in terms of malicious code, intrusion and botnet detection in this chapter. We have also pinpointed shortcomings in the detection methods based on signatures and anomalies (based on specifications). The threat detection strategies are outlined in Figure 2-1. The chapter also portrays a brief explanation of malware, its different types, and the detection techniques. Machine learning has a very important role in the current era of cybersecurity, so the chapter briefly describes the role of machine learning/deep learning detecting different types of malware.

From the review addressed so far, we can understand that the evaluation of malware is just like a game of cat and mouse. The literature review indicates that there are two significant malware analysis techniques available; this also demonstrates that many scientists cannot detect malware because malware has a wide range of analytical avoidance methods. With the development of new malware analysis techniques, malware authors react to thwart assessment with new techniques.

Chapter 3 Malware Detection Through Reverse Engineering and Feature Extraction

3.1 Introduction

Malicious software which is additionally alluded as malware is the major threat for computer users. The principal focal point of malware is, to accumulate the individual’s data without the consideration of clients and to exasperate the PC activities which makes issues for clients. There are numerous sorts of malware, for example, Trojan-horse, Virus, Rootkit, Backdoor, Worms, Spyware, Adware and so on. Yearly reports from antivirus organizations demonstrate that a large number of malicious software are made day by day. Attackers are making new software, and the new software turn out to be more advanced that they could never be recognized by the traditional discovery procedures, for example, signature-based recognition and behavior-based recognition.

Signature-based identification looks for determined bytes groupings into an object so it can recognize extraordinarily a specific kind of malicious software. The main disadvantage is that it can't identify zero-day malicious software since the new software signatures aren't stored in the database. Behavior-based recognition was created to fundamentally conquer the impediment of the signature-based method, in the way that it filters the framework's behavior to recognize the abnormal exercises rather than looking for the signature of the malware. The constraint of the behavior-based procedure is that it influences the execution time of the system and more storage is required. This method focuses on the conduct of the program when it executes. The program is marked as benign if it executes normally, else, it is set apart as a malware. By breaking down this meaning of the behavior-based method, we can specifically presume that the disadvantage of this procedure is the generation of numerous false positives and false negatives, considering the way that a legitimate program can be slammed and be set apart as malware or malware can execute as a normal program.

Current accomplishments in deep learning innovative work draw in individual's consideration. Google released TensorFlow in 2015 [103], a structure of acknowledging deep learning. All the more particularly, deep learning is a counterfeit neural network system, in which numerous layers of neurons are connected to each other with various weights and enactment capacities to take in the shrouded connection among inputs and outputs. Instinctively, input information is encouraged to the primary layer that produces diverse mixes of the information [104]. After the detection function, these configurations are fed to the new layer, and so on. Different configurations of outputs from the preceding layer can be considered as a different representation of features under the above procedures. In reverse propagation, the weights between the layers are balanced, contingent upon the separation between obviously labeled output. Deep learning approach could be viewed as a neural system with an expansive layer. After the above learning process by means of numerous layers, we can determine a superior comprehension and portrayal of recognizable features, improving the recognition precision [105].

Additionally, see that the adequacy of deep learning increments by the system measure. The most understood profound systems are convolutional neural systems (CNN) in addition to the Neural Network system. The portrayal of CNN incorporates AlexNet, VGG, GoogleNet, and ResNet [106]. All the more particularly, CNN is made out of concealed layers, completely associated layers, convolution layers, and pooling layers. The shrouded layers are utilized to expand the many-sided quality of the model. In the event that a similar number of neural is related to the information picture, the number of parameters can be altogether diminished, receiving to the capacity structure much legitimately.

3.2 Proposed Model

This segment is separated into two parts. The initial part is about information planning or processing the data and the second part is tied in with training/testing the model. Following steps describe the model briefly, followed by Figure 3-1:

1. Download malware and benign software’s from open source databases and then convert into assembly code

2. Convert assembly code to 2-tuple code

3. Opcode sequence generate the matrix

4. Generate images from binary code matrix

5. For the training and testing, divide the dataset into two parts i.e. training and testing phases

6. Use tensor flow models i.e. GoogleNet and ResNet which hold the features and labels

7. Implement the relevant model

8. Optimize the model to reduce RMSE

9. Prediction on the test dataset

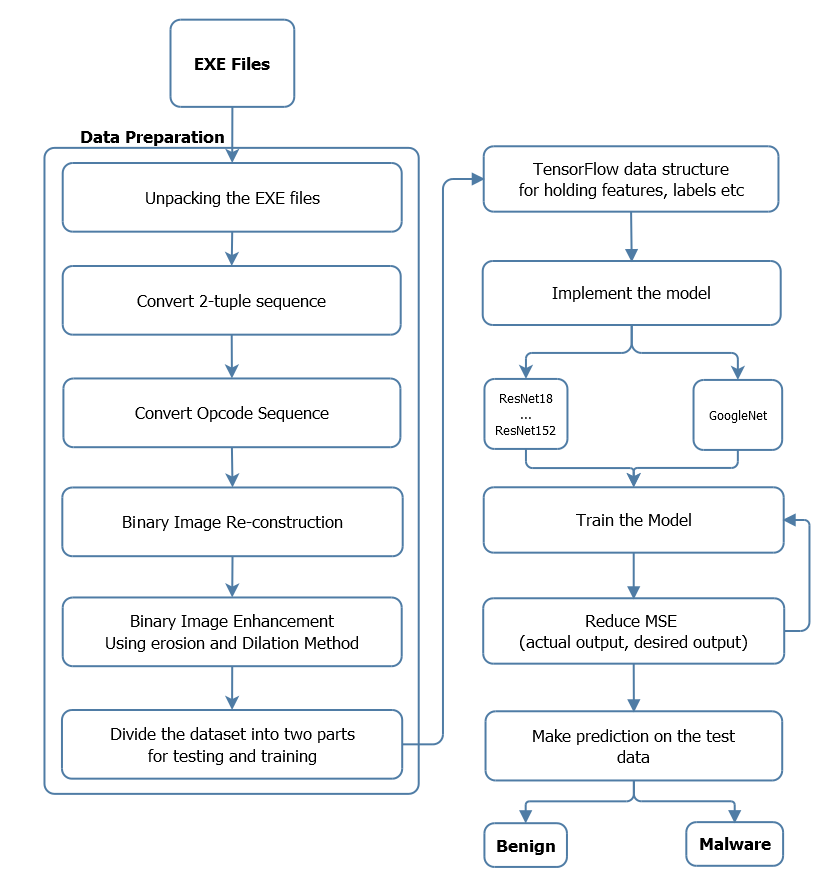


Figure 3-1 Architecture of Proposed Model i.e. from Data preparation to the prediction

Further, in our design, we partitioned our model in two stages, that is training and detection. For the preparation and the discovery of malicious code, we utilized CNN, as appeared in Figure 3-2. The outputs of the information arrangement were "images". Pictures have paired marks that is malware or benign. The recognition stage appears in Figure 3-3. In other words, .exe (executable) files were converted into images and the classifier is used to detect the malicious code.

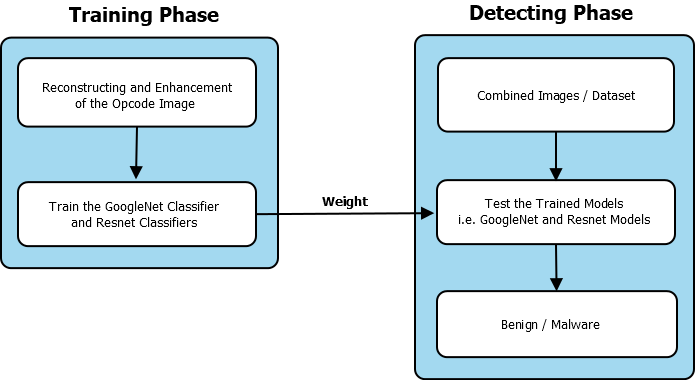


Figure 3-2 Implementation of Deep Learning

In the followings, the author explains the procedure of malware detection methodology. First of all, the author collected the software and they are classified as benign and malware. The author decompiles the software to the assembly files, finally, the author constructed gray-scale images. The author further applied CNN based models i.e. GoogleNet model and ResNet model to detect malware and compute the accuracy. The following sections describe in both models.

3.3 Data Preparation

3.3.1 Dataset

Datasets were downloaded from various sources. Malicious software dataset (malware) was downloaded from Microsoft. Similarly, the author downloaded 3000 benign software’s from open source websites. In the accompanying discussion, the datasets are described in detail.

3.3.1.1 Dataset from Microsoft Malware Classification Challenge

The Microsoft-obtained dataset comprises 9 classes in total. The dataset of 500 GB includes 21741 samples of malware. In preparation, 10868 samples are used, and the rest of the samples are used for testing.

**a) Bytes Files:** Byte file contains 10,868 training data and 10873 testing data in the Microsoft dataset. Each byte file contains a binary content hexadecimal representation.

**b) Asm Files:** Asm file contains 10,868 training samples and 10873 testing samples in the Microsoft dataset. Each Asm file derived from the IDA disassembler tool comprises a manifestation of metadata. This data includes segments of assembly commands, strings, function calls, etc.

**c) Training Labels:** MD5 Hash is the name of the file in the actual program and is used as a label for training. The training label files contain every MD5 hash and malware class to which it maps. The test data input files have not been provided with training labels.

**d) Sample Submission:** The random sample submission file shows a reasonable format for the submission of 10,873 sample files.

**e) Data Sample:** A teaser of the test and training data is included in the data sample file.

Table 3-1 Data set for Microsoft Malware Classification

|  |  |  |
| --- | --- | --- |
| **No** | **Family Name** | **Number of samples** |
| 1 | Ramnit | 1541 |
| 2 | Lollipop | 2478 |
| 3 | Kelihos\_ver3 | 2942 |
| 4 | Vundo | 475 |
| 5 | Simda | 42 |
| 6 | Tracur | 751 |
| 7 | Kelihos\_ver1 | 398 |
| 8 | Obfuscator.ACY | 1128 |
| 9 | Gatak | 1013 |

3.3.1.2 Benign files

The author gathered 3000 of clean coded files from various sources.

3.3.2 Environmental Setup

This study has used Ubuntu 64bit operating system and RAM of 8 GB. To play out the experiment, the author used Python programming language with Python libraries, for example, Tensor Flow, Docker Server, Anaconda. The Tensor Flow Library utilizes the architecture CNN.

3.3.3 Data Preparation Technique

The accompanying strategy to process the information is proposed in this chapter.

3.3.3.1 Opcode to Images

Two tools were used in this study to extract the hidden message from binary files.

i. PEID: This tool is utilized for static code

ii. Poly Unpack: This tool is utilized for dynamic code

1. Decompile Opcode: The Opcode succession has been decompiled from assembly code and after that change over 2-tuple opcode grouping.

2. Opcode Sequence: The binary image matrix multiplication is rebuilt with their statistical probabilities and information gains by such Opcode sequences. Figure 3-3 portrays the matrix, each opcode pattern of length 2 can be paired to one of the features in the matrix according to , shown in Eq. (3-3). The component value  of the matrix of the image  is measured on the basis of probabilities  and the information gain  of in the binary .

 (3-1)

The probabilities and information gain are measured on the basis of frequencies of the segments of opcodes of length 2 , can be seen in two equations i.e., Eq 3-2 and 3-3, while is the statistical probability of  in the training phase of malicious code, is the statistical probability of in entire training binaries, and  is the statistical probability of the training binaries.

 (3-2)

 (3-3)

3. Binary Image Re-construction and Enhancement: binary opcode frequency constructs images. To improve the sequences of Opcode, histogram normalizing, dilation and erosion strategies are used.

To improve the complexity between malware variation pictures and benign pictures, the histogram standardization, enlargement, and disintegration techniques are utilized to upgrade the binary pictures. Through picture upgrade, the difference of these uncommon opcode pictures would be improved.

Let be the pixel-value of the improved images; the histogram normalization method is according to equation 3-4.

 (3-4)

In this strategy of data preparation, the author can without much of a stretch recognize malware and favorable records by the visual investigation as appeared in Figure 3-3.

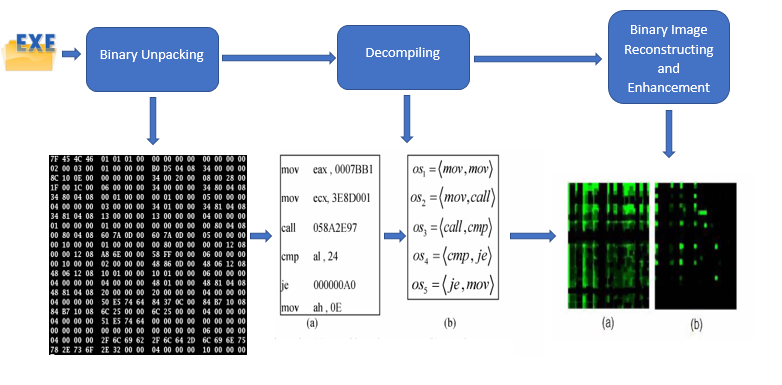


Figure 3-3 Overview architecture of Preparation Dataset using opcode Environmental Settings

3.3.4 Malware Detection by GoogleNet Model

GoogleNet was proposed based on NIN and won the ILSVRC 2014 [107]. GoogleNet uses an efficient Inception Module for reducing the parameters. The naive Inception Module, shown in Figure 3-4(a), applies parallel filter operations on the input from the previous layer so that multiple receptive field sizes can be achieved from the convolution of size 1 \* 1, 3 \* 3 and 5 \* 5. The naive structure has a huge computational complexity. To reduce the computation cost, GoogleNet uses “bottleneck” layers with 1\*1 convolution to reduce feature depth as shown in Figure 3-4(b). The 1\*1 convolution proposed in NIN contributes to reducing the dimension before the expensive parallel blocks of 1\*1, 3\*3 and 5\*5. Furthermore, the 1\*1 convolution makes the possibility of increasing the depth and width of the network. Additionally, average pooling layers plus SoftMax classifier are utilized to replace fully connected layers at the top of the convolutional layers so that a large number of parameters can be eliminated.

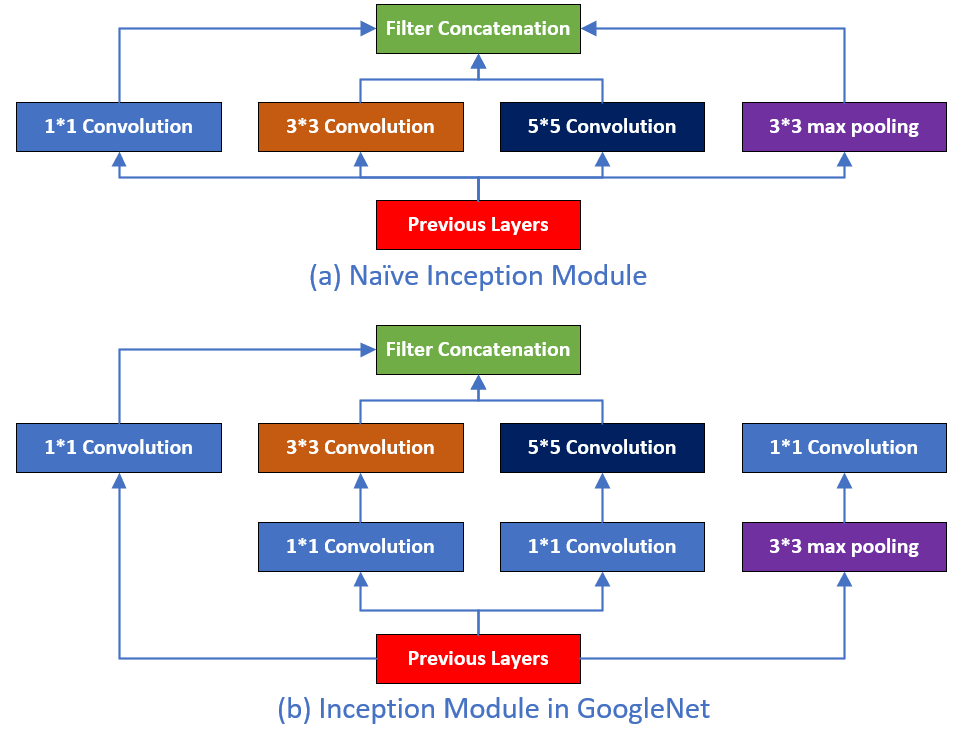


Figure 3-4 Inception modules in GoogleNet Model

The GoogleNet shows work by Google association in 2014, which contains 22 concealed layers. This research has utilized the GoogleNet CNN system since it is solid and it can be connected to the whole picture at once and after that, the author expects an effective feature extraction. However, the model evolving in Inception v2, v3, v4. This approach used inception v4 model. Two strategies of malware detection were also found.

 As the conventional size and weight of the CNN filter is 3 \* 3 or 5 \* 5, the negatively correlated byte code may be correlated when the code is converted into images. A few filters were replaced with a smaller Perceptron layer with a mixture of 1x1 and 3x3 convolutions. In this way, the dimensions were reduced inside the inception module.

 In the CNN model, pooling is a popular solution to drastically reduce the overhead calculation in traditional image recognition. In our academic research, the threat detection engine demonstrably utilizes pooling to accomplish acceleration. Although, grey-scale images are unnatural images, but developed from EXE source code.

 ReLU was used rather than non-linearity work since it is quicker than sigmoid or tanh and helps in vanishing inclination issue which emerges in lower layers of the system.

 It takes a channel, and a walk of a similar length at that point applied it to the volume of information and outputs the most extreme number in sub-district that the channel assembles around. The rationality behind it was; that the images transformed from malicious code are grey-scaled and the layers, e.g., average maximum pooling might not help much because the image has a lot of dark space.

3.3.5 Malware Detection by ResNet Model

ResNet is owned by Microsoft, and it was introduced in 2015. Latest ResNet contains 152 hidden layers. ResNet convolutional neural networks are used in our experiments, and the results show that it is accurate as compare to GoogleNet and it can be applied to all the images at a time. ResNet model is a good choice for extracting the features from images. The dubbing and innovation costs are increased in terms of time and memory but this also the fact that it gives us a high accuracy. Here dubbing means the transfer or copying of previously recorded structure of the same or a different type. In this experiment, the author applied the Tensor Flow ResNet Library which is easy to deploy and achieve more accuracy than GoogleNet.

With more and more layers stacked, deeper CNN does not lead to better performance but accuracy degradation. Its sources from that deeper models become difficult to optimize and train. K. He [108] proposed Residual Network (ResNet) to solve the problem by stacking residual blocks, which mapped the network layers to residual instead of the desired underlying input. ResNet was the winner of ILSVRC 2015. As shown in Figure 3-5(a), every residual block has two 3\*3 convolution layers. Deep Bottleneck Architecture (DBA) can be used to generate deeper networks. As shown in Figure 3-5(b), the residual block is transformed into a bottleneck architecture with three consecutive convolutional layers of size 1\*1, 3\*3 and 1\*1 separately.

As shown in Figure 3-7, a full ResNet with 34 layers groups every two adjacent layers into one residual block, while the plain version of 34-layer network in Figure 3-6 stacks the convolutional layers directly.

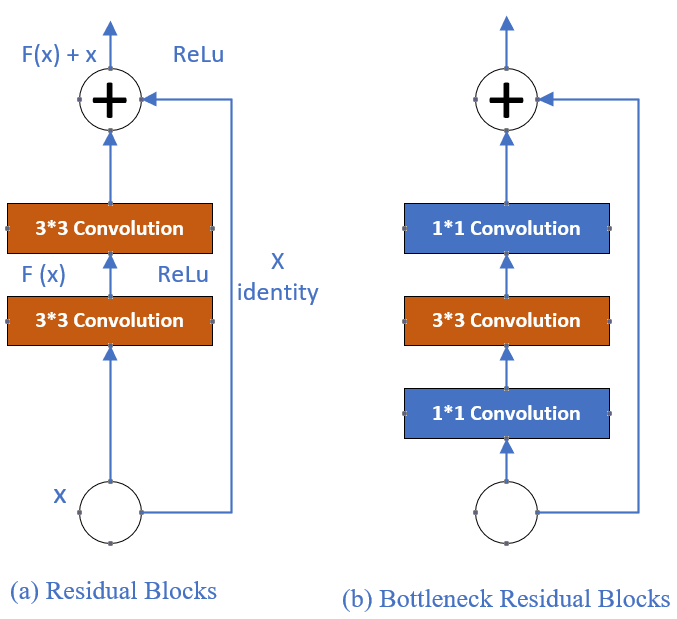


Figure 3-5 ResNet Modules, the bottleneck residual blocks are used in ResNet deeper model

Except for the residual blocks, the ResNet architecture has an additional convolutional layer at the beginning and a global average pooling layer in the end. To reduce the number of parameters and thus improve efficiency, K. He [108] implemented ResNet with DBA. As shown in Figure 3-8, the practical ResNet is built by 50 layers with bottleneck architecture, known as ResNet-50. Using a similar structure, ResNet allows for more than 100 layers stacked and avoid accuracy degradation.

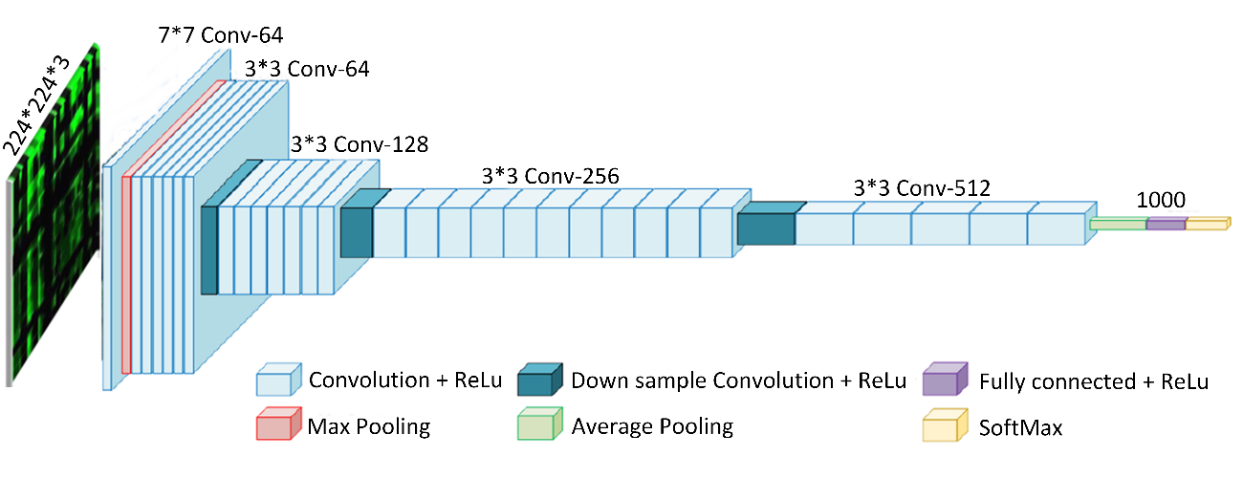


Figure 3-6 ResNet model structure of the 34-layer plain network

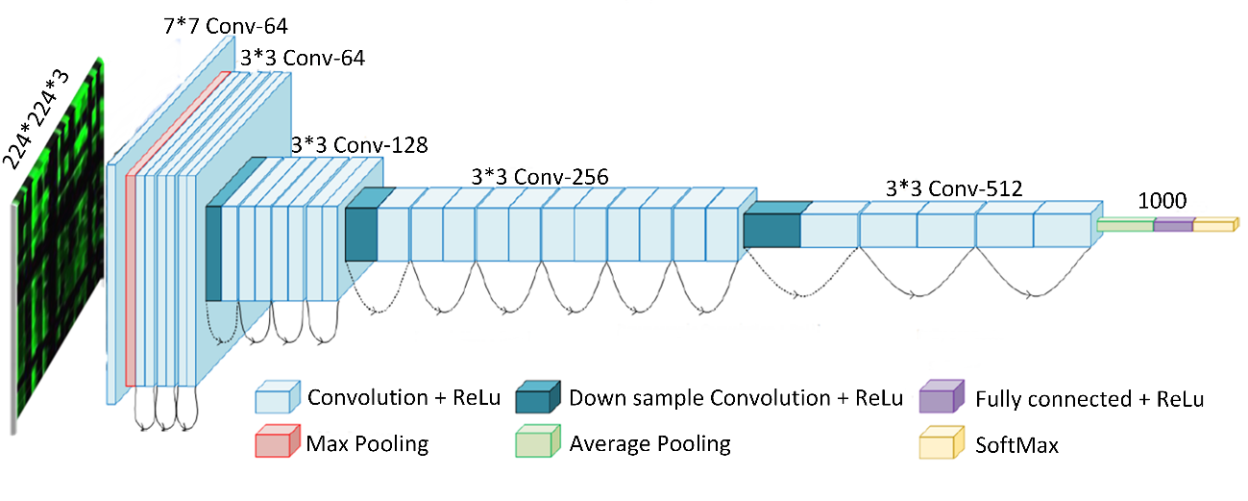


Figure 3-7 ResNet Model structure of the 34-layer residual network

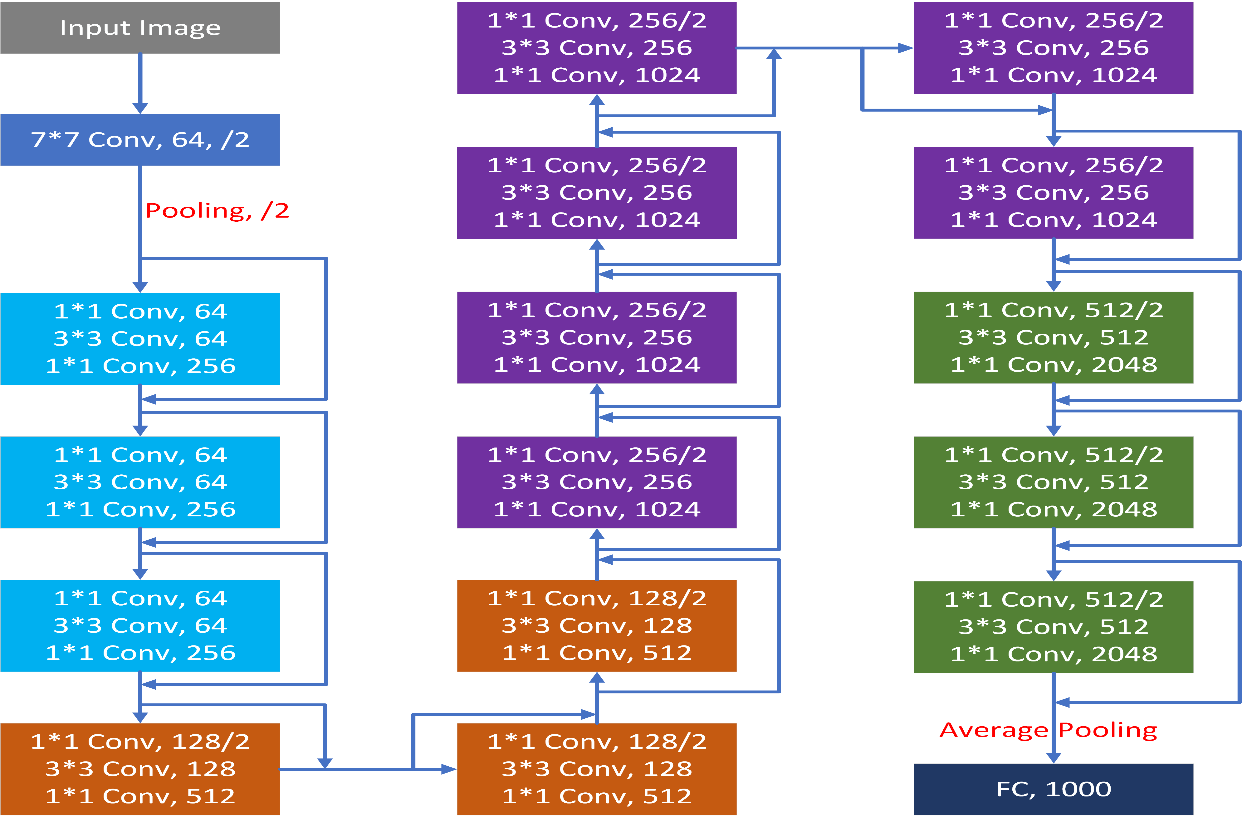


Figure 3-8 Typical structure of ResNet-50

3.4 Discussion on Implementation and Evaluation Results

We have done the experiments on openly sourced datasets from Microsoft Malware Classification Challenge. The images contained Malware and benign samples. The author adopted the Deep Neural network model, i.e. GoogleNet and ResNet model for comparison. It was observed that ResNet model has tremendous performance as compare to the GoogleNet model. We assume that if the datasets are bigger than the dataset we've been experimenting with; ResNet152 would have performed much better. ResNet (18, 50 and 101) also performed better in terms of prediction accuracy which is observed in Table 3-2 and Figure 3-9, Figure 3-10, Figure 3-11, Figure 3-12, and Figure 3-13. It, however, performed poorly in terms of running time on malware dataset which is observed in Table 3-2 and Figure 3-14 while Figure 3-15 shows the accuracy of the models. Be that as it may, when just a restricted measure of preparing information is accessible, all the more capable models are required to accomplish an improved learning capacity. It is along these lines of incredible essentially to think about how to plan deep models to gain from less preparing information, particularly for discourse and visual acknowledgment frameworks. This was apparent in our explored different avenues regarding ResNet uses of optimization algorithms to adjust the network parameters: The technique to modify the parameters in machine learning calculations is a rising theme in software engineering. In DNNs, an extensive number of parameters should be balanced. Additionally, with an expanding number of shrouded hubs, the calculation is more probably get caught in the nearby ideal. Enhancement procedures, for example, the PSO, are hence required to maintain a strategic distance from this issue. The proposed preparing calculation ought to have the capacity to extricate the highlights naturally and diminish the loss of data to moderate both the scourge of dimensional and the local optimum.

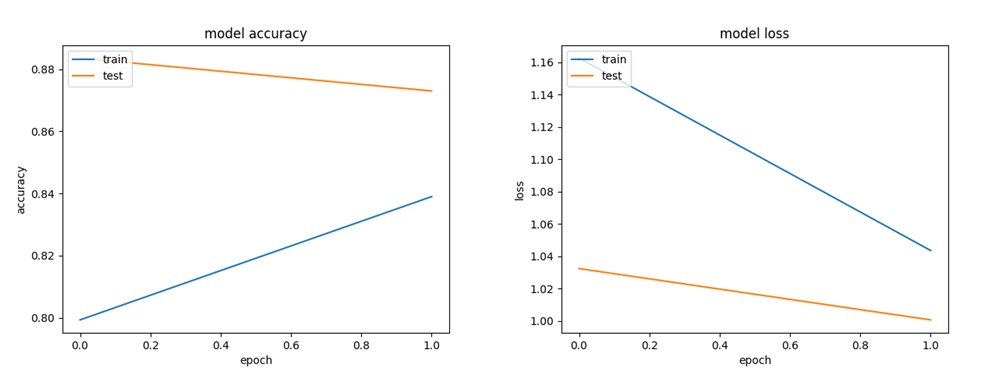


Figure 3-9 Model Accuracy and Model Loss of ResNet 18

From the plot in Figure 3-9, the author observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. The author also observed that the final training accuracy of the model is 0.83 and the testing accuracy is 0.87. Further. It was observed that the model has different execution results on training and validation datasets, i.e. labeled test. If these parallel plots begin to depart reliably, it may be an indication to quit training at a prior epoch. In this analysis, final training loss is noted as 1.0436 and validation loss is noted as 1.006. The execution time of ResNet18 model was noted as 2701 seconds.

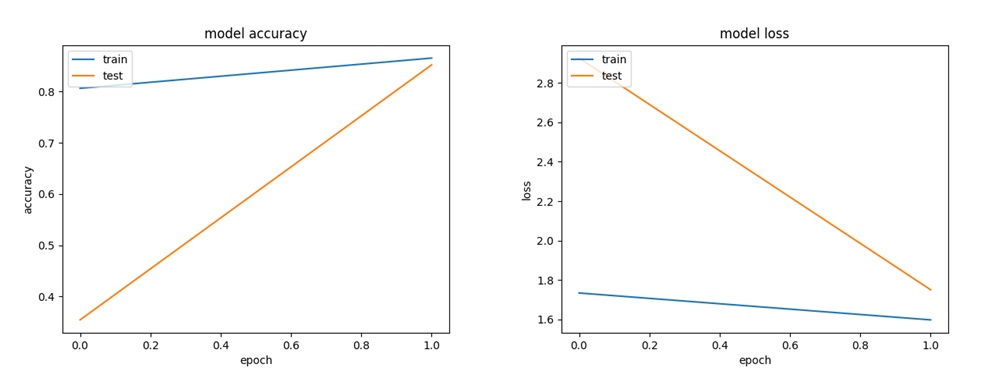


Figure 3-10 Model Accuracy and Model Loss of ResNet 34

From the plot in Figure 3-10, The author observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. The author also observed that final training accuracy of the model is 0.8651 and the testing accuracy is 0.8519. Further. The author observed that the model has different execution results on training and validation datasets. The final training loss is noted as 1.5983 and validation loss is noted as 1.7510. The execution time was noted as 4800 Seconds.

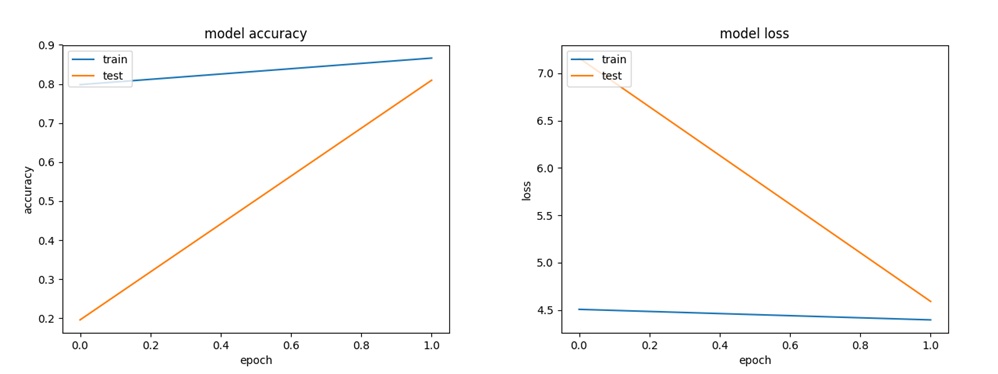


Figure 3-11 Model Accuracy and Model Loss of ResNet 50

From the plot in Figure 3-11, The author observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. Further, the author observes that the model's last preparing accuracy is 0.8662 and testing accuracy is 0.8095. The author also observed that the model has equivalent execution on both training and testing datasets. If these parallel plots begin to depart reliably, it may be an indication to quit training at a prior epoch. In this analysis, final training loss is 4.3967 and validation loss is 4.5914. Add up to execution time was noted as 5580 Seconds.

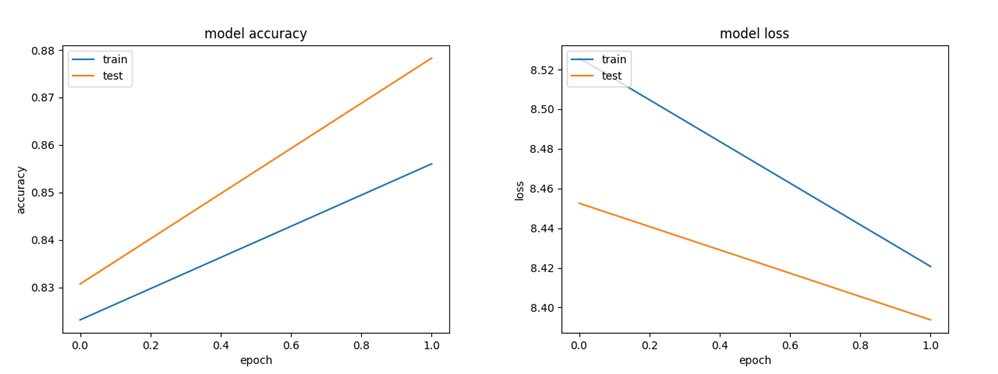


Figure 3-12 Model Accuracy and Model Loss of ResNet 101

From the plot in Figure 3-12, The author observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. Further, the author observes that the model's extreme preparing accuracy is 0.8594 and testing accuracy is 0.7884. Further. The author observed that the model has different execution results on training and validation datasets. If these parallel plots begin to depart reliably, it may be an indication to quit training at a prior epoch. In this analysis, final training loss is 8.4274 and validation loss is 8.7475. Add up to the execution time of ResNet101 was noted as 6112 Seconds.

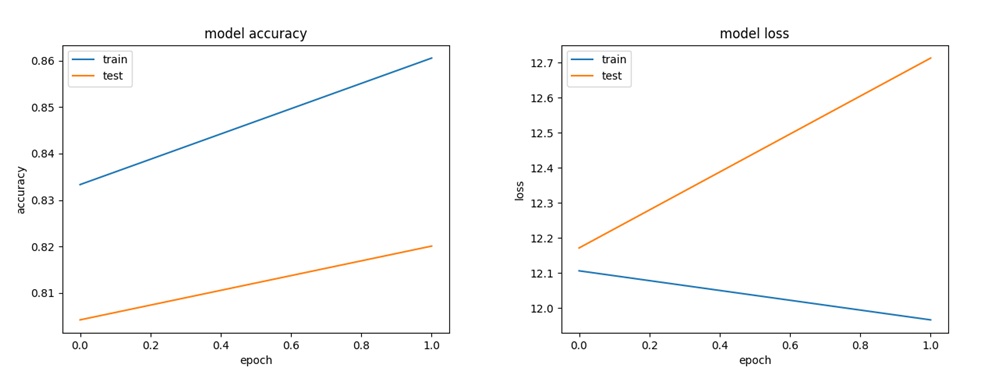


Figure 3-13 Model Accuracy and Model Loss of ResNet 152

From the plot of model accuracy in Figure 3-13, The author can see that the model's final training accuracy is 0.8798 and validation accuracy is 0.8836. In this experiment, training loss is 11.943 and validation loss is 12.05. The execution time for ResNet152 was noted as 9248 Seconds.

Table 3-2 Comparison Results of GoogleNet and ResNet Models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Training Accuracy | Testing Accuracy | Loss | Validation Loss | Time |
| GoogleNet | 0.84 | 0.745 | 0.389 | NA | 1000s |
| ResNet18 | 0.83 | 0.87 | 1.0436 | 1.006 | 2701s |
| ResNet34 | 0.8651 | 0.8519 | 1.5983 | 1.7510 | 4800s |
| ResNet50 | 0.8662 | 0.8095 | 4.3967 | 4.5914 | 5580s |
| ResNet101 | 0.8594 | 0.7884 | 8.4274 | 8.7475 | 6112s |
| ResNet152 | 0.8798 | 0.8836 | 11.943 | 12.05 | 9248s |

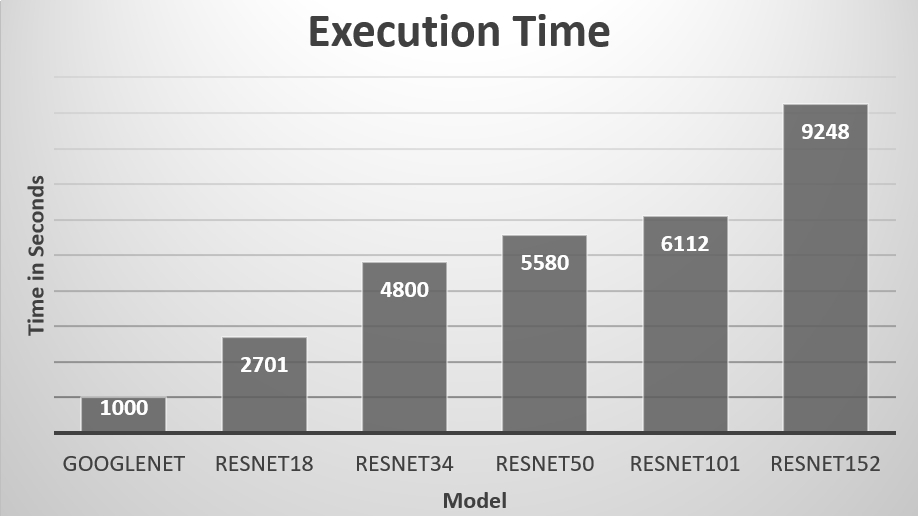


Figure 3-14 Execution Time of Different Models

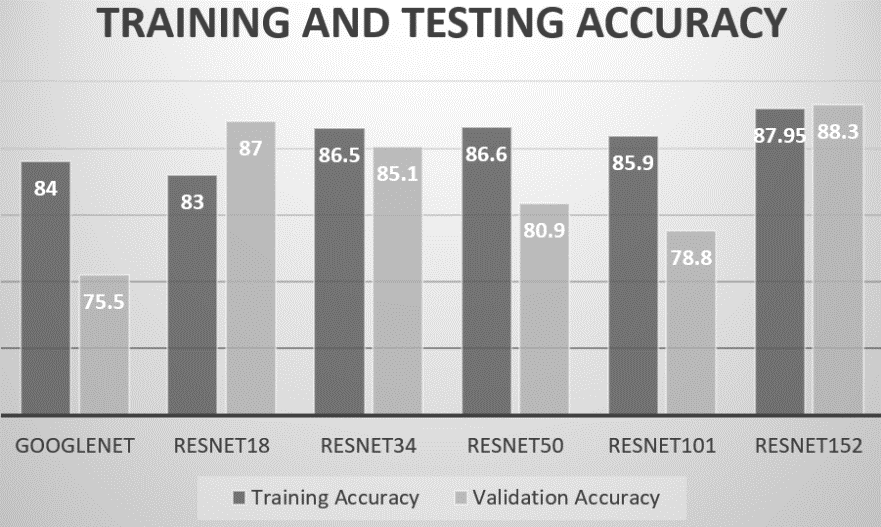


Figure 3-15 Comparison of the Training and Testing accuracy of Different Models

3.5 Summary

Having the capacity to visualizing the vindictive code as images has been a remarkable accomplishment. Numerous analysts have been utilizing this procedure for the errand of malware grouping and identification. In this chapter, it was observed that how a minor change in the picture would possibly lead to a picture misclassification and how a minor changes in the picture can eventually lead to an efficient classification. The greatest test is to locate a proficient method to defeat the vulnerability of Neural Networks. This could be accomplished via precisely malware binaries. The author observed in our study that the GoogleNet Model took less time in execution, but ResNet152 model is more accurate. The execution time of ResNet152 model is the most highly taken time as compare to GoogleNet and other models of ResNet family.

Chapter 4 An Improved Convolutional Neural Network Model for Intrusion Detection

4.1 Introduction

More and more physical devices are connected to the Internet as the development of Internet technology goes on. The connection between devices resulted in a large amount of data being generated and saved. The era of "big data" came into being; however, some valuable data is exposed due to lack of protection measures especially when the device transmits data through continuous connection, thus causing huge losses to individuals and even to the whole country [109]. Many machine learning algorithms are used for malware/intrusion detection so far. Kumar et. al. [110] used the CNN model for malicious code detection based on pattern recognition. With the increasing number of networked devices, network systems will become more vulnerable. This gives hackers an opportunity to steal data, user privacy, and trade secrets more easily [111]. Although people have tried their best to protect their important information, due to the complexity of the network system and the richness of attack methods, cyber-attacks continue to occur [112]. Given these circumstances, cyber-attack detection methods should be smarter and more efficient than ever before, in order to detect and prevent the growing hacking technology. This chapter presents a model for detecting anomalies in the network based on deep learning. Evaluation results show that this model can identify daily cyber-attacks quickly and efficiently.

Methodologies for Intrusion Detection in Networks are generally rules-based and signature-based restrictions, which are deployed to simulate potential threats at the perimeter. Attackers alter the signatures of malicious code and feasibly dodge conventional monitoring systems deployed for intrusion detection. Elejla et al. [113], discussed in their paper that the systems based on deep learning use self-learning to pinpoint or identify undetermined network breaches. Conventional cases of security including intrusion detection systems and spyware detection have been addressed using deep neural network techniques [114]

This chapter studies the network intrusion detection based on convolutional neural networks (CNN) and combines the convolution and pooling operations to better extract the feature relationships between the data. This not only fails to solve the problem of traditional machine learning models. The deep-seated mining of the relationship between data features and a better understanding of the relationships between features than general neural networks.

## 4.2 Proposed Model Design

### 4.2.1 The architecture of Convolutional Neural Network Model

As shown in Figure 4-1, the convolutional neural network structure is composed of input, convolutional layer, pooled layer, fully connected layer, and an output layer. The convolutional neural networks with different structures have different numbers of convolution and pooling layers. Assuming that the input feature of the convolutional neural network is X, and the feature map of the *i*-th layer is , the convolution process can be expressed as:

 (4-1)

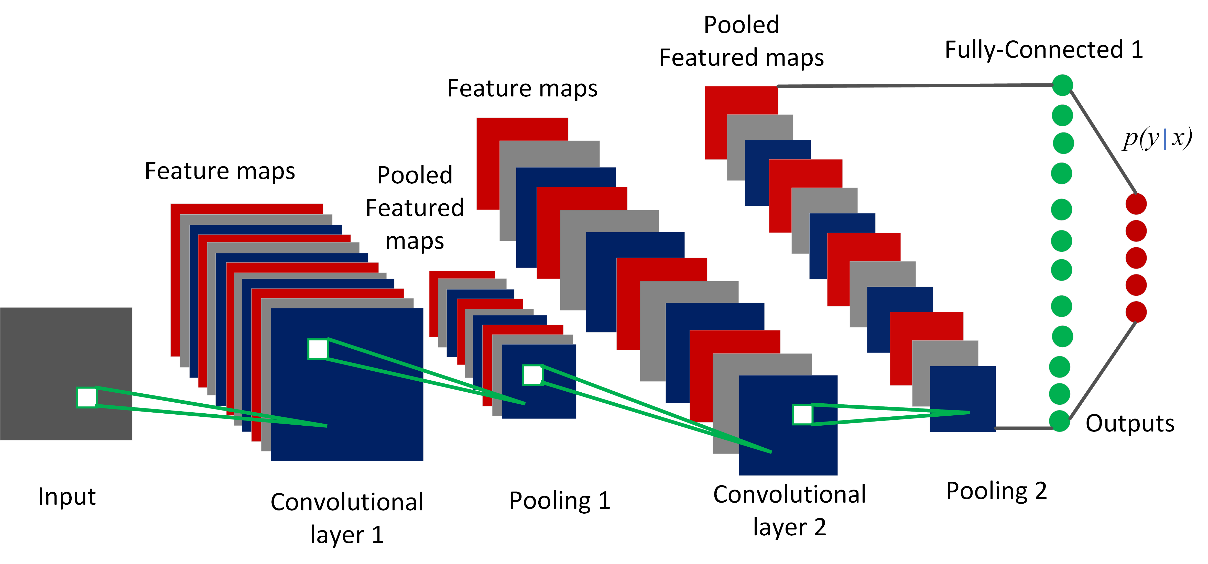


Figure 4-1 Convolutional neural network classical structure diagram

 is the weight vector of the convolution kernel of the *i*-th layer, the operation symbol represents the convolution operation and is the offset vector of the *i*-th layer, and (M) is the excitation function. In the convolution process, the convolution kernel constructs new features by repeating the convolution operation with the input features. When convolving with a convolutional kernel, the principle of “parameter sharing” is followed. That is, sharing the same weights and offsets makes the number of parameters of the entire neural network greatly reduced.

The pooling layer usually samples the feature map according to different sampling rules after the convolution layer. Assume M For the input of the pooling layer,  is the output of the pooling layer, then the pooling layer can be represented as:

 (4-2)

The structure of regular Neural Network is constructed by fully connected layers, where all neurons are fully connected with all neurons in the previous layer. The full inter-layer connection leads to a large number of weights. Besides, the neurons in each layer are learned independently without any shared connections. Therefore, the general Neural Network is hard to train yet easy to cause overfitting.

The structure of Convolutional Neural Network (CNN) improves the regular Neural Network by introducing more complex layers and adopting new techniques. Instead of connecting to all neurons in a fully-connected manner, neurons in a CNN layer are connected to a small region of the previous layer and arranged in three dimensions with width, height and depth.

We have developed an effective convolutional neural network. Three concealed layers are applied to the whole network. There is a convolutional layer and a pooling layer in each concealed layer. For each hidden layer, the total number of convolution kernels varies. The network uses convolution kernels of 2x2 and the pool of 2x2 distributed nodes to continually improve the performance of the network structure. The convolution kernel numbers in the network are different in each convolutional layer. With increasing the number of post-convolution kernels (32-64-128), maps the original features into high-dimensional space, thereby enhancing the ability to learn features. The proposed model is shown in Figure 4-2 where the SoftMax algorithm is used for classification results.

4.2.2 Intrusion Detection Model Framework

Figure 4-2 gives a brief overview of the network intrusion detection framework based on convolutional neural network algorithm used in this chapter. It can be seen in Figure 4-2, that the framework mainly consists of three steps:

**Step 1. Data preprocessing:** It mainly converts symbolic data into numerical data and then normalizes the data. Details are given section [4.3.2](#subsec_Data_preprocessing).

**Step 2. Training and Feature Extraction:** We used our designed CNN model for data training and feature extraction.

**Step 3. Classification:** We used the SoftMax classifier to classify and get the classification results.

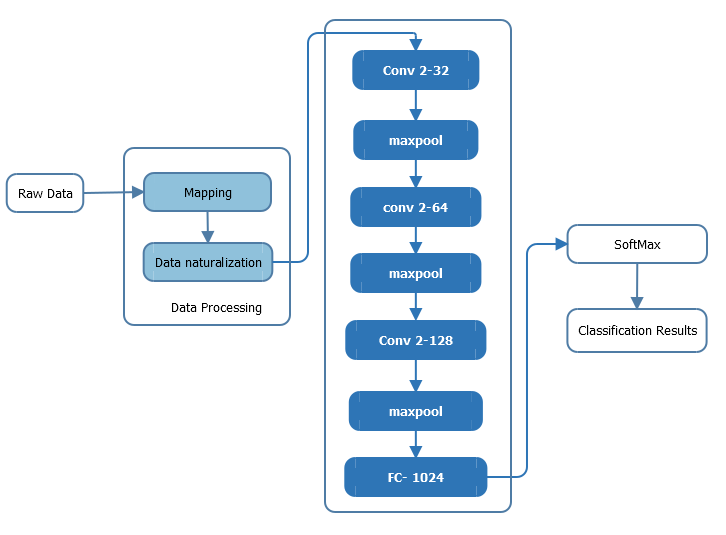


Figure 4-2 Proposed Model for Intrusion Detection

4.3 Data Preparation, Experiments and analysis

4.3.1 Dataset

The data set used in this paper is the KDD99 data set. The data set divides network intrusion into five categories: Normal, DOS, R2L, U2R, and probing. Its features represent each behavior. This chapter uses the KDD99 dataset to train the model. This data set contains 494021 training samples and 311029 test samples; the distribution of various types of invasion is shown in Table 4-1.

Table 4-1 Distribution of KDD99 datasets

|  |  |  |
| --- | --- | --- |
| Attack Type | Training Sets | Testing Sets |
| Benign | 97278 | 60593 |
| DOS | 391 458 | 229 853 |
| R2L | 1126 | 16189 |
| U2R | 32 | 228 |
| Probe | 4107 | 4166 |

4.3.2 Data preprocessing and Attack Detection

The KDD99 dataset contains 41 features per record. It contains 38 numerical features and three symbolic features. For these features, the dataset needed to be processed separately.

**1) Numerical characterization of symbolic features:** For the three symbolic features we used the one-hot-encoded method to digitize, for example, for the protocol-type feature, it contains three characters No., i.e., TCP, UDP, ICMP, we converted it to , , , i.e., the 1D vector into a 3D vector.

**2) Normalization of numerical features:** For numerical features, due to the different dimensions, the magnitudes of the numerical features are very different. Therefore, in order to eliminate the influence of dimension differences, it is necessary to carry out numerical values. The normalized formula is as follows:

 (4-3)

**3) Feature Extraction by Latent:**  In this step, the latent features are extracted using matrix factorization. In the training phase, we minimize the approximation error.

In this phase, we used one-hot-encoder for the multilayer network. Each neuron connected with pervious layer, i.e., Flow of the data is unidirectional from the previous layer. The output layer of the neural network is functioning an input to the next layer. The output computed as using a single deterministic pass. The weights are calculating by using the equation below:

 (4-4)

The activation function is defined as:

 (4-5)

The logistic equation of the positive derivative given below:

 (4-6)

**4) Attack Detection:** Each class can be labeled as benign or attack as shown in the equation below:

 (4-7)

Thus, the training dataset can be denoted as:

 (4-8)



Figure 4-3 Hyperplane of the maximum margin between two classes

Figure 4-3 demonstrates a hyperplane represented by (w, bias), in which w represents the weight and bias is built in a relatively limited area of the training subsample. This is done because the bigger the margin, the lower the classifier's particular error.

4.3.3 Setting Experimental Parameters

This experiment uses accuracy (ACC) as an evaluation index to measure the effect of the model. ACC formula is as follows:

Among them, TP is the number of samples of attack behaviors that are correctly classified;



TN is the number of samples of normal behaviors that are correctly classified;

FP is the number of samples of normal behaviors that are misclassified;

FN is the number of samples of misclassified attack behaviors.

Table 4-2 Metrics for Prediction

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Predicted |  |
|  |  | Benign | Attack |
| Actual | Normal | TN | FP |
|  | Attack | FN | TP |

4.3.4 Analysis of Experimental Results

The data used in this article is a 10% KDD99 dataset use Accuracy (AC) as a verification indicator. The convolution kernel has a convolution kernel whose length and width are both set to 2, the step length is set to 1, the length and width of the pooling layer are both set to 2, the step length is set to 2, and the pooled layer adopts max. The pooling algorithm performs down sampling using the Adam optimization algorithm to optimize the loss function. We observe changes in accuracy by setting the number of epochs in the convolutional neural network. From Figure 4-4 it can be seen that with the continuous increase in the number of epochs, accuracy is rising. It is also observed in Figure 4-4 that accuracy is growing when we increase the number of epochs.

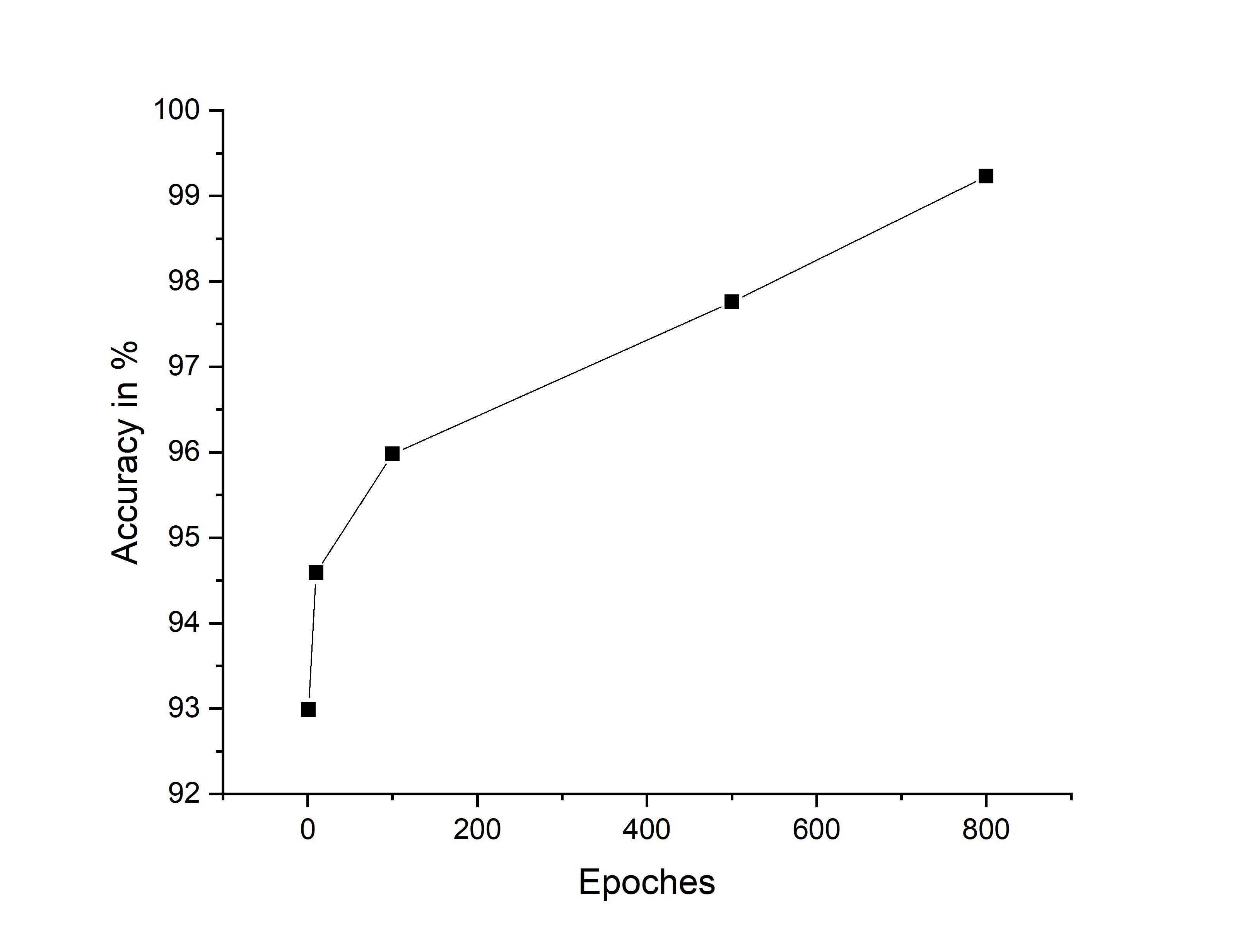


Figure 4-4 Increasing the accuracy of test results by increasing the number of epochs

Table 4-3 and Figure 4-5 compares the detection effect by accuracy of the model and SVM, DBN and CNN algorithms. It can be seen that the detection efficiency of the improved CNN model is higher than that of other algorithms. Therefore, the proposed model is effective.

Table 4-3 Comparison of SVM, DBN, and Improved CNN models

|  |  |  |
| --- | --- | --- |
| Model | | Accuracy % |
| SVM |  | 98.20 |
| DBN |  | 98.59 |
| Improved CNN | | 99.23 |

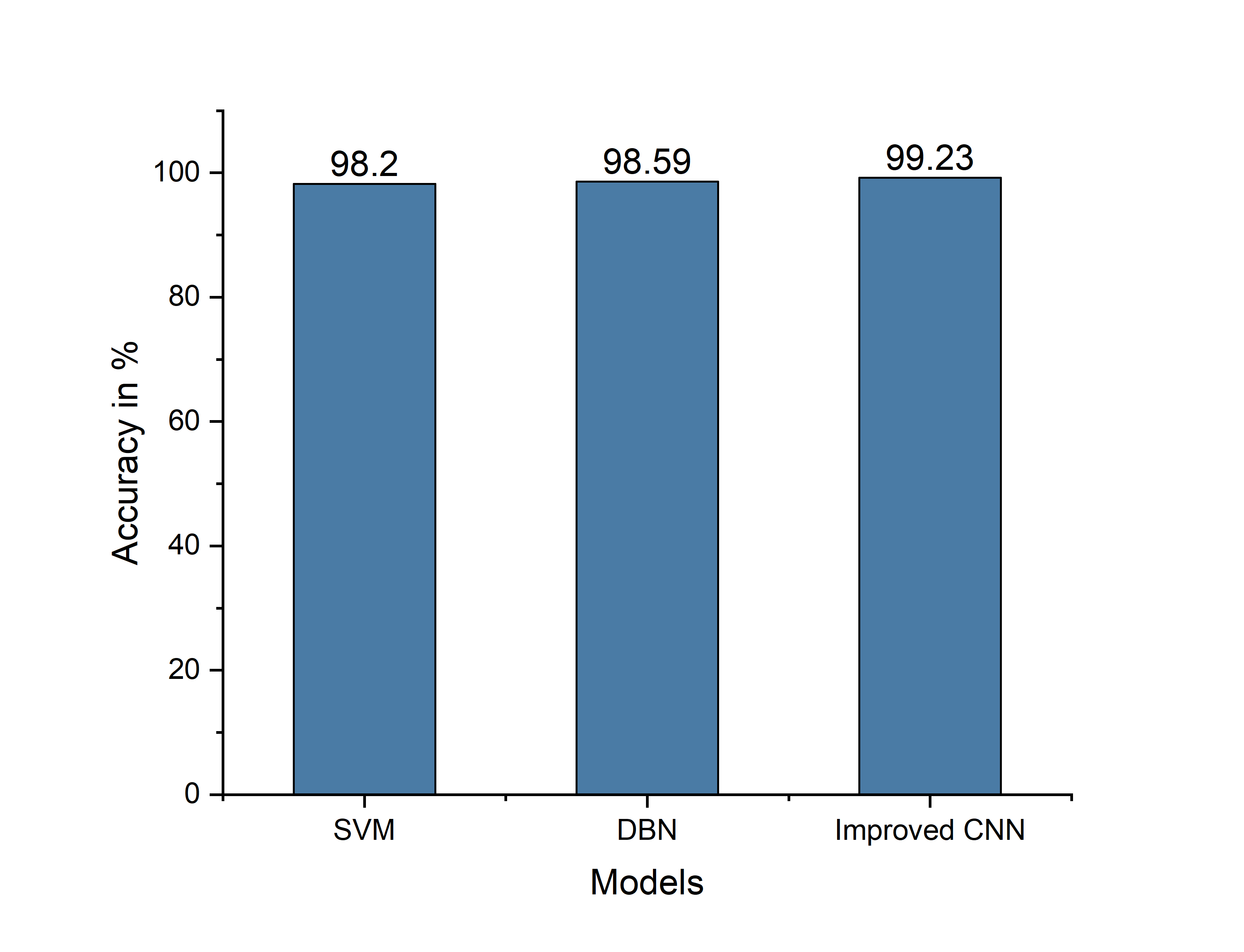


Figure 4-5 Comparison of Improved CNN and other models

4.4 Summary

The application of the convolutional neural network algorithm in intrusion detection is a new idea. This chapter proposed a methodology that combines the convolutional neural network algorithm and SoftMax algorithm. The experimental evaluations demonstrated that this model can improve the accuracy of human intrusion detection and improve the performance of human invading detection system. It is observed in the results that the efficiency is increased when we increase the number of epochs. It is also observed that the proposed model performed better as compare to SVM and DBN models.

Chapter 5 A Hybrid Technique to Detect Botnets based on P2P Traffic Similarity

5.1 Introduction

Nowadays, the network environment is highly complex, and the security problem is becoming more and more prominent. As the botnet C & C server has a higher degree of concealment, unknown programs are often used by large-scale network intruders. The botnets initiate almost all of the DDoS attacks and 80% to 90% of the spam attacks. Therefore, the botnet has become a big threat to network security and cannot be ignored. Early botnets normally used IRC and HTTP as a communication protocol, with a single failure point, and it has been easy to be detected and destroyed. Today, most of the botnets use P2P technology to create C & C (command and control) mechanisms to enhance network traffic concealment. Compared to botnets with IRC and HTTP protocols, P2P botnets without central nodes have greater threat and concealment. Therefore, P2P botnet is increasingly favored by attackers. P2P botnet detection has also become a hot research area in the field of cybersecurity.

At present, P2P applications have caused the explosive growth of Internet traffic, which is a huge challenge in terms of data storage and real-time analysis. Therefore, the network of non-P2P traffic filtering is particularly essential. This chapter aims to examine the features and strategies to detect botnets. Main contributions of this chapter are as follows:

 This chapter presents a novel approach to classify network traffic and identify botnets through machine learning algorithms.

 The classification has been done using two-stage technique. This technique covers the limitations of single stage botnet detection, e.g. class imbalance.

5.2 Approach

Botnets have been the most common threats to network security in recent times because they use various malevolent codes such as a worm, trojans, rootkit, etc. The botnets have been used to carry phishing links, to perform attacks and provide malicious services on the internet. This chapter proposes a two-stage detection method for P2P botnets, i.e., the first stage is based on port judgment, DNS query and data flow count in the session to filter non-P2P traffic, and the second stage is based on session characteristics to identify P2P botnet. The method is used on the bases of the session feature to effectively reduce the data packets to be analyzed. Furthermore, Machine Learning algorithms are used to classify and identify traffic. At the same time, we compare our experiments by using three machine learning algorithms on the datasets collected from diverse sources. The experimental results show that the Decision Tree algorithm is the most accurate for P2P botnet detection.

5.3 Proposed Model

This section describes the methods introduced in this chapter to detect bot-bling traffic in two phases. The focus of this method is on non-P2P traffic filtering and the extraction of the characteristics of the session. The architecture of the model is shown in Figure 5-1. The first stage of the model will start from the three aspects of, packet filtering rules, session characteristics, and classification algorithm. The second stage classifies the traffic as either the traffic is normal P2P or botnet traffic.

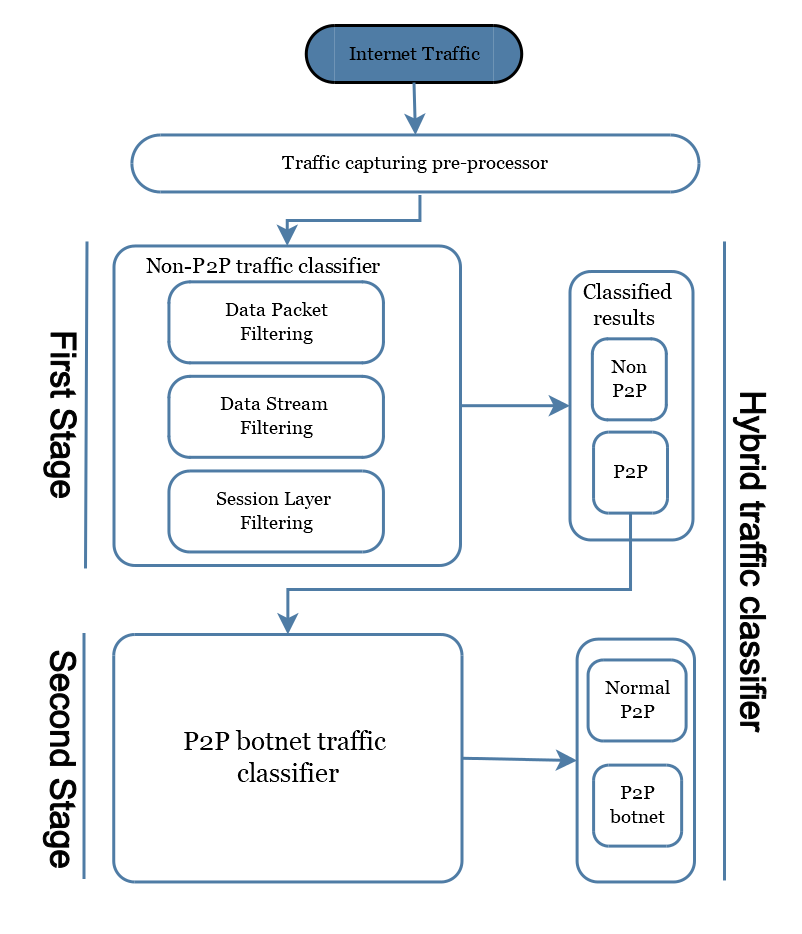


Figure 5-1 Architecture of the proposed method

5.3.1 First Stage of Traffic Classifier:

At present, port identification, signature recognition, and identification are commonly used methods for P2P traffic identification. These methods are based on stream feature [99]. However, the port identification method cannot recognize P2P applications with random ports or custom ports. DPI (Deep Packet Inspection Technology), does not identify encrypted P2P traffic [115]. Stream-based identification methods can only determine P2P applications of the partial flow and have a high false alarm rate. Therefore, we use the non-P2P well-known port filtering mechanism, DNS query, flow counting rules to filter non-P2P traffic, combined with fast heuristic P2P traffic identification method, as shown in Figure 5-2.

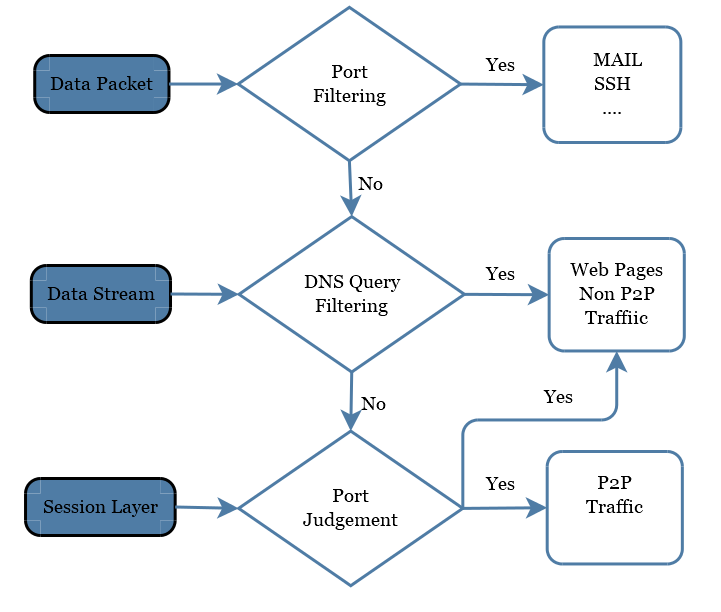


Figure 5-2 First Stage Traffic Classifier

Port filtering is a packet-level filtering method, mainly filtering the commonly used non-P2P application traffic. DNS query is a stream-level filtering method, flow counting, and port judgment is a session-level filtering method, the two rules are mainly filtered web pages and other non-P2P traffic. Among them, the port-based filtering method can identify some common non-P2P application traffic, such as SSH generally use port 22, Telnet (remote login) use port 23. Commonly used applications and their corresponding port numbers are shown in Table 5-1.

Table 5-1 Common applications and their corresponding ports

|  |  |  |
| --- | --- | --- |
| Application | | Port Number |
| SSH |  | 22 |
| TelNet |  | 23 |
| MAIL | | 25, 110, 143, 465, 220, 993, 995 |
| NetBios | | 125, 137, 139, 445 |
| Remote | | 3389 |
| FTP | | 20, 21 |
| NTP | | 123 |

In general, P2P node communication does not require domain name resolution but directly read the IPS list stored in the local configuration file to obtain IP. However, for non-P2P applications, DNS domain name resolution must be used to obtain IP. Therefore, one of the criteria for determining non-P2P network data flows such as Web and Mail, etc., is resolved by the domain name and maybe the destination IP address in the network flow.

When a user sends a Web application service request normally, the Web application uses a multi-port, parallel-requested connection to an IP address on a page. As a result, multiple data streams appear in the same session. The P2P network node communicates each time using a pair of random source and destination ports. Therefore, we can use flow counting and port determination to filter non-P2P traffic. If a session is using the TCP protocol and 80,8080 or 443 port, and the number of sessions in the flow exceeds the threshold, then the session can be considered a web page traffic session. Where the number of valid streams in a session is represented; the threshold is selected based on the number of streams that appear in the normal page access session. Using the capture tool to collect simple and relatively complex web page requests, the analysis results show that the simple web page is generally 3 to 4 connection requests, and the complexity of the page connection request is 5 to 8. Therefore, the threshold is set to 3 in this chapter.

Although this phase of the method cannot accurately detect the identified P2P applications, but it can be in the real network environment to filter out the vast majority of non-P2P traffic and a small amount of secure P2P traffic.

5.3.2 Second Stage of Traffic Classification

Through the analysis of the data flow characteristics of P2P botnet, the traffic characteristics between the zombie hosts that join the same botnet are similar. Therefore, this chapter uses the session-based strategy for feature extraction, that is, with the same destination address of the data flow in the same session, reducing the number of stream features and the number of data, thereby improving the detection efficiency.

**(i) Session Duration:**

The zombie program automatically completes P2P zombie host and other zombie host communication process, the flow of the duration is generally short and very fixed. Therefore, you can extract the average, maximum, minimum, and standard deviation of the duration of the session, and the average interval of the upstream (downstream) stream packets in the session as a feature.

**(ii) Distribution of The Flow in The Session:**

In the process of communication between two nodes in the P2P botnet, the size and transmission quantity of the transmitted packets are relatively small, and the C & C communication flow generated by the zombie host in the same botnet has great similarity. This was observed in our simulations. Therefore, we can distinguish between normal P2P network traffic and P2P botnet traffic by using the distribution of traffic in the session. The average of the maximum packet length of the upstream/downstream in the extraction session, the average of the average packet length, the average of the minimum packet length, the standard deviation of the average packet length, and the average of the number of valid packets, the standard deviation of the number of packets, the average number of bytes transmitted, and the standard deviation of the number of bytes transmitted as a feature.

5.4 Experimental Results and Analysis

5.4.1 Evaluating Metrics

We assessed the execution of our methodologies utilizing 10-fold cross-validation. The methodology of k-fold cross-validation is shown in Figure 5-3. The first example was arbitrarily apportioned into ten equivalent measured sub-tests. Nine sub-tests were utilized for training the model and the remaining one sub-test was held for the testing. The procedure was rehashed ten times, utilizing an alternate sub-sample for testing, every time. The outcomes were then found the average value for the single and final result. All tests were utilized once for validation. Furthermore, we used a wrapper method for feature selection. The mechanism of the wrapper technique is shown in Figure 5-4.

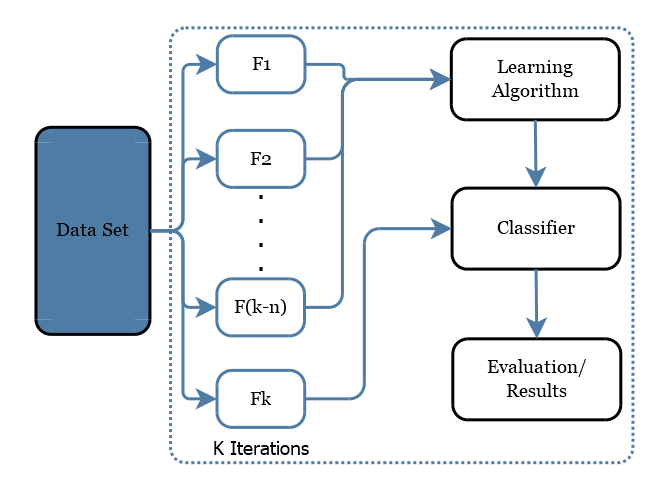


Figure 5-3 K-Fold Cross Validation

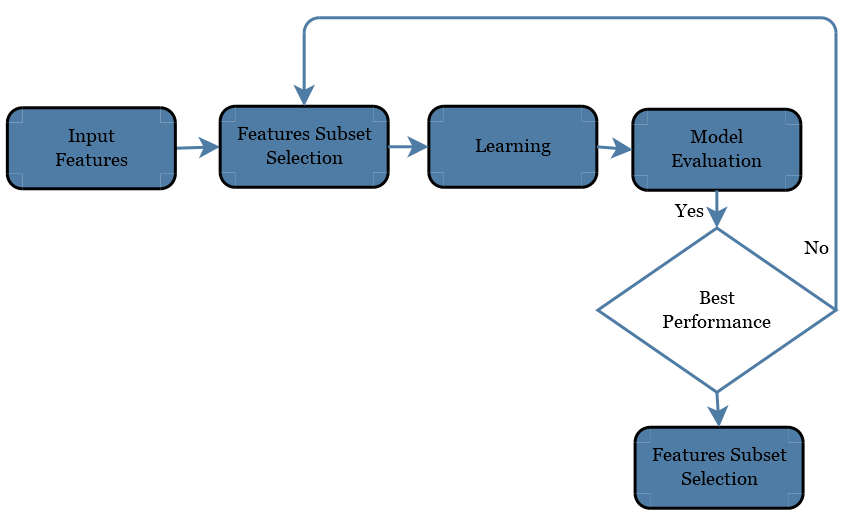


Figure 5-4 Wrapper Method for Features Selection

5.4.1.1 The accuracy of Detection Model

The percentage of correctly classified instances among the total number of instances.



5.4.1.2 False Alarm Rate

FP False positive rate—the rate of P2P recognized incorrectly.

FN False negative rate—the rate of Normal P2P recognized incorrectly as botnets.

The following formula calculates recall rate with the given number of true positives and false negatives.



The TPR is referred to as “sensitivity” or the “true positive rate” sometimes. Precision is calculated by the following formula which is also known as “positive predictive rate”:



5.5 Dataset and Experimental setup

To ensure the reliability and scalability of our proposed model, we trained our model with network flow data from a diverse variety of sources. The dataset includes different types of botnets tested in different kinds of environmental setup. We set a *VMWare* virtual environment in windows 10 and Linux operating systems. *Nfdump* was set up on the system to collect network data. *Nfdump* captures network flow and stores into *nfcapd* files. One instance of the *nfcapd* file is associated with a flow data record over time. We used *Wireshark* in a window environment to capture the network flow because *Wireshark* is an open source software which is available free of cost. The advantage of utilizing *nfcapd* in a Linux domain is that it records countless highlights of the network traffic which turn out to be very favorable in further investigation. It likewise runs discreetly out of sight utilizing insignificant handling memory and power, thus an ideal decision as a tool to gather information. Description of the famous datasets which we used in our experiments are discussed below.

CTU-13 Dataset [116], [117]: We used the dataset of CTU-13 project to do experiments because it contains thirteen various captures of different botnet samples, i.e., IRC, SPAM, CF (Click Fraud), DDoS, FF (FastFlux), PS (Port Scan), US (Compiled and Controlled by us), HTTP. The capture files are stored in the *pcap* form. The dataset of CTU-13 project is a labeled dataset with background traffic, botnet and normal. We have also downloaded non-malicious packets to combine with CTU-13 dataset.

5.6 Classifier Selection

So far, many supervised machine learning algorithms are used to classify data, e.g. Khan et al. [118][119] analyzed ResNet for malware detection using image processing technique. Kumar et al. [120] used the CNN model for malicious code detection based on pattern recognition. In this chapter, we have compared the following classification algorithms to verify the detection rate of the proposed method;

1) Naive Bayes classification algorithm,

2) Decision Tree classification algorithm,

3) ANN

These algorithms are based on session characteristics to detect P2P botnet traffic, the Decision Tree algorithm shows a high accuracy. The Decision Tree algorithm uses the binary tree as a classification tree. The principle of each classification tree is recursively from top to bottom, and its training set is obtained by returning the original training data set. In order to minimize the occurrence of the fitting phenomenon, the Decision Tree uses the Bagging random sampling method to construct the classification tree. Therefore, this chapter uses the Decision Tree classification algorithm for high-speed network environment P2P botnet traffic detection.

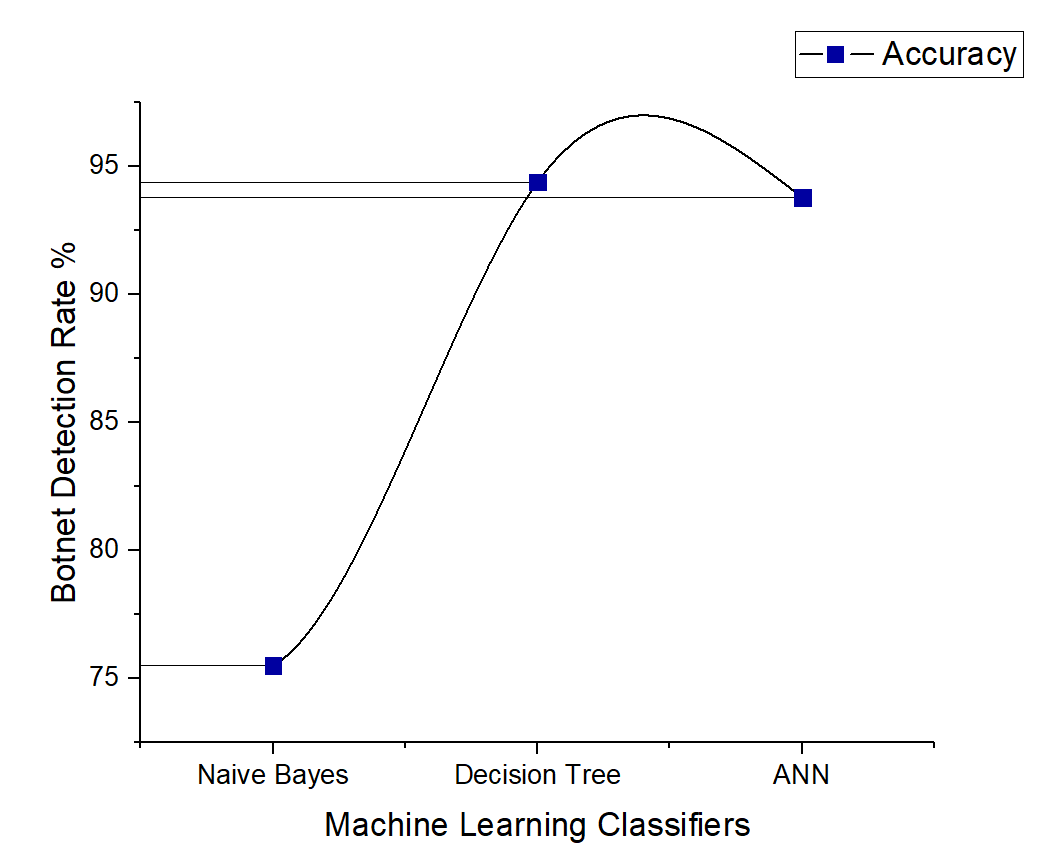


Figure 5-5 Comparison of three machine learning classifiers on P2P botnet detection

Using Naive Bayes classification algorithm and ANN, the detection rate was 75.5% and 93.8%, respectively, but the results of the Decision Tree algorithm were noted as high as 94.4%. Therefore, the Decision Tree algorithm for various types of P2P botnet traffic detection is more accurate than the other two classification algorithms. So, the Decision Tree detection algorithm based on session feature has greatly improved the detection rate of P2P botnet.

5.7 Summary

In this chapter, a hybrid technique for P2P botnet detection is proposed on the basis of session features. Firstly, non-P2P traffic was filtered from packet, stream and session level respectively. Then, P2P botnet classifiers were used to classify the Normal P2P communication and P2P botnet on the basis of session features. This study combines the advantages Detection Method Based on Flow Similarity. The validity of the proposed method is verified by using the open sourced published data set. It is noted from the experimental results, that two-stage technique can effectively detect P2P botnet traffic. We evaluated the model by comparing three different classifiers and noted that the Decision Tree classifier has higher accuracy.

Chapter 6 An Adaptive Multi-Layer Botnet Detection Technique

6.1 Introduction

Nowadays, the network environments are becoming more and more complex and so as security problems. A Botnet is a system of customized bots (computers) controlled remotely by a botmaster. A botnet can perform different noxious exercises, for example, sending spam messages, phishing, click misrepresentation, DDoS and spreading malicious programming. To viably oversee a botnet, the botmaster develops a framework of a correspondence channel to send directions to the Bots and to get results from them [121]. The fundamental contrast between a botnet and other malicious code is the structure utilized in the command and control (C&C) [122]. Compared to other malware programs which are being used to perform malicious conduct exclusively, a botnet functions as a gathering of contaminated hosts dependent on the C&C correspondence channel. A botnets system can be ordered into two principal classes dependent on the C&C foundation: brought together and decentralized C&C [123]. In incorporated botnets, the botmaster typically utilizes the C&C server to send a direction to the bots. Figure 6-1 gives a brief overview of the centralized IRC/HTTP traffic and decentralized P2P botnet traffic. As the higher degree of concealment of botnet command and control server (C & C server), the attacker uses unknown programs to create intrusion in a large-scale network.

Moreover, the botnets initiate almost all of the DDoS attacks and 80% to 90% of the spam attacks. Consequently, the botnet has become a significant threat to network security. Early botnets were easily detected due to commonly used IRC and HTTP as a communication protocol, with a single failure point. Recently, most of the botnets use P2P technology to create C & C mechanisms to enhance network traffic concealment. As compared to botnets with IRC and HTTP protocols, the P2P botnets without central nodes have greater threat and concealment. Therefore, the P2P botnets are increasingly favored by attackers. P2P botnet detection has also become a hot research area in the field of cybersecurity. At present, P2P applications have caused the explosive growth of Internet traffic, which is a massive challenge in terms of data storage and real-time analysis. So, the network of non-P2P traffic filtering is particularly important.

Recent research works on botnets among our surveyed literature focuses mainly on designing systems to detect command and control (C&C) botnets, where many bot-infected machines are controlled and coordinated by few entities to carry out malicious activities [6]. Those systems need to learn decision boundaries between human and bot activities. Therefore, ML-based classifiers are at the core of those systems and are often trained by labeled data in supervised learning environments. The most popular classifier is support vector machines (SVMs) with different kernels, while spatial-temporal time series analysis and probabilistic inferences are also striking techniques employed in ML-based classifiers. Clustering is mostly used in natural language processing (NLP), to build a large-scale system to identify bot queries [124]. In botnet detection literature, the core assumption widely shared:

 Botnet behaviors are different and distinguishable from a legitimate human user, e.g., human actions are more complex [125].

Research has been done on the state-of-the-art machine learning models of the network to simulate real-time network traffic and create honeypots. The ground reality is often heuristic or a combination, labeled by human experts; for example, the game masters visual inspections serve to detect bots in online games [126]. In retrospect, the evolution of botnet detection is clear from earlier and more straightforward uses of classification techniques such as clustering and Naive Bayes; the research focus has been expanded from the last step of classification to the important preceding step of constructing suitable metrics, that measures and distinguishes bot-based and human-based activities [125].

This chapter aims to examine the features and strategies to detect botnets. Main contributions of this paper are as follows:

 This chapter presents a multi-stage approach to classify network traffic (P2P botnet traffic and non-P2P traffic) and identifies botnets by applying machine learning classifier on network features such as port filtering, DNS query, and flow counting.

 The proposed technique of this study covers the limitations of single stage botnet detection, e.g., class imbalance.

 The accuracy of our model is 98.7 % because the threshold of false alarm rate was reduced to 3. The accuracy was improved up to 99% by considering the factor if benign files also send out search requests consistently so benign file may be reported as botnets. Additionally, it was observed the accuracy might be improved by increasing the epochs of deep learning algorithms at the expense more execution cost.

 To validate the performance of our proposed technique, we done the experiments on diverse datasets and the results are compared with five machine learning algorithms implemented for botnet detection.

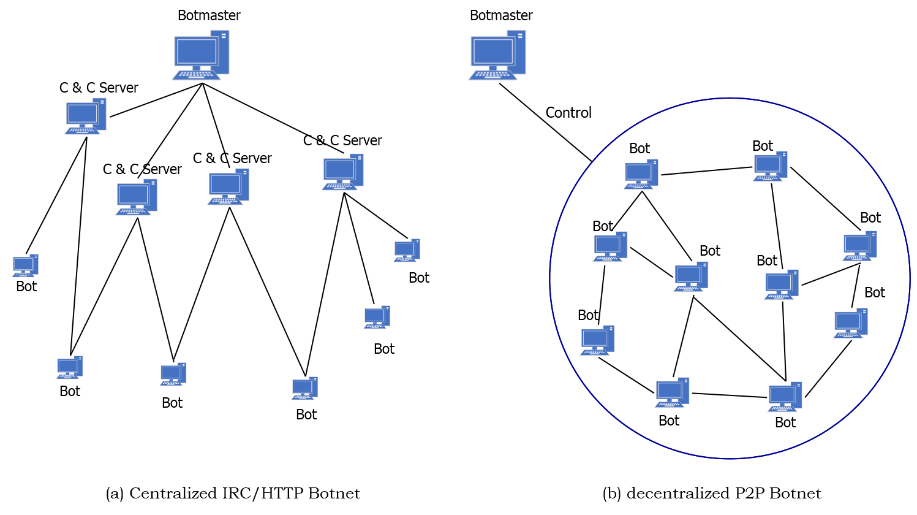


Figure 6-1 Difference between centralized and decentralized P2P botnets.

6.2 Proposed Scheme

Three unique qualities of communication, i.e., P2P botnet, non-P2P and, Normal P2P activity, are demonstrated in Figure 6-2. These three characteristics overlap with each other. If the three types of activities are grouped into a single stage mechanism, a large part of the P2P botnet traffic with normal qualities for non-P2P activity could be misclassified and few substantial P2P streams could be misclassified as countless streams due to the class imbalance problem. In this way, a single-phase mechanism doesn't have the capability to order the traffic of three different qualities at the same time.

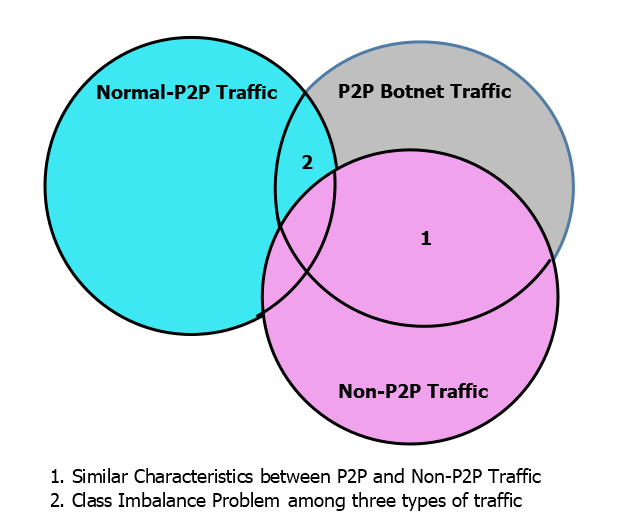


Figure 6-2 Overlapping qualities among three types of traffic.

Therefore, a multi-stage method with a specific aim of settling the limitations of the single stage mechanism is very important. While there are many conceivable techniques exist, our research proposes a multi-stage mechanism, shown in Figure 6-3. The first stage of our model manages traffic reduction by filtering the TCP control packets and feature extraction. The second stage classifies the most P2P streams as indicated by the basic attributes of non-P2P and P2P traffic. In the final stage, P2P botnet activity is detected among the P2P communication through the uncommon attributes. The correlation of error is lower among all stages since the classifiers are trained in each stage on the bases of statistics and utilizing distinctive stream feature sets. In this way, the multi-stage methodology defeats the class imbalance issue, also decreases the error rate during the P2P botnet detection process.

6.2.1 Multi-Layer Detection Method

It is a difficult task in cybersecurity to identify infected computers before the Bot exploits the host machine. Several techniques have been advocated in recent years to classify provocations to Botnet. These strategies can become worthless when the local network conditions change. All offline methods may be illegitimate to detect botnets accurately, in this scenario since they do not contain online techniques. The main objective of this chapter is therefore to incorporate an effective online approach to bot detection using reinforcement learning. The preceding chapter concentrated on the flow of network traffic, the processing of features and the offline countermeasures of bot. This section of the chapter provides a short explanation to the proposed system, along with the modules and various layers of the model. In addition, this section of the dissertation formulates the Botnet problem based on reinforcement learning, followed by a model-based algorithm to detect bot in a dynamic environment online efficiently.

This section describes the method proposed in this paper to detect bot-bling traffic in phases. The focus of this method is on non-P2P traffic filtering and the extraction of the characteristics of the session. The architecture of the model is shown in Figure 6-3. The first stage of the model will start from the traffic reduction followed by the three aspects of packet filtering rules, session characteristics, and classification algorithm. The final stage classifies the traffic as either the traffic is normal P2P or botnet traffic. The detail discussion on the entire mechanism of the proposed scheme is given in the rest of this section.

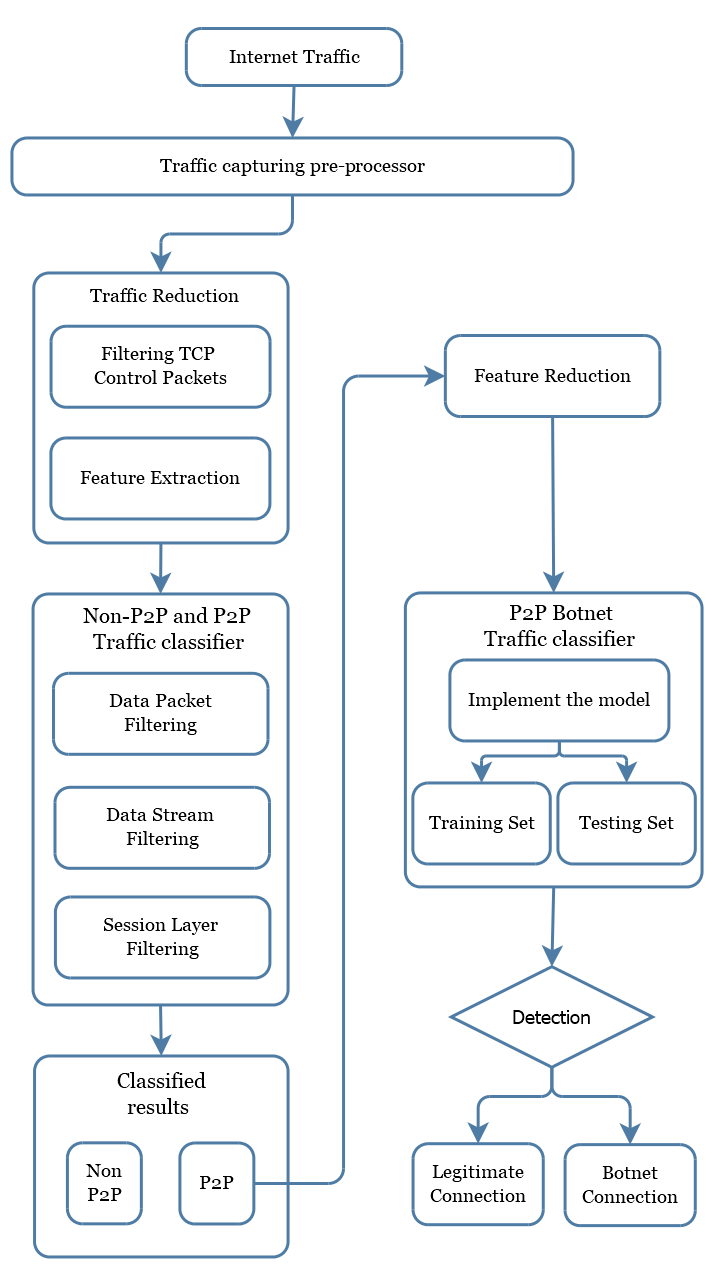


Figure 6-3 Architecture of the proposed method

6.2.2 Traffic Reduction

Reducing network traffic to detect malevolent behaviors is extremely important to manage a large amount of network traffic with limited resources, e.g., hard drive and Memory. The hardest step in the process is to determine the network activity behavior by only scanning several packets for each flow. Our research, therefore, brings a new model to reduce congestion so that bot detection systems can be deployed more easily on congested networks. In currently available identification methods for botnets, most of them use deep packet inspection (DPI) to evaluate packets content that is complicated and expensive and ineffective to simulate unidentified payload signatures [127]. It is assumed that the system can access the payload of each packet in DPI. This strategy can be especially accurate when the payload is in decrypted form. Most of the new malicious code generators, however, use the concealment strategies, such as encapsulation of protocols, obstruction and encrypting the payloads [128].

In addition, it is a costly task to evaluate all the packets on the congested networks because the amount of the packets transmitted through the networks are increasing day by day. Consequently, the DPI detection system can suffer from efficiency limited to the processing of traffic from high-speed networks [127]. This study aims to improve efficiency by reducing the amount of packets without hampering the precision rate. To accomplish the ultimate objective of this study, the selection of only TCP control packets proposes a new reduction in traffic for a Botnet identification paradigm. This detection technique is used to reduce the volume of the network traffic and will increase the efficiency of the Botnet detection model.

In this study, TCP traffic control packets are filtered to reduce network traffic volumes and improve the efficiency of the envisaged strategy. Filtering involves mainly two steps: filtering the network flow associated with the TCP protocol and obtaining the control packets SYN, ACK, FIN and RST TCP. Figure 6-4 displays the network traffic reduction process (.PCAP files). An array of TCP Control Packets lists is initialized in the flowchart. New packets (TCP Control Packets List) are added to the array by tuples through the packages until the last package in the file is reached. After the first loop i.e. For loop, the next loop examines the TCP packet header for loop, and the third loop selects packets without payload data. The flowchart's final loop gets the packet header.

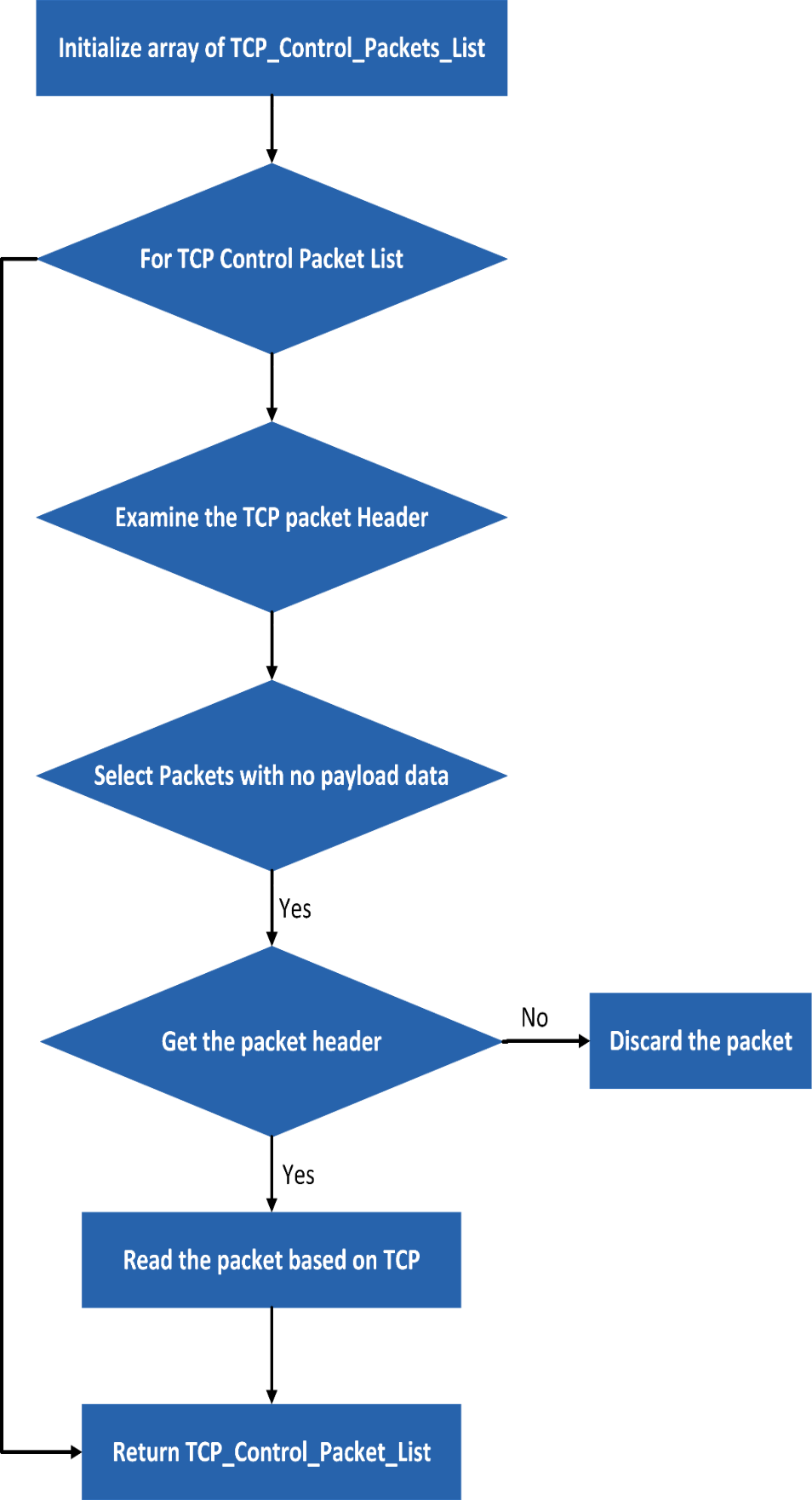


Figure 6-4 Flow chart for network traffic reduction

6.2.3 P2P and Non-P2P Traffic Classification

At present, port identification, signature recognition, and identification are commonly used methods for P2P traffic identification. These methods are based on stream feature [99]. However, the port identification method cannot recognize P2P applications with random ports or custom ports. DPI (Deep Packet Inspection Technology), does not recognize encrypted P2P traffic [36]. Stream-based identification methods can only determine P2P applications of the partial flow and have a high false alarm rate. Therefore, we use a non-P2P well-known port filtering mechanism, DNS query, flow counting rules to filter non-P2P traffic, combined with fast heuristic P2P traffic identification method, as shown in Figure 6-5.

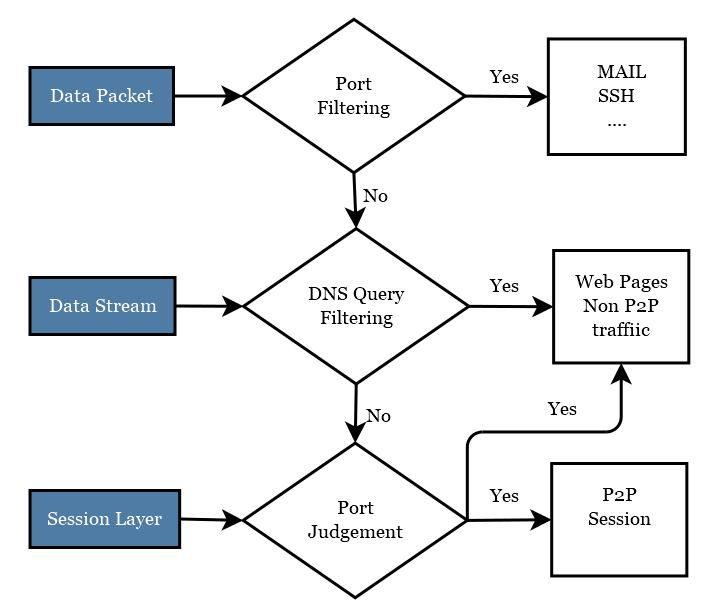


Figure 6-5 First Stage Traffic Classifier

Algorithm 6-1: Second Layer Classification

1. INPUT: Data Packet/ Data Stream/ Session Layer

2. OUTPUT: MAIL/ SSH/ Non-P2P Traffic/ P2P Session

3. Step1: Classify Data Packets by Port-Filtering:

4.      if the data packet is true, then go to MAIL/SSH

5.      else go to step 2.

6. Step2: Classify Data Stream by DNS-Query-Filtering:

7.      if true then go to Non-P2P Traffic

8.      else go to step 3.

9. Step3: Classify Session Layer Port-Judgement:

10.      if true then go to P2P Session or Non-P2P Traffic

11. Step4: **Return**

Port filtering is a packet-level filtering method, mainly filtering the commonly used non-P2P application traffic. DNS query is a stream-level filtering method, flow counting, and port judgment is a session-level filtering method, the two rules are mainly filtered web pages and other non-P2P traffic. Among them, the port-based filtering method can identify some common non-P2P application traffic, such as SSH generally use port 22, Telnet (remote login) use port 23. Commonly used applications and their corresponding port numbers are shown in Table 6-1.

In general, P2P node communication does not require domain name resolution but directly read the IPS list stored in the local configuration file to obtain IP. However, for non-P2P applications, DNS domain name resolution must be used to obtain IP. Therefore, one of the criteria for determining non-P2P network data flows such as Web and Mail etc., is resolved by the domain name and maybe the destination IP address in the network flow.

Table 6-1 Common applications and their corresponding ports

|  |  |  |
| --- | --- | --- |
| Application | | Port Number |
| SSH |  | 22 |
| TelNet |  | 23 |
| MAIL | | 25, 110, 143, 465, 220, 993, 995 |
| NetBios | | 125, 137, 139, 445 |
| Remote | | 3389 |
| FTP | | 20, 21 |
| NTP | | 123 |

When a user sends a Web application service request normally, the Web application uses a multi-port, parallel-requested connection to an IP address on a page. As a result, multiple data streams appear in the same session. The P2P network node communicates each time using a pair of random source and destination ports. Therefore, we can use flow counting and port determination to filter non-P2P traffic. If a session is using the TCP protocol and 80,8080 or 443 port, and the number of sessions in the flow exceeds the threshold, then the session can be considered a web page traffic session. Where the number of valid streams in a session is represented; the threshold is selected based on the number of streams that appear in the normal page access session. Using the capture tool to collect simple and relatively complex web page requests, the analysis results show that the simple web page is generally 3 to 4 connection requests, and the complexity of the page connection request is 5 to 8. Therefore, the threshold is set to 3.

6.2.4 Feature Extraction and Feature Reduction

Reducing the number of characteristics from the features list is the method to remove those functionalities which may marginally affect the classification [129]. Feature reduction helps in reducing the overfitting problem and the problem of data set imbalance. The reliability of reducing the features is, therefore, one of the most significant aspects affecting the precision rate of the detection model.

In the phase of feature extraction, the features are synthesized which are valuable in locating the malevolent behavior of the Bot and these characteristics are obtained in 29 iterator qualities based on 60-s connections. The interpretation of a link is obtained as the subgroup of packets received between two different networks (source port, source IP address, destination port, and destination IP address). All features are obtained explicitly from the control packet header via a thorough check of the payload material of the packets instead of previous approaches. The efficiency is therefore improved, and utilization of the computational resources is reduced, e.g., memory allocation and processing. The feature list was collected from the 60-s connection and consists of 60-s connection function vector.

The purpose of this study is to select an appropriate subset of features that will enhance the efficiency of the machine learning classifiers and reduce model complexity without drastically reducing the precision. We used a classification technique for feature reduction to eliminate less critical features in order to reduce the amount of data, to achieve better levels of accuracy.

The decision tree consists of two node types: two children's internal nodes and children's leaf nodes. Any internal node has a decision mechanism to demonstrate the next node to visit. Training samples containing a number of features begins the construction of the tree. During constructing the tree, the training set is repetitively divided into small subsets. A predicted class is assigned to each resulting node based on the decision matrix of the class distribution in the training set. The internal node test is determined on the basis of an impurity measurement to pick the selected function and threshold values. The most common impurity metric is entropy impurity [130], which we used for feature reduction is shown in Figure 6-6.

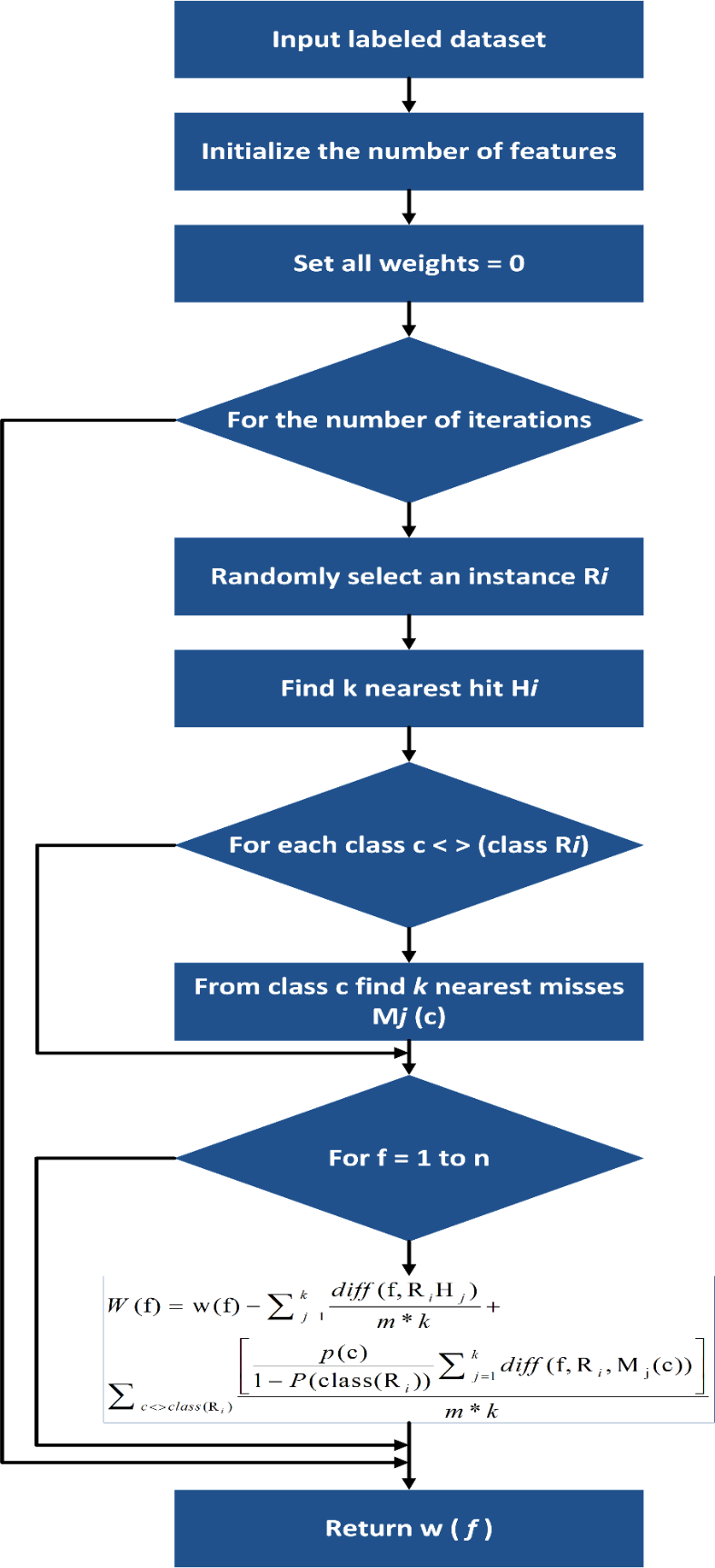


Figure 6-6 Flow chart for Feature Reduction

6.2.5 P2P Traffic Classification

Algorithm 6-2: Final Layer Classification

1. INPUT: Pre-ladled Dataset from the first stage (Non-P2P/ P2P Session)

2. OUTPUT: Normal P2P/ P2P Botnet

3. - **Training Phase:**

4. Step1: Inset Pre-ladled Dataset i.e. Non-P2P Traffic / P2P Traffic

5. Step2: Extract Flows with fine-grained features

6. Step3: Generate Classifier Model

7.     Go to Step 6.

8. - **Classifying Phase:**

9. Step4: Classified P2P Traffic

10. Step5: Get flows with fine-grained features gain the features of the p2p traffic.

11. Step6: P2P Traffic Classifier classify the P2P traffic in two classes as follows:

12.     Normal P2P, P2P Botnet

13. Step7: **Return**

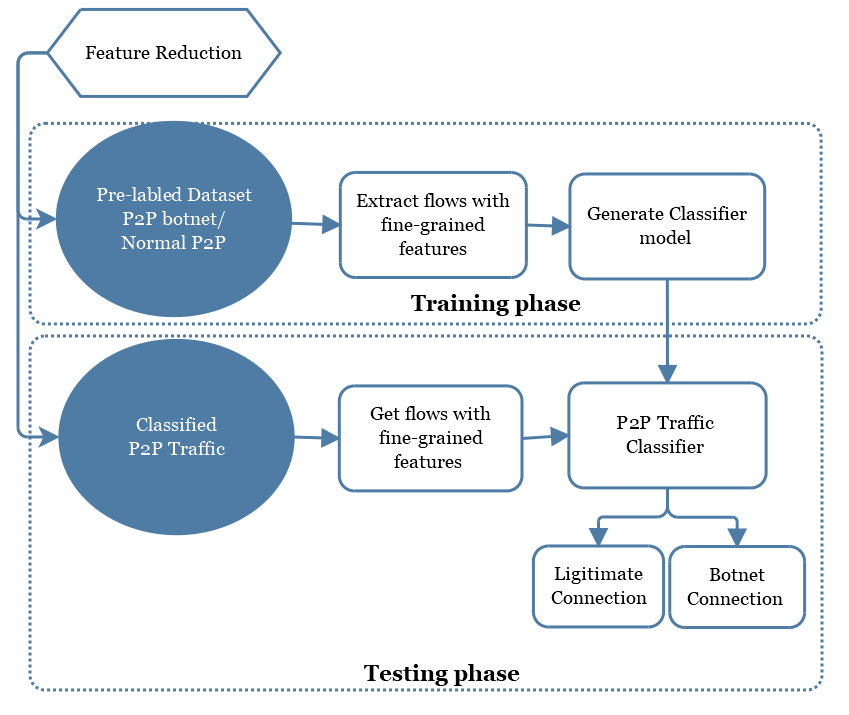


Figure 6-7 Final Layer of Traffic Classifier

Through the analysis of the data flow characteristics of a P2P botnet, the traffic characteristics between the zombie hosts that join the same botnet are similar. Therefore, this paper uses the session-based strategy for feature extraction, that is with the same destination address of the data flow in the same session, reducing the number of stream features and the number of data, thereby improving the detection efficiency. The zombie program automatically completes P2P zombie host and other zombie host communication process, the flow of the duration is generally short and very fixed. Therefore, you can extract the average, maximum, minimum, and standard deviation of the duration of the session, and the average interval of the upstream (downstream) stream packets in the session as a feature.

In the process of communication between two nodes in the P2P botnet, the size and transmission quantity of the transmitted packets are relatively small, and the C & C communication flow generated by the zombie host in the same botnet has great similarity. Therefore, we can distinguish between normal P2P network traffic and P2P botnet traffic by using the distribution of traffic in the session. The average of the maximum packet length of the upstream (downstream) stream in the extraction session, the average of the average packet length, the average of the minimum packet length, the standard deviation of the average packet length, and the average of the number of valid packets, the standard deviation of the number of packets, the average number of bytes transmitted, and the standard deviation of the number of bytes transmitted as a feature. In summary, all the features which were extracted from the session are shown in Table 6-2.

Table 6-2 Session characteristics

|  |  |
| --- | --- |
| Eigenvalues | Description |
| avg\_duration | The mean of the total duration of the different network flows in the same session. |
| std\_duration | The standard deviation of the total duration of the different network flows in the same session. |
| min\_duration | The minimum total duration of the different network flows in the same session. |
| max\_duration | The maximum total duration of the different network flows in the same session. |
| avg\_f(b)int | Average interval of uplink (downstream) packet transmission for different network flows in the same session. |
| max\_f(b)pl | Average value of the maximum value of the uplink transmission packet length for different network flows in the same session. |
| min\_f(b)pl | Average value of the minimum value of the uplink transmission packet length for different network flows in the same session. |
| std\_avg\_f(b)pl | The standard deviation of the average of the length of the packet transmitted in the uplink (downstream) of different network flows in the same session. |
| avg\_f(b)pen | Average number of valid packets transmitted upstream (downstream) of different network flows in the same session. |
| std\_avg\_f(b)pen | The standard deviation of the number of valid packets transmitted upstream (downstream) of different network flows in the same session. |
| avg\_f(b)pb | The average of the total number of bytes transmitted upstream (downstream) of different network flows in the same session. |
| std\_f(b)pb | The standard deviation of the total number of bytes transmitted upstream (downstream) of different network flows in the same session. |

6.3 Evaluating Metrics

We assessed the execution of our methodologies utilizing 10-fold cross-validation. The methodology of k-fold cross-validation. The first example was arbitrarily apportioned into ten equivalent measured sub-tests. Nine sub-tests were utilized for training the model and the remaining one sub-test was held for the testing. The procedure was rehashed ten times, utilizing an alternate sub-sample for testing, every time. The outcomes were then found the average value for the single and final result. All tests were utilized once, for validation. Furthermore, we used the wrapper method for feature selection.

6.3.1 The accuracy of Detection Model

The percentage of correctly classified instances among the total number of instances.



6.3.2 False Alarm Rate

FP False positive rate—the rate of P2P recognized incorrectly.

FN False negative rate—the rate of Normal P2P recognized incorrectly as botnets.

Recall rate is calculated by the following formula with a given number of true positives and false negatives.



The TPR is referred to as “sensitivity” or the “true positive rate” sometimes. Precision is calculated by the following formula which is also known as “positive predictive rate”:



6.4 Dataset and Experimental setup

To ensure the reliability and scalability of our proposed model, we trained our model with network flow data from a diverse variety of sources. The dataset includes different types of botnets tested in different kinds of environmental setup. We set a *VMWare* virtual environment in Windows 10 and Linux operating systems. *Nfdump* was set up on the system to collect network data. *Nfdump* captures network flow and stores into *nfcapd* files. One instance of the *nfcapd* file is associated with a flow data record over time. We used *Wireshark* in a window environment to capture the network flow because *Wireshark* is an open source software which is available free of cost. Figure 6-8 and Figure 6-9 gives a brief overview of the network capture. The advantage of utilizing *nfcapd* in a Linux domain is that it records countless highlights of the network traffic which turn out to be very favorable in further investigation. It likewise runs discreetly out of sight utilizing insignificant handling memory and power, thus an ideal decision as a tool to gather information. Description of the famous datasets which we used in our experiments are discussed below.

CTU-13 Dataset [116][117]: We used the dataset of the CTU-13 project to do experiments because it contains thirteen various captures of different botnet samples, i.e., IRC, SPAM, CF (Click Fraud), DDoS, FF (FastFlux), PS (Port Scan), US (Compiled and Controlled by us), HTTP. The capture files are stored in the *pcap* form. The dataset of CTU-13 project is a labeled dataset with background traffic, botnet and normal.

ISOT [131]: The ISOT dataset includes both malicious (Storm and Zeus botnets) and non-malicious traffic (HTTP traffic, gaming packets, and P2P applications, e.g., BitTorrent ). The Stratosphere dataset includes botnet traffic with port scanning, C & C communication, and attack.

Table 6-3 Packet counts of the different captured scenarios

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario #** | **Name** | **Total Packets** | **Displayed** |
| 1 | Neris | 323154 | 100 % |
| 2 | Neris | 176064 | 100 % |
| 3 | Rbot | 495056 | 100 % |
| 4 | Rbot-dos | 256712 | 100 % |
| 5 | Fast-flux | 45853 | 100 % |
| 6 | Donbot | 24764 | 100 % |
| 7 | Sogou | 20663 | 100 % |
| 8 | Qvod | 85735 | 100 % |
| 9 | Bot | 2129949 | 100 % |
| 10 | Bot | 66340518 | 100 % |
| 11 | Bot-2 | 3941769 | 100 % |
| 12 | Bot | 352266 | 100 % |
| 13 | Fast-flux-2 | 440625 | 100 % |
| 14 | ISOT | 371899 | 100 % |
| 15 | Benign | 14822 | 100 % |

We have downloaded non-malicious packets to combine with the other two datasets. Figure 6-8 and Figure 6-9 are the sample screenshot of the running *Wireshark* environment which records the network traffic.

*Wireshark* was used in Windows 10 environment to obtain the CSV files to analyses the data for further experiments because CSV format is compatible with most of the software tools, e.g., python, Weka, java. The values in the CSV file were separated with commas. The CSV data was split into two parts i.e. training and testing. Table 6-3 shows the detailed view of the packet counts which were captured in different scenarios. It was observed that the packets were displayed 100 % for each scenario which means that no packet drop was seen during capturing the packets.

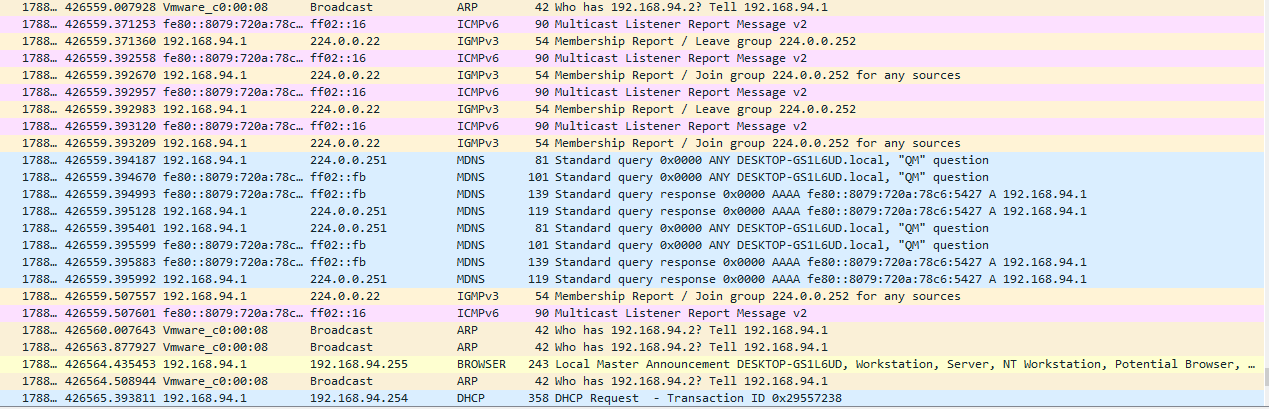


Figure 6-8 Network traffic capture with Wireshark

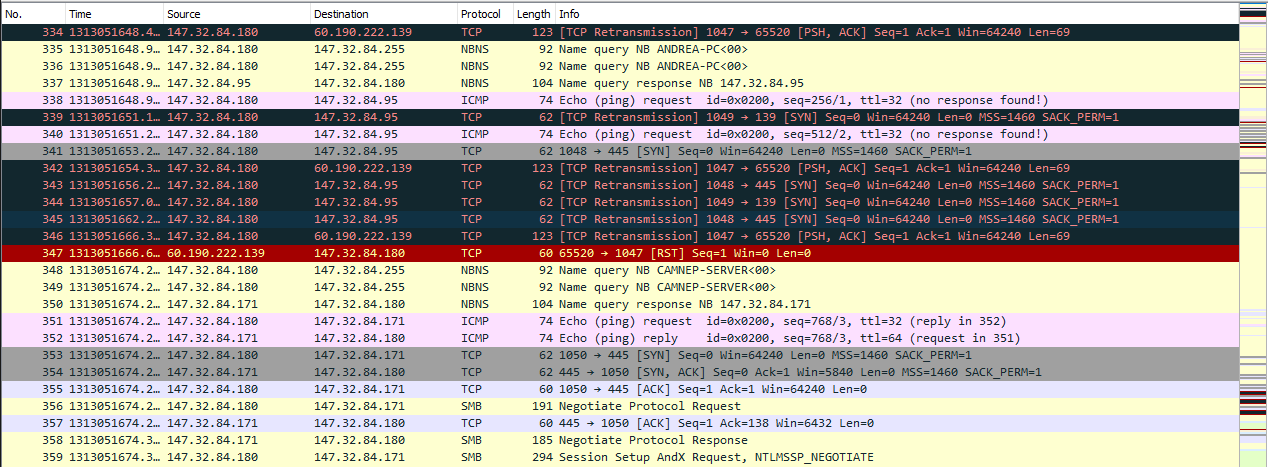


Figure 6-9 Network traffic capture with Wireshark

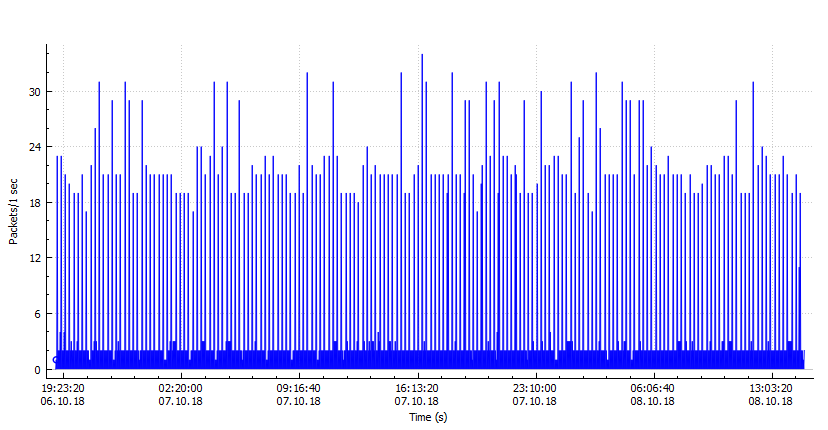


Figure 6-10 Benign Network Traffic Capture

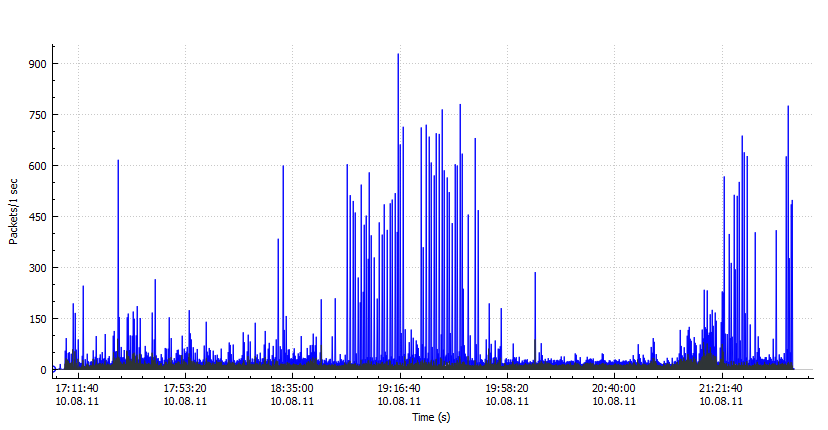


Figure 6-11 Neris Botnet Capture

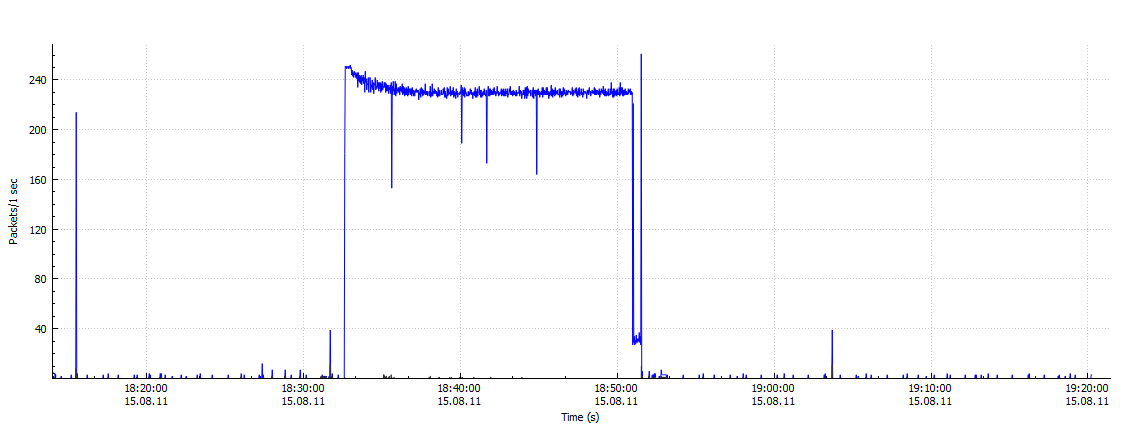


Figure 6-12 RBot Botnet Capture

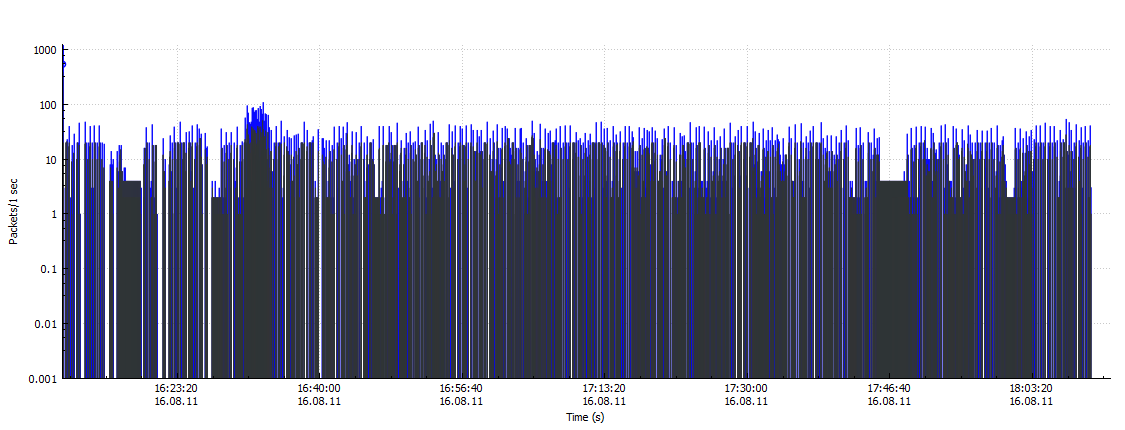


Figure 6-13 Donbot Botnet Capture

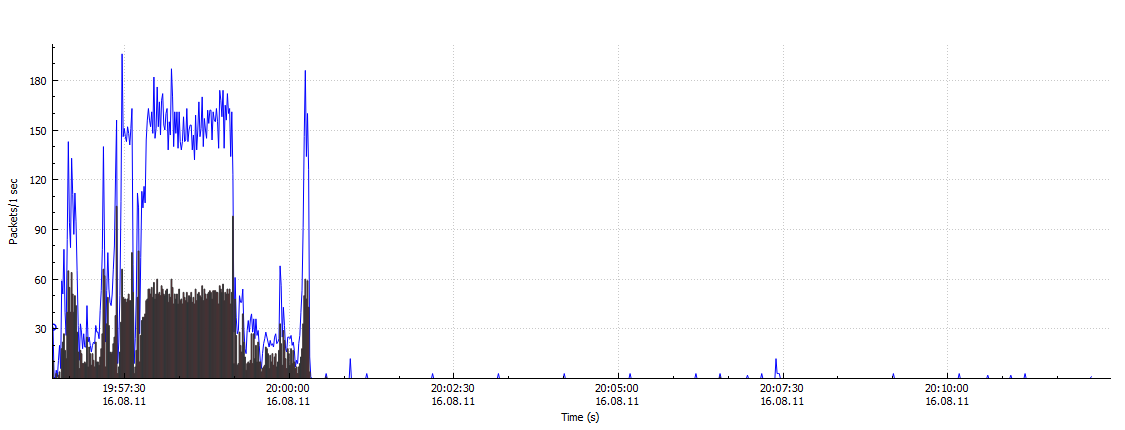


Figure 6-14 Sogou Botnet Capture

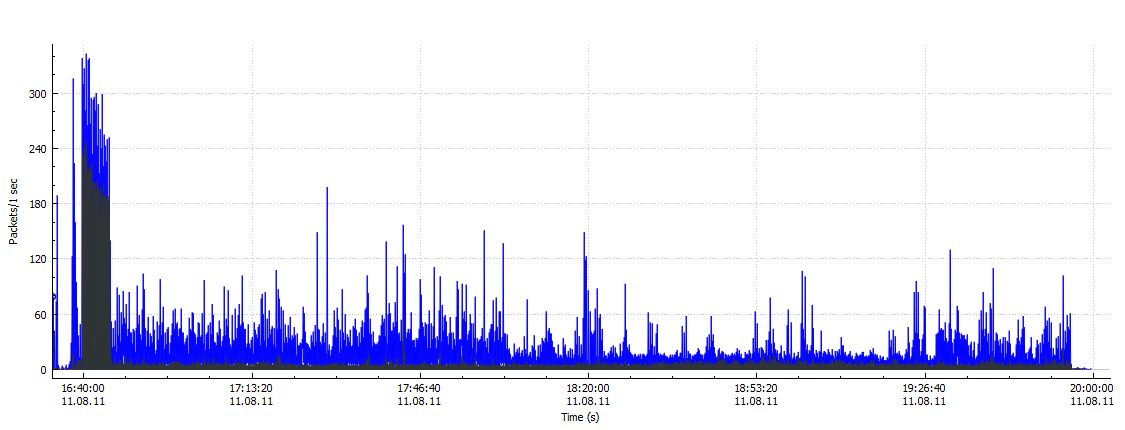


Figure 6-15 Nerris-2 Botnet Capture

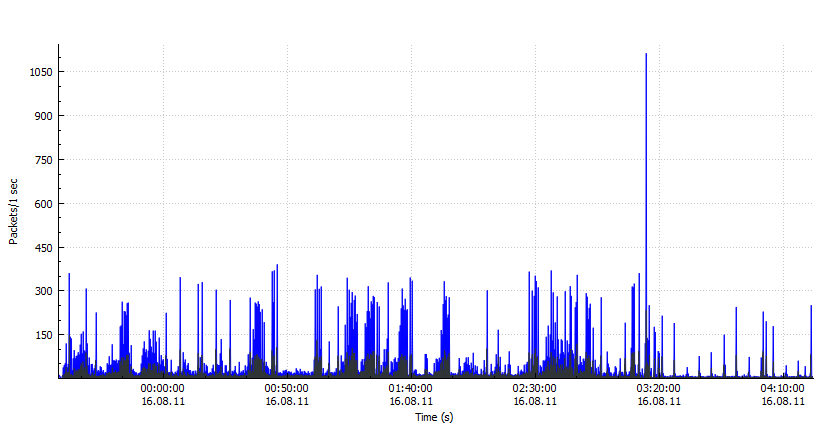


Figure 6-16 Fast-Flux Botnet Capture

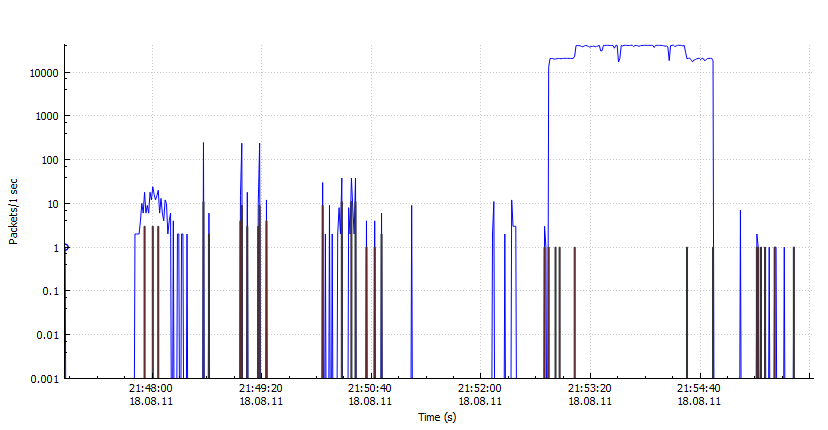


Figure 6-17 Bot-2 Botnet Capture

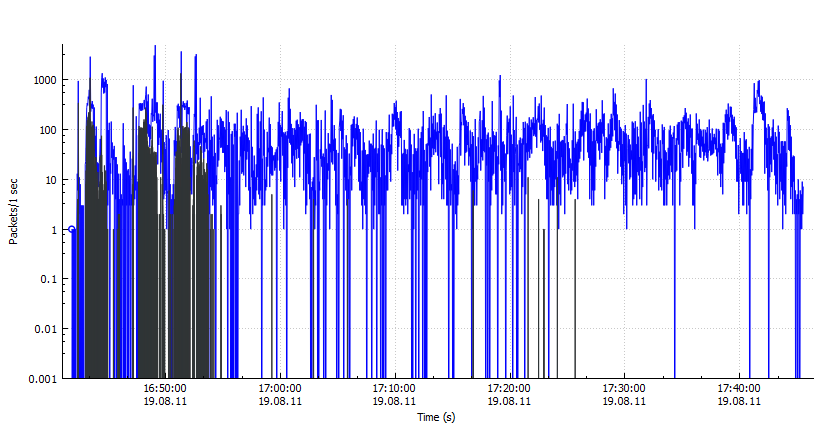


Figure 6-18 Bot-3 Botnet Capture

Figure 6-10 is the simulation of benign traffic while Figure 6-11, Figure 6-12, Figure 6-13, Figure 6-14, Figure 6-15, Figure 6-16, Figure 6-17, Figure 6-18 are the sample simulations of *Neris* botnets, *Rbot* botnets *Donbot* botnets, *Sogou* botnets Fast-Flux and other types of botnets. The *X-axis* of the graphs show the time intervals while *Y-axis* show the packet lengths per second. Blue lining in the graphs show the capture of all benign traffic while black color shows the “Bad TCP capture” or error filter catch. “Bad TCP” capture indicates of something wrong with the TCP communication in the capture file so it was observed in each and every graph that there were particular botnets in the traffic while Figure 6-10 indicates benign traffic in the network.

6.5 Detection Results and Discussion

The detection of flow based on the general use of a single type of botnet dataset as experimental data, and the use of 10-fold cross-validation strategy to analyze the classification accuracy. Therefore, this paper uses the method of 10-fold cross-validation to compare and analyze the performance of the filter-less session and the filtering feature after classification. The detection rates of the botnet are 99%, 98.9%, 98.7% and so on. The false alarm rate of normal traffic is 9%, 7.6%, 3% for thresholds 1, 2 and 3. The false alarm rate of normal traffic for thresholds 4, 5, 6, 7 and 8 is 1%. The threshold is a minimum or maximum value (developed for a characteristic, feature or variable) that serves as a benchmark for comparison or instruction and which may require a complete system review. The recognition rate of Web traffic is related to the set of thresholds, flow counting, and port identification. In this experiment, we use different thresholds (1, 2, 3, 4, 5, 6, 7, 8) to identify Web traffic. The results are shown in Figure 6-19, i.e., when the threshold reaches 3, the detection rate drops rapidly with the increase of the threshold, and the false alarm rate does not change. However, when the threshold is too small which is 1 in our case, so the false alarm rate will increase, and the detection rate does not change much. Therefore, according to the test results, we set the threshold 3 in the entire P2P traffic detection process. The results show that the classification of well-known port, DNS, and flow counting filtering is not only better than P2P without filtering, but also the false alarm rate of normal traffic is further reduced.

In general, the classification accuracy of the Decision Tree algorithm is positively correlated with the number of classification trees and the classification tree depth. However, the detection rate of the Decision Tree algorithm is negatively correlated with the number of classification trees. In order to determine the stochastic forest model used to identify P2P botnet traffic in a high-speed network environment, this paper makes a comparative analysis of the influence of a different number of the classification tree and depth of the classification tree on the basis of the detection rate. The number of the classification tree is set from 0 ~ 300, the depth of the classification tree is set from 0 ~ 50.

Based on the session characteristics, this paper uses a Decision Tree classification algorithm to classify Normal P2P and botnet traffic. The depth of the Decision Tree algorithm was set to 8, and the classification tree was set to 100, shown in Figure 6-19. The Decision Tree detection rate is 98.7% for threshold 3 of the false positive rate, while other machine learning P2P botnet traffic detection results are shown in Table 6-4 and Figure 6-20.

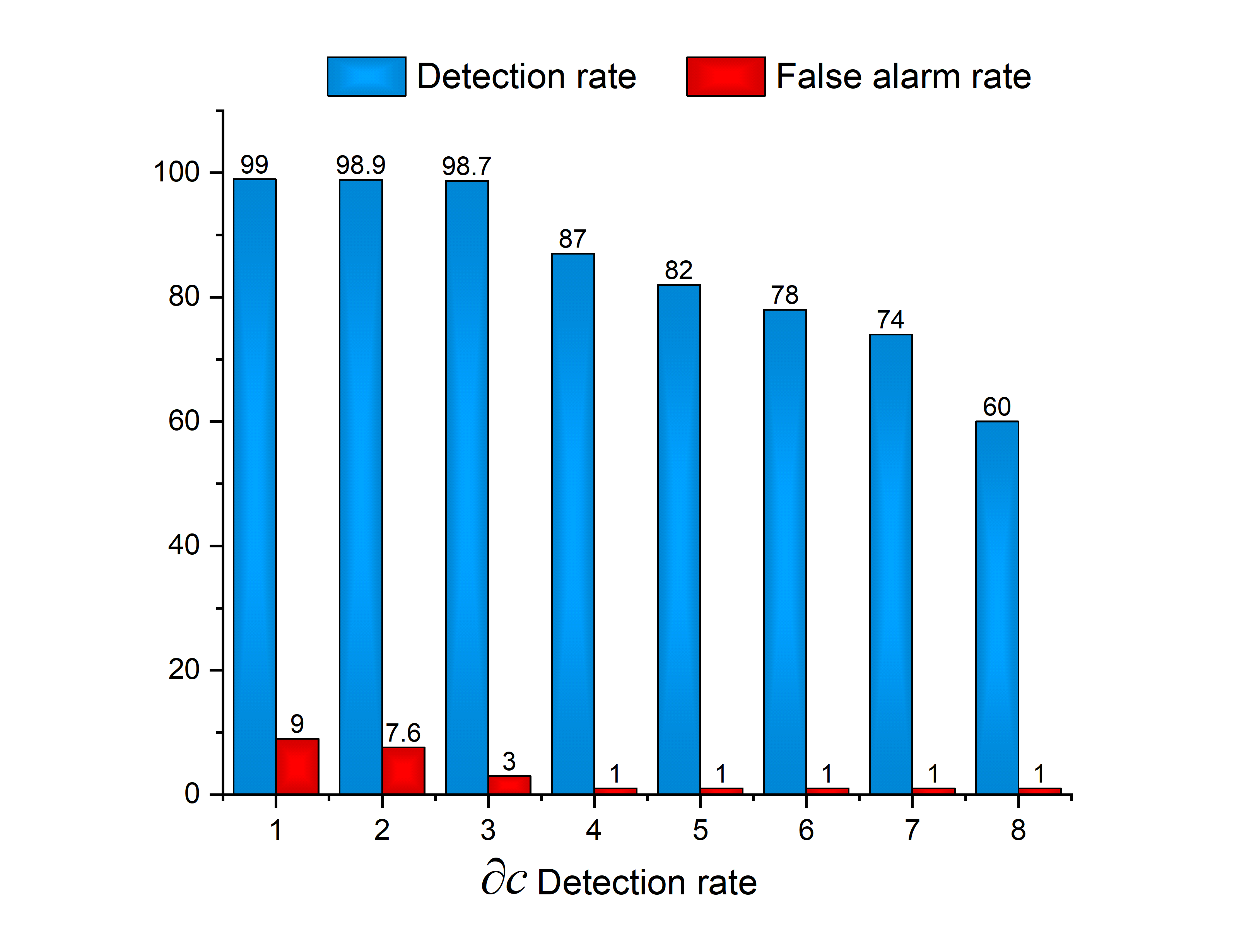


Figure 6-19 Web traffic identification under different thresholds during the training process

6.6 Comparison with other Classifiers

Comparison of the different approaches for botnet identification is a difficult task because different evaluations and experiments use different botnet samples and data sets. We have compared our proposed model with other detection models based on precision and false positive rates. We have compared five machine learning models and two other models previously published. Table 6-4 demonstrates the comparison of our results with other machine learning classifiers and published work on the bases of the network flow analysis.

Table 6-4 Comparison with other published work and other machine learning classifiers

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario #** | Approaches | **FAR** | **Accuracy %** |
| 1 | Wen-Hwa et al. [132] | 0 | 98 % |
| 2 | Fedynyshyn et al. [133] | 7.8 | 92.9 % |
| 3 | Naive Bayes classification algorithm | 3 | 75 % |
| 4 | Logistic Regression algorithm | 3 | 93.8 % |
| 5 | Artificial Neural Network (ANN) | 3 | 93.2 % |
| 6 | K-Nearest Neighbor (KNN) | 3 | 93.9 % |
| 7 | Our Proposed Model | 3 | 98.7 % |

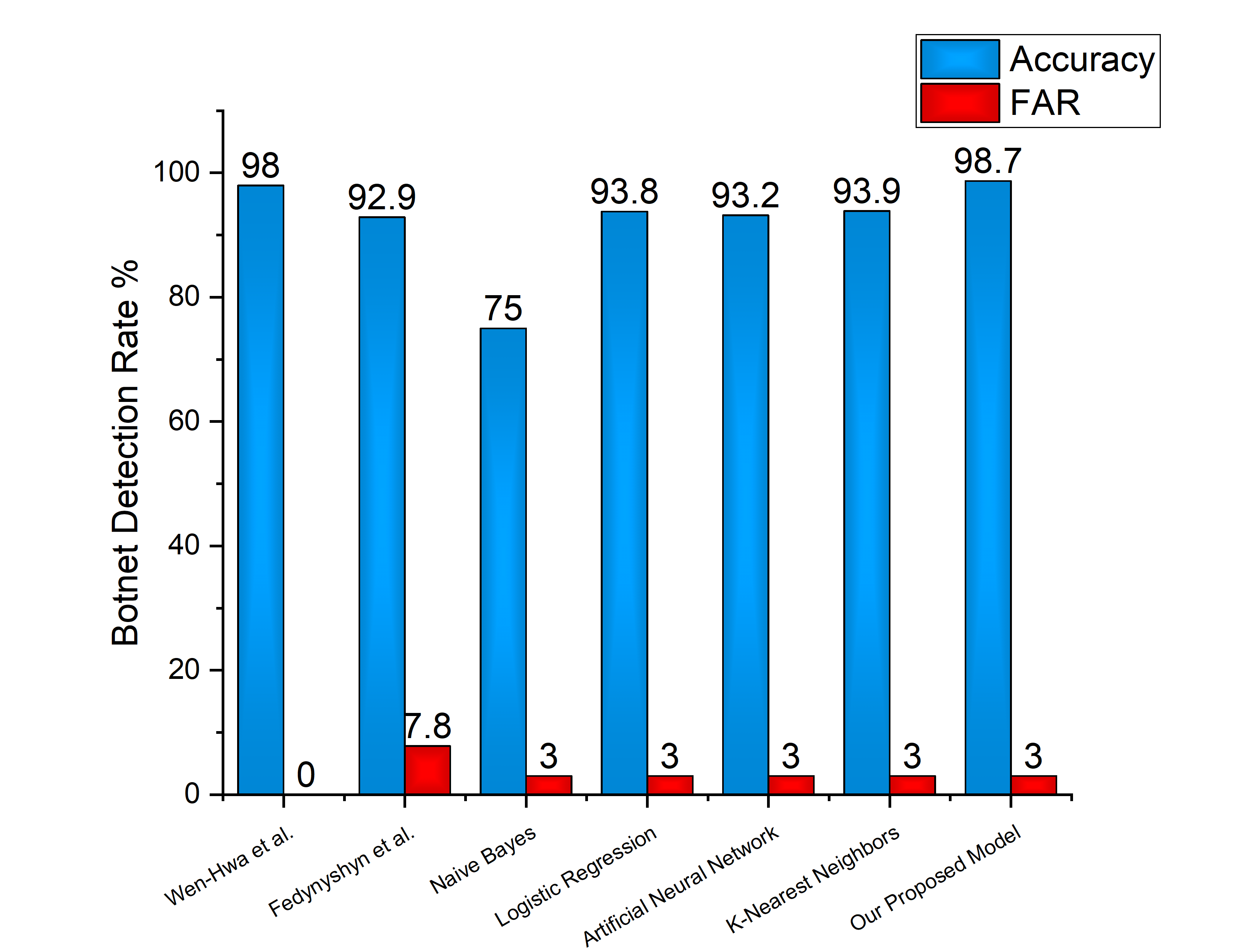


Figure 6-20 Comparison with other machine learning classifiers on P2P botnet detection

The compared algorithms are based on session characteristics to detect P2P botnet traffic, the Decision Tree algorithm shows a high accuracy. The Decision Tree algorithm uses the binary tree as a classification tree. The principle of each classification tree is recursively from top to bottom, and its training set is obtained by returning the original training data set. In order to minimize the occurrence of the fitting phenomenon, the Decision Tree uses the Bagging random sampling method to construct the classification tree. Therefore, this paper uses the Decision Tree classification algorithm for high-speed network environment P2P botnet traffic detection.

Using Naive Bayes classification algorithm, Logistic Regression, Artificial Neural Network, and K-Nearest Neighbor, the detection rate was 75.5% and 93.8%, 93.2% and 93.9%, respectively, but the results of Decision Tree algorithm were noted as high as 98.7%. Therefore, the Decision Tree algorithm for various types of P2P botnet traffic detection is more accurate than the other four classification algorithms.

6.7 Summary

This chapter of the dissertation proposes a multi-layer technique for P2P botnet detection which is again a hybrid technique. P2P traffic was filtered in terms of packets, streams and sessions level. As the network traffic is very huge in high-speed networks, so we reduce the traffic by deploying a filtering model. Then, P2P botnet classifiers were used to classify the Normal P2P communication and P2P botnet based on session features. This study combines the advantages of two different detection strategies, i.e., the traffic behavior-based detection and the network flow similarity-based detection. The validity of the proposed method is verified by using the open sourced published data set and collected dataset by setting up the network environment in the lab of cybersecurity at UESTC. It was noted from the experimental results, that multi-layer technique can effectively detect P2P botnets. We evaluated the model by comparing the five different classifiers, and two other previously published models and noted that our proposed model has a higher accuracy than others.

Chapter 7 Conclusion and Future Work

In this chapter, conclusions on the various algorithms proposed in this dissertation are first presented. This is followed by pointers to possible future research directions in relation to the current work.

7.1 Conclusion

This dissertation has presented a series of techniques, examples, issues, and topics within the area of malware detection. Some novel classification schemes have been proposed for malware detection techniques. The dissertation presented experiments in both signature-based and anomaly-based detection methods for malware detection. This dissertation investigated and studied the use of machine learning and artificial neural network configurations to simulate malicious code and malevolent behaviors by categorizing their samples in their malware families and describing, analyzing and transferring malicious code to images. In the case of supervised learning, the main concepts of feature and classification are asserted. In order to validate the proposed approaches, numerical analyses are presented in chapter 3, chapter 4, chapter 5 and chapter 6.

In chapter 3, Machine learning models conducted well on Microsoft dataset at a very high speed of training, also the most famous models i.e., ResNet and GoogleNet are analyzed in terms of speed and accuracy. Although a few effective solutions have been established to address the issues of malicious code detection and undiscovered attacks related to online activities, it is anticipated that newer versions of comparable malicious code will be more advanced and harmful.

The application of the convolutional neural network algorithm in intrusion detection is a new idea. Chapter 4, in the dissertation, proposed a novel technique that combines convolutional neural network and SoftMax algorithms. The experimental evaluations demonstrated that this model could improve the accuracy of human intrusion detection and improve the performance of human invading detection system. It is observed in the results that the accuracy is increased when we increase the number of epochs. It is also observed that the proposed model performed better as compare to SVM and DBN models.

In chapter 5, a hybrid technique for P2P botnet detection is proposed based on session features. Firstly, non-P2P traffic was filtered from packet, stream and session level respectively. Then, P2P botnet classifiers were used to classify the Normal P2P communication and P2P botnet based on session features. Chapter 5 combines the advantages of the detection technique based on Flow Similarity of the network traffic. It is noted from the experimental results, that two-stage technique can effectively detect P2P botnet traffic.

Contagion media is willing to switch from computers to mobile phone terminals, as being the most commonly used systems tend to attract intruders and attackers. Newly cloud-based interconnected systems are also easier targets and must be adequately aimed at preventing malware from penetrating them. In addition, the effective implementation of auditing and filtering, which is unlikely due to the diversity of the Internet and the lack of economic inducements for subscribers and ISPs to safeguard devices and sites, would be a way to address the issue of malicious code detection and to be more certain about the botnet detection problem.

Chapter 6 of this dissertation proposes a multi-layer technique for P2P botnet detection which is again a hybrid technique. P2P traffic was filtered in terms of packets, streams and sessions level as the network traffic is very huge in high-speed networks, so we reduce the traffic by deploying a filtering model. Then, P2P botnet classifiers were used to classify the Normal P2P communication and P2P botnet based on session features. This study combines the advantages of two different detection strategies, i.e., the traffic behavior-based detection and the network flow similarity-based detection. The validity of the proposed method is verified by using the open sourced published data set and collected dataset by setting up the network environment in the lab of cybersecurity at UESTC. It was noted from the experimental results, that multi-layer technique can effectively detect P2P botnets. We evaluated the model by comparing the five different classifiers, and two other previously published models and noted that our proposed model has a higher accuracy than others.

In all the proposed models, machine learning and deep learning classifications schemes are used. Compared with other algorithms, the proposed schemes (chapters 4, 5 and 6) demonstrate better efficiencies which make them attractive candidates for malware detection in diverse fields. These have demonstrated that though there are some weaknesses associated with the low-dimensional systems, improving their statistical properties and diffusion functions make them more efficient.

7.2 Future Work

The scope of this dissertation has been quite broad that the generic algorithms applicable to malware detection have been proposed. All algorithms have been designed to be applicable to malware detection, intrusion detection and botnet detection in diverse fields.

While the algorithm proposed in chapter 3 yielded good outcomes, it is clear from the experiment that the scheme demonstrates some weakness in terms of robustness against differential analysis. It is shown in this works that how a small change in the image could lead to miss-classification of images and how a small change in the image could lead to an effective classification. The algorithm can be improved by the introduction of a mechanism to produce good quality of images from EXE files. The algorithm is currently implemented via a serial program. The execution time of this technique can be greatly improved when performed with a parallel approach. For instance, using a parallel computing platform such as the Compute Unified Device Architecture (CUDA) may improve the performance since algebraic operations between pain image and encoded image matrices could be put into different threads for parallel execution on a Graphics Processing Unit (GPU). There are many real-world challenges. First, correctly disassembling malicious code is an open problem. Secondly, code packing (e.g., UPX) and embedded virtualizer (e.g., Themida and VMProtect) should be investigated for feature extraction.

Although the algorithm proposed in chapter 4 (An Improved CNN model for Intrusion Detection), has demonstrated robustness against various forms of attacks, good statistical properties, and good execution speed, particularly as compared to some other algorithms in literature, it is still possible to improve upon the computing cost in terms of execution time, reduce the false alarm rate and memory resource.

A hybrid botnet detection scheme is proposed in chapter 5. This algorithm demonstrates high speed and is robust against various forms of attacks. As such, it could be combined with another scheme to provide comprehensive security to networks particularly in time-critical fields such as high-speed networks as shown in chapter 6. The botnet detection technique proposed in chapter 6 can be applied to IoT environments to integrate tracking functionalities. Besides, a seamless mechanism that captures and embeds workflows in situations where the traffic goes through a number of hands, for instance in cloud environments can be considered for the purposes of security, authentication, and traceability. The ability to show newly discovered behavior during the detection phase of a scheme is not zero. The likelihood of a method based on an anomaly raising a false positive is indeed not zero. It is an open computer science major issue to develop better approaches to the appropriate behavior of a computer system. Furthermore, some fields require specific constraints that need special or dedicated algorithms in which these constraints are factored. For instance, during the network traffic capture, the encrypted protocols might not be captured; so there is a lot to be done for capturing the encrypted packets properly and to reduce the false alarm rate.

Acknowledgments

References

[1] Symantec. Internet Security Threat Report[S].2018, 23:1–89.

[2] AV-TEST. Malware Statistics & Trends Report | AV-TEST[OL].2019.

[3] D. Gavrilut,M. Cimpoesu,D. Anton,et al. Malware detection using machine learning [C].2009 International Multiconference on Computer Science and Information Technology, 2009, 735–741.

[4] C. Cao,X. Liu,Y. Yang,et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks [J].Proceedings of the IEEE International Conference on Computer Vision, 2016, 11–18–Dece:2956–2964.

[5] A. Nisioti,A. Mylonas,P. D. Yoo,et al. From Intrusion Detection to Attacker Attribution: A Comprehensive Survey of Unsupervised Methods [J].IEEE Communications Surveys & Tutorials, 2018, 20(4):3369–3388.

[6] A. Arora,S. Yadav,… K. S. R. on N. F.,et al.Handbook of Research on Network Forensics and Analysis Techniques, 2018, 117–141.

[7] J. Zhang,Y. Xie,F. Yu,et al. Intention and Origination: An Inside Look at Large-Scale Bot Queries. [J].NDSS, 2013, 1–16.

[8] V. Roto,A. Oulasvirta,T. Haikarainen,et al. You Are a Game Bot!: Uncovering Game Bots in MMORPGs via Self-similarity in the Wild [J].Ndss, 2016, (February):1–19.

[9] F. Chen,S. Ranjan,and P.-N. Tan. Detecting bots via incremental LS-SVM learning with dynamic feature adaptation [C].Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’11, 2011, 386.

[10] T. Krueger,H. Gascon,N. Krämer,et al. Learning stateful models for network honeypots [C].Proceedings of the 5th ACM workshop on Security and artificial intelligence - AISec ’12, 2012, 37.

[11] R. U. Khan,X. Zhang,and R. Kumar. Analysis of ResNet and GoogleNet models for malware detection [J].Journal of Computer Virology and Hacking Techniques, 2018.

[12] DAN GOODIN. Malware found preinstalled on 38 Android phones used by 2 companies | Ars Technica[OL].03-Nov-2017.

[13] D. Kiwia,A. Dehghantanha,K.-K. R. Choo,et al. A cyber kill chain based taxonomy of banking Trojans for evolutionary computational intelligence [J].Journal of Computational Science, 2018, 27:394–409.

[14] A. Nourian and S. Madnick. A Systems Theoretic Approach to the Security Threats in Cyber Physical Systems Applied to Stuxnet [J].IEEE Transactions on Dependable and Secure Computing, 2018, 15(1):2–13.

[15] X. Zhou,Z. Xu,L. Wang,et al. APT Attack Analysis in SCADA Systems [J].matec-conferences.org.

[16] Grim,A. Jonathan,R. Irfan,et al. Automatic Mitigation of Kernel Rootkits in Cloud Environments [C].International Workshop on Information Security Applications, 2017, 137–149.

[17] A. Matrosov,S. Bratus,and E. Rodionov. Rootkits and bootkits reversing modern malware and next generation threats [J].2019.

[18] A. Wang,W. Chang,and S. Chen. Delving into internet DDoS attacks by botnets: characterization and analysis [J].IEEE/ACM Transactions on Networking (TON), 2018, 26(6):2843–2855.

[19] B. Tran,J. Li,M. Jerry,et al. Spectral Signatures in Backdoor Attacks [C].Advances in Neural Information Processing Systems, 2018, 8011–8021.

[20] E. Dauterman,H. Corrigan-Gibbs,D. Mazières,et al. True2F: Backdoor-resistant authentication tokens [J].arXiv preprint arXiv:1810.04660, 2018, 1–23.

[21] A. Kharraz,W. Robertson,and E. K. Privacy. Protecting against Ransomware: A New Line of Research or Restating Classic Ideas? [J].IEEE Security & Privacy, 2018, 16(3):103–107.

[22] D. J. Quinlan,K. Adams,Langton,et al. Identifying malware based on a relationship between a downloader file and a downloaded file[P].08-Aug-2017, 1–10.

[23] J. A. Kraemer. System and Method for Reverse Command Shell Detection[P].Google Patents, 2018.

[24] M. Kumar,B. I.-I. J. of Applied,and U. 2018. A Study on Web Hijacking Techniques and Browser Attacks [J].ripublication.com.

[25] P. Black,I. Gondal,and R. L. Security. A survey of similarities in banking malware behaviours [J].Computers & Security, 2018, 77:756–772.

[26] A. Zimba,Z. Wang,and M. C. Information. Addressing Crypto-Ransomware Attacks: Before You Decide whether To-Pay or Not-To [J].Journal of Computer Information Systems, 2019, 1–11.

[27] M. Chatterjee,M. Namin,and S. Akbar. Detecting Web Spams Using Evidence Theory [C].IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2018, 695–700.

[28] R. Zhang,X. Chen,J. Lu,et al. Using AI to Hack IA: A New Stealthy Spyware Against Voice Assistance Functions in Smart Phones [J].arXiv preprint arXiv:1805.06187, 2018, 1–11.

[29] T. Urban,D. Tatang,T. Holz,et al.2018, 449–469.

[30] V. Stavova,L. Dedkova,V. Matyas,et al. Experimental large-scale review of attractors for detection of potentially unwanted applications [J].Computers & Security, 2018, 76:92–100.

[31] N. Weaver,V. Paxson,S. Staniford,et al. A taxonomy of computer worms [C].Proceedings of the 2003 ACM workshop on Rapid malcode, 2003, 11–18.

[32] W. Li,K. Wang,and S. Stolfo. Fileprints: Identifying file types by n-gram analysis [C].Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop, 2005, 64–71.

[33] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection [C].International Workshop on Recent Advances in Intrusion Detection, 2004, 203–222.

[34] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. [C].USENIX Security Symposium, 1998, 79–93.

[35] M. Boldt and B. Carlsson. Analysing privacy-invasive software using computer forensic methods [J].ICSEA, Papeetee, 2006, 1–17.

[36] Y. Wang,D. Beck,B. Vo,et al. Detecting stealth software with strider ghostbuster [J].Dependable Systems, 2005.

[37] F. Zareen and R. Karam. Detecting RTL Trojans using Artificial Immune Systems and High Level Behavior Classification [C].2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2018, 68–73.

[38] B. A. S. Al-rimy,M. A. Maarof,and S. Z. M. Shaid. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions [J].Computers & Security, 2018, 74:144–166.

[39] A. H. Sung,J. Xu,P. Chavez,et al. Static Analyzer of Vicious Executables (SAVE) [C].20th Annual Computer Security Applications Conference, 326–334.

[40] M. Christodorescu,S. Jha,and S. Seshia. Semantics-aware malware detection [J].Privacy, 2005 IEEE …, 2005.

[41] S. Kumar and E. H. Spafford. A generic virus scanner for C++ - IEEE Conference Publication [C].

[42] K. Ilgun,R. Kemmerer,and P. Porras. State transition analysis: A rule-based intrusion detection approach [J].IEEE transactions on, 1995.

[43] Y.-L. Wan,J.-C. Chang,R.-J. Chen,et al. Feature-Selection-Based Ransomware Detection with Machine Learning of Data Analysis [C].2018 3rd International Conference on Computer and Communication Systems (ICCCS), 2018, 85–88.

[44] J. C. Rabek,R. I. Khazan,S. M. Lewandowski,et al. Detection of injected, dynamically generated, and obfuscated malicious code [C].Proceedings of the 2003 ACM workshop on Rapid Malcode - WORM’03, 2003, 76–82.

[45] F. Castaneda,E. Sezer,and J. Xu. WORM vs. WORM: preliminary study of an active counter-attack mechanism [C].Proceedings of the 2004 ACM workshop on Rapid Malcode, 2004, 83–93.

[46] A. Souri,M. Norouzi,and P. Asghari. An analytical automated refinement approach for structural modeling large-scale codes using reverse engineering [J].International Journal of Information Technology, 2017, 9(4):329–333.

[47] A. Souri,N. J. Navimipour,and A. M. Rahmani. Formal verification approaches and standards in the cloud computing: A comprehensive and systematic review [J].Computer Standards & Interfaces, 2018, 58:1–22.

[48] H. Hashemi,A. Azmoodeh,A. Hamzeh,et al. Graph embedding as a new approach for unknown malware detection [J].Journal of Computer Virology and Hacking Techniques, 2017, 13(3):153–166.

[49] S. D. Nikolopoulos and I. Polenakis. A graph-based model for malware detection and classification using system-call groups [J].Journal of Computer Virology and Hacking Techniques, 2017, 13(1):29–46.

[50] R. Kumar,Z. Xiaosong,R. U. Khan,et al. Effective and Explainable Detection of Android Malware Based on Machine Learning Algorithms [C].Proceedings of the 2018 International Conference on Computing and Artificial Intelligence - ICCAI 2018, 2018, 35–40.

[51] Y. Ye,L. Chen,S. Hou,et al. DeepAM: a heterogeneous deep learning framework for intelligent malware detection [J].Knowledge and Information Systems, 2018, 54(2):265–285.

[52] A. Pektaş and T. Acarman. Classification of malware families based on runtime behaviors [J].Journal of Information Security and Applications, 2017, 37:91–100.

[53] M. Damshenas,A. Dehghantanha,K.-K. R. Choo,et al. M0Droid: An Android Behavioral-Based Malware Detection Model [J].Journal of Information Privacy and Security, 2015, 11(3):141–157.

[54] N. Milosevic,A. Dehghantanha,and K. K. R. Choo. Machine learning aided Android malware classification [J].Computers and Electrical Engineering, 2017, 61:266–274.

[55] H.-T. Lee,D. Kim,M. Park,et al. Protecting data on android platform against privilege escalation attack [J].International Journal of Computer Mathematics, 2016, 93(2):401–414.

[56] M. Siddiqui,M. C. Wang,and J. Lee. Detecting internet worms using data mining techniques [J].Journal of Systemics, Cybernetics and Informatics, 2009, 6(6):48–53.

[57] M. Egele,T. Scholte,E. Kirda,et al. A survey on automated dynamic malware-analysis techniques and tools [J].ACM computing surveys (CSUR), 2012, 44(2):1–42.

[58] L. Nataraj,V. Yegneswaran,P. Porras,et al. A Comparative Assessment of Malware Classification Using Binary Texture Analysis and Dynamic Analysis [J].Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, 2011, 21–30.

[59] D. Kong and G. Yan. Discriminant malware distance learning on structural information for automated malware classification [C].Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013, 1357–1365.

[60] R. Tian,L. M. Batten,and S. C. Versteeg. Function length as a tool for malware classification [J].3rd International Conference on Malicious and Unwanted Software, MALWARE 2008, 2008, 69–76.

[61] M. Hall,F. E. Holmes,G. Holmes,et al. The WEKA data mining software: An update [J].SIGKDD Explorations, 2009, 11 (1).

[62] I. Santos,C. Laorden,and P. G. Bringas. Collective classification for unknown malware detection [C].IEEE International Conference on Security and Cryptography (SECRYPT), 2011, 251–256.

[63] I. Santos,J. Devesa,F. Brezo,et al. Opem: A static-dynamic approach for machine-learning-based malware detection [C].International Joint Conference CISIS’12-ICEUTE 12-SOCO, 2013, 271–280.

[64] M. F. Zolkipli and A. Jantan. An approach for malware behavior identification and classification [C].3rd International Conference on Computer Research and Development (ICCRD), 2011, 191–194.

[65] C. Willems,T. Holz,and F. Freiling. Toward automated dynamic malware analysis using cwsandbox [J].IEEE Security and Privacy, 2007, 5(2):32–39.

[66] W. Hardy,L. Chen,S. Hou,et al. DL4MD: A Deep Learning Framework for Intelligent Malware Detection [C].Proceedings of the International Conference on Data Mining (DMIN), 2016, 61–67.

[67] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features [C].2015 10th International Conference on Malicious and Unwanted Software (MALWARE), 2015, 11–20.

[68] Z. Yuan,Y. Lu,and Y. Xue. Droiddetector: android malware characterization and detection using deep learning [J].Tsinghua Science and Technology, 2016, 21(1):114–123.

[69] Abou-Assaleh,T. Cercone,K. Nick,et al. N-gram-based detection of new malicious code [C].28th Annual International Conference on Computer Software and Applications, 2004, 41–42.

[70] D. Reddy and P. AK. N-gram analysis for computer virus detection [J].Journal of Computer Virology Springer, 2006, 2(3):231–239.

[71] R. Moskovitch,C. Feher,N. Tzachar,et al. Unknown malcode detection using opcode representation [J].Intelligence and Security Informatics, 2008, 204–215.

[72] J. Zhang,Z. Qin,H. Yin,et al. Malware Variant Detection Using Opcode Image Recognition with Small Training Sets [C].25th International Conference on Computer Communication and Networks (ICCCN), 2016, 1–9.

[73] F. Tong and Z. Yan. A hybrid approach of mobile malware detection in Android [J].Journal of Parallel and Distributed Computing, 2017, 103:22–31.

[74] J. P. NDERSON. Computer Security Threat Monitoring and Surveillance[R].Technical Report, James P. Anderson Company, 1980.

[75] S. Aljawarneh,M. Aldwairi,and M. Y. Science. Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model [J].Journal of Computational Science, 2018, 25:152–160.

[76] X. Pan,Y. Luo,and Y. Xu. K-nearest neighbor based structural twin support vector machine [J].Knowledge-Based Systems, 2015, 88:34–44.

[77] D. H. Shin,K. K. An,S. C. Choi,et al. Malicious traffic detection using K-means [J].The Journal of Korean Institute of Communications and Information Sciences, 2016, 41(2):277–284.

[78] A. Azab,M. Alazab,and M. Aiash. Machine Learning Based Botnet Identification Traffic [C].2016 IEEE Trustcom/BigDataSE/ISPA, 2016, 1788–1794.

[79] S. M. H. Bamakan,H. Wang,and Y. Shi. Ramp loss K-Support Vector Classification-Regression; a robust and sparse multi-class approach to the intrusion detection problem [J].Knowledge-Based Systems, 2017, 126:113–126.

[80] H. Mohamad Tahir,W. Hasan,A. Md Said,et al. Hybrid machine learning technique for intrusion detection system [C].5th International Conference on Computing and Informatics (ICOCI) 11-13 August 2015, Istanbul, Turkey, 2015, 464–472.

[81] G. Zhao,K. Xu,L. Xu,et al. Detecting APT malware infections based on malicious DNS and traffic analysis [J].IEEE Access, 2015, 3:1132–1142.

[82] S. Jha,K. M. C. Tan,and R. A. Maxion. Markov Chains, Classifiers, and Intrusion Detection. [C].csfw, 2001, 1:206.

[83] S.-J. Horng,M.-Y. Su,Y.-H. Chen,et al. A novel intrusion detection system based on hierarchical clustering and support vector machines [J].Expert systems with Applications, 2011, 38(1):306–313.

[84] N. Gao,L. Gao,Q. Gao,et al. An intrusion detection model based on deep belief networks [C].Advanced Cloud and Big Data (CBD), 2014 Second International Conference on, 2014, 247–252.

[85] M. R. G. Raman,N. Somu,K. Kirthivasan,et al. A hypergraph and arithmetic residue-based probabilistic neural network for classification in intrusion detection systems [J].Neural Networks, 2017, 92:89–97.

[86] S. Peddabachigari,A. Abraham,C. Grosan,et al. Modeling intrusion detection system using hybrid intelligent systems [J].Journal of network and computer applications, 2007, 30(1):114–132.

[87] H. Zhang,G. Lu,M. T. Qassrawi,et al. Feature selection for optimizing traffic classification [J].Computer Communications, 2012, 35(12):1457–1471.

[88] C.-M. Chen,G.-H. Lai,and P.-Y. Young. Defense Joint Attacks Based on Stochastic Discrete Sequence Anomaly Detection [C].2016 11th Asia Joint Conference on Information Security (AsiaJCIS), 2016, 74–79.

[89] S. Wang,Q. Yan,Z. Chen,et al. Detecting Android Malware Leveraging Text Semantics of Network Flows [J].IEEE Transactions on Information Forensics and Security, 2018, 13(5):1096–1109.

[90] M. Albanese,S. Jajodia,and S. Venkatesan. Defending from Stealthy Botnets Using Moving Target Defenses [J].IEEE Security & Privacy, 2018, 16(1):92–97.

[91] F. Haddadi and A. N. Zincir-Heywood. Botnet behaviour analysis: How would a data analytics-based system with minimum a priori information perform? [J].International Journal of Network Management, 2017, 27(4):e1977.

[92] M. Alazab. Profiling and classifying the behavior of malicious codes [J].Journal of Systems and Software, 2015, 100:91–102.

[93] M. Alazab,S. Venkataraman,and P. Watters. Towards understanding malware behaviour by the extraction of API calls [C].2010 Second Cybercrime and Trustworthy Computing Workshop, 2010, 52–59.

[94] J. He,Y. Yang,X. Wang,et al. Peersorter: Classifying generic P2P traffic in real-time [C].Proceedings - 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Systems, 2015, 605–613.

[95] J. Zhang,R. Perdisci,W. Lee,et al. Building a Scalable System for Stealthy P2P-Botnet Detection [J].IEEE Transactions on Information Forensics and Security, 2014, 9(1):27–38.

[96] R. S. Abdullah,M. F. Abdollah,Z. A. M. Noh,et al. Preliminary study of host and network-based analysis on P2P Botnet detection [C].2013 International Conference on Technology, Informatics, Management, Engineering and Environment, 2013, 105–109.

[97] C. Yin. Towards accurate node-based detection of P2P botnets [J].The Scientific World Journal, 2014, 2014:1–10.

[98] Z. David,T. Issa,S. Bassam,et al. Botnet detection based on traffic behavior analysis and flow intervals [J].Computers & Security, 2013, 39:2–16.

[99] M. A. R Sharifnya. DFBotKiller: Domain-flux botnet detection based on the history of group activities and failures in DNS traffic [J].Digital Investigation, 2015, 12:15–26.

[100] A. L. Buczak and E. Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection [J].IEEE Communications Surveys & Tutorials, 2016, 18(2):1153–1176.

[101] W. Ye and K. Cho. P2P and P2P botnet traffic classification in two stages [J].Soft Computing, 2017, 21(5):1315–1326.

[102] W. Ye and K. Cho. Hybrid P2P traffic classification with heuristic rules and machine learning [J].Soft Computing, 2014, 18(9):1815–1827.

[103] M. Abadi,A. Agarwal,P. Barham,et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems [C].Conference on Language Resources and Evaluation, 2016, 3243–3249.

[104] G. Dahl,J. Stokes,and L. Deng. Large-scale malware classification using random projections and neural networks [C].IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, 3422–3426.

[105] O. S. Adebayo and N. A. Aziz. Static Code Analysis of Permission-based Features for Android Malware Classification Using Apriori Algorithm with Particle Swarm Optimization. [J].Journal of Information Assurance & Security, 2015, 10(4):152–163.

[106] H. Bennasar,A. Bendahmane,and M. Essaaidi. An Overview of the State-of-the-Art of Cloud Computing Cyber-Security [C].2017, 56–67.

[107] S. Chritian,L. Wei,J. Yangqing,et al. Going deeper with convolutions [C].Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), 2015, 1–9.

[108] K. He,X. Zhang,and S. Ren. Deep residual learning for image recognition [C].Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, 770–778.

[109] S. Venticinque and A. Amato.Security and Resilience in Intelligent Data-Centric Systems and Communication Networks, 2018, 123–141.

[110] R. Kumar,Z. Xiaosong,R. U. Khan,et al. Malicious code detection based on image processing using deep learning [C].ACM International Conference Proceeding Series, 2018.

[111] J. West. A Prediction Model Framework for Cyber-Attacks to Precision Agriculture Technologies [J].Journal of Agricultural & Food Information, 2018, 1–24.

[112] W. Leonard.Disciplinary Convergence in Systems Engineering Research, 2018, 145–156.

[113] O. E. Elejla,B. Belaton,M. Anbar,et al. Intrusion Detection Systems of ICMPv6-based DDoS attacks [J].Neural Computing and Applications, 2018, 30(1):45–56.

[114] M. Bhusan,R. Rathore,and A. Jamshed.2018.

[115] C. Wang,X. Zhou,F. You,et al. Design of P2P Traffic Identification Based on DPI and DFI [C].2009 International Symposium on Computer Network and Multimedia Technology, 2009, 1–4.

[116] S. García. Malware Capture Facility Project. CVUT University.[OL].2013.

[117] S. Garcia,M. Grill,J. Stiborek,et al. An empirical comparison of botnet detection methods [J].computers & security, 2014, 45:100–123.

[118] R. U. Khan,X. Zhang,R. Kumar,et al. Evaluating the performance of ResNet model based on image recognition [C].ACM International Conference Proceeding Series, 2018.

[119] R. U. Khan,X. Zhang,R. Kumar,et al. Analysis of resnet model for malicious code detection [C].2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2017, 239–242.

[120] R. Kumar,Z. Xiaosong,R. U. Khan,et al. Malicious Code Detection based on Image Processing Using Deep Learning [C].Proceedings of the 2018 International Conference on Computing and Artificial Intelligence - ICCAI 2018, 2018, 81–85.

[121] X. Ma,J. Zhang,J. Tao,et al. DNSRadar: Outsourcing malicious domain detection based on distributed cache-footprints [J].IEEE Transactions on Information Forensics and Security, 2014, 9(11):1906–1921.

[122] S. Zhao,P. P. C. Lee,J. Lui,et al. Cloud-based push-styled mobile botnets: a case study of exploiting the cloud to device messaging service [C].Proceedings of the 28th Annual Computer Security Applications Conference, 2012, 119–128.

[123] X. Ma,X. Guan,J. Tao,et al. A novel IRC botnet detection method based on packet size sequence [C].Communications (ICC), 2010 IEEE International Conference on, 2010, 1–5.

[124] Method and apparatus to detect non-human users on computer systems [J].2018.

[125] M. Guerar,A. Merlo,M. Migliardi,et al. Invisible CAPPCHA: A usable mechanism to distinguish between malware and humans on the mobile IoT [J].Computers & Security, 2018, 78:255–266.

[126] H. Jiang,J. Nagra,and P. Ahammad. SoK: Applying Machine Learning in Security - A Survey [J].2016.

[127] B. B. Gupta and O. P. Badve. Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a cloud computing environment [J].Neural Computing and Applications, 2017, 28(12):3655–3682.

[128] P. Wang,L. Wu,B. Aslam,et al.Propagation phenomena in real world networks, 2015, 183–214.

[129] N. Papernot,P. McDaniel,X. Wu,et al. Distillation as a defense to adversarial perturbations against deep neural networks [C].2016 IEEE Symposium on Security and Privacy (SP), 2016, 582–597.

[130] K. Kim. A hybrid classification algorithm by subspace partitioning through semi-supervised decision tree [J].Pattern Recognition, 2016, 60:157–163.

[131] S. Saad,I. Traore,A. Ghorbani,et al. Detecting P2P botnets through network behavior analysis and machine learning [C].2011 Ninth Annual International Conference on Privacy, Security and Trust, 2011, 174–180.

[132] W.-H. Liao and C.-C. Chang. Peer to peer botnet detection using data mining scheme [C].Internet Technology and Applications, 2010 International Conference on, 2010, 1–4.

[133] G. Fedynyshyn,M. C. Chuah,and G. Tan. Detection and classification of different botnet C&C channels [C].International Conference on Autonomic and Trusted Computing, 2011, 228–242.

Research Results Achieved During the Study for Doctoral Degree