City Research Online

City St George's, University of London

Citation: Meng, X. (2018). An integrated networkbased mobile botnet detection system. (Unpublished Doctoral thesis, City, Universtiy of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/19840/

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

AN INTEGRATED NETWORK-BASED MOBILE BOTNET DETECTION SYSTEM

Xin Meng

Department of Computer Science

City, University of London

This dissertation is submitted for the degree of

Doctor of Philosophy

City, University of London April 2018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 65,000 words including appendices, bibliography, footnotes, tables and equations and has less than 150 figures.

Xin Meng

June 2017

Acknowledgements

I would like to express my greatest appreciation to my supervisor Prof. George Spanoudakis. I am always motivated by his full support and his knowledge of the field during the whole period of my doctoral studies. There were times of the research for this thesis when I wondered if I would ever finish it, but his professional support and his personality gave me the strength to keep going further with the research. He is always patient, and his encouragement and advice are valued greatly. My thesis work would not have been accomplished without his help, aspiring advice and expert guidance.

I would like to acknowledge my deep appreciation to all the colleagues from School of Mathematics, Computer Science & Engineering, Department of Computer Science.

Special thanks go to my parents and my wife for their endless love.

Abstract

The increase in the use of mobile devices has made them target for attackers, through the use of sophisticated malware. One of the most significant types of such malware is mobile botnets. Due to their continually evolving nature, botnets are difficult to tackle through signature and traditional anomaly based detection methods. Machine learning techniques have also been used for this purpose. However, the study of their effectiveness has shown methodological weaknesses that have prevented the emergence of conclusive and thorough evidence about their merit.

To address this problem, in this thesis we propose a mobile botnet detection system, called MBotCS and report the outcomes of a comprehensive experimental study of mobile botnet detection using supervised machine learning techniques to analyse network traffic and system calls on Android mobile devices.

The research covers a range of botnet detection scenarios that is wider from what explored so far, explores atomic and box learning algorithms, and investigates thoroughly the sensitivity of the algorithm performance on different factors (algorithms, features of network traffic, system call data aggregation periods, and botnets vs normal applications and so on). These experiments have been evaluated using real mobile device traffic, and system call captured from Android mobile devices, running normal apps and mobile botnets.

The experiments study has several superiorities comparing with existing research. Firstly, experiments use not only atomic but also box ML classifiers. Secondly, a comprehensive set of Android mobile botnets, which had not been considered previously, without relying on any form of synthetic training data. Thirdly, experiments contain a wider set of detection scenarios including unknown botnets and normal applications. Finally, experiments include the statistical significance of differences in detection performance measures with respect to different factors.

The study resulted in positive evidence about the effectiveness of the supervised learning approach, as a solution to the mobile botnet detection problem.

Contents

Contents ix
List of Figures xiii
List of Tables xv
List of Abbreviations xvii
List of Equations xix
Chapter 1 Introduction 1
1.1 Research background 1
1.2 Motivation 6
1.3 Overview of approach 7
1.4 Research hypothesis and objectives 8
1.5 Contribution 9
1.6 Thesis outline 10
Chapter 2 Literature Review 13
2.1 Background of botnet 13
2.1.1 Definition 13
2.1.2 Taxonomy 15
2.1.3 Lifecycle of botnet 19
2.1.4 Comparison different type of malware 22
2.2 Conventional botnet detection techniques 23
2.2.1 Incidents 23
2.2.2 Botnet creation techniques 25
2.2.3 Taxonomy of detection techniques 27
2.2.4 Comparison 37
2.3 Network traffic anomaly detection technique 43
2.3.1 Introduction of anomaly detection and intrusion detection 43
2.3.2 Classification of anomaly-based detection technique 46
2.3.3 The future trend of anomaly detection technique 52
2.4 Machine Learning 53
2.4.1 Introduction of machine learning algorithm 53
2.4.2 Machine learning algorithms 55
2.4.3 Evaluation criteria for machine learning 73
2.5 Mobile botnet 78
2.5.1 Mobile botnet accidents 79

2.5.2 Mobile botnet creation 82
2.5.3 Detection techniques 85
2.5.4 ML based botnet detection techniques comparison 88
2.5.5 Mobile botnet detection specificity 92
2.5.6 Open issues 95
Chapter 3 The MBotCS Detection System 97
3.1 Overall framework 97
3.2 Introduction of components 100
3.2.1 Mobile Verification Component 100
3.2.2 Data Broker 100
3.2.3 Notification Broker 101
3.2.4 Feedback Broker and Processor 101
3.2.5 Data Analyser 102
3.3 System Security 103
3.3.1 Communication Security 103
3.3.2 MVC Application Security 105
Chapter 4 Experimental Evaluation of Mobile Botnet Detection 107
4.1 Overview 107
4.2 MVC of MBotCS System on mobile device 110
4.3 Implementation of MVC Component 112
4.3.1 Network traffic capture 112
4.3.2 Pcap parse and pre-processor 113
4.3.3 WEKA based machine learning analyser 114
4.3.4 User interface 114
4.4 Analysed normal and botnet applications 115
4.5 Experiments for network traffic analysis 118
4.5.1 Workflow of experiments 118
4.5.2 Experiment I 125
4.5.3 Experiment II 139
4.5.4 Experiment III 146
4.6 Experiments for system call analysis 150
4.6.1 Overview 150
4.6.2 Methodological setup of the experiments 151
4.6.3 Basic statistical analysis 155
4.6.4 First experiment: KBKN scenario 157
4.6.5 Second experiment: UBKN scenario 165
4.6.6 Third experiment: UBUN scenario 174

4.6.7 Overall discussion & threats to validity 180
4.7 Conclusion of experiments 183
Chapter 5 Conclusion 184
5.1 Discussion 184
5.2 Summary of contributions 186
5.3 Further research 188
5.3.1 Future research directions 188
5.3.2 Planned and related work within MBotCS 189
References 191
Appendix A Key Implementation Code of Android Application 217
A.1 PCAP file parse 217
A.2 Machine learning analyser 221
A.3 Android intent service 228
A.4 User interface 229
Appendix B System Call Monitor Bash Script 232
Appendix C Key Implementation Code of Broker 235
Appendix D Key Implementation Code of Analyser 242
Appendix E Normal Application Actions 244
Appendix F Tables of Experiments Result 246
Appendix G Training Dataset Feature Selection 251

List of Figures

Figure 2-1 - Four types of communication structure of botnet [69] 16
Figure 2-2 - Adapted version of botnet lifecycle 21
Figure 2-3 - The hierarchical classification of botnet detection techniques 28
Figure 2-4 - The feature of Rxbot data packet 32
Figure 2-5 - The example of point anomalies 44
Figure 2-6 - The example of contextual anomaly 45
Figure 2-7 - The unit of neural network 66
Figure 2-8 - Simple neural network model 66
Figure 3-1 - The overall architecture of MBotCS 99
Figure 4-1 - Architecture of MVC on mobile devices 111
Figure 4-2 - The GUI of MBotCS 112
Figure 4-3 - Workflow of experimental training 118
Figure 4-4 - Tshark command for extracting features 120
Figure 4-5 - Normal stream result visualisation 128
Figure 4-6 - Infect stream result visualisation 129
Figure 4-7 - Comparison between packet and stream dataset 130
Figure 4-8 - Comparison of ML algorithms 134
Figure 4-9 - ROC curve of infect stream based on six classifier algorithms 135
Figure 4-10 - ROC curves of evaluation of selected botnet detection systems 136
Figure 4-11 - Infect recall across infect malware family 141
Figure 4-12 - Infect FPR across infect malware family 141
Figure 4-13 - Infect Prec across infect malware family 141
Figure 4-14 - Comparison of ML algorithms 143
Figure 4-15 - Battery consumption 148
Figure 4-16 - The ML-analyser execution time 149
Figure 4-17 - Experimental set up 151
Figure 4-18 - Performance of normal across different time interval dataset 160
Figure 4-19 - Performance of botnet across different time interval dataset 161
Figure 4-20 - Average performance of time interval dataset 162
Figure 4-21 - Performance of normal across different malware family dataset 168
Figure 4-22 - Performance of botnet across different malware family dataset 169
Figure 4-23 - Average performance of malware family dataset 170
Figure 4-24 - Performance of normal across different malware family dataset 176

Figure 4-25 - Performance of botnet across different malware family dataset.....177
Figure 4-26 - Average performance of malware family dataset 178
Figure 4-27 - Average TPR, FPR, PRC and AUC measures for botnet applications in all three experiments. 180

List of Tables

Table 2-1 - The accidents of conventional botnet 24
Table 2-2 - Taxonomy of botnet detection techniques 29
Table 2-3 - The summary of the criteria for botnet detection 39
Table 2-4 - Confusion matrix 74
Table 2-5 - List of potentially harmful behaviours of mobile botnet 80
Table 2-6 - The accidents of mobile botnet 81
Table 2-7 - ML based botnet detection approaches 91
Table 4-1 - Botnet malware families 116
Table 4-2 - Normal applications 117
Table 4-3 - Pattern-matching library 121
Table 4-4 - AVONA analysis results 123
Table 4-5 - Ranking of algorithms for infected stream traffic in experiment I 132
Table 4-6 - The AUC of six classifiers based on stream dataset 135
Table 4-7 - Outcome of analysis of variance for experiment I 137
Table 4-8 - Ranking of algorithms for infected stream traffic in experiment II 142
Table 4-9 - Outcome of analysis of variance for experiment II 144
Table 4-10 - The specs of GT-I9228 147
Table 4-11 - Structure of primitive system call data set 153
Table 4-12 - Structure of derived dataset 154
Table 4-13 - Statistical significance of system call frequency differences 156
Table 4-14 - Outcomes of analysis of variance for experiment 1 163
Table 4-15 - Outcomes of analysis of variance for experiment 2 171
Table 4-16 - Outcomes of analysis of variance for experiment 3 179
Table F-1 - Results of experiment I of network traffic 246
Table F-2 - Results of experiment II of network traffic 247
Table F-3 - Performance measures in experiment 1 of system call 248
Table F-4 - Performance measures in experiment 2 of system call 249
Table F-5 - Performance measures in experiment 3 of system call 250
Table G-1 - Network Traffic Feature Selection 251

Table G-2 - System Calls Dataset Feature Selection 251

List of Abbreviations

Abbreviation	*Meaning*	*Page*
ACC	Accuracy	75
ADB	Android Debug Bridge	152
ANOVA	Analysis of variance	77
API	Application Programming Interface	34
APNS	Apple Push Notification Service	19
AUC	Aera Under the Curve	75
BPS	Bits per second (bps), a data rate unit	19
C2DM	Cloud to Device Messaging	19
C4.5	C4.5 is an extension of Quinlan's earlier ID3 algorithm	62
C&C	Command-and-Control	55
CBLOF	Cluster-Based Local. Outlier Factor	50
CER	Crossover Error Rate	40
COF	Connectivity-based Outlier Factor	49
COER	Crossover Error Rate	38
CPU	Central Processing Unit	147
CSV	Comma-Separated Values	121
DDOS	Distributed Denial of Service	1
DNS	Domain Name System	27
FAR	False Acceptance Rate	39
FN	False Negative	40
FP	False Positive	40
FPR	False Positive Rate	40
GCM	Google Cloud Messaging	19
GPRS	General Packet Radio Service	95
GUI	Graphical User Interface	107
HPCC	Hybrid Peer to Peer Command and Control	26
HTTP	Hypertext Transfer Protocol	5
ID3	Iterative Dichotomiser 3	60
IDS	Intrusion Detection System	5
I/O	Input/Output	156
IP	Internet Protocol	17
IRC	Internet Relay Chat	1
KBKN	Know Botnet and Known Normal	10
KBTA	Knowledge-based Temporal Abstractions	88
KNN	K-Nearest Neighbour	48

KTT	Karush–Kuhn–Tucker	72
LOF	Local Outlier Factor	48
ML	Machine Learning	7
MPNS	Microsoft's Push Notification Service	19
MTU	Maximum Transmission Unit	131
MVC	Mobile Verification Components	97
NFC	Near-Field Communication	85
NIDS	Network Intrusion Detection System	37
NIPS	Network Intrusion Prevention System	37
NNA	Nokia's Notifications API	19
NN	Neural Network	65
NPV	Negative Predictive Value	75
P2P	Peer to Peer	5
PCAP	Packet CAPture (Data file created by Wireshark)	107
RAM	Random-Access Memory	147
RD	Relative Density	48
RMSE	Root-Mean-Squared Error	76
ROC	Receiver Operating Characteristics	38
SCM	System Call Monitor	153
SMC	Simple Matching Coefficient	48
SMO	Sequential Minimal Optimisation	70
SMS	Short Message Service	2
SMTP	Simple Mail Transfer Protocol	28
SVM	Support Vector Machine	55
Tbps	Terabytes Per Second	2
TN	True Negative	40
TP	True Positive	40
TPR	True Positive Rate	40
TTTS	Training-To-Test Data Set Size	145
UBKN	Unkonw Botnet and Known Normal	10
UDP	User Datagram Protocol	120
UFCC	URL Flux-based Command-and-Control	26
UGA	Username Generation Algorithm	26
UNUB	Unknown Normal and Unknown Botnet	10
URL	Uniform Resource Locator	83
VPN	Virtual Private Network	190
VRC	Value Range Criteria	109

List of Equations

Equation 2-1 TCP work weight equation 35
Equation 2-2 Similarity between DNS traffic 35
Equation 2-3 Equation True Positive Rate 40
Equation 2-4 False Positive Rate 40
Equation 2-5 Precision 40
Equation 2-6 Euclidean distance between two points 48
Equation 2-7 Simple matching coefficient equation 48
Equation 2-8 Bayes's theorem 56
Equation 2-9 Bayes's theorem: conditional probability of features I 57
Equation 2-10 Bayes's theorem: conditional probability of features II 57
Equation 2-11 Bayes's theorem: conditional probability of features III 58
Equation 2-12 ID3 algorithm: entropy equation 61
Equation 2-13 ID3 algorithm: information after attributes split 61
Equation 2-14 ID3 algorithm: information gain equation 61
Equation 2-15 C4.5 algorithm: split information 62
Equation 2-16 C4.5 algorithm: information gain ratio 62
Equation 2-17 KNN algorithm: distance compute equation 64
Equation 2-18 KNN algorithm: vote equation 64
Equation 2-19 Three layers NN algorithm: output of layer 2 67
Equation 2-20 Three layers NN algorithm: output of layer 3 67
Equation 2-21 Three layers NN algorithm: overall cost function 67
Equation 2-22 gradient descent 68
Equation 2-23 Neural network algorithm: cost function 68
Equation 2-24 Support vector machine: objective function 70
Equation 2-25 Support vector machine: decision function 70
Equation 2-26 decision boundary of SVM 71
Equation 2-27 to Karush–Kuhn–Tucker conditions 71
Equation 2-28 Cohen's Kappa formula 75
Equation 2-29 Root-mean-squared error formula 77
Equation 4-1 K-fold classification error of a classifier 124

Chapter 1 Introduction

1.1 Research background

Botnets malware has become one of the most serious threats to networks which can be defined as "A network of remotely controlled systems used to coordinate attacks and distribute malware, spam, and phishing scams" [1]. Unlike traditional malware, botnets spread easily and have a wider range impact. Although there is significant research on detecting and defending botnets, most of the defence systems still stay at a primary stage. In 2007, experts give an estimation that approximately 100 to 150 million computers which comprise about 16–25% of all current 600 million computers which connected to the Internet were already controlled by botnets [2, 3]. Since the appearance of botnets, there has been a continuous stream of news regarding damages caused by different botnets. An IRC-based botnet which infected 10,000 machines to perform a DDOS attack and spread junk email was discovered in 2004 [4]. In June 2008, the Shadow Server Foundation gave an estimation that the number of botnets had exceeded 450,000 machines [5]. In 2009, the Carbon Footprint of e-mail Spam report estimated that 62 trillion spam emails are sent globally every year, and the majority of these spam emails are sent via botnets [6]. Troj.MDK botnet malware on Android platform which was discovered in 2012 is estimated to have been hidden in more than 11,000 malicious apps and infected more than 1 million mobile devices in China [7]. In June 2013, Microsoft and the FBI launched a joint strike for breaking up the Citadel botnet which has stolen more than $500 million (£323 million) from bank accounts and infected more than 5 million computers [8]. In 2016, DDoS attacks make their mark on all the digital internet threats around the world. ATLAS tracked nearly 124,000 DDoS attack events each week in 6 months from January 2015 to June 2016. Apart from the increment of frequency, the peak attack size also growth 73% compared with 2015 from 354Gbps to 579Gbps [9]. The most five significant DDoS attacks make use of

Internet of Things(IoT) botnet and leverage insecure devices to conduct the attack [10]. Unlike other types of Botnet depend on computing devices such as computer and smartphone, IoT botnet can be largely made up of IoT devices such as digital cameras and digital video recorder (DVR) player [11]. Mirai Botnet is one of most famous IoT botnet with significate growth pace because the source code is published on Github [12]. According to the report of Level 3 Threat Research Labs, the Mirai bots has been reached to 493,000[13]. In October 2016, the Dyn suffered a DDoS attack regarding as the most severe one in 2016 which performed by Mirai Botnet including approximately 100,000 bots. Packet flow reached to nearly 50 times higher than its normal volume during the attack and the attack peak size reach almost 1.2 Terabytes Per Second (Tbps). The attack leads to several high-profile websites suffered service interruptions and went offline [14, 15]. In November 2016, A botnet which consisted of at least 24,000 bots located in more than 30 countries made a DDoS attack for at least 5 Russian major banks [16]. It is no doubt that botnets have become one of the most serious security problems to the Internet.

With the popularisation of mobile phone and development of the wireless mobile internet, the mobile devices are becoming the target of the botnets. According to Cisco, 497 million new mobile devices and connections were sold in 2014 [17]. Another recent report has published that the global mobile devices and connections have grown to 8.0 billion in 2016, up from 7.6 billion in 2015. It also forecasts that mobile-cellular subscriptions will grow to 11.6 billion by 2021 [18]. Market reports also show that since 2012 Google's Android operating system has overtaken other smartphone operating systems and is currently the market leading mobile OS and is expected to get more than 80% market share until 2019 [19]. Along with the growth in the use of mobile devices, there has also been a growing number of mobile malware systems, often in the form of mobile botnets. According to KASPERSKY [20], a mobile botnet is defined as a collection of applications, which are connected through a network and communicate with each other and a remote server (known as the "botmaster") in order to perform orchestrated attacks (e.g., remotely executed commands, information stealing, SMS dispatching). Nowadays, the performance of the smartphone and tablet is similar to the PC. Along with the enormous potential benefits of the mobile network, many mobile botnets have appeared in the real network. The simplification of the mobile devices and the lack of the

network safety consciousness of most of the common users of such devices make botnet a serious problem. According to [20], 148,778 mobile malware applications had been detected at the end of 2013, and nearly 62% of them were part of mobile botnets. In 2016, the number of malicious installation packages grew considerably, amounting to more than 8.5 million which is three times more than 2015 [21]. The first mobile botnet was an iPhone botnet, known as *iKee.B* [22], which was traced back in 2009. *iKee.B* was not a particularly dangerous botnet because iOS is a closed system. Unlike it, Android, which is an open system, has become a major target for mobile botnet creators. *Geinimi* was the first Android botnet that was discovered in 2010 [23]. Other Android botnets include *Android.Troj.Mdk*, i.e., a Trojan found in more than 7,000 apps that have infected more than 1m mobile users in China, and *NotCompatible.C*, i.e., a Trojan targeting protected enterprise networks [24]. Research on mobile botnets (see [25] for detailed surveys) has looked at device specific botnet detection [26] as well as mobile botnet implementation principles and architectures for creating mobile botnets [27, 28].

The current situation of botnets research is still staying the initial stage. Although a lot of the solution has proposed by some researchers, there is hardly any mature botnets detection and defence system which is used in the real environment. Most of the methods for confronting with botnets malware still rely on the signature-based antimalware software. There is a bottleneck for the conventional platform botnets detection which is the evaluation measurement according to analyses most of the detection techniques. Because of the bottleneck, it is hard to evaluate the various kinds of detection technologies. In our research, we will give an as far as possible objective evaluation for several conventional detection approaches based on several criteria.

Research on the detection of mobile botnets has intensified over the last few years. The techniques developed for this purpose range from static analysis of application code [26, 29, 30], analysis of application fingerprints [31], signature-based detection [32-34], anomaly based detection [35-38] and detection based on machine learning techniques [39-41]. In addition to the detection techniques, some research focuses on mobile botnet implementation principles [25, 42, 43] and new mobile botnets architectures [22, 27, 28, 44-59]. The purpose of these research is to make a prediction of new types of mobile botnets that may occur in

future. Currently, mobile users can only use software that detects the malware based on signatures of known mobile botnets. However, as in botnets, the botmaster can update the malicious code of the bots continuously. So more dynamic detection approaches that are capable of detecting unknown future mobile botnets are required.

Through a literature review, we find several surveys and reports are giving an overview of botnets in different aspects. However, most of them just concentrate on the conventional platforms such as traditional computers, routers, switches and so on. Because of large scale compromise of the mobile botnet and the increase of mobile botnet research, we will give a more detailed state of the art of mobile botnets.

McCarty [60] provide a brief description of botnet including the structure and principal illegitimate purposes based on Honeynet Project which can be regarded as the earlier research. Puri [61] presented a comprehensive overview of the botnet based on IRC including the mechanism, attack, target and the possible defending methods of an IRC-based botnet. The survey even gives a list of some known bots (The link in the list is invalid now). Rajab et al. [62] deploy multifaceted distributed data collection infrastructure which can capture the activities of botnets to demonstrate the botnets phenomenon. Through their measurement methodology, they perform a comprehensive measurement analysis that reflects the prevalence, spreading and growth patterns, structure, lifetime and efficient size of the botnets.

The survey in [63] offers a brief overview of botnet based on the existing research on every aspect. It not only makes an anatomy of Bot but also discusses the techniques of botnet detection and defence. It divides the detection into two main approaches, one is Honeynet based method, and the other relies on passive traffic monitoring. This survey also found the rare research on the defence technologies against botnet which only concentrate on the spam detection and enterprise solutions. Liu et al. [64] make a summary for the most of the direction of botnet research. They discuss primary concepts of botnet including structure, exploitation, lifecycle and topology and introduce several relevant attacks, detection, tracing, and countermeasures. The short survey [65] plain introduction of the related research directions, addressing infection mechanisms, malicious behaviour, command and control models, communication protocols, botnets detection and defence against botnets. They also present a

simple case study of early IRC-based botnet worm – SpyBot [66]. Shin and Im [67] present a description of botnet basic knowledge and botnet defence method. The emphasis of this survey is topologies and consequences of botnets. Zhang et al. [68] introduce the principle and mechanism of fluxing in botnet which includes fast fluxing and domain fluxing. Moreover, they also make an investigation of research on fluxing botnet detection. Silva et al. [69] present a comprehensive review that broadly discusses the botnet problem. The survey gives a presentation of a comprehensive tutorial-like study addressing the botnet problem in general firstly and lists the timeline of some important bots from 1993 to 2011 and their main features. It also makes a summary of detection and defence techniques which contain nearly all the existing research.

Further research on the subject includes works which provide taxonomy according to a particular aspect of botnets. Dagon et al. [70] present a taxonomy of botnets based on a topological structure which divides into three categories(centralised, peer-to-peer, and random). They even measure their utilisation by using four metrics: effectiveness, efficiency, robustness and average available bandwidth. Zeidanloo et al. [71] provides a classification of botnets C&C channels which are divided into three models (centralised, decentralised and hybrid) and evaluate well-known protocols (e.g. IRC, HTTP, and P2P) which are being used in each of them. They also give a taxonomy of botnet detection techniques which classify into two approaches [72]: Honeynet and IDS(Intrusion Detection System). Czosseck et al. [73] propose a comparatively overall botnet taxonomy based on usage which consists of four features: users of botnets, motivations of botnet usage, functionality applied and way of infection. The literature even examines some instances according to their taxonomy. Hachem et al. [74][51] present a classification that reflects the life cycle and current resilience techniques of botnets, distinguishing the propagation, the injection, the control and the attack phases.

Although the challenges of botnet detection are discussed as the last part of majority surveys, there is also some literature which pays attention to the challenges separately. Rajab et al. [52] draw a conclusion that it remains some challenges to estimate the size of botnet through presenting different metrics for counting botnet membership and show different size estimates for a large number of botnets they traced. Aviv and Haeberlen [53] outline several current

challenges when evaluating botnet detection systems. The survey analyses the evaluation methods for existing botnet detection system and finds that it is hard to find appropriate test traces for botnet detection system because of some issue such as realism, sensitive information and so on. Brezo et al. [54] sets out the main lines of current research in botnet detection and presents the limitation of the existing botnet detection. The recent literature [55] shed light on some of the challenges for establishing botnet Emulation systems. Moreover, they discuss various techniques used to address or alleviate these problems.

Through a large number of literature reviews, we divided these researches of botnet into four main categories based on the research direction.

- Botnet survey and mechanism research
- Botnet detection on conventional platform
- Design and detection of mobile botnet and malware

In our research, we will make an in-depth discussion about state of the art for the botnet according to these research directions.

1.2 Motivation

In this research, the main target is to design a high-efficiency detection system to detect the attack of known and unknown mobile botnet. Some reasons are listed as follows:

- Mobile botnets have posed a severe threat to the property and individual privacy. With the increasing of the number and the performance of mobile devices, more and more important transactions are allocated to the mobile devices. Such as almost all bank transactions can be finished by the mobile application and most of the personal information can be stored on the mobile device even for the business and government confidential information. In contract, the measure of protection and the safety awareness for mobile devices are still weak at present. Especially in the developing country and regions, mobile device users prefer to install applications obtained from non-trustworthy sources to save money, which is likely to carry malicious codes that can infect the device. Therefore, hackers try to infect a large number of mobile devices

and establish botnet that will be capable of performing attacks, causing severe damage and loss [7, 75].

- Limited measurement to detect unknown mobile botnets. Currently, the only protective measures for mobile devices is anti-malware software. Most of the anti-malware software is signature-based detection technique which can only detect the known botnet malware. However, the botnet malware is frequently upgraded, and new botnet malware appears continuously. Therefore, the detection of unknown botnet malware is an important issue that needs to be solved.
- The use of ML techniques for botnet detection has been applied before. However, existing research has not: (a) provided a comprehensive coverage of mobile botnets and range of system calls, (b) considered systematically different detection scenarios arising from combination of known and unknown normal applications and botnets, and (c) conducted any systematic analysis of the sensitivity of the performance of ML classifiers against some key dimensions for the practical applications of detection as for example the statistical significance of performance differences observed across different ML classifiers and types of botnets. Also, existing research has been restricted by arbitrary and not experimentally tested assumptions about the range of system calls that should be taken into account in botnet detection.

1.3 Overview of approach

In this thesis, we propose a proactive approach for detecting unknown mobile botnets that we have implemented for Android devices. Our approach is based on the analysis of traffic data and system call of Android mobile devices using machine learning (ML) techniques and can be realised through an architecture involving traffic monitors and controllers installed on them.

In the first stage, we perform the classification experiments on the server side. First of all, a large amount of data including network traffic and system call captured from mobile botnet malware applications and normal applications. Then some features are generated for machine learning classification according to the analysis of the captured network traffic data. After

labelling data, we choose some atomic machine learning algorithms and develop some aggregate machine learning algorithms to classify the data based on the selected features.

In the second stage, we focus on the implementation of the mobile botnet detection system on the client side which is the Android platform. Firstly, we implement the network traffic and system call capture component which can monitor all the traffic pass through the mobile device and system call invoked by specified applications. The component can also pre-process these data to a standard format for the machine learning classifiers. Then a machine learning analyser is deployed to classify the traffic data by using the training dataset that generated at the first stage. At last the detection system shows the warning for the suspicious traffic.

The experimental study that we report in this thesis has been aimed to overcome the above limitations and investigate a number of additional factors, notably: (a) the merit of aggregate ML classifiers, (b) the sensitivity of the detection capability of ML detectors on different types of botnet families, and (c) the actual cost of using ML detectors on mobile devices in terms of execution efficiency and battery consumption. Furthermore, our study has been based on a mobile botnet detection system that we implemented and deployed on a mobile device.

1.4 Research hypothesis and objectives

The research objectives of this thesis have been as follows:

- **Objective 1:** To undertake and produce a comprehensive survey of the botnet and mobile botnet research.
- **Objective 2:** To design a botnet detection system that can operate on mobile devices, to detect unknown mobile botnet with network traffic and system call based on the use of machine learning techniques.
- **Objective 3:** To implement the new mobile botnet detection system on Android devices, addressing the open issues identified in Section 1.2
- **Objective 4:** To provide an experimental evaluation of the approach.

To achieve our research target, some of the research hypotheses are presented as follows:

- **Research Hypotheses I:** It will be possible to detect mobile botnets using machine learning with the feature generated by the captured network traffic of the botnet and normal applications on Android devices. Because there are existing several researches about using machine learning with network traffic to detect malware on the desktop.
- **Research Hypotheses II:** It will be possible to detect mobile botnets using machine learning with the feature generated by the invoked system call of the botnet and normal applications on Android devices. Because there are existing several researches about using machine learning with Linux system call to detect malware on the desktop.
- **Research Hypotheses III:** It will be possible to detect mobile botnets using machine learning method with the feature of the frequency of system calls in the different time interval. Because there are existing several researches about using pattern of Linux system call to detect malware on the desktop.
- **Research Hypotheses IV:** Using aggregated machine learning algorithm to detect mobile botnets will improve detection accuracy over competing for the atomic algorithm. Because there are existing several scenarios for using aggregated machine learning algorithms to improve the performance.
- **Research Hypotheses V:** The mobile botnet detector utilising the ML approach can be built for and deployed on a mobile phone to detect botnets without depleting the energy of it and without affecting the performance of other mobile applications (benign) on the phone. Because the performance of the mobile device is increased rapidly in recent several years and is close to the desktop.

1.5 Contribution

For our research, we can summarise our contributions on mobile botnet detection as follows:

- We present the results of an experimental study on the use of ML algorithms for the detection of mobile botnets operating on Android devices, based on the network traffic data captured on the mobile devices. Our experiments have shown that

algorithm J48 and Box-Half+ have the best performance distinguish the botnet and normal by using the network traffic data.

- We present the results of an experimental study on the use of ML algorithms for the detection of mobile botnets operating on Android devices, based on the analysis of system (i.e., Android OS) calls, whose aim has been to address the above limitations. This experimental study not only atomic but also box ML classifiers using supervised learning. The performance of ML classifiers a wider set of detection scenarios than existing work, namely detection of known botnets and known normal applications (KBKN scenario), unknown botnets and known normal applications (UBKN scenario), and unknown botnets and normal applications (UNUB scenario). A comprehensive set of Android mobile botnets, which had not been considered previously, without relying on any form of synthetic training data. The statistical significance of differences in detection performance measures with respect to ML algorithms, system call aggregation periods, normal and botnet applications, and different types of botnet families. We have also implemented botnet detection system running on the mobile device by using the classifier trained in previous experiments and evaluated the effect of our approach with respect to its effect on the overall performance and battery consumption of mobile devices. The system has a low energy effect on the battery consumption of the device with only 0.5% of the total battery during the period of the experiment. Moreover, the J48 algorithm has fast average execution time with only 1.216 seconds.
- We proposed and developed a network-based mobile botnet detection system named MBotCS. The design and implementation of every component in the system were described in detail. Meanwhile, we also provided several solutions for solving the security issue of the intercomponent communication and the privacy of the application which is deployed on the end client.

1.6 Thesis outline

The rest of this thesis is organised as follows.

Chapter 2 reviews recent studies about the botnet detection including the background knowledge, conventional botnet, the anomaly detection techniques and machine learning. Mobile botnet researches are also reviewed at last

Chapter 3 present the architecture of MBotCS and solutions for security issues.

Chapter 4 presents the design of our experiments and the analysis of the results obtained from them. Moreover, the chapter reports on the results of the experiments that we conducted to evaluate the merit of different ML algorithms for the botnet detection by mining system calls captured on Android OS.

Finally, Chapter 5 reviews the objectives and outlines conclusions and plans for future work.

Chapter 2 Literature Review

2.1 Background of botnet

2.1.1 Definition

According to [76], botnets is defined as “A collection of Internet-connected programs communicating with other similar programs to perform tasks”. It divides botnets into legal botnets and illegal botnets. The botnets were derived from IRC in 1993, and the early bots can perform much beneficial and even vital functions for managing the IRC automatically. Unfortunately, more and more botnets have been developed for malicious purposes. Our research focuses on the illegal botnets.

Except the [76], there are many papers or articles give a definition for illegal botnets. The report [1] describe botnets as “A network of remotely controlled systems used to coordinate attacks and distribute malware, spam, and phishing scams”. The report [77] define botnets as “A networks of Internet-connected end-user computing devices infected with bot malware, which is remotely controlled by third parties for nefarious purposes”. The paper [78] regards botnets as “A collections of computers infected with malicious code that can be controlled remotely through a command and control infrastructure”. The paper [79] define botnets as “A network of compromised computers that are remotely controlled by malicious agents”. The report [80] define botnets as “A group of malware

infected computers also called “zombies” or bots that can be used remotely to carry out attacks against other computer systems”.

According to these definitions of botnets, the similarity of different description is that botnets consist of three key components including the compromised computers in the network, the attacker who control these computers and the remote control channel. Although the current botnet is more complex than the first known IRC botnet-Eggdrop, the basic components of botnets never change. In our context of work, the Botnet can be defined as collections of computers infected with malicious code (Bots) that can be controlled remotely by the attacker (Botmaster) through a command and control infrastructure (C&C). The following terms to describe the three basic components of a botnet.

- Bots: The definition of a bot varies within the literature. Some researchers regard the devices where malicious programs run as bots (the term “bot” is derived from “robot”) [61, 62, 68, 81]. Others regard the malicious programs themselves as bots [63, 69, 76, 82]. Regardless of such differences, however, bots can be defined as the computational entities of the botnet that have malicious behaviour causing some harm and which exhibit this behaviour under control by the owner of the botnet.

- Botmaster: Botmaster is usually defined as the human operator of botnets which can control the bots to execute commands [28, 62]. The extra responsibilities include keeping the bot online, maintaining the errors in bots and updating the malicious code for new features [69].

- Command and Control (C&C): C&C signifies the commands that the botmaster sends to bots to instruct them to perform malicious tasks. C&C may be transmitted using different network communication protocols such as IRC [83], HTTP [84] and SMS [85] and so on.

2.1.2 Taxonomy

Botnets can be classified according to different criteria. Much literature tries to summarise their taxonomy. The literature [61, 86] use the set of commands, the topology of C&C, the propagation mechanism, and the exploitation strategy utilised by the attacker to classify the botnet. The literature [87] made a contribution to botnet taxonomy by listing three topologies of Command and Control Models(centralised, peer-to-peer, and random). The survey [69] further generalise C&C architecture into four types including Centralized C&C, Decentralized C&C, Hybrid model C&C and Random model C&C which is considered relatively integrated taxonomy for C&C. The literature [74] give a fine-grained taxonomy based on the different phases of the botnet lifecycle. It divided the C&C into four dimensions including Model &Topology, Application &Protocol, Communication initiation and Communication direction and gave a taxonomy based on every dimension.

Even though the current literature has contained nearly all possible taxonomy of botnet, there is few papers try to combine these and give a comprehensive taxonomy. We just list the criteria and the taxonomy in existing literature and provide a description and explanation in detail.

2.1.2.1 Communication structure of botnet

This criterion is concerned with the topology of the command & communications within a botnet. According to it, botnets can be distinguished into a Centralised Model botnet, Decentralised Model botnet, Hybrid Model botnet [88] and Random Model botnet [69] based on the topology of botnet [72, 89]. We also present the four type of communication structure in Figure 2-1.

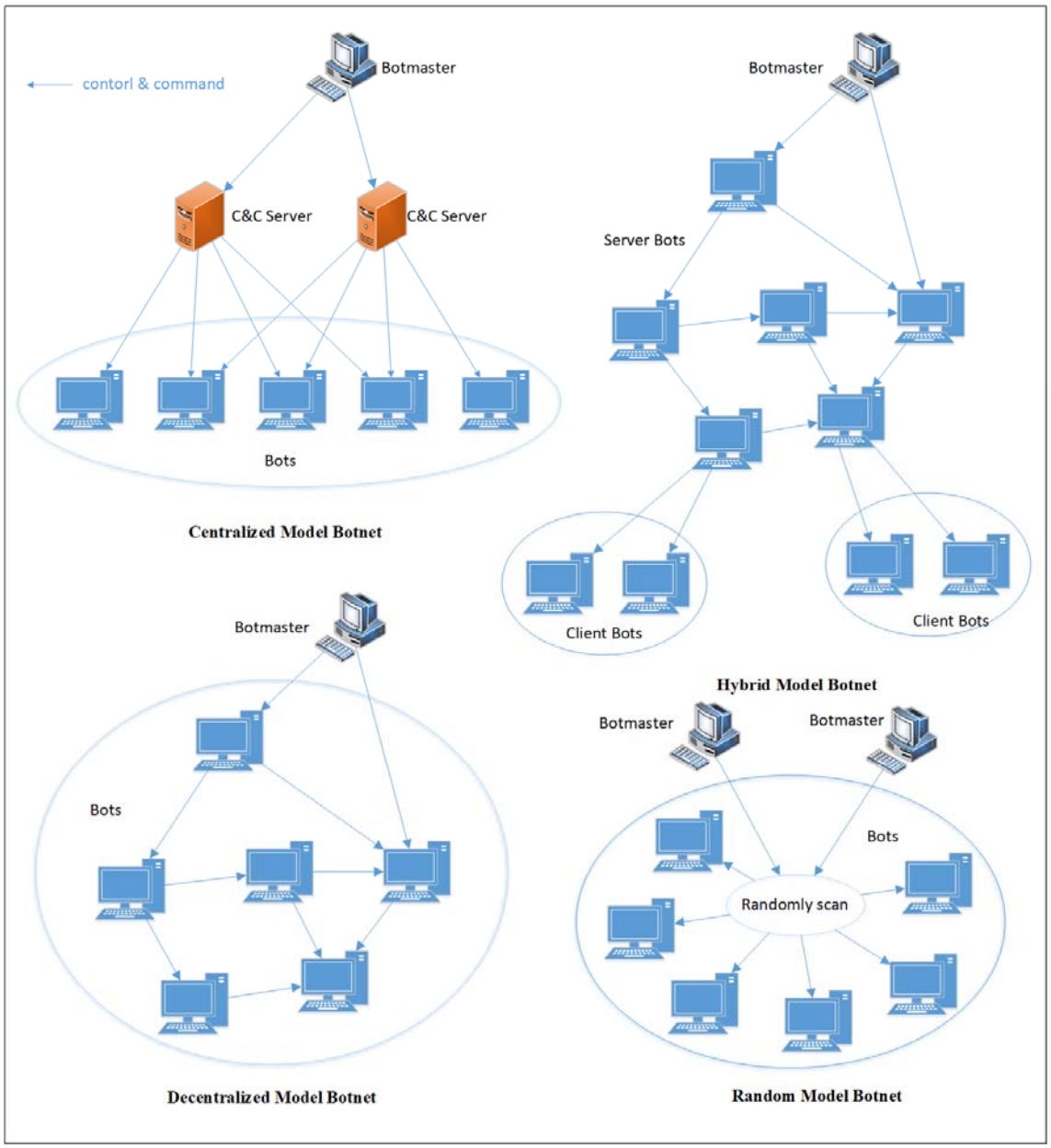


Figure 2-1 - Four types of communication structure of botnet [69]

- Centralised Model botnet: The centralised model is the older type of topology which has one central point being responsible for exchanging commands and data between the botmaster and bots.
- Decentralised Model botnet: In the decentralised model botnet, there is no centralised point for communication, and each Bot makes some connections with the other bots so that the botmaster has multiple accesses for bots.
- Hybrid Model botnet: The hybrid model divides the bots into servant bots and client bots which construct local centralised and global decentralised structure [88]. The servant bots group contains bots that have static, non-private IP addresses and are accessible from the global Internet. Moreover, the client bots group contains the bots with dynamically allocated IP addresses, private IP addresses and behind firewalls disconnected from the global Internet. The client bots cannot accept incoming connections.
- Random Model botnet: This model is first introduced by Cooke et al. [87] as the "botnet model of tomorrow". The connection between botmaster and bots in this communication structure is not fixed. So if the controller of the botnet wants to perform an attack, the botmaster operates by scanning the Internet randomly and sending a command to every host in the network. Once botmaster receives the specify reply, it means that a bot is found. The design of such a system would be relatively simple, and the detection of a single bot would never compromise the full botnet.

2.1.2.2 Communication protocol

This criterion is concerned with the communication protocol which is used in the command&control between botmaster and bots [63, 65]. In most of the literature, botnets are assumed to communicate through just three protocols, namely IRC, HTTP and P2P.

However, additional communication protocols have also been used in transmitting command&control in botnets. So botnets can be distinguished into IRC Protocol, HTTP Protocol, P2P Protocol, SMS Protocol and Push notification services.

- IRC protocol: This is the most common and oldest protocol used by botmaster to communicate with their bots. The IRC protocol is mainly used to support simple text–based chatting environments and has been designed to support not only one-to-many conversations but also one-to-one conversations. This feature of IRC allows botmaster to deploy command&control simply. However, with the enhancement of network security awareness, more and more businesses or individuals start to greater use of network firewall and anti-malware software. Default TCP service port for IRC is 6667, and it can be easily blocked and filtered by security software or device. So IRC botnets are dying off [90].
- HTTP protocol: The use of HTTP as a botnet C&C communication protocol has increased in recent years. HTTP has the advantage of being the primary protocol for web browsing, which means that botnet traffic may be harder to detect and block. Hence, with the use of the HTTP protocol, which is the most popular Internet traffic, botnet usually bypass security devices. Such as Andbot [28] can bypass the warning of traffic monitoring software to access background Internet.
- P2P protocol: the P2P network is a distributed and decentralised network architecture.The individual nodes in the network act as both suppliers and consumers of resources. Recently, more advanced botnets have used P2P protocols for their communications. The main advantage of P2P protocols is that they can avoid single-failure of centralised botnets [91].
- SMS protocol: The SMS protocols is one of the ideal C&C protocol for communication in mobile botnets. It is because SMS is supported nearly by all the mobile phones and is quite simple and reliable [49].

- Push notification services: This protocol establishes a style of Internet-based communication where the request for a given transaction is initiated by a publisher or a central server [92]. Moreover, the clients are designed to receive the message passively. It is contrasted with *pull notification service* and the clients initiate the request for transmission of information from the server. This technology has been used in different smartphones platform widely, such as Apple's Push Notification Service (APNS) [93] for iOS, Blackberry's Push Service (BPS) [94], Google Cloud Messaging (GCM) [95] service for Android, Microsoft's Push Notification Service (MPNS) [96] for Windows Mobile Phone, and Nokia's Notifications API (NNA) [97] for Symbian devices. Shuang et al. [49] proposed the use of this protocol for mobile botnets using C2DM [98] which is an Android push notification service.

2.1.2.3 Infected target platform

Mobile botnets have already received much attention as a branch of botnets because of the widespread use of smartphones in social, business, and military [69]. So we can divide the botnet into conventional platform botnet and mobile botnet. Moreover, we will discuss mobile botnet in Section 2.5 in detail.

2.1.3 Lifecycle of botnet

Most of the surveys of botnets have given a description of a life cycle of botnets [69, 74, 81, 99]. Most phases in these life cycle models are common but there also some differences. However, within the ten years of botnet development history, the mechanism of current botnets has become more complex. So we want to divide the phases in the life cycle of botnet into two parts: basic phrases and enhance phrases. Moreover, we present

an adapted version of botnet life cycle in Figure 2-2 based on the literature [69, 74, 81, 99].

Basic phrases are the phrases that must be set up by botnets for performing an attack to bots or other targets. Different names have identified such phases. For this survey, the basic phrases of the botnet lifecycle are the Infection phase, Connection phase, Control phase and Attack phase. These phases are shown as in Figure 2-2 with a solid block.

Additional phases, which we call "enhance" phases, are phases which can be included optionally for increasing the performance of a botnet. These phases are the Second Injection, Propagation, Maintenance and Updating phase and are shown in Figure 2-2 with dotted blocks.

Firstly, botmaster should develop specific botnets malware programs to infect the hosts so that these hosts become bots (1: Infection phase). There are many approaches to infect the vulnerable hosts. Some botmasters inject the malware code into attractive software and publish them to wait for downloading and running. This method usually used in mobile botnet malware [58]. However, the majority of botnet infect vulnerable hosts actively through what we call "Second Infection phase" (2: Second infection phase). More specifically, during the initial infection phase, the botmaster just searches for some target hosts for known vulnerability and, if it identifies any, it infects them with a script, which, in the secondary infection phase, is executed by the infected hosts to fetch the actual botnet malware programs from the malware server [69, 81]. Because of the diversity of protocol in secondary infection, this type of infection is more difficult to detect.

In addition to direct infection, botmasters may also take advantage of propagation mechanisms to expand the number of bots in botnets (3: Propagation phase). At this phase, once hosts are infected by botnet malware, they continue to try to spread the

malware thus saving the time and reducing the workload of botmaster in enlarging the botnet [46, 64].

Although there are different types of implementations for botnets, they all must realise the connection between bots and botmaster. It is designated as the Connection Phase in [81], or Rallying Phase in [100] (4: Connection phase). Once the connection is established between them, bots can get commands and updates from botmaster. Also, the botmaster can receive reports from bots through the connection channel (5: Control phase). When the botmaster wants to attack the infected bots, it sends commands to these bots performing the Attack phase (6: Attack phase). The last phase of the enhanced life cycle is the maintenance and updating phase. This phase is necessary if the botmaster wants to keep control of the infected bots with continuous system update for a long time. For example, they may need to upgrade the binary of botnets malware to evade constantly updated detection techniques and to change new, different C&C server for concealing themselves [81]. The connection and control phases in Botnet life cycle are unique to other malware, so our detection system focuses on these two phases.

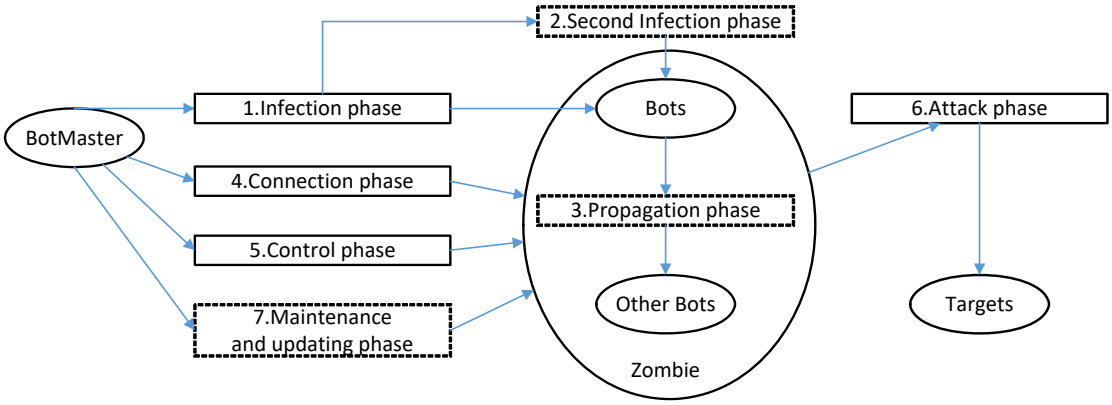


Figure 2-2 - Adapted version of botnet lifecycle

2.1.4 Comparison different type of malware

There are various types of malware that have different ways of propagating and infecting. Several of the more commonly known types of malware are worms, Trojans, bots and so on. We will compare these types of malware in this section.

Worms: Worm has the ability to replicate functional copies of themselves to perform the similar type of damage. Meanwhile, worms are standalone software and do not require a host program or human help to propagate. The propagating methods include exploiting a vulnerability on the target system and using social engineering to trick users to execute them [101]. *PrettyPark* is one of typical worm which spread by email and tries to send itself to the email addresses in registered address book periodically [102].

Trojans: A trojan is one type of malware that looks legitimate. The main difference between Trojans and worm is that Trojan does not have replication ability generally. They usually spread through user interaction such as opening an email attachment or running an infected file. The major task of a Trojan is to provide backdoors for other malicious programs to enter host system then destroy or steal valuable data without permission [103]. *JS.Debeski* is one of typical trojan which can delete several important system files [104].

Bots: Bots or robots are automated processes that interact with other network services without the need for human interaction. The general usage of bots includes gathering information, stealing passwords, relaying spam and launching DDoS attacks. Bots can be used for either good or malicious intent. A malicious bot is designed with the self-propagating feature to infect a host and can communicate with the control server. Depending on the remote control, their infection rate and the tactic is more effective than that of worms [105]. *Zeus* is one of the typical bots which is used to harvest banking credentials and financial information from users of infected devices [106].

2.2 Conventional botnet detection techniques

In this section, we focus on traditional botnets, discussing firstly known incidents/attacks of conventional botnet malware in the last ten years, and then presenting botnet creation and detection approaches. We also present a taxonomy of conventional botnet detection techniques and give a comparison of such techniques based on different criteria.

2.2.1 Incidents

A conventional botnet is originated from the IRC-based botnet, and the first recorded botnet is Eggdrop which was published in 1993 and last updated at 2011 with Eggdrop1.6.21 [107]. The early botnet is designed based on IRC channel on account of the featured protocol as well as the unencrypted and span long time periods connection between client and server. Agobot [108] and SDBot [109] are two typical IRC botnet which has drawn much attention. Agobot, also known as Gaobot, is a family of computer worms whose first version was written by Axel Ago Gembe in 2002. After that, the number of versions rapidly increased to around 1200 in two-year which leads to makes Agobot a challenge to examine [110]. SDBot botnet appeared around as early as 2004, but it continues to make waves still now [111]. Kharouni et al. [109] carried out some researches special for the variant of SDBot - BKDR_SDBOT.COD and gave a detail of the botnet mechanism.

Table 2-1 - The accidents of conventional botnet

Year	Name	Estimated Size	Brief Introduction	Reference
2004	Bagle	230,000	Bagle is a mass-mailing computer worm affecting all versions of Microsoft Windows.	[112, 113]
2006	Rustock	150,000	Rustock is a multi-component family of rootkit-enabled backdoor Trojans, which were historically developed to aid in the distribution of 'spam' e-mail.	[114, 115]
2007	Srizbi	450,000	Srizbi was the world's largest or second-largest botnet depending on expert reports before November 2008.	[116, 117]
2007	Akbot	1,300,000	Akbot is an IRC controlled backdoor, which provides an attacker with unauthorised remote access to the compromised computer.	[118, 119]
2007	Cutwail	1,500,000	The Cutwail botnet is a botnet mostly involved in sending spam e-mails which founded around 2007.	[120-122]
2007	Zeus	13 million	The primary target of Zeus is stealing banking information. It spreads mainly through drive-by downloads and phishing schemes.	[123-127]
2007	Storm	160,000	The name comes from the subject line about a recent weather disaster in an e-mail with this spam. At its height in September 2007, it could be as large as 50 million computers. However, it began to decline in late 2007.	[128-131]
2008	Waledac	80,000	The Waledac is a P2P-based botnet and was detected in December 2008. It takes advantage of real-world events and occasions and uses them to trick users into performing specific actions as social engineering.	[132-134]
2008	Sality	1,000,000	Sality is a P2P-based file infector that spreads by infecting executable files and by replicating itself across network shares. Sality was first discovered in 2003.	[135, 136]
2008	Conficker	10.5 million	Conficker was first detected in November 2008.It is robust and secure distribution utility for disseminating malicious software and has significant evolution from versions A to E.	[137-142]
2008	Asprox	15,000	The Asprox botnet was discovered around 2008; It was initially used exclusively for sending phishing emails and performing SQL Injections into websites to spread Malware.	[143-146]
2009	Festi	250,000	The Festi botnet is involved in email spam and denial of service attacks which first discovered around Autumn 2009.	[147-150]
2009	Grum	560,000	Grum is traced back to as early as 2008 and shut down in July 2012. It was reportedly responsible for 18% of worldwide spam traffic and capable of blasting 18 billion spam emails per day.	[121, 151, 152]
2010	TDL4	4,500,000	TDL-4 is the fourth generation of P2P-based TDL botnet. The new features of TDL-4 ensure the assessment to infected computers even in cases the botnet control centres are shut down.	[153-155]
2010	Kelihos	300,000+	The Kelihos botnet is a P2P-based botnet mainly involved in the theft of Bitcoins and spamming which is discovered around December 2010.	[154, 156, 157]
2013	Chameleon	120,000	The Chameleon botnet is notable for the size of its financial impact: at a cost to advertisers of over 6 million dollars per month. It is also the first botnet found to be impacting display advertisers at scale.	[158-160]

Except for the IRC-based botnet, new generation botnet later emerged with different communication protocols such as HTTP and P2P protocol. The Zeus and Conficker are recognised as the largest scale of botnets in last ten years. Zeus [127] is an HTTP-based botnet which is estimated to infect millions of compromised computers (estimated 13 million publish by Microsoft at 2012 [126] and approximately 3.6 million only in the United States [125]). Conficker, also known as Downup, is a P2P-base botnet [142]. Because of the advanced malware techniques, it is difficult to estimate the size of the botnet. However, it is regarded as one of the largest known botnets which infected around 10.5 million until July 2009 [141] and could potentially infect 300 million [140]. Table 2-1 shows the majority of the most threatening conventional botnet in recent ten years.

2.2.2 Botnet creation techniques

One of the main reasons that make network security more vulnerable is the continuous update of the malware software and the appearance of new invasion technologies. Consequently, prediction of new means of attack is one of the keys and open research direction in network security.

In the research of botnet, there is also some literature which concentrates on the creation of designation of the new type of botnet or botnet architecture. To be well prepared for future attacks, it is not enough to study how to detect and defend against the botnets that have appeared in the past [88]. The research of botnet creation is very meaningful, and it can not only make some predictions of the possible botnet in future but also increase our understanding of the mechanism of botnets. From an operator's perspective, understanding the deployment strategy of a botnet is critical for defending against malicious attacks on an operational network. Certainly, it is also possible that this research may be used by some hackers to deploy a more stealthy and robust botnet and enhance their control of botnet [58].

Wang et al. [88] present a design of an advanced hybrid P2P botnet which was aimed to construct a botnet which is harder to be shut down, monitored, and hijacked and perform more harmful attacks. Considering some problems faced by current botmaster and many network related issues, they provide a summary the six features should be realised in the next generation botnet. The hybrid P2P botnet contains two groups of bots: the server bots which can be accessed from the global Internet and the client bots which do not accept incoming connections. Based on this structure, they design a complete set of botnet components including Command & Control Architecture, initial construction and advanced construction. The robustness and the possible defending strategies are discussed at the end of the paper.

Starnberger et al. [161] have designed and implemented a P2P based botnet named Overbot to address the weakness of current botnet designs. The stealth control and command channel and the encryption of this channel are the main focus of their work. Botmaster does not reveal any information when capturing about other nodes of the botnet. The message sent by each bot should be encrypted by a public-private key pair owned by the botmaster so the botmaster can only identify them. Even though they present a more stealth botnet protocol, they also discuss possible countermeasures with this type of botnet. One of the methods is a statistical analysis of the requests and the nodes in botnet will issue more requests than normal nodes. The other one is probing the connected nodes by the captured nodes.

Liu et al. [162] introduce a recoverable hybrid C&C botnet named CoolBot. CoolBot combines the decentralised hybrid P2P C&C (HPCC) and the centralised URL Flux-based C&C (UFCC) to enhance and coordinate C&C mechanisms dynamically. The URL Flux-based C&C is established on Web 2.0 services, such as microblogs and social network sites and depends on the *Username Generation Algorithm* (UGA) to get the actual command published by the botmaster. They have also analysed the recoverability of the two C&C architectures. Through the simulation evaluation for the botnet, they draw

a conclusion that CoolBot is not only defending against some popular attacks but also could recover the C&C channel even if the majority of critical resources are destroyed within a tolerable delay.

2.2.3 Taxonomy of detection techniques

With the expansion of botnets and new types of then, botnet detection has become a major research thread in the last ten years. Botnet detection is regarded as the first and primary action to combat with this network security threat. Numerous botnet detection architectures and methods have been proposed by researchers in around the world.

Several botnet detection taxonomies have proposed in the literature [63, 64, 69, 71, 81, 163]. The survey in [63] divided botnet detection and tracking methods into two main approaches. The first approach is based on Honeynet. The second is based on passive traffic monitoring by observing data traffic in the network to look for suspicious communications that may be provided by bots or C&C servers. Tyagi et al. [163] describe a similar taxonomy, but they refine the second approach with three more specific categories which are Behaviours-based detection, DNS-based detection and Data-mining based detection. They even divided the behaviour-based detection into signature-based and anomaly-based. The work in [64] proposed a taxonomy including Honeynet, IRC(Internet Relay Chat)-based Detection, DNS Tracking Detection which contains traffic analysis and anomaly activities analysis.

Feily et al. [81] consider although Honeynet is mostly used to understand the mechanism and technology characteristics of botnets, it is not enough to detect bot infection. Therefore, they just focus on passive network traffic monitoring and analysis and classify these techniques into signature-based detection, anomaly-based detection, DNS-based detection and mining-based detection.

Zeidanloo et al. [71] proposed a relatively improved taxonomy of botnet detection techniques which give a hierarchical classification structure. Then the survey [69] present a refinement for the taxonomy of [71]. Both of [69] and [71] propose the whole botnet detection technique can be classified into Honeynet-Based and Intrusion Detection System (IDS). The second one can be further divided into two categories: signature-based detection and anomaly-based detection which consists Host-based and network-based detection techniques. The survey in [69] has made a further distinction between active and passive monitoring in network-based detection techniques. The active monitoring relies on the dynamically injecting test packets into the network or sends data packets to servers for analysis. On the contrary, the passive monitoring just uses monitor devices to watch and analyse the traffic as it passes by [164]. For passive monitoring, there are numerous protocols for analysis which contain P2P, SMTP, IRC, and DNS. The hierarchical classification should be present as Figure 2-3 [69].

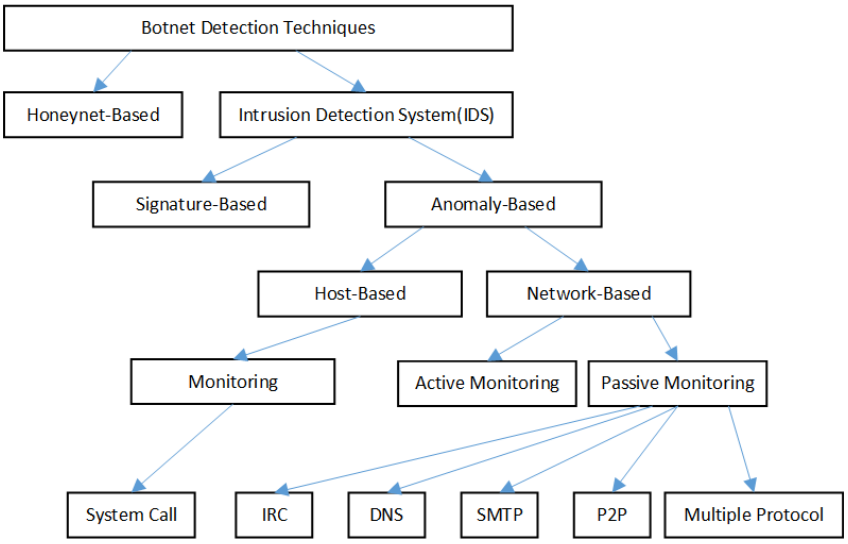


Figure 2-3 - The hierarchical classification of botnet detection techniques

According to existing study about the taxonomy of botnet detection techniques, we can present a taxonomy which contains Honeynet and Instruction Detection System (IDS). Then we introduce two primary dimensions for categorising IDS: the content of detection,

the range of detection. The content of detection is the data source which is used in detection techniques for analysis such as the traffic on the router, the log file and so on. The content of detection can be further divided into signature-based detection and anomaly-based detection. Moreover, the range of detection is used to distinguish the location of deployment of detection techniques which can be classified into host-based detection and network-based detection.

Table 2-2 presents a taxonomy of botnet detection techniques based on our additional dimensions. We will give an explanation for each class or dimensionalities in the allowing content of this section. Moreover, the specific detection techniques will be introduced at next section conforming to this taxonomy.

Table 2-2 - Taxonomy of botnet detection techniques

Honeynet		
[60, 62, 86, 87, 165]		
Instruction Detection System(IDS)		
Range / Content	Host-based	Network-based
Signature-base	[166]	[167-169]
Anomaly-base	[1, 170-172]	[173-179]

2.2.3.1 Honeynet

The honeynet is a very old technique which appeared in [180][157] initially. It is a technique for collecting information for botnets. A Honeynet is a network which contains several honey pots with intentional vulnerabilities; whose purpose is to invite attacks and analyse them. The ultimate goal of Honeynet is to understand the mechanism of attacker's activities and methods thus providing information to help increase network security. The administrator of Honeynet can open some ports or install botnet malware software on devices in Honeynet with specific monitor tools to trigger botmasters to attack the Honeynet. After a period and enough information is collected, the owners of Honeynets may be able to learn and understand the mechanism of the attack of different botnets. The

honeynet is the only type of tool for collecting information of botnet. Thus, they need to collaborate with some other techniques enabling botnet detection. McCarty [60] first proposed the method to detect botnet with Honeypot in 2003. They aimed at designing a firewall for honeypot to prevent inbound attacks without raising suspicions of the attacker. They also give an analysis for the IRC botnets monitored by the honeypot detection system. Then there is some literature discussed how to use Honeynet to realise botnet tracking and detecting [62, 86, 87, 165]. Rajab et al. [62] present infrastructure of honeypot to analyse botnets, and give a description of the inherent diversity of botnet activities. Barford et al. [86] analyse four widely-used Internet Relay Chat (IRC) botnet codebases by using honeypot. Their study reveals the complexity of botnet software which is the base for the defence strategies. Cooke et al. [87] set up an experiment to measure botnets on a real network and show the serious of botnet problem today.Vrable et al. [165] presented Potemkin, a scalable virtual honeynet system for botnet detection, which is inappropriate for long-term botnet tracking.Although Honeynet is useful in understanding botnets, it has some limitations for botnet detection [69].

- The Honeynet can only track limited scale of exploited activities. After the bot infected by malicious code, the botmaster can change their attack strategy at any time with new command control and even to update the malicious code injected to the bot to change the behaviours.
- The Honeynet cannot detect bots without using propagation methods other than those based on scanning, spam and web-driven downloads. More and more Botnet use the more diversified method to affect and communicate with bots.
- The Honeynet can only report information about the infected machines. The major function of Honeynet is monitoring and collecting information for analysis.

Therefore, we will not discuss the detail of the method in next section.

2.2.3.2 Signature-based

Signature-based botnet detection is the most widely used and mature detection technique at present. Nearly all the anti-malware software takes advantage of signature-based techniques to detect botnet malware. Just as its name implies, this technique detects botnets malware by the signature of the code or the data which is used in the botnet components. There are several kinds of literature have found some signature of specific botnet [167-169]. Goebel et al. [167] present a botnet detection technique mainly based on the signature of suspicious IRC nicknames, IRC servers, and uncommon server ports. Kugisaki et al. [168] confirmed that signature of connection between bots and IRC server when the server refused the connection from the Bot. Wurzinger et al. [169] make use of the signature of the response of bots after receiving commands from botmaster according to the same family of botnets.

Snort is a free and open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS) [166]. Through customization by configuration, the program can be widely used to detect a variety of probes or attacks. As a consequence, it also can be configured for recognising some special flow characteristic to achieve botnet detection.

The thesis [181] give a typical signature-based detection case for IRC botnet whose name is *Rxbot* [182]. Firstly, the features of data packet should be known before detection procedure. To obtain the feature of the *Rxbot*, we can run *Rxbot.exe* in the virtual machine as the bots and execute the instruction ".pencmd" to communicate with the server. Meanwhile, the traffic of IRC should be monitored at the host, and the feature of the *Rxbot* data packet is shown in

The features can be summarised in the following items:

1. The first 6 bytes is fixed: channel name to connect
   3a 72 42 6f 74 7c (:r Bot)

2. The first 6 bytes is fixed: channel name to connect
   3a 72 42 6f 74 7c (:r Bot)

3. The last 29 bytes is fixed: the response of the command
   3a5b434d445d3a2052656d6f7465207368656e6e2072656164792e0d0a

   (:[CMD]: Remote shell ready)

So after mastering the feature of the data packet, the detection system just need to apply some regular expression to filter the traffic of the host. Once matching successfully, the system can determine *Rxbot* has infected the hos*t*.

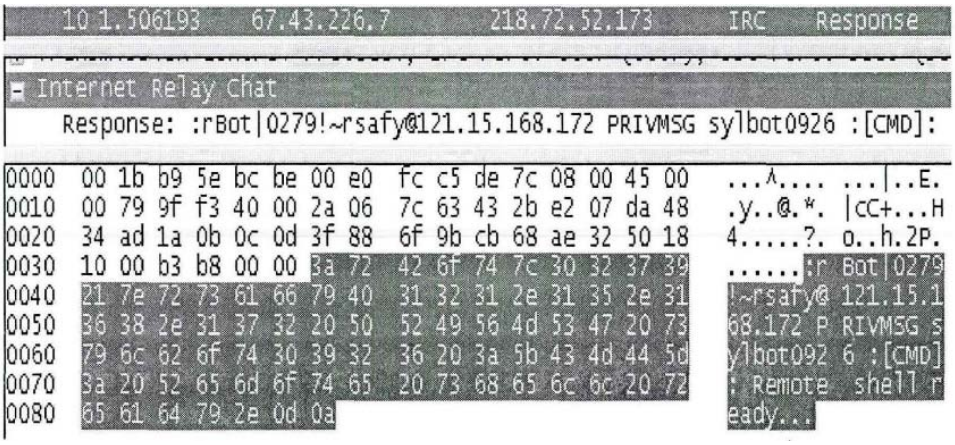


Figure 2-4 - The feature of Rxbot data packet

Goebel et al. [167] propose detection method named *Rishi* which is regarded as representative of signature-based techniques citing by many surveys of botnet detection. *Rishi* identifies whether the hosts are contaminated or not by the evaluation of IRC nickname. The IRC-based botnet is the earliest of botnets, and the mechanism was first revealed by [61]. This method can be divided into three steps. Firstly, the detection system collects traffic from the router and statistic the IRC channel. Secondly, connection objects should be created for each IRC channel to store relative information. This information includes the time of suspicious connection, IP address and port of suspected source host, IP address and port of destination IRC server, channels joined and utilised nickname. At

last, the nickname gathered by the system is passed to an analysis function that realises a scoring function to estimate whether malware infects suspicious input host or not. The higher the score a nickname evaluates, the more likely it is an infected bot malware trying to contact its C&C server.

Kugisaki et al. [168] aimed to observe new features of IRC-based bots. After monitoring the port which used by IRC channel, they discovered some different communication flow between the clients with specific IP address and other clients. Further, they conclude that the bots usually repeat transmitting "NICK" and "USER" until the IRC connection succeeds. Based on this characteristic, the detection system records the ratio of communication interval to IRC server to evaluate the risk of infection for the devices.

The literature [169] present a detection system that relies on the signature of detection models. The fact of botnets establishes the detection model that after bots receive commands from the botmaster, which will respond some message in a specific way. They divided the automatic model generation into three parts which contain command model generation, response model generation and mapping models into Bro signatures (Bro is a network intrusion detection system for monitoring suspicious or irregular events of network activity [183]). Through the analysis of 446 network traces, a total of 70 detection models are produced by the detection system which controlled by 18 different bot families (IRC1-IRC16, HTTP and P2P).

2.2.3.3 Anomaly-based

Anomaly-based detection is regarded as the most extensively studied research direction of botnet detection techniques and most efficient for detecting unknown botnet. The core idea of this approach is to compare the current status of devices or network with the normal situation, and if there is some difference between these situations, the anomaly-based detection system needs to judge whether there is a botnet infection or not

based on some procedure. For example, the paper [184] regard the log file of Windows Application Programming Interface (API) functions calls which are made by communication applications as the criterion to detect the botnet. They monitor the activities of bots by analysing the size of the log file generated by every bot. The normal situation is that different hosts have diverse changing curve. However, a high correlation between the hosts represents an abnormal situation.

The distinction between normal and abnormal situations is an actual thread of research efforts in this detection category. There are several techniques that advocate anomaly-based detection ([173-179]), and most of them consider network traffic anomalies. This is plausible, since the necessary communication between the botmaster and bots in a botnet creates some inevitable anomalies in the network traffic, such as abnormally high traffic volume and/or network latency in a period, network traffic channelled through special ports, network packages generated by some unusual system behaviour [61] and so on.

Other approaches are based on a different abnormal situation related to the correlation of bots (e.g.[184-189]). More specifically, there are possibly some similarities amongst the bots which belong to the same botnet. For example, bots perform similar communication pattern with same botmaster [189]. To analyse the correlation of each bot, they insert some statistical information into the traffic for monitoring. Once the similarity statistical information is detected in the traffic from different bots, the detection system will give some warning for the suspicious malware behaviours.

Owing to the acceleration of botnet update frequency and more and more new type of botnet, finding some methods to detect the unknown botnet is urgent. Therefore, abnormal-based detection techniques have become a promising research field in botnet detection.

Binkley et al. [190] present anomaly-based detection algorithm for IRC-based botnet mesh relatively early. The algorithm is based on the fact that botmaster needs scan the IRC-based botnet frequently to keep the contract with the bots. Therefore, the frequency of TCP connection establishment can reflect the probability of infected botnet. The paper proposes an index named TCP work weight which means the ratio of the auxiliary information in TCP connection. The Equation 2-1can compute it. The larger the TCP work weight, the higher the probability of infection by botnets.

Equation 2-1 TCP work weight equation

$$\omega = (S_s + F_s + R_r)/T_{sr} \quad (2\text{-}1)$$

The TCP synchronised data packet tuple contains (*IP source address, SYNS, SYNACKS, FINSSENT, FINSBACK, RESETS, PKTSSENT, PKTSBACK*). The ω donate the TCP work weight, the S_s denote the count of SYNC and SYNACKS, F_s denote the count of FINSSENT, R_r denotes the count of RESETS and T_{sr} denote the total count of TCP data packet.

Choi et al. [191] take advantage of the difference of group activities of DNS traffic between botnet network DNS and legitimate DNS. The group activity is the average proportion of same IP address in two IP lists which are requested at two different times with same domain name query. The similarity can be quantified through the Equation 2-2. Moreover, the more similar it is the more probability of infected by botnet malware.

Equation 2-2 Similarity between DNS traffic

$$s = 1/2 \times (C/A + C/B)(A \neq 0, B \neq 0) \quad (2\text{-}2)$$

In another anomaly-base detection method in [184, 185], the log file is regarded as the main monitor objective. The log file comes from the API socket function calls that are used by communication programs, and the detection system records the size of the log file. Through the comparison between the normal user behaviour and the botnet behaviour, there is a remarkable difference of the fluctuation curve of log file size. The paper also gives two cases to evaluate their method. First one is to compare the behaviour

of Internet Explorer and SDBot and the second one is monitoring the mIRC clients and the SDBot.

Gu et al. [175, 179, 192, 193] proposed a series of the abnormal-based detection system. The paper [192] proposes the BotHunter which detects the botnet by IDS-Driven Dialogue Correlation. Through tracking the two-way communication flow between internal assets and external entities by Snort, the detection system realises an evidence trail approach for recognising successful bot infections. The basic principle of the system is based on understanding Bot infection sequences and concluding the sequences into several independent dialogues. Then BotHunter can construct the model of infection dialogues process and assert a minimum of required conditions for bots infection. They even evaluate capabilities of the detection system in a virtual network and a live Honeynet and validate low false positive rates in two operational production networks. Based on the BotHunter, Gu et al. proposed two detection systems whose name are BotSniffer and BotMiner respectively [175, 179]. BotSniffer focuses on the analysis the correlation of activities and message in bots belonging to the same network. There are two core algorithms to evaluate the correlation: *Response-Crowd-Density-Check* Algorithm for message response and *Response-Crowd-Homogeneity-Check* for activity response. If the similarity of the different bots exceeds the threshold, the network will be regarded as infection with botnet by the system. They evaluate the BotSniffer on multiple network traces captured from campus network with only a total of 11 false positives (FPs) on four of the IRC traces and no FPs resulted from group analysis.

BotMiner [179] is a detection system which contains clustering analysis of network traffic for the protocol. There are two clustering processes in the system which can accurately and efficiently group similar C&C traffic patterns. *A-plane* clustering is designed for activity traffic, and C-plane clustering is designed for C&C communication traffic. Once the clustering results are obtained, the cross-plane correlation will be

performed to cross-check clusters in the two planes to find out intersections that reinforce evidence of a host being part of a botnet.

2.2.3.4 Host-based

Host-based detection techniques are techniques where the detection system collects information only from the host device without any communication with other devices in the same network. Early detection systems and anti-malware software are based on this technique (e.g., [1, 170-172]). Although there are some limitations, it is more convention and flexible to deploy the host-based detection system in real projects and more efficient against some specific types of attack such as *download attack* and *onset infection* [194].

Along with the enhancement of connection and diversity of the behaviour in the modern botnet malware, the host-based detection technique is no longer appropriate for comprehensive botnet detection system.

2.2.3.5 Network-based

Network-based detection techniques are the main trend of botnet detection system [173]. This technique concentrates on the traffic and communication on the network. Silva et al. [69] have classified the network-based techniques according to the communication protocol that they are based on IRC, DNS, SMTP, P2P and multiple protocol techniques. Several works focus on how to analyse the information of every protocol or data package in the network for botnet detection.

2.2.4 Comparison

2.2.4.1 Criteria of comparison

To compare the botnet detection approaches, we use some criteria for comparison. Because of some challenges for botnet evaluating, there is hardly any literature discussing

the comparison of the current botnet detection systems. The literature [170] enumerates some challenges for experiment with botnet detection systems. The different of data set and experiment environment is the biggest obstacle for comparing and evaluating current botnet detection system. Also, considering of the privacy of institutions, the experiment data cannot be shared with other organisations or persons.

However, we can make a summary of the evaluation criteria from the literature which propose the detection system with evaluation data. Because if the literature wants to prove their detection system is more effective or more efficient than others, they must present some result of the comparison in theory or experiment.

From the current botnet detection methods, there are three criteria which are widely used in detection techniques literature for comparison. The first one is *crossover error rate* (COER), the second one is *Detection Rate* and *False Positive* (FP), and the last one is *Receiver operating characteristics* (ROC) curve.

After explaining every criterion of comparison in botnet detection, we give a summary of these criteria in Table 2-3 with the detail of the criteria in every detection approaches.

Table 2-3 - The summary of the criteria for botnet detection

Detection approaches \ Criteria		ROC curve[1]			Crossover Error Rate	False Positives (Rate)	Detection Rate
		FPR	TPR	Youden Index			
Log Correlation Based [184]		0.000%	79.000%	0.79			
BotTrack [174]	BotTrack-Kademlia	0.300%	97.000%	0.97			
	BotTrack-Chord	6.000%	100.000%	0.94			
	BotTrack-Koorde	6.000%	75.000%	0.69			
Disclosure [195]	(N1, MinFlows=20)	6.000%	86.027%	0.80			
	(N1, MinFlows=50)	9.000%	94.521%	0.86			
	(N2, MinFlows=20)	8.000%	82.740%	0.75			
	(N2, MinFlows=50)	8.000%	84.000%	0.76			
BotHunter [192]	BotHunter-SLADE	1.999%	99.200%	0.97			
	BotHunter-PAYL	0.927%	72.200%	0.71			
BotMosaic [176]	SdBot-500ms				$2.8 * 10^{-3}$		
	SdBot-2000ms				$3.52 * 10^{-13}$		
	SpyBot-500ms				$2.32 * 10^{-8}$		
	SpyBot-2000ms				$7.55 * 10^{-11}$		
BotMiner [179]	IRC-rbot					0.003(Rate)	100%
	IRC-sdbot					0.003(Rate)	100%
	IRC-spybot					0.003(Rate)	75%
	IRC-N					0(Rate)	99.6%
	HTTP-1					0.003(Rate)	100%
	HTTP-2					0.003(Rate)	100%
	P2P-Storm					0(Rate)	100%
	P2P-Nugache					0(Rate)	100%
Rishi[167]						5	78.43%
Automatically Generating Models[169]						11	88%
BotSniffer[175]						0.0016 (Rate)	100%

- **Crossover Error Rate**

Crossover error rate (COER) [176] is a comparison criterion for different biometric devices and technologies. The COER is based on two rates which are the false acceptance rate (FAR) and the false rejection rate (FRR). The FAR of the system typically means the

[1] We use the Youden Index [238] to select the best performance point on the ROC curve.

ratio of the number of false acceptances which should be rejected. Moreover, the FRR is contradictory metric with FAR which measures the ratio of the number of false rejections which should be accepted. When the two ratios are equal and cross over, the point is COER (It is also called as CER [196]).

In botnet detection, the FAR express the ratio of detection system makes a faulty judgment to recognise normal hosts as bots. On the contrary, the FRR express the ratio to make a mistaken identity for bots as normal hosts. So, the botnet detection techniques should lower the two metrics as much as possible and the lower COER of the detection system, the better detection effect.

- **True Positive Rate, False Positive Rate and Precision**

The *True Positive Rate* (TPR), the *False Positive Rate* (FPR) and *Precision*. These measures are used typically for the evaluation of ML based classification [39, 174, 178, 179, 186, 192, 197-199]. If we regard normal as positive and infect as negative respectively: True positive (TP) is the number of normal data that were correctly classified by an algorithm. True negative (TN) is the number of infect data that were correctly classified. False positive (FP) is the number of normal data that were incorrectly classified by an algorithm. False negative (FN), is the number of infect data classified as normal.

Based on these measures TPR, FPR and Precision are calculated as follows:

Equation 2-3 Equation True Positive Rate

$$TPR = TP/(TP + FN)\ (aka\ Recall) \tag{2-3}$$

Equation 2-4 False Positive Rate

$$FPR = FP/(TN + FP) \tag{2-4}$$

Equation 2-5 Precision

$$Precision = TP/(TP + FP) \tag{2-5}$$

The total number of bots within the wild network is hard to estimate. Therefore, the ratio of detected bots within the complete network is difficult measured. Some literature

gives some other evaluations methods to verify their system. The paper [113] gives a comparison result with the detection method Blast-o-Mat [147] and gives the better performance to it.

- **Receiver Operating Characteristics**

Receiver Operating Characteristics (ROC) is a more comprehensive evaluation criterion for botnet detection system. The ROC was first used for signal system and then widely used in medicine, radiology and social sciences. It is also proved useful for the evaluation in computer science. The ROC is based on two important concepts: true positive rate (TPR) and false positive rate (FPR). The true positive rate reveals the number or the ratio of successful detection samples which is also called hit rate or sensitivity [200].

The ROC curve is the graph whose horizontal axis is false positive rate and vertical axis if true positive rate (TPR). With the variation of the threshold setting, the detection system will get a different pair of TPR and FPR. Ideally, FPR is lowest, and the TPR is highest. However, the threshold at the point of tangency on the ROC curve is the best configuration. The area under the ROC curve is also regarded as a criterion for detection effect.

2.2.4.2 The result of comparison

Because this is not a uniform criterion in botnet detection techniques and these is no organisation or institution proposing some standard for evaluation and comparison of the botnet detection techniques, we just summary the comparison based on the evaluation of the various botnet detection literature as far as possible.

The papers [174, 184, 192, 195, 201] evaluated their detection system based on ROC curve or give enough experimental data for ROC analysis. Al-Hammadi et al. [184] set

different percentage of log file size as a threshold from 0 to 100(%) to get ROC curve for their Log Correlation based Detection system.

Bilge et al. [195] evaluate their detection system Disclosure at two networks: Inter-University Network (N1) and Tier 1 ISP (N2). There are two types of the threshold for ROC curve. The one is ClassThresh which is the boundary separating benign scores from malicious scores and the other on is the MinFlows which is the minimum number of observed flows to a particular server to provide accurate results. Their set the MinFlows at two values: 20 and 50 to get 4 ROC curve graphics.

François et al. [174] evaluate BotTrack, a P2P botnet detection system, on three types of P2P network (Kademlia, Chord, Koorde). Moreover, the number of the bots known is also a factor to affect the ROC curve. They just only summary the situation of 0% bots known.

Despite Gu et al. [192] did not give a ROC curve to evaluate their detection system BotHunter, they present the performance of the system with PAYL (a payload-based anomaly detector [202]) and SLADE (Statistical Payload Anomaly Detection Engine) based on the difference of desired false positive rate. So we transfer these two performances into ROC curves to make a comparison with other detection systems.

2.3 Network traffic anomaly detection technique

According to Section 2.2.3, there are two types of detection techniques for botnet detection based on the network traffic, i.e., signature and anomaly based detection techniques. Because of the rapid update of the malicious code on the mobile device and the unstable of the mobile network, the anomaly-based detection technique is better than signature-based detection techniques in research and application. This section will introduce the network traffic anomaly detection techniques that can be used for mobile devices and network.

2.3.1 Introduction of anomaly detection and intrusion detection

The first uniform architecture of intrusion and intrusion detection was proposed in 1980 by a report on *Computer Security Threat Monitoring and Surveillance* [203]. The intrusion detection system can detect the malicious behaviour by the unauthorised access through monitoring the configuration of the host. If the host is under attack, the system can raise the alarm for the users of the host to take corresponding measures to protect the host.

The intrusion detection system can be classified into Signature-based detection and Anomaly-based detection. The signature-based detection technique can only detect the harmful behaviours comparing with predefined black or white behaviour feature library. However, the anomaly-based detection technology is focusing on identifying the anomaly behaviours which are not consistent with the normal behaviours of the host. Apparently, the signature-based detection techniques rely on the completeness of the feature library, which can only detect the known attack. The anomaly-based detection can identify the unknown abnormal behaviours independently of the specific information for the malware.

For mobile botnet detection, the malicious code can be updated by the controller frequently. Moreover, the malware feature library on the mobile devices cannot be updated in time with the unstable cellular network. So, signature base detection will lose effectiveness when the remote update changes the feature of the detection target. Furthermore, most of the malware tend to change their feature to evade the detection of anti-virus software. Therefore, the anomaly base detection is suitable for the mobile botnet detection system. Even the malware is upgraded, the well-designed anomaly based detection system can distinguish them with normal behaviours.

Anomalies are patterns in collected data that are consistent with a well-defined database of normal behaviour that can be divided into the following categories [204]:

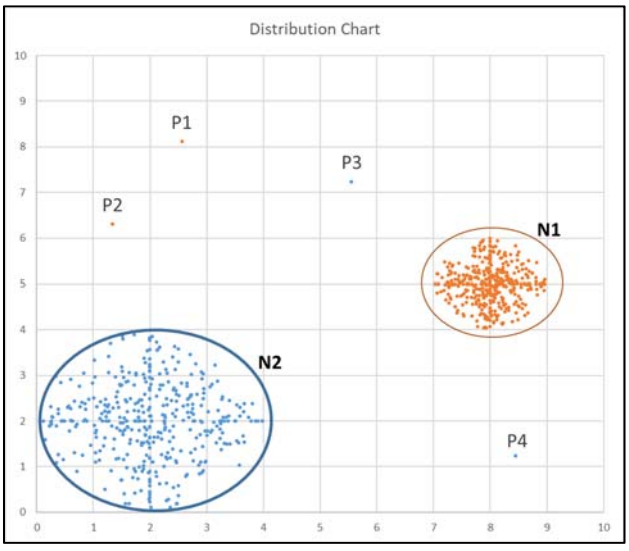


Figure 2-5 - The example of point anomalies

1. Point Anomalies

This type of anomaly cares about the individual data instance. If a single data instance can be regarded as different on the rest of data, then this instance can be termed as a point anomaly. The example can be seen from the Figure 2-5 and P1 to P4 are point anomalies hence they lie outside of the boundary of N1 and N2 which are considered as the normal

range. Point Anomalies is the simplest type of anomaly and is the focus of the majority of research on anomaly detection.

2. Contextual Anomalies

If a data instance represents unusually in a particular context (but not other contexts), then it is termed as a contextual anomaly or conditional anomaly. The notion of a context is based on the structure of the data set in specified scenario. The Figure 2-6 provides an example of contextual anomaly which illustrates the variation of traffic flow over time. The quantity of traffic between 10:00 AM to 2:00 PM reach 160 which is higher than normal level in the other period. This abnormal situation can be regarded as a contextual anomaly.

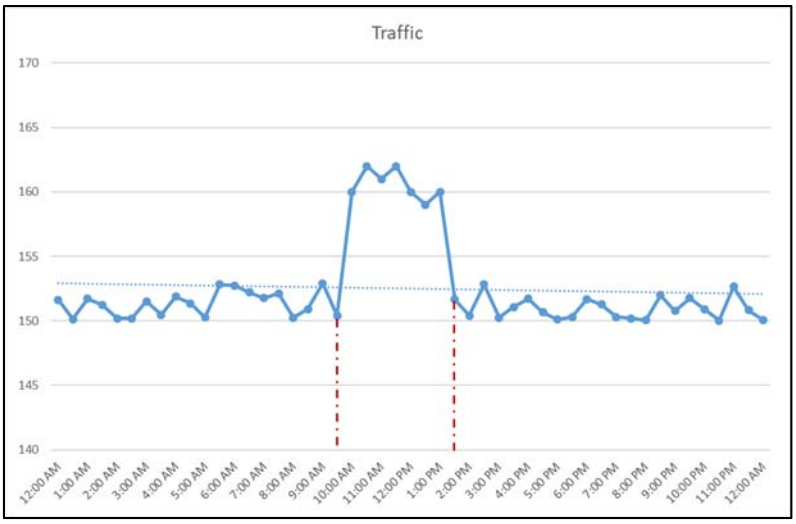


Figure 2-6 - The example of contextual anomaly

3. Collective Anomalies

If a collection of related data instances is abnormal with respect to the entire data set, it is termed as a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous [204]. For example, if the situation of high traffic in the period from10:00 AM

to 2:00 PM occurs every day in one month (Figure 2-6), then it could be regarded as a collective anomaly.

The general procedure of anomaly detection techniques contains three steps: (1) Establish normal behaviours library. (2) Capture the data. (3) Comparing the captured data with the normal behaviours library to find the anomaly instances. Although this general procedure for realising anomaly based detection of misbehaviour seems to be simple, it is complicated by the following key factors:

1. It is difficult for defining a normal pattern or region with all possible normal behaviour. Hence, the boundary between normal and anomalous behaviour is often not very clear.
2. Updates of malicious code may change what constitutes abnormal behaviour and make normal patterns no longer apply for the updated malware.
3. The very concept of an anomaly is different in different application domains. Thus, applying a technique developed for one domain to another is not straightforward.
4. There is a major issue that whether the labelled data is available for validation or training of models used by anomaly detection techniques.
5. There may be noise contained in the training data for building the behaviour pattern which is similar to the actual anomalies.

2.3.2 Classification of anomaly-based detection technique

2.3.2.1 Classification based

There are two sets of data training and testing datasets for classification based anomaly detection method. According to analyse the training data by some algorithms, the classification can generate a model which also known as classifiers. Then the classifier is used to classify the testing datasets into different classes [205, 206]. Therefore, the

classification based anomaly detection can be processed by two phases: training phase and testing phase. The theoretical basis of classification based anomaly detection techniques is assuming that the classifier can distinguish the benign and malicious classes with the features extracted from the dataset [204].

The classification based anomaly detection techniques can be classified as one-class and multi-class. The difference between the two categories is the number of the class in the training dataset. For one-class anomaly detection, there is only one normal class for all training data. So any instance in testing dataset fall out of the outlier of classification of the training dataset is an anomaly. As for multi-class anomaly detection, the training dataset can be labelled as more than one normal class. So the testing instance needs to be compared with by any of the normal class patterns to determine whether the instance is abnormal or normal.

The key issue for the classification-base anomaly detection technique is how to build a suitable classifier from training dataset to classify the testing dataset into different class accurately. In general, the machine learning technique is usually used for classification to increase the accuracy of the classifier that will be introduced in next section.

2.3.2.2 Nearest neighbour based

The theoretical basis of nearest neighbour based anomaly detection techniques is assuming that the when we map some features of observation data to the coordinate system, the normal instances will occur neighbouring location. Meanwhile, the anomalies instances will occur in the other neighbouring location far from the normal instance set.

The key issue for the nearest neighbour based anomaly detection technique is how to compute the distance between the different instance in the dataset. In general, there are several methods to calculate this distance as follows:

1. If the features are continuous, Euclidean distance [207] is usually used than others [206]. The Equation 2-6 can compute the Euclidean distance of point p and q.

Equation 2-6 Euclidean distance between two points

$$Dis_{Eulid}(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2-6}$$

2. If the features are categories, the simple matching coefficient (SMC) is often used. SMC is a statistic for comparing the similarity and diversity of datasets which is computed by the following Equation 2-7 [208]. Sometimes, the more complex distance measures can be used [209].

Equation 2-7 Simple matching coefficient equation

$$SMC = \frac{number\ of matching\ attributes}{number\ of\ attributes} = \frac{f_{11}+f_{00}}{f_{10}+f_{01}+f_{11}+f_{00}} \tag{2-7}$$

where:

$$f_{00} = number\ of\ attributes\ where\ x\ is\ 0\ and\ y\ is\ 0$$
$$f_{01} = number\ of\ attributes\ where\ x\ is\ 0\ and\ y\ is\ 1$$
$$f_{10} = number\ of\ attributes\ where\ x\ is\ 1\ and\ y\ is\ 0$$
$$f_{11} = number\ of\ attributes\ where\ x\ is\ 1\ and\ y\ is\ 1$$

3. If there is more than one feature for the dataset, we usually compute the distance of single feature and then compute the combined distance.

Broadly speaking, the nearest neighbour based anomaly detection techniques can be divided into two groups: K-nearest neighbour (KNN) and Relative Density (RD). The KNN algorithms are one of instance-based machine learning which will be introduced in next section. Moreover, the RD is the ratio of the density of a substance to the density of a given reference material [210]. So the RD based anomaly detection technique estimate the density of the neighbourhood of each data instance. An instance that locates in a neighbourhood with low density is declared to be anomalous while an instance that lies in a dense neighbourhood is declared to be normal. The Literature [211] proposed the Local Outlier Factor (LOF) which is a method to detect the anomaly using the relative

density. For a normal instance, its local density will be similar to the local density of its neighbours. While for an abnormal instance, its local density will be lower than that of its nearest neighbours. Tang et al. [212] proposed an improvement LOF method that is called Connectivity-based Outlier Factor (COF), whose the selection of K neighbours is different with LOF. The selection process is as follows: we first add one instance to the neighbourhood set. Then we add the instance whose distance to the existing neighbourhood set is minimum among all the other instances of the neighbourhood set. The rest can be done in the same manner, and the number of the neighbourhood will reach size k. The research of [213] enhances the LOF to detect the spatial anomalies in climate data. Moreover, Ye et al. [214] improve the LOF to support the anomaly detection with category attribute.

The advantages of nearest neighbour based techniques are as follows: (1) The most of the advantage of nearest neighbour based techniques is that they do not make any assumptions regarding the generative distribution for the data. Instead, they are purely data driven. (2) When the training dataset is very small, using semi-supervised techniques, perform better than unsupervised techniques. (3) The method can be used for adapting to the other different data type, and it is the only required definition of an appropriate distance measure for the given data.

The disadvantages of nearest neighbour based techniques are as follows: (1) If some of the normal instances do not have enough close neighbours, the technique cannot label them correctly based on the unsupervised learning algorithm. (2) If some of the normal instances do not have enough similar close neighbours, the false positive rate is very high based on the semi-supervised learning algorithm. (3) The calculations of the distances between each instance and other instances have a high time complexity. (4) Definition of distance measures between instances is a challenge when the training data is complex. However, the performance of the technique greatly depends on the distance measure.

2.3.2.3 Clustering based

Clustering based technique is used to converge similar instances into same clusters. Two different approaches can be used for clustering-based anomaly detection. They are unsupervised clustering and semi-supervised clustering. Clustering based anomaly detection techniques can be divided into three categories.

1. The first category is based on the assumption: Normal data instances can form a cluster in the data, on the contrary, abnormal instances do not belong to any cluster. This technique trains dataset by the clustering algorithms and determines the instances that do not belong to any cluster as an abnormal instance. The literature of [215-217] use the clustering algorithms that do not force all instances to be part of one cluster.
2. The second category is based on the assumption: Normal data instances are near the centre of the cluster, on the contrary, abnormal instances are far from the centre of the cluster. This technique first uses the clustering algorithms to achieve clustering and then compute the distance to the centre of the closest cluster for every instance as discrimination index. The literature [218] use Self-Organizing Maps, K-means Clustering and Expectation Maximization to cluster the training dataset and detect the anomaly by computing the distance to the centre point of the cluster.
3. The third category is based on the assumption: Normal data instances can form dense and large clusters, on the contrary, abnormal instances can only form sparsely small clusters. The anomaly instances probably constitute some small clusters. Therefore, the anomaly cluster can be detected through defining the threefold of size and dense of the cluster. The literature [219] proposed the cluster-based local outlier factor (CBLOF) to detect the anomaly by comparing the size of the cluster and the distance to the centre point of the cluster.

The advantages of clustering based techniques are as follows: (1) Clustering-based techniques can operate in an unsupervised mode. (2) Clustering-based techniques can be adapted to other complex data types through replacing the clustering algorithm to process the particular data types easily. (3) The time complexity of the testing phase is very low because the test instances only need to compare a small number of cluster.

The disadvantages of clustering based techniques are as follows: (1) Performance of clustering based techniques depends on the effectiveness of clustering algorithm heavily in recognising the cluster structure of normal instances. (2) Some clustering algorithms force every instance to be assigned to one of the clusters that might result in anomalies setting assigned to a large cluster. (3) Some clustering based techniques are practical only when the anomalies do not form significant clusters among themselves. (5) The computational complexity is high which is usually $O(N^2)$ because of the clustering algorithms.

2.3.2.4 Statistical based

The theory of statistical-based anomaly detection technique is based on the assumption: normal data instances occur with relatively high probability by the stochastic model, on the contrary, abnormal instances occur with relatively low probability. In general, the technique constructs the statistical model of the normal instance and then apply the model to the testing instances to find whether the instance has low probability occurring in the model that is regarded as abnormal. There are two types of statistical techniques: parametric techniques and non-parametric techniques.

Parametric statistics are statistics which assumes that the normal dataset has come from a type of probability distribution with parameters Θ and probability density function $f(x, \Theta)$ *(x is an observation).* The parameters Θ are estimated from the training dataset. Then we assign an anomaly score for every test instance using the function above. There

are three models for the distribution of the training dataset: Gaussian Model-Based, Regression Model Based and Mixture of Parametric Distributions Based.

Non-parametric statistics are statistics not based on parameterized probability distributions. Unlike parametric statistics, non-parametric statistics make no assumptions about the probability distributions do not have fixed number of parameters. There two specific statistical techniques can be used to detect the anomaly: Histogram Based and Kernel Function Based.

The advantages of statistical techniques are: (1) Performance of statistical techniques is high when the training dataset follow a certain distribution. (2) The anomaly score calculated by statistical technique can be used as additional information to make decision. (3) Statistical techniques can use unsupervised setting without any label information of data set if the distribution is robust for the training and testing dataset.

The disadvantages of statistical techniques are: (1) One of the most disadvantages of statistical techniques is that they depend on the assumption that the dataset follows a particular distribution. (2) How to choose the best statistic is hard work [220], after confirming that the dataset is followed a certain distribution. (3) The different feature may follow a different distribution. Therefore individual feature of the anomaly instance maybe is very frequent, but the combination these feature is very infrequent.

2.3.3 The future trend of anomaly detection technique

There are several promising trends for anomaly detection technique:

1. The number of research of contextual and collective anomaly detection technique will increase.
2. More distributed anomaly detection technique will occur because of the across localisation data.

3. The real-time requirement of the detection will increase because the detection techniques will be used for the wireless network and mobile devices.
4. The anomaly detection techniques will be a requirement to think about the data privacy.

2.4 Machine Learning

Machine Learning is widely used in the anomaly detection. This technique is also the theoretical basis of our mobile botnet detection system. So we will introduce the machine learning technique in detail in this section.

2.4.1 Introduction of machine learning algorithm

The earliest definition of Machine Learning (ML) can be tracked back to 1959, Arthur Samuel defined it as a "Field of study that gives computers the ability to learn without being explicitly programmed" [221]. A widely more formal definition is provided in [222] as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

Machine Learning is a multidisciplinary cross-discipline, involves the theory of probability, statistics, approximation theory, convex analysis, algorithm complexity theory and so on different subjects. It studies how to use computer simulation or human learning behaviour, to gain new knowledge or skills, reorganise the existing knowledge structure to improve its performance.

Machine learning has been more widely adopted in many areas such as medicine, chemistry, bioinformatics, business and computing. It is the core of artificial intelligence and the basic way to achieve the computing intelligence. In the present information age, the application of machine learning has penetrated into every field of computer science. It can make the search engine more accurate and intelligent. The computer can recognise the speech and handwriting precisely by using the machine learning. It also can be used for control the robot locomotion. The anomaly detection is one of the most machine learning application. Moreover, compared with traditional detection algorithms, the novel ML algorithms can significantly increase the accuracy of detection without reducing the processing speed.

In general, the machine learning can be classified into four broad categories depending on whether the learning signal is available for the algorithm: Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning [223].

1. *Supervised learning:* In supervised learning, the input data in each group which are referred to as “training data” has a clear identity or label, such as “spam” and “not spam” in the anti-spam system and digital value of picture in the number handwriting recognition system. The supervised learning is to establish a learning process that comparing the predicted value with the actual identity of training data continually to adjust the prediction model constantly until the model prediction results reach the desired accuracy. The most common application scenarios of supervised learning are classification problems and regression problems. The representative algorithms of supervised learning have Logistic Regression [224] and Back Propagation Neural Network [225].
2. *Unsupervised learning:* In unsupervised learning, the input data is not specially labelled by clear identity. The learning model is to deduce the internal structure and hidden patterns in the data. The most common application scenarios of

unsupervised learning include the study of association rules and clustering. The representative algorithms of unsupervised learning have Apriori algorithm [226] and k-mean algorithm [227].

3. *Semi-supervised learning:* Under this learning method, the input data is partly labelled. The learning model can be used to predict, but, first of all, learning the inner structure of data model is necessary to organise data reasonably for the forecast. The most common application scenarios of semi-supervised learning include classification and regression that extend the supervised learning. These algorithms usually try to perform modelling on the unlabelled data and then prediction on the labelled data. The representative algorithms of unsupervised learning have *Graph Inference* [228] or *Laplacian SVM* [229].)
4. *Reinforcement learning:* In reinforcement learning, the labelled input data is not only for checking whether the modelling is right or not (supervised learning), but also feedback to the modelling to adjust the model at once such as regulating threshold for decision. The most common application scenarios of reinforcement learning include dynamic statistic and robot locomotion. The representative algorithms have *Q-Learning* [230] and *Temporal difference Learning* [231].
5. *Deep learning:* The Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts [232]. There are two key features, one is the composition of models which includes multiple stages or layers of non-linear information processing. The other one is the representation of learning method which focuses on successively higher and more abstract layer [233].

2.4.2 Machine learning algorithms

This section we will introduce the common algorithms that have been used for mobile botnet detection and discuss the advantages and disadvantages of each of them.

2.4.2.1 Naïve Bayes Classification

The general Bayesian classification is one kind of classification algorithms that are based on the Bayes' theorem. This classification algorithm will assign the label expressed by the value of a feature to the testing instances, and the labels belong to the finite set. All the Bayesian classification algorithms are based on one assumption that the value of a particular feature used in the classifier is independent with any other feature. Naïve Bayes classifier is the simplest one of the Bayes classification algorithms. The Bayes' theorem will be introduced firstly, and then we will discuss the procedure of Naïve Bayes classification algorithm.

Bayes' theorem defines the probability of an event, based on the conditions probability that might be related to the event in probability theory and statistics. Bayes' theorem can be stated mathematically as the following Equation 2-8:

Equation 2-8 Bayes's theorem

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \tag{2-8}$$

Where A and B are events.

P (A) and P (B) denote the probabilities of A and B without regarding to each other.

P (A | B) denotes conditional probability, is the probability of A given that B is true.

P (B | A), is the probability of B given that A is true.

The key idea of Naïve Bayes classification algorithm is very simple: For the testing instances and existing classes, we compute the conditional probability of every class given the value of the instance and then compare them. The class with the max of the conditional probability will be the class that the instance belongs to. The procedure of this classification is shown as follows:

1. Set $x=\{a_1,a_2,a_3...a_m\}$, x is an instance to be classified and a is one of the features used for the classifier.
2. Set $C=\{y_1,y_2,...,y_n\}$ is category set that contains *n* categories.
3. Compute the conditional probability of every category given the value of the instance x. $P(y_1|x),P(y_2|x),P(y_3|x),...,P(y_n|x)$
4. If $P(y_k|x)=\max\{P(y_1|x),P(y_2|x),P(y_3|x),...,P(y_n|x)\}$ then we classify the instance x to the category k

So now the key is how to calculate the conditional probability of step 3. It is based on Bayes' theorem as follows:

1. First to prepare a training dataset including instances with known category.
2. The statistic the conditional probability of every feature given the category based on the training dataset:

Equation 2-9 Bayes's theorem: conditional probability of features I

$$\begin{Bmatrix} P(a_1|y_1),P(a_2|y_1),...,P(a_m|y_1); \\ P(a_1|y_2),P(a_2|y_2),...,P(a_m|y_2); \\ ... \\ P(a_1|y_n),P(a_2|y_n),...,P(a_m|y_n); \end{Bmatrix} \tag{2-9}$$

3. According to the Bayes' theorem, we can get:

Equation 2-10 Bayes's theorem: conditional probability of features II

$$P(y_i|x)=\frac{P(x|y_i)P(y_i)}{P(x)} \tag{2-10}$$

Because the denominators for all classes are constant, we can maximise numerator. Meanwhile, as the feature attributes are conditional independence, so there are:

Equation 2-11 Bayes's theorem: conditional probability of features III

$$P(x \mid y_i)P(y_i) = P(a_1 \mid y_i)P(a_2 \mid y_i)...P(a_m \mid y_i)P(y_i) = P(y_i)\prod_{j=1}^{m} P(a_j \mid y_i) \tag{2-11}$$

According to the analysis mentioned above, the Naïve Bayes classification is divided into three phases:

1. Preparation Phase: In this stage, a training dataset that has labelled different category need be prepared. Moreover, then the appropriate attributes should be extracted from the training data.
2. Classifier Training Phase: In this stage, the classifier is trained by statistical analysing the frequency of every category in the training dataset and the conditional probability of every feature.
3. Application Phase: The task of this stage is using a classifier to classify the testing instances.

There are some advantages and disadvantages for Naïve Bayes classification algorithm [234].

Advantages:

1. Fast to train (single scan). Fast to classify.
2. Not sensitive to irrelevant features.
3. The algorithm can handle real and discrete data.

Disadvantages:

1. Assumes independence of features.
2. Practically, dependencies existing among variables cannot be modelled.

2.4.2.2 Decision Trees

A decision tree is a flowchart-like structure (binary tree or non-binary tree). Each branch represents the outcome of the test, and each leaf node represents a category label (a decision made after analysing all the attributes). The procedure of making a decision using decision tree is to test the attributes of test instance from the root node. According to the value of attributes belonging to test instance and the constraint of the tree, the right branch will be chosen. Traversing the entire feature from top to bottom of the tree and reaching the leaf node that is last decision to make. There are three types of nodes in the decision tree: (1) Decision nodes (2) Chance nodes (3) End nodes. The process of the decision tree is intuitive and easy to be understood. The primary problem to use decision tree for learning is the construction of decision tree based on the training dataset [235].

Different from the Bayes' classification algorithms, the construction of decision tree does not depend on domain knowledge. The process of constructing decision tree is to build a topology structure of the relationship among all the attributes of attribute selection measure on the training dataset. The primary step of building decision tree is the split of the attribute. The attribute split is constructing different branches according to the difference of feature attributes at one of the node. The goal is ensuring the instances under the branch belong to the same category as far as possible. There are three situations for splitting attributes:

1. If the attribute is discrete and no requirement for a binary tree: Every attribute is regarded as one branch.
2. If the attribute is discrete and the request for a binary tree: Defining a subset of attributes and dividing two branches as "belong to the subset" and "not belong to the subset".

3. If the attribute is continuous: Defining a value as split point (*split_point*) and dividing two branches as "greater than *split_point*" and "less and equal than *split_point*".

A fundamental issue in decision tree learning is the attribute selection measure that is an attribute splitting rule to divide the training dataset into different branches. The algorithms ID3 and C4.5 are the common algorithms for attribute selection measure.

ID3 Algorithm

The main ideas behind the ID3 algorithm is a feature is chosen as the next level of the tree if its splitting produces the most information gain [236].

In the decision tree, each non-leaf node represents an input attribute, and each arc between two nodes represents a possible value of that attribute. A leaf node represents the expected value of the output attribute when the path from the root node to that leaf node describes the input attributes. In an idealised decision tree, the input attribute to each non-leaf node has a requirement: amongst all the input attributes, the input attribute is the most informative about the corresponding output attribute. It is because the output attribute can be predicted by using the fewest possible questions on average. Moreover, the degree of how informative a particular input attribute can be represented by entropy which is utilised in communication systems to describe the measure of uncertainty [237]. It is fundamental in modern information theory. The detail of the procedure of ID3 algorithms is shown as follows [235]:

Algorithm goal: Select the attribute with the highest information gain for splitting.

1. Let p_i denote the probability that an arbitrary tuple in *D* belongs to the category C_i, estimated by:

$$\frac{|C_{i,D}|}{|D|}$$

2. The entropy (expected information) needed to classify a tuple in D which can be denoted by:

Equation 2-12 ID3 algorithm: entropy equation

$$Info(D) = -\sum_{i=1}^{m} p_i log_2(p_i) \tag{2-12}$$

3. Information required to classify D by using *A* to split *D* into *v* partitions, and the expected information can be denoted by:

Equation 2-13 ID3 algorithm: information after attributes split

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} info(D_j) \tag{2-13}$$

4. At last the information gained by branching on attribute A can be denoted by:

Equation 2-14 ID3 algorithm: information gain equation

$$gain(A) = info(D) - info_A(D) \tag{2-14}$$

5. After computing the information gain of all the attributes, we select attribute with the max information gain for splitting different branches after the current node. Then applying the procedure above for the child notes recursively until all the attributes are used.

Even though ID3 algorithm can search complete hypothesis space and the resulting search is much less sensitive to the error in individual training instances. There are still some disadvantages for the algorithm. ID3 algorithms maintain only a single current hypothesis as it searches through the space of decision trees, so it loses the capabilities that follow explicitly representing all consistent hypothesis. Meanwhile, ID3 perform without backtracking in its search. Therefore there is some probability of converging to locally optimal solutions that are not globally optimal. Moreover, if there is only one label

for the attribute, the ID3 will choose it as spitting attribute that is helpless for classification. The C4.5 algorithm is one of improved decision tree algorithm from the ID3 algorithm.

C4.5 Algorithm

The C4.5 algorithm is regarded as an extension of an earlier ID3 algorithm developed by Quinlan (known in Weka as J48 for Java). C4.5 is often regarded as a statistical classifier because the decision trees generated by C4.5 can be used for classification [238].

For C4.5 algorithm use the gain ratio to select the attribute for splitting. Based on the ID3 algorithm, it defines the split information as follows:

Equation 2-15 C4.5 algorithm: split information

$$split_info_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} log_2(\frac{|D_j|}{|D|}) \tag{2-15}$$

Moreover, then the gain ratio is defined as:

Equation 2-16 C4.5 algorithm: information gain ratio

$$gain_ratio(A) = \frac{gain(A)}{split_info(A)} \tag{2-16}$$

The last step that is same with the ID3 algorithm is to select an attribute with the max information gain ratio for splitting different branches after the current node. Then applying the procedure above for the child notes recursively until all the attributes are used.

Decision tree algorithms have advantages and disadvantages, which can be summarised as follows [239]:

Advantages:

1. Decision tree algorithms are easy to understand and interpret.
2. Decision tree algorithms can get a relatively better result even with a little dataset.
3. Decision tree algorithms can be combined with other decision techniques.
4. Decision tree algorithms can search entire hypothesis space, and the resulting search is much less sensitive to errors in individual training instances.

Disadvantages:

1. The information gain from constructing the decision tree is biased for those multi-value attributes, because of data including categorical variables with a different number of levels [240].
2. The algorithm has high space complexity with a large number of sample. Moreover, the run-time complexity matches to the depth of decision tree, which is relative to tree size and thereby to the amount of the sample [241].

2.4.2.3 K-Nearest Neighbours

K-Nearest Neighbours (K-NN for short) is one type of instance-based learning (also called lazy learning). Instead of performing explicit generalisation, the K-NN algorithm compares testing instances with training instances stored in memory to make a decision which category the testing instance should be allocated to [242].

There are two phases for K-NN classification algorithm: (1) Determination of the K nearest neighbours for the test instance. (2) Determination of the common category among the K nearest neighbours. The general procedure of K-NN algorithms is described as follows [243]:

1. Assuming that there is a training dataset D whose size is $|D|$ and the instances in the dataset are denoted as $x_i (x \in [1, |D|])$. These instances are described by the feature set F whose values have been normalised to the range [0, 1]. The category

set is denoted as *Y*. Every instance is labelled with the corresponding category $y_i(y_i \in Y)$.

2. When a testing instance *q* is given as input to the algorithm, we first calculate the distance between *q* and all $x_i(x \in [1,|D|])$ based on the features in set *F*. The method to compute the distance has been introduced in Section 2.3.2.2. We use the generic form as follows:

Equation 2-17 KNN algorithm: distance compute equation

$$d(q,x_i) = \sum_{f \in F} w_f \delta(q_f, x_{if}) \tag{2-17}$$

3. After the K nearest neighbours are selected, the most straightforward approach to determining the class of *q* is to assign the majority category among the nearest neighbours to the testing distance. A general technique to achieve this is to get to vote on the class of the testing instance with votes weighted by the inverse of their distances, which is as follows:

Equation 2-18 KNN algorithm: vote equation

$$Vote(y_i) = \sum_{c=1}^{k} \frac{1}{d(q,x_c)^n} l(y_j, y_c) \tag{2-18}$$

Thus, the vote assigned to a class y_j by neighbour x_j is 1 divided by the distance to that neighbour. The function $l(y_j, y_c)$ will return 1 if the categories of y_j and y_c are matched and return 0 otherwise.

The K-NN algorithm has the following advantages and disadvantages:

Advantages:

1. Because it is an instance-based algorithm, the cost of the learning process is zero. Meanwhile, the algorithm can be understood and implemented easily.
2. The KNN algorithm has a probability of error that is less than twice of Bayes based on certain reasonable assumptions [244].
3. The KNN algorithm is particularly suitable for the training data that has multiple class labels. For example, for the assignment of functions to genes based on expression profiles [245].

Disadvantage:

1. The number of K should be determined when using this algorithm. Changing K can change the predicted results.
2. The algorithm must compute the distance and sort all the training dataset at each prediction [246]. It is a time-consuming process if there are a large number of training instances.

2.4.2.4 Neural Networks

Neural networks (NNs) are a group of statistical learning models derived from biological neural networks of the animal brain and are used to approximate functions that can depend on a large number input dataset [247].

Neural networks are mathematic model containing a large number of nodes (also called “neurones” or “units”) which connected each other. There is a particular output function, which is called activation function. Each connection between two nodes represents a weighted value for the signal through the connection, which is corresponding to the memory of artificial neural network. The output of the neural network is by the network connection mode, the different weights and activation function.

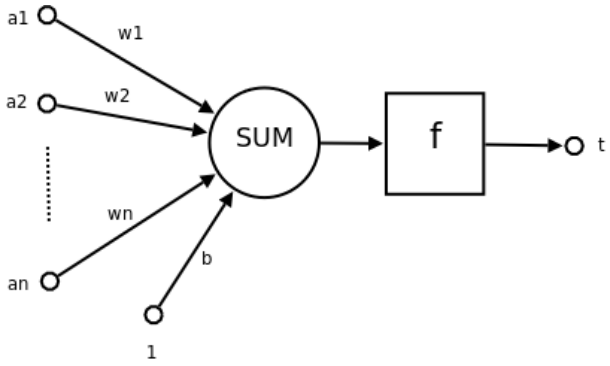


Figure 2-7 - The unit of neural network

We first introduce the unit node of the neural network that can be shown in Figure 2-7. The a_1 to a_n are inputs for the unit and w_1 to w_n is the weight for each, f is the activation function. To avoid this dilemma, a third input typically referred to as a bias input is required for the unit. A bias input should have the value of one always and is also weighted with value b. The unit can be denoted as: $t = f(\overrightarrow{W}\overrightarrow{A}' + b)$.

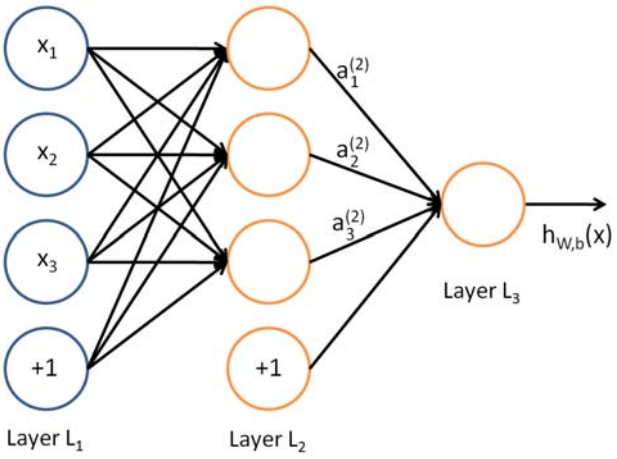


Figure 2-8 - Simple neural network model

The key of the neural network is the construction of the model based on the training dataset. For example, the Figure 2-8 is one of the simple neural networks. A common form of multilayer feedforward network consists of three parts: Input layer (*Layer L_1*),

output layer (*Layer L3*) and hidden layer (*Layer L2*). The input of labelled "+1" in layer L1 are bias units and correspond to the intercept term. This example neural network also has three input units, three hidden units, and one output unit. We use $W_{ij}^{(l)}$ to denote the weight associated with the connection between unit j in layer l, and unit i in layer l+1. $b_i^{(l)}$ is the bias associated with unit *i* in layer *l+1*. We use $a_i^{(l)}$ to denote the output value of unit *i* in layer *l*. Given a fixed parameter weight (W) and bias (b), the hypothesis $h_{W,b}(x)$ can be defined to output a real number for the neural network. We first introduce the forward propagation [248] process as follows:

1. Compute the output Layer L2:

Equation 2-19 Three layers NN algorithm: output of layer 2

$$\begin{aligned} a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\ a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\ a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \end{aligned} \tag{2-19}$$

2. Compute the output Layer L3:

Equation 2-20 Three layers NN algorithm: output of layer 3

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \tag{2-20}$$

Then we can use backpropagation algorithm [225] to learn with training dataset. We suppose that the training data set denoted as $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, *x* is the input data, and the *y* is the output as the category. The overall cost function can be defined as follows:

Equation 2-21 Three layers NN algorithm: overall cost function

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \end{aligned} \tag{2-21}$$

The first term in the definition is an average sum-of-squares error term. The second term is a regularisation term that tends to decrease the magnitude of the weights and helps prevent overfitting.

This cost function can be used both for classification and regression problems. For classification, let y=0 or 1 represent the two class labels. For regression problems, scale the outputs to ensure that they lie in the [0, 1] range. The goal of the artificial neural network is to minimise $J(W,b)$ as a function of W and b. To achieve this objective, the algorithm first initializes these two parameters with small random value and then apply optimisation algorithms. The gradient descent is one type of optimisation algorithms which can be described as follows:

Equation 2-22 Gradient descent

$$\begin{aligned} W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b) \end{aligned} \tag{2-22}$$

Where α is the learning rate

Finally, we use the function below to compute the partial derivatives of the cost function:

Equation 2-23 Neural network algorithm: cost function

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)} \tag{2-23}$$

The Neural Network (NN) algorithm has several advantages and disadvantages that can be summarised as follows [249]:

Advantages:

1. The neural network algorithm can be used for general application and can generate a variety of patterns with high accuracy.
2. The neural network algorithm uses a static and non-linear function to fit the parameters of a particular function based on the training dataset. Meanwhile, there are some multiple functions to choose.
3. It is possible to detect nearly all complex nonlinear relationships between input data and outputs data based on the training dataset.

Disadvantages:

1. The neural networks algorithm cannot determine the optimal number of nodes, hidden layers, functions and so on. Because the process of neural networks is a black box, it is difficult to find the errors in the process of learning. Therefore this algorithm is a lack of ability to reason about their output in a way that can be effectively communicated [250].
2. Training time for the neural network is usually much longer than other machine learning algorithm such as decision trees [251].

2.4.2.5 Support Vector Machines

Support vector machines (SVMs) are one of supervised learning models that can be used for classification and regression. The SVM algorithm depends on the concept of *decision boundary* of the different categories in the training dataset. The decision boundary is a hypersurface that partitions the underlying vector space into two sets and one for each class. There are several types of SVM algorithms. In the following, we present the basic two-class support vector machine [252].

We suppose the training dataset $\Omega = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, and the instances $x_i \in R^n$ and $y_i \in \{-1,1\}$ in a space where x_i and y_i is the i^{th} input and output. The

hyperplane of function ϕ is assumed which can create a non-linear decision boundary between two categories. The function can be denoted as $w^T x+b=0$, with $w \in F$ and $b \in R$. There is only one category of training instances in any one side of the hyperplane. The distance between the closest points from each category and the hyperplane is equal. The variables ξ_i are introduced to keep some instances lying within the margin as penalizes. Moreover, the constant variable C is also introduced to determine the trade-off between a large margin and a small error penalty. Therefore, the objective function of the SVM algorithms can be described as follows:

Equation 2-24 Support vector machine: objective function

$$
\begin{aligned}
& min_{w,b,\xi_i} = \frac{\|w\|^2}{2} + C\sum_{i=1}^{n}\xi_i \\
& \text{subject to:} \\
& y_i(w^T\phi(x_i)+b) \geq 1-\xi_i \quad \text{for all } i=1,\ldots,n \\
& \qquad\qquad \xi_i \geq 0 \quad \text{for all } i=1,\ldots,n
\end{aligned}
\tag{2-24}
$$

Now we can use Lagrange multipliers to solve this minimization problem and the decision function for one instance can be denoted as follows (α_i are the Lagrange multipliers):

Equation 2-25 Support vector machine: decision function

$$
f(x) = \text{sgn}(\sum_{i=1}^{n}\alpha_i y_i K(x,x_i)+b) \tag{2-25}
$$

Next, let us introduce the Sequential minimal optimisation (SMO) which is one of the simple versions of SVM and it is widely used in many machine learning toolkits such as WEKA [253]. The target of the SMO is to find a function to divide the training instances into different categories. To maximise the decision boundary in SMV, we need to minimise the value of:

Equation 2-26 decision boundary of SVM

$$\frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i \tag{2-26}$$

w is the parameter and the greater the value, the more obvious for a boundary. C is behalf of the penalty factor. If one instance has a deviation to the category that it is should belong to, the greater of the value of factor C, the more degree to rectify this situation. ξ_i is the slack variable. According to Karush–Kuhn–Tucker conditions, we can convert the problem of minimising the value the function above to find the optimal solution that conforms to these conditions [254]:

Equation 2-27 to Karush–Kuhn–Tucker conditions

$$\begin{cases} \alpha_i = 0 \Leftrightarrow y_i u_i \geq 1 & (a) \\ 0 < \alpha_i < C \Leftrightarrow y_i u_i = 1 & (b) \\ \alpha_i < 0 \Leftrightarrow y_i u_i \leq 1 & (c) \end{cases} \tag{2-27}$$

α_i are the Lagrange multipliers

1. For condition (a), the α_i is normal classification.
2. For condition (b), the α_i is support vector which is on the boundary.
3. For condition (c), the α_i is between two boundaries.

Therefore, the following conditions will be not satisfied:

1. $y_i u_i \leq 1$ but $\alpha_i < C$. It should be $\alpha_i = C$
2. $y_i u_i \geq 1$ but $\alpha_i > C$. It should be $\alpha_i = 0$
3. $y_i u_i = 1$ but $\alpha_i = C$ or $\alpha_i = 0$. It should be $0 < \alpha_i < C$

The next step is to search all the unsatisfied situation and update α_i. Because of another constraint condition $\sum_{i=1}^{l} \alpha_i y_i = 0$, the α_i and α_i will be updated simultaneously by the function:

$$\alpha_i^{new} y_i + \alpha_j^{new} y_j = \alpha_i^{old} y_i + \alpha_j^{old} y_j = constant$$

According to the convex quadratic programming, we can get

$$\alpha_j^{new} = \alpha_j + \frac{y_j(E_i - E_j)}{\eta}$$
$$E_i = u_i + y_i$$
$$\eta = k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)$$

Base on the constraint of $0 < \alpha_j < C$, we can get the analytical solution of α_j:

$$\begin{cases} L = max(0, a_j - a_i), H = max(C, C + a_j - a_i) & if\ y_i \neq y_j \\ L = max(0, a_j + a_i - C), H = max(C, a_j - a_i) & if\ y_i = y_j \end{cases}$$

$$\alpha_i^{new} = \alpha_i + y_i y_j (\alpha_j - a_j^{new,clipped})$$

α_i can be found according to the KTT, and α_j can be found according to the condition of $max|E_i - E_j|$, then we define:

$$b_1 = b - E_i - y_i(\alpha_i - a_i^{old})k(x_i, x_i) - y_j(\alpha_j - \alpha_j^{old})k(x_i, x_j)$$
$$b_2 = b - E_j - y_i(\alpha_i - a_i^{old})k(x_i, x_i) - y_j(\alpha_j - \alpha_j^{old})k(x_i, x_j)$$

Then we update b according to

$$b := \begin{cases} b_1 & if\ 0 < \alpha_i < C \\ b_2 & if\ 0 < \alpha_j < C \\ (b_1 + b_2)/2 & otherwise \end{cases}$$

Finally, we update all the α_i, y and b to finish the SMO model to learning on the training dataset.

2.4.3 Evaluation criteria for machine learning

In Section 2.2.4, we have given a short introduction for how to compare the different botnet detection techniques. We will make a detailed analysis of the methodology of the evaluation of machine learning. For evaluating the machine learning algorithm, we first to discuss the measurement for performance and the statistical test. Moreover, then the sampling techniques and the choice of the appropriate dataset will be introduced at the end.

2.4.3.1 Performance Measurement

1. Confusion Matrix

The confusion matrix is a particular table layout allowing visualisation of the performance of machine learning algorithms. In general, the columns of the matrix represent the predicted class value, and the rows of the matrix represent the actual class value for the instances. The confusion matrix can be described in Table 2-4. According to the figure, there are four types of the classification [255]:

True positive (TP): The instances are predicted as positive. Meanwhile, the actual class value of these instances is positive. It is also called hit.

False positive (FP): The instances are predicted as negative. Meanwhile, the actual class value of these instances is positive. It is also called false alarm or Type I error.

False negative (FN): The instances are predicted as positive. Meanwhile, the actual class value of these instances is negative. It is also called miss and Type II error.

True negative (TN): The instances are predicted as negative. Meanwhile, the actual class value of these instances is negative. It is also called correct rejection.

Table 2-4 - Confusion matrix

Confusion Matrix		Predicted class value	
		Positive	Negative
Actual class value	Positive	True positive(TP)	False positive(FP)
	Negative	False negative(FN)	True negative(TN)

For the machine learning algorithm, we want to high TP and TN and low FP and FN intuitively. There are also important performance measures that are generated from the confusion matrix [255].

True positive rate (TPR): TPR = TP/ (TP+FN), It is also called Recall and Sensitivity.

False positive rate (FPR): FPR = FP/ (FP+TN), It is also called Fallout.

False negative rate (FNR): FNR = FN/ TP+FN), It is also called Miss Rate.

True negative rate (TNR): TNR = TN/ (FP+TN), It is also called Specificity (SPC).

Positive predictive value (PPV): PPV = TP/ (TP+FP), It is also called Precision.

False omission rate (FOR): FOR = FN/ (FN+TN).

False discovery rate (FDR): FDR = FP/ (TP+FP).

Negative predictive value (NPV): NPV = TN/ (FN+TN).

Accuracy (ACC): ACC = (TP + TN)/ (TP + TN + FP + FN).

Positive likelihood ratio (LR+): LR+ = TPR/FPR.

Negative likelihood ratio (LR−): LR− = FNR/TNR.

Diagnostic odds ratio (DOR): DOR = LR+/LR−.

Receiver Operating Characteristics (ROC): ROC is a more comprehensive evaluation criterion for machine learning. It is based on two important concepts: TPR and FPR. The ROC curve is the graph whose horizontal axis is FPR, and the vertical axis is TPR. With the variation of the threshold setting, the detection system will get a different pair of TPR and FPR. Ideally, FPR is lowest, and the TPR is highest. However, the threshold at the point of tangency on the ROC curve is the best configuration.

The area under ROC curve (AUC): The area under the ROC curve is also regarded as a criterion for detection effect that is called AUC.

2. Cohen's Kappa Measure

Cohen's kappa coefficient measurement is a statistic that measures the degree of agreement among raters for categorical items. It can solve the problem of classification could be a result of coincidental concordance between the classifier's output and the label generation process is not taken into account. The formula of Cohen's Kappa is as follows [256]:

Equation 2-28 Cohen's Kappa formula

$$\kappa = \left(P_{\mathrm{o}} - P_e^C\right) / \left(1 - P_e^C\right) \tag{2-28}$$

P_o denotes the probability of overall agreement across the label assignments between the classifier and the actual process, and P_e^C denotes the chance agreement across the labels and is defined as the sum of the proportion of examples assigned to a class times the percentage of true labels of this category in the data set.

3. Cost Curves

Cost curves are more practical than ROC curves because it can show what class probabilities one classifier is preferable over the other. The cost curve is a graph of the costs of production as a function of total quantity produced which is derived from economy [257].

It is based on two concepts: Error rate and Probability of an example being from the positive class that can be denoted as $P(+)$. The Cost-curves is the graph whose horizontal axis is $P(+)$, and the vertical axis is an Error rate. To know the meaning of $P(+)$, we need to introduce three sets of instances that have a different proportion of positive instances. $P_{train}(+)$ is the percentage of positive instances in the training dataset used to learn the algorithm. $P_{test}(+)$ is the proportion of positive instances in the dataset used to build the classifier's confusion matrix. $P_{deploy}(+)$ is the percentage of positive instances when the classifier is deployed for the testing dataset. For the cost curves, the $P(+)$ should use $P_{deploy}(+)$.

4. The Root-Mean-Squared Error (RMSE)

The Root-Mean-Squared Error (RMSE) is usually used for regression, but can also be used with probabilistic classifiers. The formula for the RMSE is as formula [258]:

Equation 2-29 Root-mean-squared error formula

$$RMSE(f) = sqrt\left(1/m \sum_{i=1}^{m} (f(x_i) - y_i)^2 \right) \quad (2\text{-}29)$$

Where m is the number of test examples, f is the classifier's probabilistic output on x_i and y_i the actual label.

2.4.3.2 Statistical Testing

According to the number of the algorithms and domain included in the test, we can divide the statistical test into three types: The comparison of 2 algorithms on a single domain, the comparison of 2 algorithms on multiple domains and the comparison of multiple algorithms on multiple domains.

Analysis of variance (ANOVA): The analysis of variance (ANOVA) is a collection of statistical models that used to analyse the differences between group means and their associated procedures [259]. In the ANOVA setting, the observed variance in a specified variable is divided into components attributable to different sources of variation. For the simple form, ANOVA provides a statistical test of whether or not the means of several groups are equal, and therefore generalises the t-test to more than two groups. As doing multiple two-sample t-tests would result in an increased chance of committing statistical type I errors, ANOVAs are beneficial for comparing (testing) more than three means (groups or variables) for statistical significance. The F value in ANOVA which is generated by F-test can be used to analyse whether the means between two groups of data are significantly different. If the calculated F value is larger than the F critical value (F statistic which can be found in F table), the null hypothesis can be rejected which means the two set of data are significantly different.

2.5 Mobile botnet

The overflow of mobile malware follows the development of the smart mobile devices. However, because of the isolation and immature of the traditional mobile malware, their harm is limited, and they are easy to be detected by general anti-virus mobile software with the signature of the existing malware sample.

The mobile devices in using such as smartphones and tablets have increased impressively recently. According to the Cisco's report, more than 0.5bn (526 million) mobile devices were activated and connected to networks in 2013 [260]. The global mobile devices and connections have grown to 8.0 billion in 2016, up from 7.6 billion in 2015. It also forecasts this number will grow to 11.6 billion by 2021 [18]. Recent figures also show that Google's Android operating system has overtaken other platforms platform since 2012 and is currently the market leading mobile OS and is expected to get more than 80% market share until 2019 [19].

Along with the growth in the use of the high speed of mobile network (2.3 billion active mobile-broadband subscriptions worldwide in 2013 according to [261]), there has also been a growing number of mobile malware, often in the form of mobile botnets. The mobile botnets can be defined as a collection of network connected applications on mobile devices communicating with other similar applications and a remote server to perform tasks. The controllable features of botnet enable more effective and sophisticated attacks on mobile devices (e.g., dynamically execute a remote command, long-term information stealing) making mobile botnet a prominent form of new generation of mobile malware. According to KASPERSKY [20], 148,778 mobile malware apps had been detected at the end of 2013, and nearly 62% of them are elements of mobile botnets. In 2016, the number of malicious installation packages grew considerably, amounting to more than 8.5 million which is three times more than 2015 [21]. The first mobile botnet, i.e., the iKee.B iPhone

botnet [22], was traced back in 2009. However, due to iOS being a closed system, it did not prove to be a particularly harmful mobile malware. In the case of Android, however, its open source nature gave an ample opportunity for developing mobile botnets. Geinimi, a piggybacks game app, became the first discovered Android botnet in 2010 [23], and since then more and more hackers have started producing mobile botnets for Android devices. The Android.Troj.Mdk Trojan, for instance, has been found in more than 7,000 apps and infected more than a million mobile users in China in 2013 [7]. NotCompatible.C is another example of new Trojan threatening protected enterprise networks [24].

The current state of mobile botnet incidents in real networks is reviewed in the next section. It is followed by a summary of the existing research on mobile botnets in Section 2.5.2 and 2.5.3 which contain two aspects: mobile botnet creation and mobile botnet detection. A comparison of current machine learning based mobile botnet detection techniques are discussed in Section 2.5.4. We also discuss the specificities of mobile botnets with other mobile malware and traditional botnets in Section 2.5.5. Lastly, we analyse trend in mobile botnet techniques and the open issues in correspondent detection techniques in Section 2.5.6.

2.5.1 Mobile botnet accidents

One of the important aspects of the analysis of mobile botnet malware is the harm and the general mode of attack. The analysis of harmful behaviours and the general attack means of mobile botnet malware are meaningful to detect and defence the corresponding malware. Through summarising the accidents of the mobile botnet, we list the potential attacks which can be performed by these types of mobile botnet malware in Table 2-5.

Table 2-5 - List of potentially harmful behaviours of mobile botnet

Index	The potential harmful behaviours
A	Capable of rooting the vulnerable Android phones.
B	Evade the detection from mobile anti-virus software.
C	Collect information and phoning home. (IMEI number, phone model, Android OS version)
D	Elevate its privilege to root.
E	Capability to install or remove any packages without users' awareness.
F	Send SMS to the premium-rate telephone number to get profit.
G	Remove messages related to the premium-rated telephone number.
H	Thoroughly remove evidence of their activities.
I	Steal user's accounts and other credential information.
J	Built-in promotion mechanism to install other instances of malware.
K	Automatically confirm the subscription of premium-rate SMS services without users' awareness.
L	Capability to dynamically load and execute remotely downloaded code from Blog.
M	Detect the existence of anti-virus software and attempt to shut down the security software.
N	Encrypt information bypass detection.
O	Install itself as a device administration app.
P	Abusive use of notification bar.

Basing on this harmful behaviour of mobile botnets, we make a summary of the major mobile botnet accidents in recent years. Every accident includes the year first detected, the infected platform, the way of spread and the harmful effect. We also give some impacts of some more serious mobile botnet malware according to reference. The complete accidents of mobile botnets are shown in Table 2-6.

Table 2-6 - The accidents of mobile botnet

Ref.	Incidents name	Y	Device	The way of spread	Effect	Impact
[262]	iKee.B	2009	iOS	Scan the iOS devices by SSH	**C**	T-Mobile's Dutch IP infected.
[23, 263]	Geinimi	2010	Android	Piggybacks on apps	**C E**	Infect significant number of Android devices
[264, 265]	DroidDream	2011	Android	Piggybacks on apps	**C E O**	There over 50 applications were infected with it.
[266-268]	DroidKungFu1,2,3	2011	Android	Piggybacks on apps	**A B C D E**	
[269]	YZHCSMS Trojan	2011	Android	Piggybacks on apps	**F G H**	
[270, 271]	Plankton	2011	Android	Piggybacks on apps	**C I**	
[272, 273]	GoldDream	2011	Android	Piggybacks on apps	**C E F**	
[274, 275]	HippoSMS	2011	Android	Piggybacks on apps	**F** S **G**	
[276, 277]	SndApps	2011	Android	Piggybacks on apps	**C J** B	
[278]	NickiBot	2011	Android	Piggybacks on apps	**C**	
[279, 280]	RogueSPPush	2011	Android	Piggybacks on apps	**K** G	
[281, 282]	GingerMaster	2011	Android	Piggybacks on apps	**A C E**	the first one that utilises a root exploit against Android 2.3
[263, 283]	DroidDeluxe	2011	Android	Rogue App or Fake malware[2]	**A C I**	
[284, 285]	AnserverBot	2011	Android	Piggybacks on apps	**L M**	Being injected into some (20+) legitimate apps
[286]	DroidCoupon	2011	Android	*Standalone or Fake malware*	**A C E**	
[287, 288]	BeanBot	2011	Android	Piggybacks on apps	**C F**	
[289, 290]	SMS Android Trojan DroidLive	2011	Android	Piggybacks on apps	**C F O**	It infects more than ten distinct Android apps.
[291, 292]	Android. Master	2012	Android	Piggybacks on apps	**C I E**	11,000/$547,500 to $3,285,000 per year
[293, 294]	RootSmart	2012	Android	Piggybacks on apps	**A E**	affect between 10,000 and 30,000 devices per day.
[295]	PushBot	2012	Android	Piggybacks on apps	**E P**	
[296]	DKFBootKit	2012	Android	Piggybacks on apps	**E**	Infect 1,657 devices in the past two weeks. 100 malicious apps
[297, 298]	TigerBot	2012	Android	Piggybacks on apps	**C F**	
[299, 300]	UpdtKiller	2012	Android	Piggybacks on apps	**B C F G**	
[301]	ZitMo	2012	Android Blackberry	*Standalone or Fake malware*	**C**	
[7]	Android.Troj.mdk	2012	Android	Piggybacks on apps	**C E N**	Infects 100,000 Chinese smartphones and 11,000 malicious apps samples
[302]	Eurograbber (variant of ZitMo)	2012	Android BlackBerry	Fake software security upgrade[3].	**C**	infected more than 30,000 users Stolen an estimated 36 million Euros

[2] Rogue App or Fake malware: It's mean that some malware disguise themselves as some legitimate applications with some attractive function, such as password recovery and so on.

[3] Phishing message leading downloads the Trojan onto their PC. Trojan hijacks security upgrade to send SMS with fake upgrade URL to install software on mobile device.

2.5.2 Mobile botnet creation

Because the mobile botnet is relatively new threaten for the current mobile network environment, there are not so many typical mobile botnet samples detected in the wild. So many literature do research for proposing new mobile botnet as preventive measures for the future detection system. Some creations of mobile botnet concentrate on the feature of the mobile network to strengthen one or more procedures in the botnet.

Mulliner et al. [53] propose a cellular botnet relatively early that describe what it takes to build a mobile botnet in detail. The paper first discusses the critical procedures that contain infection pathway, Command and Control (C&C) protocol and communication strategies. C&C channel is the most important part of a botnet as well as for mobile botnet. There are two major modes of transmission of C&C command. The first one is a P2P-based approach and the second one is an SMS-based approach that is much harder to observe, analyse and disrupt by security software. To transfer a large meaningful volume data, the paper first proposes the SMS-HTTP hybrid C&C channel. The basic idea for the hybrid structure is dividing the communication into SMS and HTTP. The encrypted files are stored on some websites as a command, and their URLs are encoded into the SMS which should be sent to the random bots. This method can decrease the size of controlled SMS and conceal the real commands. They even give an implement of necessary parts of the botnet and evaluate it in Wi-Fi connection and mobile data connection.

Xiang et al. [28] design another mobile botnet named Andbot. They first analyse the challenges of the constructing mobile botnet and propose the corresponding solutions for these challenges including stealthy and resilient of C&C channel, low-cost in charges, traffic, and battery. The paper describes a control mechanism called URL Flux. The basic principle is that the bots try to connect the Web 2.0 servers (such as blog and microblog) with a sequence of generated usernames until the connection is established successfully and get the information from the new feed on the servers. A particular C&C architecture

and process of Andbot are presented. The botmaster in Andbot binds the encrypted and signed commands into a JPG picture and divides the URL of the picture and the picture into different Web2.0 servers which improve the concealment of the C&C channel.

The SMS is usually applied to the C&C channel in mobile botnet as one of the important unique features in mobile devices. The literature [46, 47] propose an improved SMS based heterogeneous mobile botnet. They design a heterogeneous structure SMS communication for the botnet which contains botmaster, collection node, Bot server, region Bot server and bots. Collection nodes are the response for receiving the valuable information from all the nodes of the botnet. The Bot servers, as well as the region Bot servers, are key nodes in the structure which undertake the tasks of searching and forwarding the nodes in next layer. They also make an evaluation of nodes capacity and connectivity for the structure in mathematic. Hamandi et al. [44] demonstrate a type of botnet application with two levels of topology structure that targets Android devices specifically. They give more detail of how to build an SMS-based botnet in Android OS, focusing on the components: *main activity*, *listening service* and *permissions* in the application. The SMS payload is designed to contain verifier to verify the message originated from the Botmaster and all the information of the command.

Hua et al. [49] proposed an SMS-based mobile botnet using *Flooding Algorithm*. The structure of the botnet is P2P, and after the botmaster first sends out the command to any other node, the process will not stop. Whenever any node receives the command, they will continually transmit the command to their other neighbour nodes until the forwarding count reaches preconfigure threshold. The key problem of the simple flooding algorithm is how to choose neighbourhood nodes to propagate the command. A helper server is introduced in the paper to support the selection of neighbour with a certain probability.

The social network applications such as Facebook, Twitter and MySpace have had become one of the general applications installed in the smart mobile phone. So there is

some literature taking advantage of the feature of the social network to establish the mobile botnet. Faghani et al. [45] present a new cellular botnet named *SoCellBot* that makes use of social networks to infect the mobile devices as bots and uses social networks messaging systems as communication channels between bots. The paper discusses three parts of the botnet which contain the propagation mechanism, C&C channel and topology. There are two main methods to recruit the mobile devices into the botnet: one is to exploit vulnerabilities of the operating systems, and the other is using social engineering techniques to trick users to install the application. The topology is based on the knit groups and friendship on the social network that stabilise the connection between botmaster and bots. They simulate their model in two set experiments with four scenarios to evaluate the propagation efficiency and the communication efficiency. Yue et al. [52] put forward a mobile botnet based on the Twitter and SMS control. They also propose two common algorithms for constructing the network communication topology which synthesises the Twitter message and SMS.

As the functions of mobile devices are continuously enhanced, the services on the mobile devices are gradually diversifying. Meanwhile, some specific services can be exploited for designing mobile botnet. The literature [58] present a new mobile botnet called *cloud-based push-styled* mobile botnets based on one of the important services provided by Google on Android OS devices named Cloud to Device Message Service (C2DM)[4]. C2DM is a service that helps developers send data from servers to their applications on Android devices [98]. So the paper proposes an architecture for pushing the command for application server as the botmaster to the mobile devices. Considering the single C2DM mechanism is easy to be detected by the system, they design an enhanced architecture using two sets of C2DM registrations to move the botmaster for

[4] C2DM has been officially deprecated as of June 26, 2012.And it has been replaced by the new version of C2DM, called Google Cloud Messaging for Android (GCM)

application server into another mobile device. They even give a discussion of large-scale botnet building solution dividing the bots into several groups by using the same username in the same group. At last, they evaluate the botnet at three aspects containing stealthiness in control and data plane, efficiency in resource consumption and controllability.

The advancements of the mobile device hardware are not only the performance improvement but also the hardware with new features continuously is added into the mobile devices. For example, more abundant sensors are placed into the mobile devices such as light sensor, gravity sensor, near-field communication (NFC) and so on. These new features of hardware can also be used as the communication channel in the botnet. Hasan et al. [48] propose a new idea for the mobile botnet that is based on the sensors on the mobile. Thinking of the popularity of sensor in the mobile device, the paper supposes that the sensors can also be used for communication between the mobile devices as botnet command and control channel. The out-of-band C&C can be divided into *steganographic* channels and *non-steganographic* channel. The difference between the two type channels is whether the trigger signal is hidden inside another signal or not. Then they give a detail description of how to use the audio, light, magnetic and vibrational sensor channel to transmit the command to bots. The communication through the sensor is hard detected by monitoring the cellular or wireless communication networks. However, the stability is one of the most difficult problems to be solved.

2.5.3 Detection techniques

The traditional mobile malware detection systems have been deeply studying in the past few years, and some of these systems can partially be used for mobile detection. So, we will also discuss two types of detection techniques in the following part. One is the detection method designing specially for the mobile botnet. The other one is general

mobile malware detection system designing for all mobile malware. At last, we also do a review for the detection techniques by using machine learning algorithms.

2.5.3.1 Special mobile botnet detection techniques

Mobile botnet detection has been studied by Vural et al. [35, 303]. In [303], the authors propose a detection technique based on network forensics and give a list of the metrics for building an SMS behaviour profiles to use in detection. Based on these profiles the compare information gathered from some network forensics tool to establish if it is normal. Autocorrelation is used to calculate the value of every metric in the list, and a fuzzy function is used for comparison with normal behaviour. In [35], they improved the accuracy of their approach by introducing an artificial immune system based detection. Although it is an anomaly-based detection technique, their approach still relies on network forensics.

Seo et al. [26] propose a static analysis tool, called DroidAnalyser, to identify potential vulnerabilities of Android apps and root exploits. Their tool is signature-based. More specifically, they define some suspicion signatures as Dalvik Executable file (i.e., a *.dex* file in Android install package *.apk*). Their analysis tool processes mobile behaviour in two stages: (a) a signature matching state and (b) a search of app code using pre-fixed keywords. Using the outcomes (a) and (b), DroidAnalyser generates a measurement of the suspicious level of the detected application. DroidAnalyser gives suggestions before installing an application on a mobile device but cannot detect infection by malware at runtime.

Another system, called *Copper-Droid* that can perform dynamic behavioural analysis of Android malware automatically is presented in [304]. There is also some online file analysis system which can detect suspicious application install files such as *Andrototal* [305], *SandDroid* [306], *App360Scan* [307], *MobileSandbox* [308] and so on.

2.5.3.2 General mobile malware detection techniques

Some of the methods generated for the detection of general mobile malware can detect some mobile botnets.

In [37], the authors show a monitor of Symbian OS and Windows Mobile smartphone that extracts features for anomaly detection. Based on ML they analyse the monitoring log to detect features of normal/infected situations. Due to mobile hardware limitations, their ML based complex intrusion detection system runs on a remote server.

The approach in [29] makes two contributions: (1) it includes a static analysis system for analysing Android Market applications and providing detailed and readable reports to the user, and (2) it uses automated reverse-engineering and refactoring of binary application packages to mitigate security and privacy threats driven by users' security preferences. Their approach is based on a novel probabilistic diffusion scheme for detecting anomalies that may indicate malware using device usage patterns. The Android Application Sandbox (*AASandbox*) [309] has also been used to perform both static and dynamic analysis on Android programs to automatically detect suspicious applications, based on the idea that the detection result of neighbours is important in evaluating indicators of malware.

Zhou et al. [31] propose a fast and scalable approach to detect "piggybacked" Android applications, i.e., apps that attach some destructive payloads or malware code. Two techniques are used for this purpose: (a) a module decoupling technique that partitions source code of an application into primary and non-primary modules and (b) a feature fingerprint technique that extracts various semantic features (from primary modules) and converts them into feature vectors. Using this approach, the authors have collected more than 1,200 malware samples cover the majority of Android malware families from August 2010 to October 2011, and have systematically characterised them from various aspects.

Shabtai et al. [34] proposed a knowledge-based approach for detecting known classes of Android mobile malware based on temporal behaviour patterns. The basic idea is to train a classifier to detect different types of applications based on application features. This approach has been implemented in Andromaly [310] and was subsequently improved with Knowledge-based Temporal Abstractions (KBTA) [311]. KBTAs were used to derive context-specific interpretations of applications from timed behaviour data.

2.5.4 ML based botnet detection techniques comparison

In the following, we review techniques developed to detect mobile botnets on Android mobile platforms. Our review focuses on techniques, which use machine-learning algorithms for this purpose, as this approach has been the driver of the experimental investigation discussed in the paper. Table 2-7 provides an overview of such techniques, which are known to us.

Each of these techniques in the table is described by:

- The normal applications (*Normal Apps (N)*) that it has been applied to. N apps are distinguished into the apps used to train, and the apps used to test the ML algorithms.
- The botnet (or other malware) applications that it has been applied to *Botnets (B)*. B apps are also distinguished into the apps used to train, and the apps used to test the ML algorithms.
- The *features* used to classify an app as N or B and whether some *pre-filtering* of them was applied by the technique prior to the ML training phase.
- The *ML algorithms* applied and tested by it.
- The method/source of *classification* of the feature sets as N or B features, which was used to train the algorithms (not applicable in techniques using unsupervised learning).

- The *set up* of the experimental evaluation of the technique including the use of user interactions in forming the training data set and the evaluation *scenarios* used in the testing phase. The latter are distinguished by whether known or unknown N and B apps were used in testing. This distinction results in four scenarios: known Bs & known Ns (KBKN), unknown Bs & known Ns (UBKN), known Bs & unknown Ns (KBUN), and unknown Bs & unknown Ns (UBUN).
- The *performance* measures reported for the technique, i.e., the true (botnet) positive rate (TPR), false (botnet) positive rate (FPR) and precision (PRC) (see Section 2.4.3.1 for definitions of these measures).
- Whether any *sensitivity analysis* has been carried out by a technique to explore whether its performance varies across different ML algorithms (*Alg*), the feature data aggregation period (*Agp*), normal apps (*N ap*), botnets (*B ap*), features (*Fet*), and if the statistical significance of any observed differences have been validated (*Val*).

As it can be seen from Table 2-7, the features that have been used for mobile botnet detection are related to OS system calls [312-315], the permissions that different apps declare for Android devices ([316, 317]) and which are used to grant them access to certain device actions, device usage by the apps (e.g., SMS dispatches, use of device's camera through calls to the API of the device) [315, 316], device usage by the user (e.g., SMS dispatches, use of device's camera) [315, 316, 318], and external communication activities [39, 40, 179, 315].

Of the techniques focusing on system calls (as we do in the study of this paper), only two, i.e., Crowdroid [313] and MADAM [315], specify their experimental set up the insufficient level of detail for forming an assessment of the merit of the approach. Crowdroid uses unsupervised learning (k-means), and MADAM uses supervised learning but only one algorithm (KNN). However, none of them considers the full spectrum of scenarios, i.e., KBKN, UBKN and UNUB, and has carried out a validated sensitivity

analysis of performance as that we do in this study. More specifically, MADAM has only tested known botnet scenarios (KB**), and Crowdroid has considered only UBUN scenarios. Furthermore, both these techniques considered a more limited set of botnets than we did. Two botnets in these sets were also considered in our study (i.e., *PJApps*/Crowdroid and *DroidDream*/MADAM). It should also be noted that neither Crowdroid nor MADAM used real botnet data in their training phase: Crowdroid used data from a botnet developed by its producers for this purpose and MADAM used system call vectors from Trojanised malware and synthetic "botnet" call vectors with high numbers of system calls, as this appeared to distinguish botnets from normal apps in their preliminary experiments. The latter was created from the Trojanised malware vectors by interpolation.

For the remaining two techniques, which analysed OS system calls but not with a clearly specified experimental set up (e.g., unclear scenarios, not clear which botnets they used for testing), it should be noted that both of them used supervised learning, i.e., [312] and [314] and carry out pre-filtering of OS calls before training the ML classifiers. Also, none of them carried out any thorough and validated sensitive analysis, which is important in the light of the variance of the reported performance measures.

Table 2-7 - ML based botnet detection approaches

Technique / Platform	Normal Apps (N)	Botnets (B)	Features[1]/ Pre-filtering	ML Algorithms	Classification	Set Up	Performance	Sensitivity analysis
Amda[312] (Android emulator)	Training: 126 (Src: Google Play + other) Test: as above	Training: 250, not clear Test: not clear	Features: OS calls[2] Pre-filtering: Yes	NB, J48, RF, MLR	Virus Total	User Inter: simulated; Scenarios: not clear	TPR: 0.74; FPR: n/a PRC: n/a;	No
Aung & Zaw [317] (Offline)	Training: 500 Test: not clear	Training/Test: not clear	Features: permissions[3] Pre-filtering: Yes, info gain	K-means clustering, J48, RF, CART	Not clear	User Inter: simulated; Scenarios: Not clear, KBKN likely (WEKA)	TRP: 0.85 – 0.98 FPR: 0.08 – 0.16 PRC: 0.85 – 0.92	No
Crowdroid [313] (Android)	Training: SteamyWindow, Monkey Jump 2 Test: as above	Training: self-written malware (Trojan); Test: PJApps HongTouTou	Features: OS calls (all) Pre-filtering: Yes, info gain	k-means clustering (k=2)	Known N and B/M apps	User Inter: real, per N and 10 per B; Scenarios: UBUN	TRR: n/a; FPR: n/a PRC: 0.85	No
STREAM [316] (Android)	Training: 408 Test: 24[4]	Training: 1330 Test: 23[5]	Features: Permissions and device usage (37 features) Pre-filtering: Yes, not clear	RF, NB, NN(MLP, Bayes, MLR	Known N and B/M apps	User Inter: simulated (10000 for each N) Scenarios: KBKN, UBUN	KBKN: TPR: 0.87 – 0.97; FPR: 0.14 – 0.44; PRC: n/a; UBUN: TPR: 0.48 – 0.95; FPR: 0.16 – 0.33; PRC: n/a	Alg: Yes; Agp: No; N ap: PRC; B ap: PRC; Fet: No; Val: No
Masud et al. [314] (Android tablet)	Training/Test: Not specified	Training/Test: Not specified	Features: 5 different sets; not clear (one set with OS calls Pre-filtering: info gain, x[2]	NB, KNN, J48, MLP, RF	Not clear	User Inter: real, 2 hrs Scenarios: Not clear, KBKN likely	TPR: 0.17 – 0.90; FPR: 0.03 – 0.67; PRC: n/a;	Alg: No; Agp: No; N ap: No; B ap: No; Fet: Yes; Val: No
SCREDENT [318] (Android, logging)	Training: Not specified Test: Not specified	Training/Test: Not specified	Features: User triggered device activities[7] Pre-filtering: No	SVM	Users and SVM (not clear)	User Inter: not clear; Scenarios: Not clear	No data	No
Feizollah et al.[39](Android, offline)	Training: 6 see[11] Test: as above	Training: see Table 1 Test: see Table 1	Features: Ext. comms Pre-filtering: No	J48, KNN, SVM, NB, MLP	Known N and B/M apps	User Inter: not clear; Scenarios: KBKN	TPR (B): 0.93 – 0.99; FPR(B): 0.006 – 0.07; PRC (B): n/a	Alg:Yes; Agp: No; N ap: No; B ap: No; Fet: No; Val:No
MBotCS [40] (Android)	Training: 12, See Appendix Test: 12, See Appendix	Training: see Table 1 Test: see Table 1	Features: Ext. comms Pre-filtering: No	J48, KNN, SVM, NB, MLP, Box algs	Known N and B/M apps	User Inter: simulated; Scenarios: KBKN, UBKN	KBKN: TPR (B): 0.06– 0.99; FPR(B): 0.002 – 0.96; PRC (B): 0.62– 0.91; UBKN: TPR (B): 0.08 – 0.98; FPR(B):0.03 –0.95; PRC (B):0.52–0.66	Alg: Yes; Agp: No; N ap: Yes; B ap: Yes; Fet : No; Val:No
MADAM [319] (Android 4.0.1)	Training: 56 apps not specified Test: None	Training: Troanised malware[9] (Src: Contagio): Test: As in training	Features: OS calls[10]; SMS number; device idleness Pre-filtering: No	KNN		User Inter: Synthetic B data in training; Scenarios:KB	TPR (B): 0.66 – 1.0 FPR (B): n/a PRC (B): n/a	No
Zhao et al. [315] (non mobile)	Training: real traffic data, Warcraft Gamingpackets, Azureus Test: as above	Training: Storm, Waledac Test: Storm, Waledac Weasel (self-written), BlackEnergy	Features: Ext. comms[8] Pre-filtering: No	J48;	Known malware and normal apps	User Inter: Replayed N traffic mixed with B traffic; Scenarios: KBKN, UBKN	KBKN: TPR (B): 0.983 – 0.99; FPR(B): 0.01 – 0.017; PRC (B): n/a UBKN: TPR (B): 0975 – 0.9975; FPR(B):0.0025– 0.0225; PRC (B): n/a	Alg:No; Agp : Yes N ap: No; B ap: Yes Fet:Yes; Val: No
BotMiner [179] (non mobile)	Training: Multiple (network traffic capture) Test: as above	Training: rbot, sdbot, spybot, IRC-N, HTTP-1, HTTP-2, Storm, Nugache Test: as above	Features: External comms stats, device activities[6] Pre-filtering: int. comms, 1-way traffic	X-means clustering; Cross-plane correlation	Traffic of separate execution of Bs	User Inter: real (10-days logs; Scenarios: Not clear, KBKN likely	TPR (B): 0.75 – 1.0 FPR (B): 0.0 – 0.03 PRC: n/a	Alg: No; Agp: No; N ap: No; B ap: No; Fet No; Val:No

1. Feature types: OS calls, permissions on device, ext. comms/network flows (network APIs), intent features, use of device hardware components (e.g., device camera), device activities (scanning, downloading, exploit attempts) – see for a categorization of possible botnet features [320]
2. Read, write, Brk, getpid, Sigprocmask, Recv, lseek, Open, Msgget, Close, Semget, Semop, Clone, System_224, Dup, Fork, Ioctl, mprotect
3. Internet, change cnf, write_sms, send_sms, call_phone
4. quiz.companies.game, battery.free, android.reader, papajohns, androidPlayer, pregnancytracker, stylem.wallpapers, templerun, airpushdetector, unveil, mahjong, songkick, bbc.mobile.news.ww, mycalendarmobile, imdb, pinterest, craigslistfree, hydra, bfs.ninjump, tumblr, OMatic, box, gtask
5. Beauty.Girl, XrayScanner, mymovies, CallOfDuty, DWBeta, android.installer, txthej, bowlingtime, barcode, luckjesusblessings, topGear, Ipad2App, ad.notify1, gone60, skyscanner, antimosquitos, sipphone, rommanager, paintpro, zanti, youLoveLiveWallpaper, fingerprint
6. scanning, downloading, exploit attempts
7. Accept call, Change battery status, Uninstall App, Set time zone, Receive SMS, Activate 3G, Change wallpaper, Turn on GP; Install APKs Sensor: Accelerometer, Turn off GPS; Change clock; Turn on Airplane mode; Set ringer; Sensor: Gyroscope; Turn off Airplane mode Cancel Call / Hang up; Turn on screen, Set volume; Sensor: Rotation/Pitch; Turn of Terminal Connect AC; Send SMS; Get location, Go Home, Lock Terminal, Turn off 3G; Bright auto, Set bright auto, View SMS
8. TCP & UDP flow data and aggregated metadata: SrcIp, SrcPort, DstIp, DstPort, Protocol, AvePayloadPacketLength (AvePL), VarPL, NumPackets (NP), NP/sec, FirstPacketSize, TimeBetweenPackets(TBP), Number of reconnects, Address flow ratio
9. Lena.B, Moghava, TGLoader, OpFakeA, NickySpyB, Gone in 60 sec, KMin, Lotoor, DroidDream, Droid Kung Fu
10. open, ioctl, brk, read, write, exit, close, sendto, sendmsg, recvfrom, recvmsg
11. Facebook, Twitter, Chrome, Google R, Flipboard, YouTube

2.5.5 Mobile botnet detection specificity

Even though several mobile botnets break out in recent years and the mechanism has been unveiled through tracking and analysing, how to detect the mobile botnet malware before they perform the attack to cause a loss is still a severe problem. So, different between the mobile malware detection techniques and the conventional botnet detection techniques are the important reference material for designing mobile botnet detection system. We will compare these techniques with the requirement of mobile botnet detection and make the perspective of future mobile botnet detection techniques.

2.5.5.1 Mobile botnet and conventional mobile malware

Although the mobile botnet malware is one of the mobile malware, the several new features of mobile botnet make it more threatening than the other malware. We just list the three features and give a detail discussion of the insufficient of current mobile malware detection techniques and systems.

- Communication

The main characteristic of mobile botnet malware is the ability of communication with botmaster and other bots. The communication channel between the bots and botmaster is called C&C channel. From current mobile botnet incidents and the researches about the creation mobile botnet, we can find that there are a diversity of communication channels including SMS-based, HTTP-based, Hybrid-based (SMS and HTTP), Push Service-based, Social Network Message based and even Sensor-based.

Though the extra communication with the botmaster and other bots increase the risk of was detected, there are some techniques designing to hide these communication messages as far as possible such as URL flux techniques [33]. However, the communication which is based on the C&C channel enhances the mobile malware with

several features. For example, the mobile malware on the bots can change the type and target of attack according to the different command from the botmaster. The botmaster can adjust their attack strategy based on the current situation through the communication.

Consequently, if we want to build a mobile botnet malware detection system, how to unveil the communication of the mobile botnet is the highest priority issue to solve.

- Upgrade

As the mobile malware problem is serious, there are more and more organisations, and companies give some concentration on the mobile anti-malware software. According to the analysis of the suspicious application in the real network, the malware databases constantly update.

So for the controller of the malware or the attacker, they need to continuously upgrade their mobile malware to evade the new means of detection. The general mobile malware without the communication with the attacker cannot actively upgrade. They can only spread the new version of the malware to the network and wait for the mobile devices to be infected again. However, the botnet mobile malware can actively get the update package according to the command that sent by the botmaster. This mechanism is the enhance phrase in the botnet lifecycle which has discussed in Section 2.1.3.

- Latency of malware

Once infected by the most of the conventional mobile malware, the harmful behaviour will be performed at regular intervals. However, the activities of botnet malware are controlled by the remote server. So, without the attack command from the botmaster, the botnet malware can hide in the mobile device without any harmful features. It can make the botnet malware escape the active scanning by some anti-malware software.

2.5.5.2 Mobile botnet and conventional botnet

Because of the accumulation of the conventional platform botnet detection research, the ideal solution for mobile botnet detection approach is making a transplant of the current conventional platform botnet detection method. However, there are some gaps between the mobile botnet and conventional platform botnet detection. We give several important factors which should be separated discussed in the mobile botnet detection.

- The C&C Channel.

C&C channel is the most important part of the botnet malware. Through the discussion of the taxonomy of the convention platform botnet and mobile botnet, we can find that there is a more diverse C&C channel for the mobile botnet. The special channels for C&C give some trouble for detection system, but they can also be regarded as new detected objective in mobile botnet detection approach.

SMS as the special characteristic on the mobile phone has been used in most of the mobile botnet creation [44, 47, 49, 50, 52, 53]. The SMS is text-based and system-independent which supported nearly by all the existing phones. The other feature of SMS is the simple and reliable. Literature [35] defines a measurement based on the feature of the SMS to reveal the abnormal behaviour on the mobile devices.

There is also some mobile botnet combining multiple communication channels to realise the C&C channel, such as the paper [49] combine the SMS and HTTP to transfer the commands. The Andbot [28] combines the SMS with the microblog feed as the C&C channel to transfer information between botmaster and bots.

- Data Source for Detection

According to the review of the conventional platform botnet detection system, we can know that considerable part of the approaches monitors the traffic on the router of the traditional network to detect the botnet of the network. However, it is difficult to get the

data source like that on the mobile network. Except for the Wi-Fi network, most of the mobile devices are connected to the mobile network such as GPRS, 3G and 4G.

Consequently, the detection data source for mobile botnet detection may be limited to the mobile device itself in the current situation. Moreover, digging more information on the mobile device to detect is the work of the future mobile detection approaches.

- The Damage

Though the function between the mobile device and the conventional platform is more and more similar, the mobile handset as a part of the daily life has more important additional responsibility than the conventional platform. So, we can find that there is more damage in the mobile botnet and the attacker can also get more profit from the mobile botnet.

The potential harmful behaviours of a mobile botnet which lists in the Table 2-5 show the attack method of the current mobile botnet. Except the attackers can get the interest directly through the premium-rate telephone number, they can also control the botnet to spread spam email, SMS and junk mobile application to earn a profit. The other characteristic of the mobile botnet is that the period of the damage is shorter than conventional platform botnet. Once some mobile devices infected the malware, the botmaster can control the Bot to perform a harmful attack to get money as soon as possible.

2.5.6 Open issues

Though some mobile botnet detection systems have been proposed in recent years, most of them are signature-based, and host-based detection approaches. There are some limitations for these detection methods:

1. Only apply to the known mobile botnet; Most of existing mobile botnet detection techniques do not discuss how to detect the unknown botnet. Meanwhile, the update of the botnet is very frequency. So, the current detection method is hard to keep up with the pace of change of botnet.
2. Not fully consider the limitation of mobile device performance; Though the mobile had grown more capable, compare with the desktop the resource of mobile is still limited. So, the detection on the mobile device should concern about the performance such as CPU usage and battery consumption.
3. The individual mobile device is hard to collect enough information for analysing; The mechanism of botnet decides that whether the infected script is active or not is controlled remotely. Meanwhile, anti-detection techniques are widely used in new generation botnet. It is hard to unveil the whole pattern of a botnet in single mobile.

By contrast, most of the conventional platform botnet detection systems are network-based and anomaly-base detection methods that can take full advantage of the collaboration of different hosts in the network and the clustering phenomenon of the botnet malware.

So how to design a network-based and anomaly-based botnet detection system for mobile platform is an open issue.

Chapter 3 The MBotCS Detection System

3.1 Overall framework

Our mobile botnet detection approach, called MBotCS, has been implemented as a system that has the architecture shown in Figure 3-1. This system has six main types of components. These are the *mobile verification components (MVC), the data broker, the notification broker, the feedback broker, the data analysers* and *the feedback processor*.

MVCs are components which are deployed on the individual mobile devices. Each mobile device has its own MVC. After it is deployed, an MVC captures the traffic and system call of specified applications on the device at run-time, pre-process the packets and calls in it and prepare them for transfer to the *data analyser* for the detection of suspicious behaviour. This transfer takes place through the *data broker*. MVC also receives control signals indicating suspicious behaviour from the *data analyser* and acts according to them and a local user policy. Control signals block all packets and calls that have the characteristics that made the *data analyser* to detect them as “infected”.

The two components of *data broker* and a *notification broker* are responsible for transmitting the traffic and system calls collected by the MVCs of different mobiles to a *data analyser* and transmitting notifications (control signals) produced by the data analyser in the opposite direction. The two brokers transmit information based on a publish-subscribe event reporting infrastructure. It enables keeping the analysis and

detection capabilities separate from the actual mobile devices, which is important for ensuring the scalability of the implementation in the presence of large numbers of mobile devices and traffic analysers. The publish-subscribe architecture is appropriate for real mobile device networks in which mobile devices may come and go quickly and unpredictably. It is because it does not overload the mobile communication infrastructure with message transmissions required for monitoring [321]. The brokers maintain subscriptions to the "channels" between publishers of messages and their respective subscribers.

When the system enrols a new mobile device, the mobile device will send an advertisement message to the *data broker* to notify that they will publish captured data to the analyser. Following this, the *data broker* will subscribe the mobile device to receive the captured data and forward to the *data analyser*. In parallel, the mobile device will subscribe the *notification broker* to receive the notifications message generated by the analyser.

The *data analyser* is the core component of the detection system. It analyses the mobile traffic and system calls that it receives from the MVCs installed on the mobile devices through the *data brokers*, at runtime based on training that has been carried out using the machine learning algorithms discussed in Chapter 2, and the data sets held in the *training database* of MBotCS. MBotCS might deploy more than one *data analysers* depending on the number of the mobile devices and the volume of their captured data. All *data analysers*, however, are trained using the same training dataset, i.e., the *Training Database* in Figure 3-1.

The *feedback processor* in MBotCS is a component that has been introduced to expand and optimise the training data. More specifically, when the *data analyser* finds some malicious traffic or system calls, it sends a warning notification signal to the mobile device. The mobile users are given the opportunity to provide feedback for the warning

manually. This feedback is passed on the *feedback processor* component, which may decide to add the traffic or system calls sample into the training database.

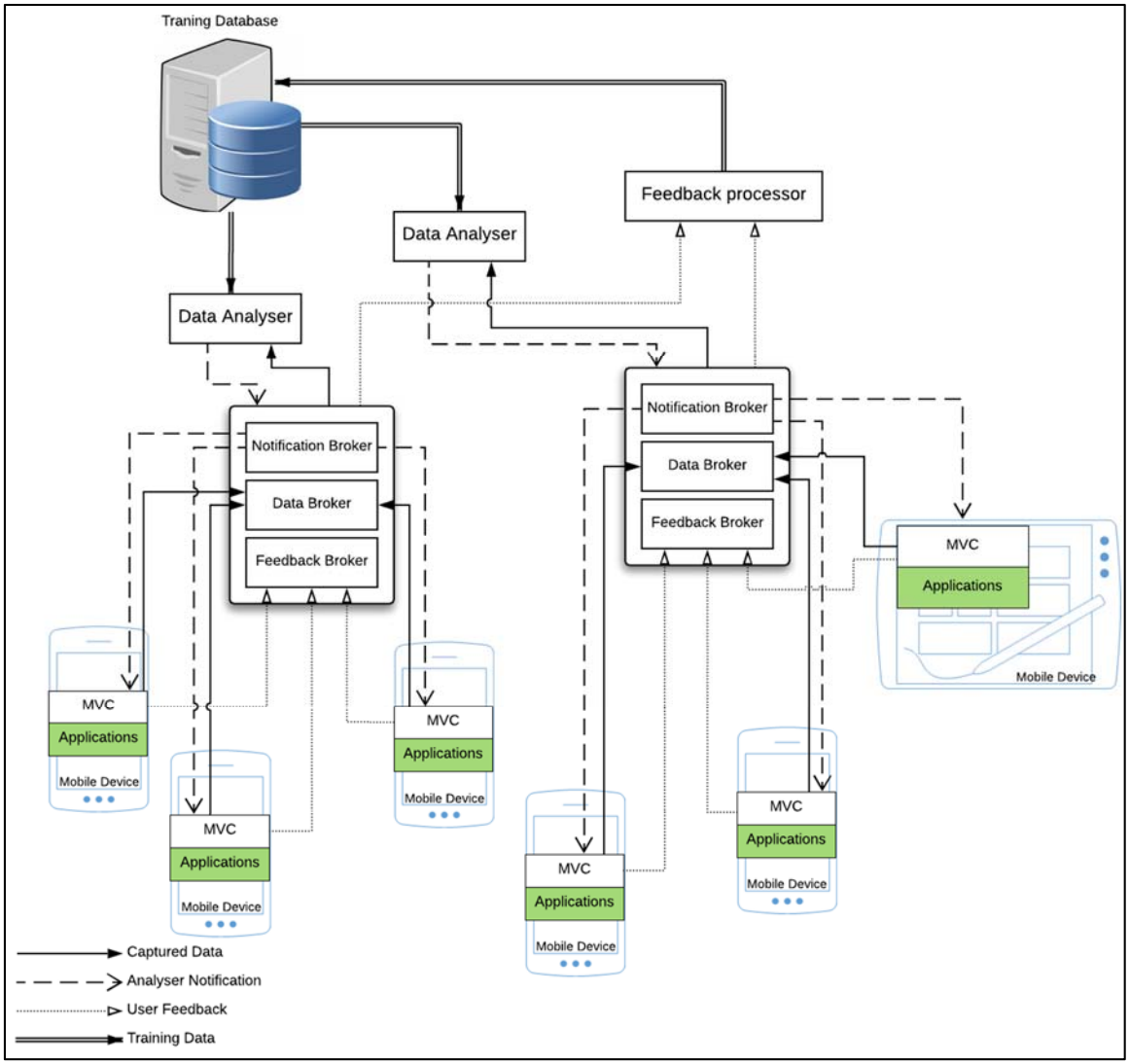


Figure 3-1 - The overall architecture of MBotCS

3.2 Introduction of components

3.2.1 Mobile Verification Component

The Mobile Verification Component (MVC) is deployed on the end client mobile device in MBotCS system. Firstly, the MVC has the ability to collect the traffic and system call on the mobile device. The Android VpnService API [322] provides the possibility to monitor the traffic easily on the mobile and *tPacketCapture* [323] application is used to capture and save the traffic data as a file with PCAP format. The mobile system call data is captured by Strace [324]. And we develop a bash shell script to make use of Strace to intercept the required system call with proper format for analysis. The detail of how to use the *tPacketCapture* and strace bash shell script to collect the traffic and system call can be found in Section 4.3.

Apart from the data collectors, there is a data pre-processor module in the MVC. Before passing data to machine learning classifier, we need to convert the data to the acceptable format for the classifier. The pre-processing includes data clean, feature selection, data statistical analysis etc. The specific data process workflows for traffic and system call are described with the corresponding experiments in Section 4.1.

Finally, the MVC has a user interface to notify the user the malicious behaviours with useful information and help the user to block suspicious processes.

3.2.2 Data Broker

The data broker is the component that is responsible for receiving the data from the MVC. We use the Cloud Firestore [325] which is a real-time database to receive the data in real-time. A real-time database is a database system which uses real-time processing to handle workloads whose state is constantly changing [326]. Every MVC has a unique

real-time database entry path on the Data Broker, and the MVC uses server-sent events HTTP request [327] to subscribe the corresponding database entry. When there is new data collected and pre-processed on MVC, the data will be pushed to the database entry immediately. Meanwhile, the MVC can receive the status receipt to confirm the data is received by Data Broker successfully.

3.2.3 Notification Broker

The Notification Broker also uses the real-time database technique as a data publisher whose role is different with the Data Broker (which is data subscriber). When data analyser detects the malicious behaviour for the submitted data, it will push an alert notification to the corresponding MVC database entry in Notification Broker. Then the Notification Broker publishes the information automatically without waiting for the connection with the MVCs. The MVCs that observe the specific database entry will receive the alert notification in real-time.

Benefited from the publish/subscribe pattern architecture, the notification broker can be subscribed by multiple MVC and other services such as warning centre which can make a better strategy to block or prevent the specific malicious activity based on the entire system.

3.2.4 Feedback Broker and Processor

The feedbacks include positive feedback and negative feedback. When the MVC receive the notification about the malicious warning for the specific application installed on the mobile device. The MVC will suggest the user block or uninstall the application based on the warning information. However, there's still some probability that some normal applications produce several behaviour patterns which are similar to malicious. So, if the warning is false positive, the user still has the opportunity to ignore it and send

feedback to the analyser to improve the classifier. In Feedback Processor, the customised policy can be configured to the analyser for separate MVC. Taking advantage of user-specific feedback, the warning notification could be more personalize and intelligent. Certainly, the positive feedback can be used to update the training dataset to improve the performance of analyser.

Feedback Broke and Processor are also based on the real-time database. MVC push user feedback to the specific Feedback Broker. Then the Feedback Processor monitors the change of Feedback Broker database entry to get the content of feedback and generates a policy registry for the specific MVC in the data analyser.

3.2.5 Data Analyser

The Data Analyser is responsible for processing collected data and generate the classification result. It has two modes: cloud mode and offline mode. If the MVC is available to access the network, the cloud mode will be enabled. However, if the mobile device cannot access the network or the MVC has no permission to access the network on mobile, the offline mode will be activated alternatively. Cloud mode has higher priority because of the relatively higher system resource consumption for machine learning analysis.

The core of the data analyser is a Java machine learning engine which is based on open source machine learning analyser. Apart from making use of the atomic algorithms providing by WEKA [253] directly, we also make several aggregated algorithms that extend the Vote [328] class. Meanwhile, an enhanced version of Evaluation [329] class is developed to support more analyser result output for research analysis. The detail of data analyser information can be found in Section 4.1 experiments introduction.

3.3 System Security

Because the overall architecture of the system is cloud-based, meanwhile the mobile device is based on Android which is open source system. So there are several security issues need to be considered.

3.3.1 Communication Security

All the network connections between the components use the transport layer security (TLS) [330]. Transport Layer Security whose predecessor is Secure Sockets Layer (SSL) is a family of cryptographic protocols that utilize X.509 certificates [331] and asymmetric encryption to secure the HTTP connection. The TLS can verify the identity and prevent the man-in-the-middle attacks.

In order to prevent the data leaking and data tampering, A custom private/public key encryption and signature process to protect the communication thoroughly. For every component, a pair of private/public keys is generated during the deployment. Meanwhile, every component maintains a public key list of existing components in the system and keep the list up to date. Assume we transfer a set of data *D* from component *A* to component *B* (the component could be MVC, broker, etc). The private and public keys of A and B are denoted as (Pr_a, Pu_a); (Pr_b, Pu_b). Then we will give a solution to use these resources to make the transferred data secure and tamper-resistant. The sending and receiving processes are described separately and both have two phases:

- Sending process (on component A):
  - Phase1 (encrypt data): During this phase, the plaintext data will be encrypted by using the asymmetric encryption algorithm such as RSA [332] with the public key Pu_b of component B (receiver party). Then the data will be change to $D_{encrypt} = Encrypt(D, Pu_b)$.

- Phase2 (sign data): The encrypted data will be signed with the private key of component A. Then the final data is ready to send over the network. The final data could be denoted as

$$D_{final} = D_{encrypt} + Signature$$

$$Signature = Signature(Digest(D_{encrypt}), Pr_a)$$

The digest algorithm such as MD5 [333] is used to improve the performance of the asymmetric signature algorithm.

- Receiving process (on component B):
  - Phase1(verify signature data): The component B will receive the final data which is generated by Phase2 of Sending process. Firstly, the component B need to verify the signature of in the final data with the public key of component A: $Digest = Verify(\text{Signature}, Pu_a)$. If the digest is matched with received data, the component B will trust that the received data is not tampered by mid man.
  - Phase2(decrypt data): After making sure the data is the original data sent from the component A, the component B start to decrypt the data to plaintext by using the same asymmetric encryption algorithm in Sending process with the private key of component B: $D = Decrypt(D_{encrypt}, Pr_b)$. Finally, the component B get the plaintext data send from component A.

Now let us analyse why this solution can make sure the data cannot be stolen or be tampered. Firstly, because the transferred data is encrypted by the receiver public key, so the only parties who hold the corresponding private key can decrypt the data in theory. So even the hackers get the data by some methods such as sniffing, they cannot get the plaintext data without the private key. Secondly, because we add the signature of the sender to the data and the signature can be verified by the sender public key. This signature can only be generated by the parties who hold the corresponding private key. So even the hackers tamper the data, they cannot make a new signature which can be

verified successfully with sender public key. Certainly, the validity of this solution is based two points: one is the private key should be kept in the component safely. The other one is the public keys list should be protected by a certificate authority to make sure all the public keys should be matched with the corresponding private key of components.

3.3.2 MVC Application Security

The other security issue is the protection of MVC. Because the MVC is deployed on the Android mobile device and the MVC is Java-based Android application, so there are a lot of methods to decompile the android installer file (APK). If the application is decompiled by the attackers, they will understand the mechanism of detection and analysis which help them to prevent the detection. So, the anti-decompile techniques should be applied to the MVC development.

There are several techniques can be used to prevent malicious decompile behaviour. The most general one is the Proguard [334] which is a tool to obfuscate Android application source code. After shrinking the source code of the application by Proguard, the stack trace will be difficult to read because the method names are obfuscated. So even the hackers decompile the installer file, they will get obfuscated code that is hard to analyse.

The other one is programming the part of the application with Android Native Develop Kit (NDK) which is more advanced technique. The Android NDK is a toolset that lets you implement parts of your app in native code by using languages such as C and C++ [335]. The original intention is helping reuse code libraries written in other languages for certain types of apps. However, this method can also help to protect the source code from decompiling. During the development of MVC, the sensitive and important parts of code such as feature selection process are programmed by C++ language and build a library.

Then the Java-based code use the NDK to load the library and execute the corresponding functions that are exposed in the library.

Through these protection methods, the MVC which is deployed on the end client will be hard to be decompiled and statically analysed.

Chapter 4 Experimental Evaluation of Mobile Botnet Detection

4.1 Overview

The prototype implementation of MBotCS that was described in Chapter 3 has been used in a set of experiments that were carried out to evaluate our approach. The purpose of these experiments was to evaluate:

(1) The accuracy of the classifications of mobile applications (as "normal" or "infected" applications participating in botnets) produced by it.

(2) The performance in terms of energy consumption and execution time on Android devices. These two criteria were selected as they constitute key performance indicators for our approach.

The ML algorithms that we used in the experiments were the five supervised machine learning algorithms and the group of machine learning box algorithms, which we discussed in Section 2.4. More specifically, we used the following atomic ML algorithms

- *Naïve Bayes* [336].
- *Decision Tree (J48)* [241].
- *K-nearest neighbour (KNN)* [242].

- *Neural Network (NN) Perceptron* [247].
- *Support Vector Machine (SVM) [337]*.

These atomic ML algorithms we chose are used widely in current research based on the information in Table 2-7 (Naïve Bayes [used in 5 approaches], Decision Tree J48 [used in 5 approaches], *K-nearest neighbour* [used in 4 approaches], Neural Network [used in 4 approaches]; Support Vector Machine [used in 3 approaches]).

In addition to atomic algorithms, we used *ML boxing*, a technique where classifications of the individual ML algorithms are aggregated in order to improve the accuracy of results. For this purpose, we used three different aggregation methods:

- *ML-BOX (AND):* In this aggregate classifier, an instance of the dataset was classified as infected if ALL the individual classifiers indicated it as infected. Otherwise, the instance was classified as normal.
- *ML-BOX (OR):* In this aggregate classifier, an instance of the dataset was classified as infected if AT LEAST ONE the individual classifiers indicated it as infected. Otherwise, the instance was classified as normal.

ML-BOX (HALF): In this aggregate classifier, an instance of the dataset was classified as infected if MORE THAN HALF of the individual classifiers indicated it as infected. Otherwise, the instance was classified as normal.

ML boxing was used to aggregate: (a) the results of all individual classifiers and (b) the results of only J48 and KNN as these algorithms outperformed the rest in the single algorithm based classifications (see Section 4.5.2). In the following, we will refer to the outcomes of (a) as “ML-BOX (.)” and the results of (b) as “ML-BOX+ (.)”. In the case of ML-BOX+(HALF), if the J48 and KNN algorithms classified the instance of the dataset in the same class, ML-BOX+(HALF) generated the same common classification.

When J48 and KNN were in disagreement, ML-BOX+ (HALF) generated a classification based on the outcome of the three remaining classifiers only.

The experiments were based on capturing and analysing network traffic and system calls from both malware botnet and normal applications. The analysis of network traffic was the focus of the first set of experiments. The analysis of system calls was the focus of the second set of experiments. Before demonstrating the results of these sets of experiments, we introduce the experimental methodology that was used to set up and carry out the experiments. This includes the selection of the mobile botnet and normal applications that we used in the experiments; the overall workflow for carrying them out; the ways of capturing the data sets used for analysis; and the ways of measuring the performance of different algorithms.

To evaluate and compare the results arising in the different experiments, we used the following performance measures which have been introduced in Section 2.4.3.1:

- True Positive Ratio (TPR) also called Recall
- False Positive Ratio (FPR)
- Precision (Prec)
- Area Under Curve (AUC)

Beyond the point measures provided above, we also used value range (VR) criteria to characterise the performance of an algorithm as “very good” or “weak” with respect to the individual measures based on value ranges. The VRCs used for this purpose were:

- TPR: VERY GOOD if TPR ≥ .9, WEAK if TPR < .8
- FPR: VERY GOOD if FPR ≤ .05, WEAK if FPR > .1
- PRC: VERY GOOD if PRC ≥ .9, WEAK if PRC < .8

4.2 MVC of MBotCS System on mobile device

The architecture of the MVC components in MBotCS system, which are deployed on mobile devices, is shown in Figure 4-1. There are four main components of this type: the *data pre-processor*, *machine learning analyser (ML analyser)*, *user interface* and the *training dataset*. The architecture also uses *tPacketCapture* [323] and *strace* to capture mobile traffic and system call, respectively. *Gsam Battery Monitor* application is used for monitoring the mobile battery consumption. The mobile device with deploying these components is called monitoring-enabled mobile (MEM).

All the traffic passing through the mobile device is captured by tPacketCapture and stored in the pcap file. Moreover, the system call that invoked by specified applications is captured by strace and stored in log file. Both of two files will be persisted on the SD card of the mobile device. The data pre-processor reads the pcap file periodically and converts any incremental (new) data that it finds in it into the standard structure file for the ML analyser. Meanwhile, read the system call log file and generate the required formatted data entry for ML analyser. The ML analyser trains the classifiers by the training dataset and classifies the captured traffic and system call in real-time as infected or normal. Traffic and system call classifications are shown on the user interface, warning the users to block suspicious applications (i.e., applications that generated traffic classified as infected).

During the stage of infection detection, the captured traffic and system calls will be analyzed continuously. When the infection is detected, users get warnings through a GUI is shown in Figure 4-2. The malicious data analysis result will be highlighted with red colour relative to the grey colour. Meanwhile, the corresponding log will be recorded with the detail information for the detected infection which can help the user to locate the malicious application or process on the device.

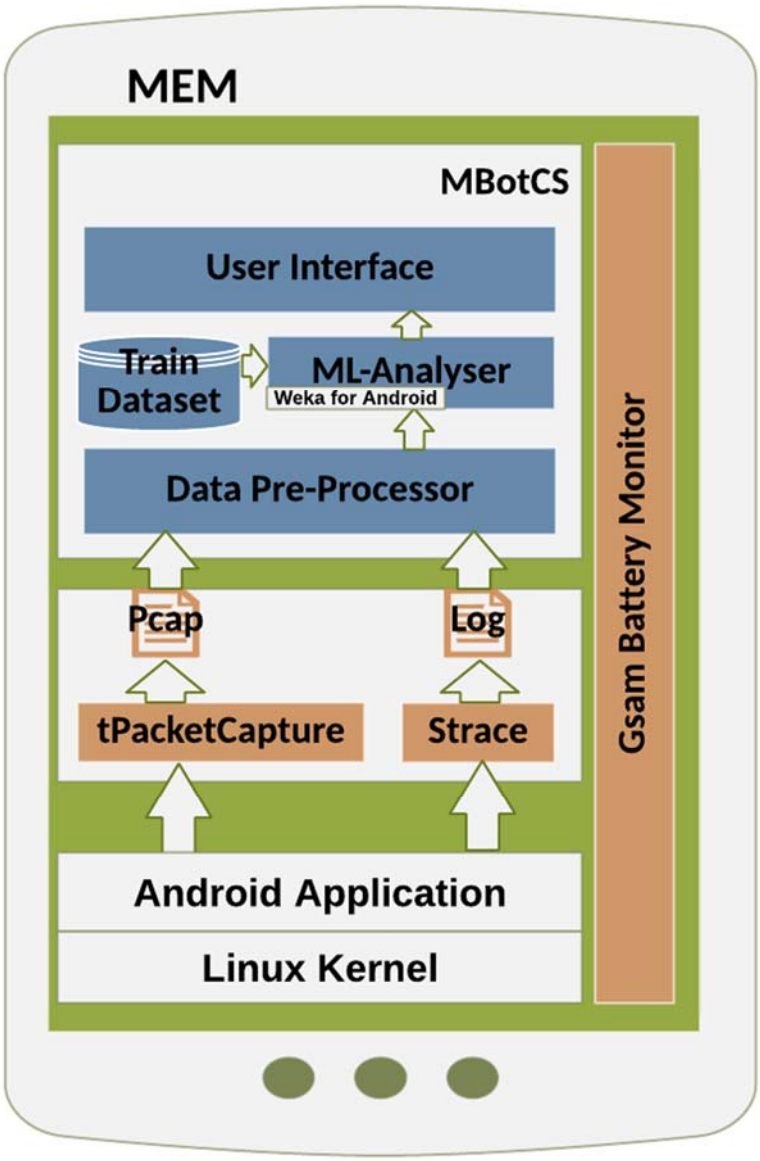


Figure 4-1 - Architecture of MVC on mobile devices

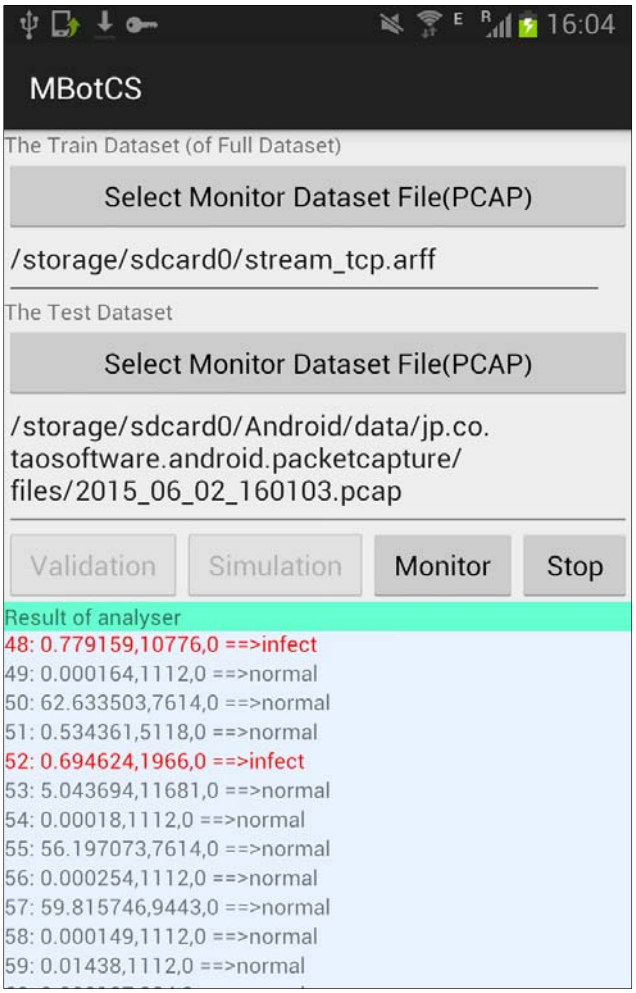


Figure 4-2 - The GUI of MBotCS

4.3 Implementation of MVC Component

4.3.1 Network traffic capture

We use *tPacketCapture* for capturing the network traffic and pre-configure training dataset with the application. For the advanced user, we also designed a button for

selecting the specific training dataset file which is shown in Figure 4-2. In the scenario, we select the monitor target PCAP file as:

/storage/sdcard0/Android/data/jp.co.taosoftware.android.packetcapture/files/2015_06_02_160103.pcap

When we launch the application of *tPacketCapture*, it will create one pcap file named with the current date and time (*2015_06_02_160103.pcap)* for collecting network traffic.

4.3.2 Pcap parse and pre-processor

One of the most important for MBotCS is the PCAP file parser component. Our implementation is based on the *JNetPcap* [338], an open source Java library for network analysis. This library contains a Java wrapper for nearly all *libpcap* library native calls and provides a large library of network protocols.

We also use *JFlowMap* to parse the PCAP file which is one of data structure in the *JNetPcap*. This structure can filter the flow information, called stream, in the PCAP file. The code for converting PCAP to the stream data using the *JFlowMap* is as follows:

```
JFlowMap superFlowMap = new JFlowMap();
pcap.loop(Pcap.LOOP_INFINITE, superFlowMap, null);
Iterator iterator = superFlowMap.entrySet().iterator();
```

To improve the effectiveness of parsing the PCAP file, we also studied the structure of the file and tried to read it incrementally. The PCAP file is read at regular intervals and incrementally. For this, we need to remember the last visit point and next time to read from the last visit point. Meanwhile, according to the PCAP file structure, we need to construct the new temporary PCAP from the monitored file. The code that supports this is shown in APPENDIX A.1. In the code, *TimerTask* is used for reading the PCAP file

repeatedly. For every time read the file, the context information such as the current line will be stored and used for next reading action.

4.3.3 WEKA based machine learning analyser

To realise the Machine Learning Analyser, we used *Weka-for-Android*, i.e., a Java library modified from the WEKA for adapting Android platform. The used library implements the atomic machine learning algorithms that we discussed in Section 2.4:

```
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.classifiers.functions.SMO;
import weka.classifiers.lazy.IBk;
import weka.classifiers.trees.J48;
```

In addition, we implement six aggregate algorithms, which combine the results of the atomic machine learning algorithms. These algorithms are *BOX-AND*, *BOX-AND+*, *BOX-OR*, *BOX OR+*, *BOX HALF* and *BOX HALF+*, which will be discussed in Section 4.1. The code of this component is shown in APPENDIX A.2.

4.3.4 User interface

The user interface of the MVC is not very complex; the user needs to choose the storage location for capturing traffic and system call in real-time and advance user can custom the training dataset. The two buttons: *Monitor* and *Stop* can be applied to launch and terminate real-time detection. The detection result will be displayed at the bottom of the screen. The UI design XML document (for Android) is shown in APPENDIX A.4.

4.4 Analysed normal and botnet applications

To collect data for our experiments (both the first and second set), we deployed 12 normal applications and 163 mobile botnet malware applications.

The botnet malware applications that we used in the experiments were grouped in 12 families, which are shown in Table 4-1. These applications were selected from the *MalGenome* project [339], according to their level of pandemic risk (according to the number of captured infected samples when these malware families were discovered). MalGenome has collected more than 1200 Android malware applications, the vast majority of which (i.e., more than 90%) are botnets.

The normal applications that we used were: Chrome, Gmail, Maps, Facebook, Twitter, Feedly, YouTube, Messenger, Skype, PlayNewsstand, Flipboard, and MailDroid. These applications were selected due to their popularity. Also, to be certain about their genuineness, all of them were downloaded from the Android Official APP Store (Google Play). The normal applications that we used are shown in Table 4-2.

Based on the dataset we used for training and testing in ML algorithm, we define three types of scenarios:

- Known botnets and known normal applications (KBKN scenario): The training dataset includes data in testing dataset both of botnets and normal application.
- Unknown botnets and known normal applications (UBKN scenario): The training dataset only includes data in a testing dataset of a normal application. The data of botnet in the testing dataset is totally different with the data in training dataset.
- Unknown botnets and normal applications (UNUB scenario): Both of botnets and normal application data are totally different between training and testing dataset.

Table 4-1 - Botnet malware families

Malware Family	No	Description	pandemic risk
1. AnserverBot	20	This a Trojan botnet that aims to remote control users' cell phones. The infected host app has two hidden apps with names *anservera.db* and *anserverb.db*. Moreover, when host app runs, it will pop up fake upgrade window to mislead user install *anservera.db.* At runtime, *anservera.db* and host app can dynamically execute the command in *anserverb.db.[285]*	187 samples captured
2. BaseBridge	20	This a Trojan botnet that attempts to send premium-rate SMS messages. When the infected app is installed, the malware will run some malicious (BridgeProvider, AdSmsService, PhoneService, ZlPhoneService) in silence [340].	122 samples captured
3. DroidDream	12	This botnet hijacks applications, steal the phone information such as IMEI, IMSI number and forwards them to botmaster. Meanwhile, it can also download and install other apps then tracked them in the background [264, 265, 341].	62 samples captured
4. DroidKung Fu3	20	This botnet forwards information to the remote server and downloads additional payload. This malware makes use of encryption for the remote server URL and exploited code to evaded static code analyser. [342].	309 samples captured
5. DroidKung Fu4	20	This botnet is a more sophisticated version of DroidKungFu which equipped with a new mechanism to protect from anti-virus software detection by obfuscating remote control URLs and masquerading an embedded app as the official Google Update [343] [344].	96 samples captured
6. Geinimi	20	It is a Trojan botnet that opens a backdoor to perform several functions such as sending SMS message and stealing sensitive information to the remote server [345].	69 samples captured
7. GoldDream	20	It is a Trojan botnet that monitors incoming and outgoing SMS message, phone calls and sends the detail to log back to the remote server. Meanwhile, it also collects other sensitive data such as subscriber ID and SIM card's serial number [346].	48 samples captured
8. KMin	2	It is a Trojan botnet that attempts to send user sensitive information such as device ID, subscriber ID and the current time to a remote server [347].	53 samples captured
9. Pjapps	20	It is a Trojan botnet that opens a backdoor and retrieves commands from servers such as push(<smscontent>,<smsurl>,<tel>), blacklisting, response blocking and so on. By using these commands, the botmaster can control infected device to send a premium-rate message and steal sensitive information [348].	59 samples captured
10. Plankton	9	It is a botnet that forwards user sensitive information to a server such as a device ID and IMEI number and collects the browser history and modifies the browser's bookmarks [349].	20 samples captured

Table 4-2 - Normal applications

Name	**Description**
Chrome	Chrome is a popular free web browser developed by Google.
Gmail	Gmail is a free advertising-supported email service provided by Google
Maps	Maps is a mobile web mapping service application provided by Google
YouTube	YouTube is a video-sharing website owned by Google since late 2006.
Play Newsstand	Play Newsstand is a digital newsstand and news aggregator application.
Facebook	Facebook is a famous online social networking service.
Twitter	Twitter is an online micro social networking service
Messenger	Messenger is an application providing instant messaging service.
Flipboard	Flipboard is a social-network aggregation mobile app.
Feedly	Feedly is a mobile application that can subscribe news.
Skype	Skype is a telecommunications application providing video chat and voice calls.
MailDroid	MailDroid is a WebDAV/POP3/IMAP Idle Push mail client on Android device

4.5 Experiments for network traffic analysis

4.5.1 Workflow of experiments

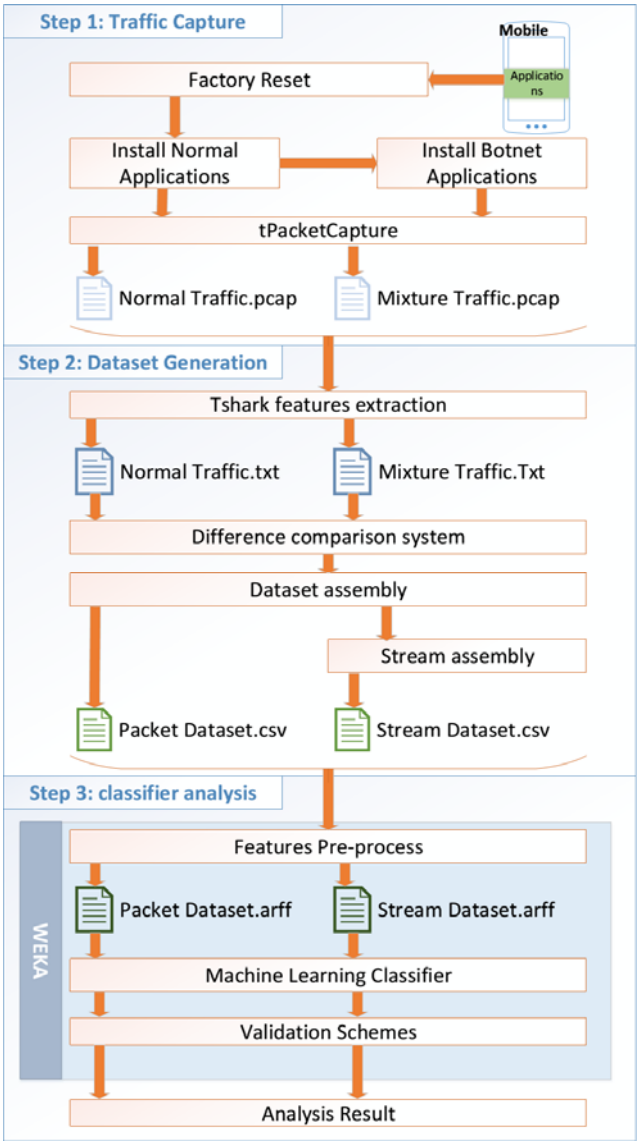


Figure 4-3 - Workflow of experimental training

Figure 4-3 shows the workflow that was used to set up the experiments. This workflow consisted of 3 main steps: (1) the capture of mobile device traffic for further analysis (traffic capture), (2) the generation of the data set for the experiments (dataset generation), and (3) the experimental use of different ML classifier algorithms as the basis for training the traffic analyser (classifier analysis). These steps are discussed in detail in the following.

The mobile device that we used in the experiments was a Samsung Note 1st generation (GT-I9228) running Android version 4.1.2 (i.e., *Jelly Bean*). Jelly Bean was the most frequently used version of Android with more 50% of installations in November 2014 [350], when the execution of these experiments started. To avoid interference with other applications, the mobile device used for the experiment was reset to the default Android OS settings before the experiments started.

4.5.1.1 Network traffic dataset generation (first set of experiments)

To generate the experimental traffic data, we created two different set ups of the mobile device. The first set up (*set up A*) contained only normal applications. The second set up (*set up B*) contained both normal applications and malware. A device with each of these two set-ups was used to generate traffic, over a 24-hour trial period. Over the 24-hour period, in the case of *set up A,* we carried out 120 transactions using only the normal applications of the set up. The same transactions were also executed at exactly the same time in the trial period of *set up B*, which also lasted 24 hours. Our assumption behind this experimental design was that, whilst using the normal applications of *set up B* to carry out the 120 transactions, the mobile botnet malware families that were part of the set up would also be activated by themselves or by their botmaster and would generate infected traffic. This assumption was correct as we discuss below.

To capture raw traffic data from the mobile device, we used *tPacketCapture* and stored it in the *pcap* file. The captured *pcap* file was processed to generate a structured dataset for further analysis. In particular, the traffic data in the *pcap* file were processed to extract features that we considered important for the classifier analysis phase, namely the *Source IP Address, Destination IP Address, Protocol, Frame Duration, UDP Packet Size, TCP Packet Size, Stream Index*[5] [351] *and the HTTP Request URL*. To extract these features from the raw packet lines within the pcap traffic file, we used *Tshark* [352], i.e., a command line pcap file analysis application integrated into Wireshark. The command Tshark batch file used to extract the above features is shown in Figure 4-4.

```
   setlocal enabledelayedexpansion

   set outputFormat=.txt

   for %%f in (*.pcap) do (

   tshark –r %%f -o tcp.calculate_timestamps:true -n -T fields -e
ip.src -e ip.dst -e ip.proto -e frame.time_delta -e udp.length -e
udp.stream -e tcp.len -e tcp.stream -e http.request.uri
>%%f%outputFormat%
```

Figure 4-4 - Tshark command for extracting features

The packet traffic data that were obtained from this step were further processed in order to label them as “normal” or “infected”. This step was performed by a script that we developed to compare the mixed traffic file generated by *set up B* with normal traffic file generated by *set up A*. More specifically, to label the different packets in the traffic of set up A and set up B, we considered three features of the packets: the *Source IP Address,* the *Destination IP Address* and the used *Protocol*. The set of legitimate (i.e.,

[5] Stream index is a number applied to each TCP conversation seen in the traffic file.

non-infected) combinations of values of these features was established by analysing the normal traffic data generated from set up A first. These combinations were subsequently expanded further through combinations with legitimate public IP addresses taken from Google Public IP address [353]. Based on this, we generated a three feature pattern-matching library, an extract of which is shown in Table 4-3. Subsequently, every packet in the mixed traffic generated by *set up B* was compared with the patterns in the library. If the packet had a combination of values for Source IP Address, Destination IP Address, and Protocol matching a pattern in the library, it was labelled as "normal". Otherwise, it was labelled as "infected".

Table 4-3 - Pattern-matching library

Normal Pattern			Public IP Address Pattern		
Source IP	Destination IP	Protocol	Source IP	Destination IP	Protocol
10.8.0.1	74.125.71.100	6		216.239.32.0	
74.125.71.100	10.8.0.1	6		216.239.32.1	
10.8.0.1	74.125.71.95	6		2.14.192.0	
…..	…..	…..		…..	

Subsequently, we combined the packets with the labels and exported them in *CSV* format (a universal dataset format). Furthermore, as TCP traffic is a stream-oriented protocol (i.e., TCP packets are part of instances of integrated communication between a client and a server, known as streams), we also grouped the individual packets into streams, following TCP. This process yielded two separate data sets: (a) the *packet dataset* and (b) the *stream dataset*. The grouping of packets into streams was based on a flag in TCP packets called *Stream Index*, which indicates the communication stream that each packet belongs to. Thus, an element in the stream dataset was formed by assembling all the packets, which had the same stream index. Streams were labelled as "infected" if they had at least one packet within them that had been labelled as "infected", and "normal" otherwise.

A preliminary analysis of the datasets generated by the two set ups indicated that all *domain name system* (DNS) packets, which used the *user datagram protocol* (UDP) had been labelled as normal. Hence, UDP traffic was excluded from further analysis and the training phase focused on TCP traffic only. This was plausible as botnets involve a series of communications between the botmaster and the mobile botnets that are based on TCP traffic [354]. Following the packets and stream labelling, the features used for training were: Packets/Stream Frame Duration, Packets/Stream Packet Size, and Arguments Number in HTTP Request URL (Table G-1). Overall, the traffic capture and labelling process produced two datasets for the 3rd phase of our experiments (i.e., the classifier analysis phase): (1) the TCP packets dataset, which included 13652 infected packets and 20715 normal packets; and (2) TCP stream dataset, which included 1043 infected streams and 563 normal streams.

Before carrying out the classifier analysis phase, we also performed a single factor Analysis of Variance (ANOVA) statistical analysis on the datasets. This was in order to obtain an initial (and crude view) of possible indicators of packets and streams that could have an effect on their classification. This analysis showed that the frame duration, the TCP packet size and the arguments number of HTTP requests were three potentially important differentiators to take into account in the traffic analysis training phase.

Table 4-4 shows the results of this analysis for these three features. According to it, frame duration was found to have different average values in infect and normal datasets (i.e., data from the mobile botnet and normal applications, respectively). However only in the case of the packet data set this difference was statistically significant. This is shown by the F-test: the F-value from the data set (172.9) was larger than the F-critical value (F-value: 172.9 > F-crit: 3.842). The average values of frame duration were different between infect and normal traffic in the case of the stream dataset, but the relevant difference was not statistically significant (F-value: 1.44 < F-crit: 3.842).

Table 4-4 - AVONA analysis results

Packet Dataset						
Frame Duration						
SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Normal	7997	77812.4	9.7302	2602.083		
Infect	11750	28858.6	2.45605	676.1407		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	251785	1	251785	172.9199	2.49361E-39	3.841929825
Within Groups	2.9E+07	19745	1456.08			
Total	2.9E+07	19746				
TCP size						
SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Normal	7997	1135272	141.962	126885.7		
Infect	11750	2518838	214.369	172342		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2.5E+07	1	2.5E+07	162.0654	5.59522E-37	3.841929825
Within Groups	3E+09	19745	153934			
Total	3.1E+09	19746				
Argument number in HTTP request						
SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Normal	7997	358	0.04477	1.258626		
Infect	13846	2757	0.19912	2.606575		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	120.771	1	120.771	57.15394	4.1887E-14	3.841884621
Within Groups	46152	21841	2.11309			
Total	46272.8	21842				
Stream Dataset						
Frame Duration						
SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
normal	644	6192.87	9.61626	2216.41		
infect	1043	79562.3	76.2822	1972424		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	1769550	1	1769550	1.449752	0.228736818	3.846983477
Within Groups	2.1E+09	1685	1220588			
Total	2.1E+09	1686				
TCP size						
SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
normal	644	6539185	10154	6.51E+08		
infect	1043	2870130	2751.8	47240458		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.2E+10	1	2.2E+10	78.52476	1.96272E-18	3.846983477
Within Groups	4.7E+11	1685	2.8E+08			
Total	4.9E+11	1686				
Argument number in HTTP request						
SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
normal	644	1566	2.43168	90.97666		
infect	1043	2761	2.64717	36.63931		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	18.4896	1	18.4896	0.322262	0.570327351	3.846983477
Within Groups	96676.2	1685	57.3746			
Total	96694.6	1686				

On the basis of the F-test, the TCP packet size was found to be a statistically significant differentiator for both the packet and the stream data sets.

The arguments number of in HTTP requests was found to have a different variance between infect and normal for stream dataset. For package dataset, the F-value was larger than F-crit (F-value: 57.1 > F-crit: 3.842) which show the statistically significant difference.

On the basis of this analysis, we considered these features to be suitable for machine learning analysis.

4.5.1.2 Validation of training

To validate the experimental training results, we used three validation schemes. These were based on *K-fold cross-validation,* and *10% split validation*.

K-fold cross validation is a common technique for estimating the performance of an ML classifier [355]. According to it, in a learning training involving *m* training examples, the examples are initially arranged in random orders, and then they are divided into k folds. A classifier is then trained with examples in all folds but folds *i* (*i* = 1 . . . k), and its outcomes are tested using the examples in fold *i*. Following this training-testing process, the classification error of a classifier is computed by:

Equation 4-1 K-fold classification error of a classifier

$$\mathrm{E} = \sum_{i=1..K} n_i/m \tag{4-1}$$

where n_i is the number of the wrongly classified examples in fold i and m is the number of training examples.

Based on this scheme, we used 90-10% 10-fold and 50-50% 2-fold cross-validation, which are two typical validation approaches in ML. Split validation is simpler as it divides the training dataset into two parts, one part containing data used only for training and

another part containing data used only for testing. In 90-10% split validation, 90% of the data are selected as training dataset and 10% as the test dataset.

The analysis of the performance of classifiers was also based on two different formulations of the training and test data sets. In the first formulation (*experiment I*), both the training and the test datasets could include data from the same malware family, although the two data sets were disjoint. Hence, in this experiment, classifiers could have been trained with instances of traffic from a malware family that they needed to detect. In the second formulation (*experiment II*), the training and test data sets were restricted to include only data from different malware families. Hence, in this experiment, the classifiers were tested on totally unknown malware families (i.e., malware families whose infected data traffic had not been considered at all in the training phase).

4.5.2 Experiment I

4.5.2.1 Purpose

As discussed in Section 4.5.1.1, all the UDP protocol traffic in the dataset was labelled normal and was filtered out in the subsequent analysis. Thus, the attributes that we used in the experiment were frame duration, TCP packet size and the number of arguments in the URL of HTTP requests. Also, classifications were performed separately for the stream and packet data sets using all basic classifier algorithms. Hence, we carried out 30 groups of basic algorithm experiments (5 classifier algorithms × 3 validation schemes × 2 data sets) and 18 groups of ML-BOX experiments (6 box algorithms × 3 validation schemes × 1 dataset).

4.5.2.2 Set up

According to the workflow of the experiments, two sets of the dataset are prepared for machine learning analyser: packet dataset and stream dataset. Then we input two sets of the dataset into the WEKA toolkit and configure different algorithms to perform the machine learning analysis.

4.5.2.3 Results

The results of the experiments for the atomic and aggregate classifiers from experiment 1 are shown in Table F-1 (In Appendix F). The table shows the recall, precision and FPR measures for stream and packet data separately and for different validation set ups (90-10% 10-fold validation, 50-50% 2-fold validation, and 10-90% split validation). The main overall observation from Table F-1 was that the results in the case of packet level traffic were not encouraging and that the results for stream traffic were considerably better. Base on the Table F-1, we generated a group of visualisation charts to demonstrate the result.

The Figure 4-5 and Figure 4-6 visualise the measures of Recall, FPR and Precision for Normal applications and Infect applications separately with stream dataset. For each measure, we put the result of different dataset schemes together for comparison. According to the charts of Normal Recall and Normal FPR, both of Recall and FPR of normal application are relatively low for SVM, MNN and ML-BOX(OR) algorithms. On the contrary, the Naïve Bayes and ML-BOX(AND) algorithms have high value for these two measures, especially for 2-fold cross-validation dataset scheme. The 10-fold cross-validation dataset scheme has the best Recall for all ML algorithms except ML-BOX (AND which is slightly lower than 2-fold cross-validation dataset scheme. However, FPR of 10-fold cross-validation dataset scheme is higher than others except for J48, KNN and

ML-BOX(AND). The performance of precision measure is better across the normal and infect application.

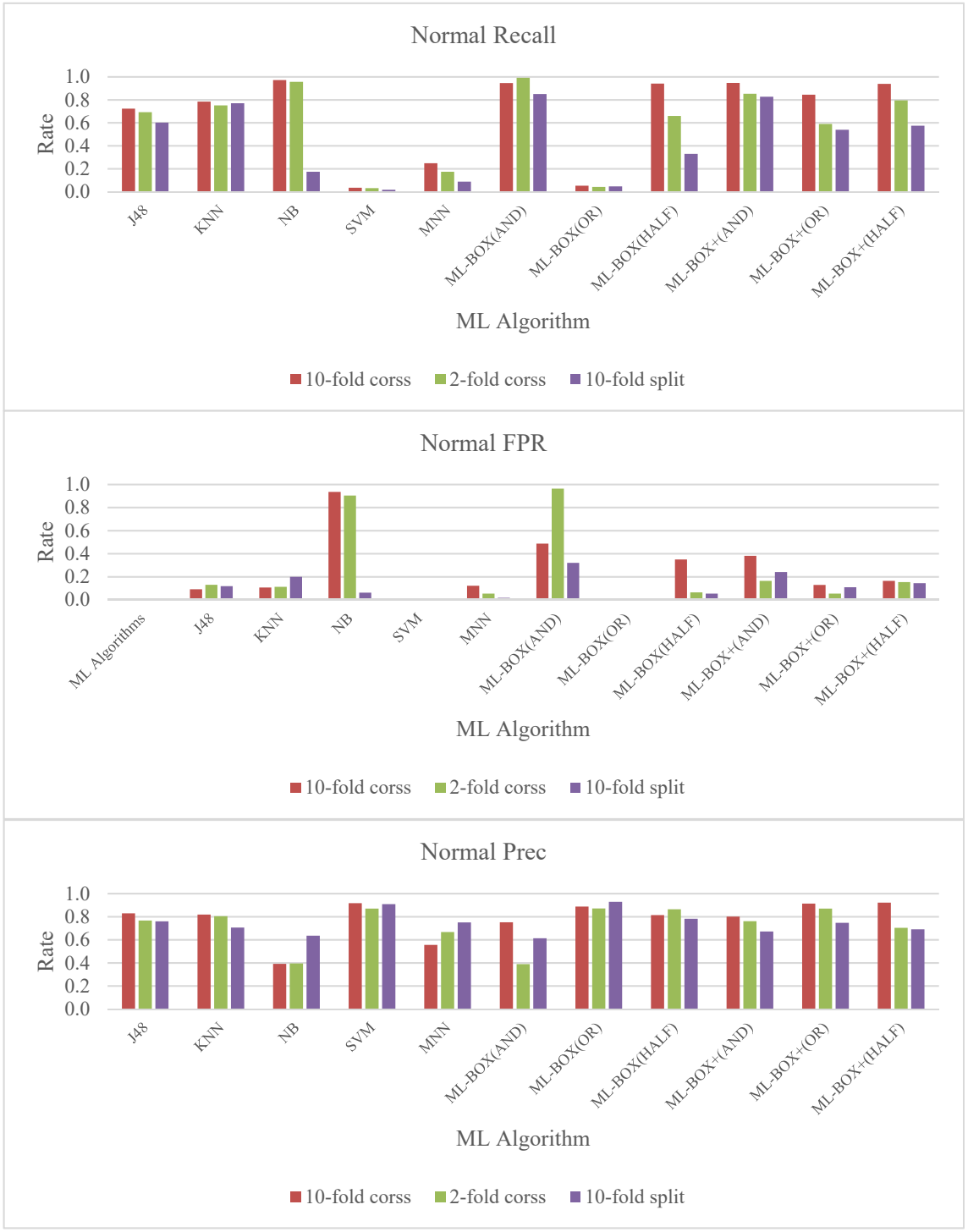


Figure 4-5 - Normal stream result visualisation

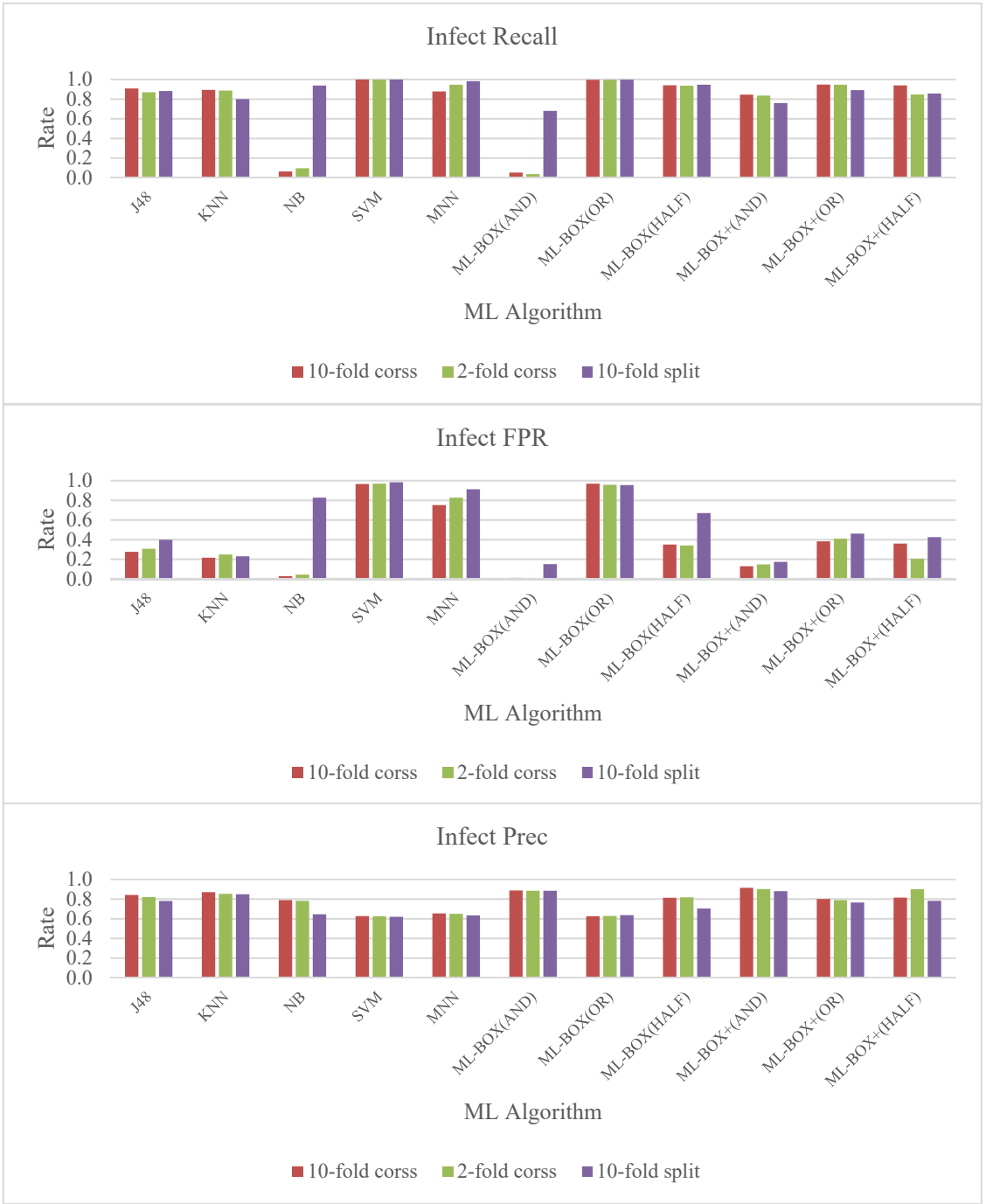


Figure 4-6 - Infect stream result visualisation

(Keys) cross: cross-validation, split: split-validation

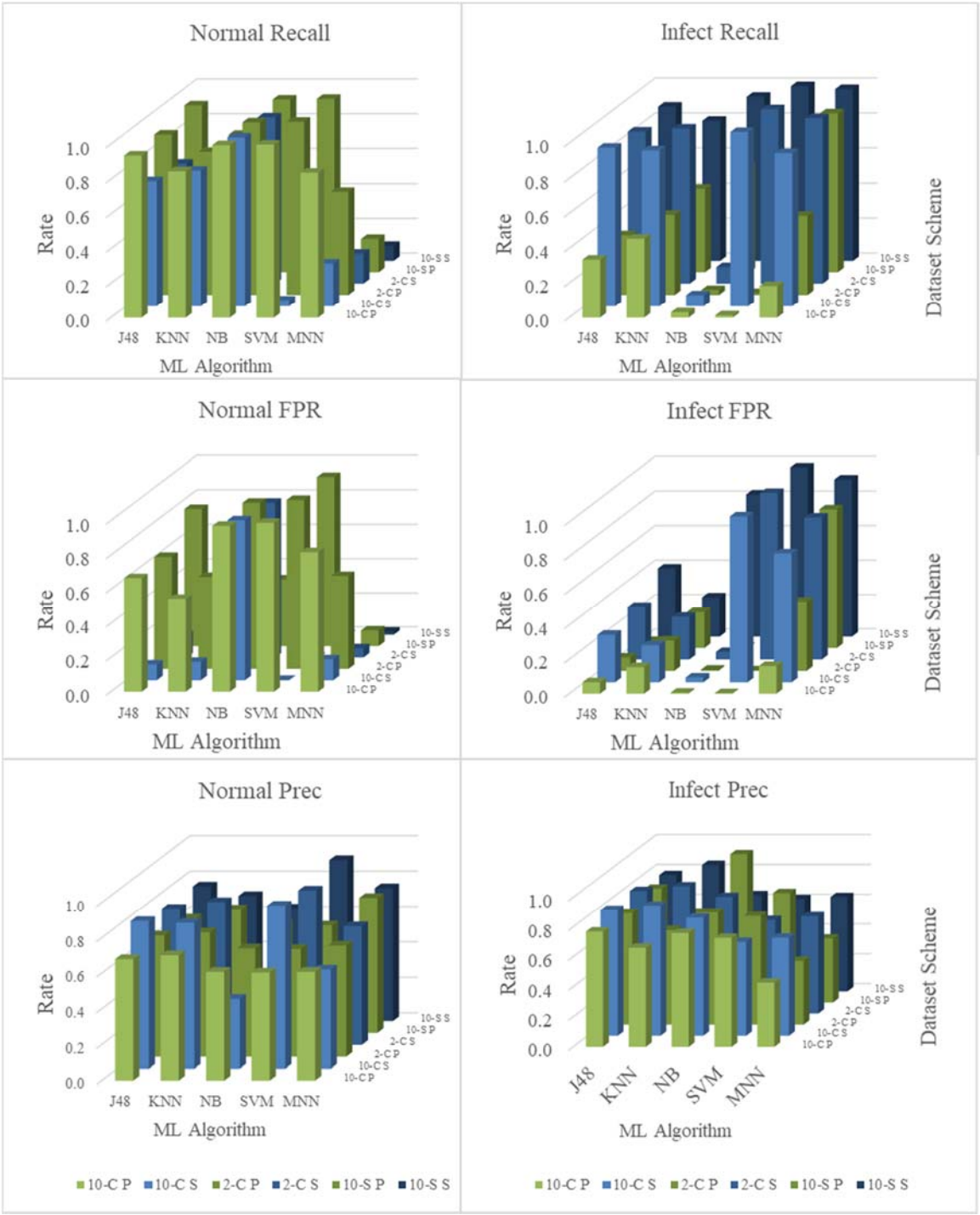


Figure 4-7 - Comparison between packet and stream dataset

Figure 4-7 illustrates the comparison between Packet dataset and Stream dataset performance of atomic algorithms. The x-axis is the algorithms, and the y-axis is the dataset schema with the flag of packet or stream (*10-C P* means 10-fold cross validation packet and *10-C S* means 10-fold cross validation stream). In order to observe the result of packet and stream obviously, we colour the packet result with green and stream result with blue. The figure demonstrates that the performance of packet result in normal applications is slightly better than a stream for J48, KNN and NB algorithms. However, in the infect application, the performance of packet is significantly lower than stream result. The similar observations can be found in normal and infect FPR. The precision of stream dataset is better than packet both in normal and infect applications nearly for all the algorithms. Therefore, the streams dataset should be chosen as a training dataset for learning.

4.5.2.4 Analysis

The result could be explained as follows. In TCP communication, the server and client should make a connection by a 3-way handshake, then send transfer data (payload packets) in fragments to stay below a maximum transmission unit (MTU). Also for each data transfer, the receiver sends an acknowledgement signal packet (ACK signal). Finally, the initiator sends a FIN signal packet to end the communication. In our experiments, data of normal and infected applications were labelled by the source and destination IP address of each traffic instance. In the case of the packet dataset, there was a large number of FIN and ACK packets labelled as “infected” due to the used IP addresses. The remaining features of these packets, however, were similar to FIN and ACK packets labelled as “normal”. Thus, the classifiers could not distinguish between them. In the case of the stream dataset, however, FIN and ACK packets were grouped into single streams, and hence their own characteristics did not feature prominently in the training and testing data sets. Hence, the classifiers were not misled by these signal packets in cases where they

had the same features as payload packets and, consequently, the performance of the stream dataset was better than that of the packet dataset.

Table 4-5 - Ranking of algorithms for infected stream traffic in experiment I

	Precision (ave)		FPR (ave)		Recall (ave)	
Classifier Split	**90-10**	**50-50**	**90-10**	**50-50**	**90-10**	**50-50**
Naïve Bayesian	.788 /3	.781/ 3	.028 /1	.043 /1	.064 /5	.096 /5
J48 Tree	.842 /2	.821/ 2	.276 /3	.307 /3	.908 /2	.870 /3
MNN	.654 /5	.650/ 5	.752 /4	.826 /4	.877 /4	.946 /2
KNN	.870 /1	.853 /1	.216 /2	.248 /2	.893 /3	.887 /4
SVM	.626 /4	.625/ 4	.966 /5	.969 /5	.998 /1	.997 /1
ML-BOX(AND)	.887 /2	.884 /3	.011 /1	.008/ 1	.053 /6	.036 /6
ML-BOX(OR)	.625 /6	.627 /6	.969 /6	.958 /6	.996 /1	.996 /1
ML-BOX(HALF)	.813 /4	.817 /4	.349 /3	.340 /4	.941 /3	.936/ 3
ML-BOX+(AND)	.914 /1	.902 /1	.129 /2	.148 /2	.845 /5	.835/ 5
ML-BOX+(OR)	.801 /5	.789 /5	.382 /5	.410 /5	.947/ 2	.945 /2
ML-BOX+ (HALF)	.814 /3	.900 /2	.359 /4	.205 /3	.939 /4	.847 /4

Focusing on stream traffic only, Table 4-5 shows the average recall, TPR and precision across for the two validation schemes with the best outcome (i.e., the 90-10 and 50-50 k-fold validation) in the case of infected traffic, and the relative ranking of each algorithm given the each of the evaluation measures. In the case of KNN, for example, the table shows ".870 /1" under precision for the 90-10 validation scheme. This means that the precision of KNN was .870 for the 90-10 scheme and that this algorithm was ranked 1st amongst the atomic algorithms. The results show a mixed picture. In particular, KNN and J48 were the best two atomic algorithms in terms of precision; KNN and J48 were the best two atomic algorithms in terms of recall, and Naïve Bayesian and KNN were the best two atomic algorithms in terms of FPR. The outcome was the same in the case of precision and FPR for the 50-50% scheme, but in this case, the ranking of atomic algorithms changed for recall (SVM still turned out as best but was followed by MNN).

The results of aggregated algorithms were, in general, better than those of atomic algorithms in this experiment. In particular, the ML-BOX(OR) and ML-BOX+(OR) algorithms produced the best recall for infected traffic (i.e., about 99% and 95%,

respectively) for both validation schemes. In terms of precision and FPR, the best two algorithms were ML-BOX+(AND) and ML-BOX(AND), albeit the different order of their ranking under each of these measures. ML-BOX(OR) and ML-BOX+(OR) yielded a higher recall than the individual algorithms because they classified as infected the union of the streams classified as such by any of these algorithms (i.e., a superset of all the sets of infected streams returned by the individual algorithms). ML-BOX(AND) and ML-BOX+(AND) yielded a higher precision than individual algorithms as they classified as infected the intersection of the streams that were classified as such by these algorithms (i.e., a subset of all the sets of infected streams returned by the individual algorithms).

Comparing the results of different validation schemes, the results in terms of precision and FPR in the case of 90-10% 10-fold validation were better than those of the 50-50% 2-fold validation for most algorithms, although no notable differences amongst these two schemes were observed for recall.

With the purpose to compare the performance of different algorithms intuitively, we produce the Figure 4-8 which demonstrate the infect application average measure across all ML algorithms. Thought the SVM, MNN and ML-BOX(OR) algorithms have a high recall; the corresponding FPR is also very high. Both of NB and ML-BOX(AND) have the low merit of recall and FPR. Considering other 6 ML algorithms (J48, KNN, ML-BOX(HALF), ML-BOX+(AND), ML-BOX+(OR), ML-BOX+(HALF)) that have relatively acceptable performance, the ML-BOX+(AND) represent the lowest FPR and highest Precision.

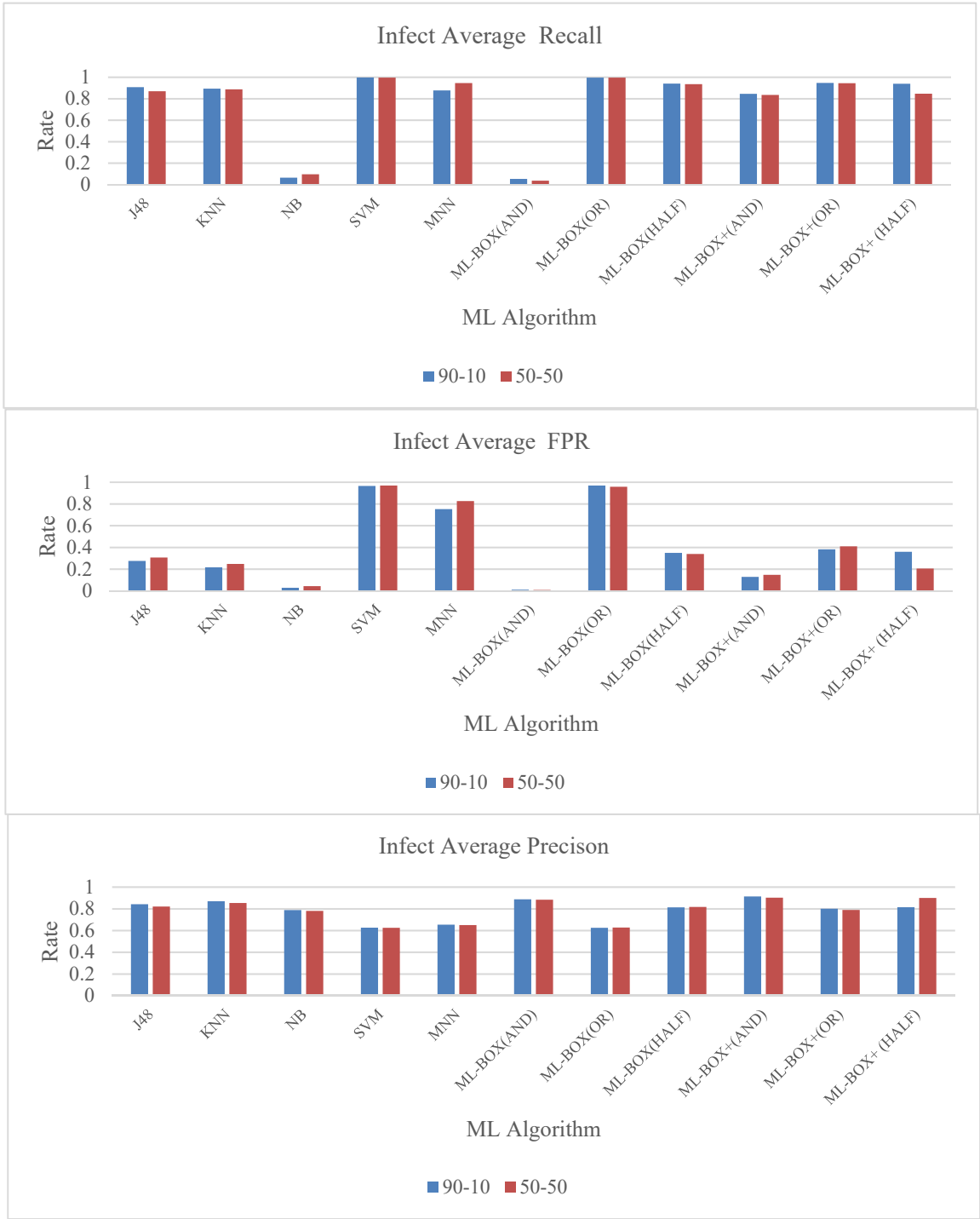


Figure 4-8 - Comparison of ML algorithms

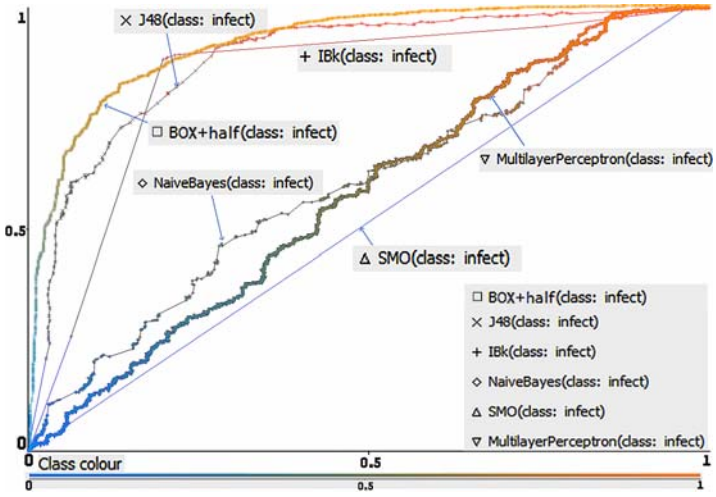


Figure 4-9 - ROC curve of infect stream based on six classifier algorithms

Another comparison of the performance of different classifier algorithms can be made using the receiver operating characteristic (ROC) curve. The ROC curve is generated by plotting the true positive rate (TPR) on the vertical axis (x-axis) against the false positive rate (FPR) on the horizontal axis (y-axis) at various threshold settings. Getting higher TPR and lower FPR is the objective of classification. So the closer the curve gets to the left upper corner, the better the performance is it. Figure 4-9 illustrates the ROC curve for six different classifiers for infected traffic in the stream dataset and the 10-fold cross validation. As shown in the figure, the performance of the J48 and KNN algorithms was better than the performance of the other three basic algorithms, and the aggregated algorithm ML- BOX+(HALF) yielded the best result of all algorithms.

Table 4-6 - The AUC of six classifiers based on stream dataset

Classifier	**AUC**
Naïve Bayesian	0.601
J48 Decision Tree	0.882
Multi-layer NN	0.573
K-Nearest Neighbours	0.836
SVM	0.516
ML-BOX+(HALF)	0.919

The measure of the area under the ROC curve, known as *area under the curve* (AUC), is also widely used for comparison of the performance of different classifications. Table 4-6 presents the AUC for the 6 ROC curves of Figure 4-9. Based on AUC, the J48 algorithm has the largest AUC measure across all basic algorithms competitors (0.882) and can, therefore, be assumed to be the best basic algorithm for the classification. However, the KNN algorithm is close to it with an AUC of 0.836. All the other basic algorithms have had low AUC measures. Overall the ML-BOX+(HALF) have had the best AUC (i.e., 0.919) across all algorithms.

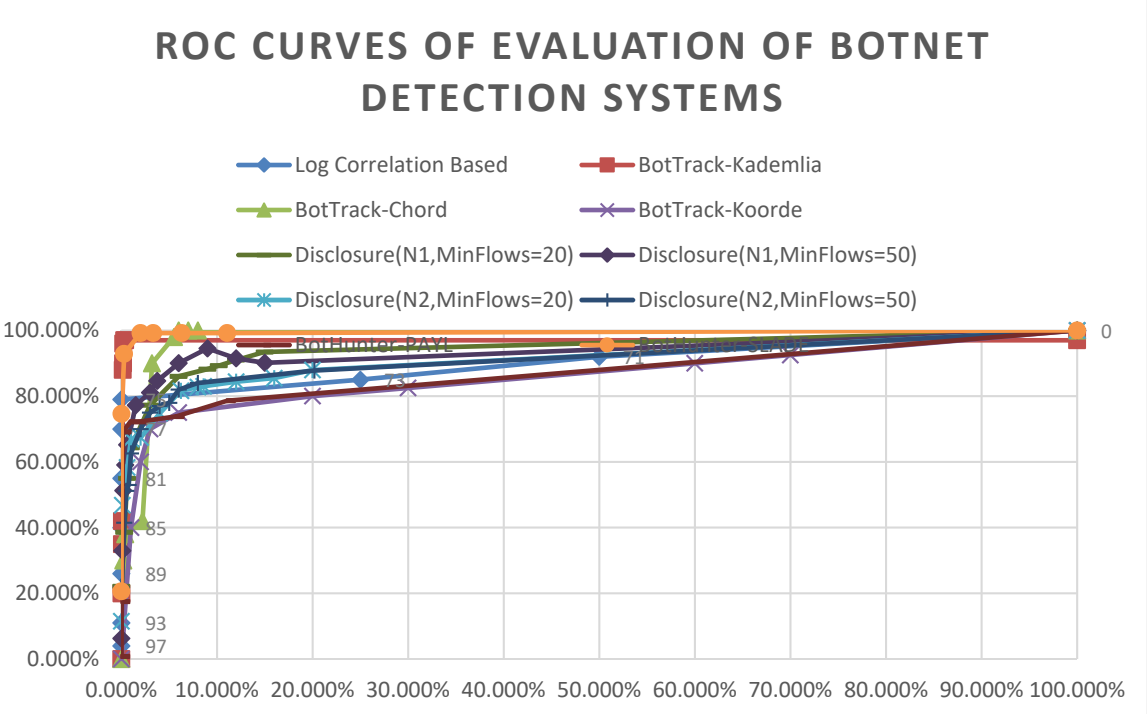


Figure 4-10 - ROC curves of evaluation of selected botnet detection systems

The shows the summary of the selected literatures in Section 2.2.4 which includes the evaluation based on the ROC curve. Although the data sets in the experiment are different, they have the similar trend for the ROC curve. Moreover, we can find the area under the ROC curves of BotHunter and BotTrack on Chord P2P network is larger than others, so

they have better performance. Meanwhile, the performance of J48, KNN and ML-BOX+(HALF) have similar performance with these exiting researches.

Table 4-7 - Outcome of analysis of variance for experiment I

(A) ATOMIC ML CLASSIFIERS

Measure	Source of Variation	SS	df	MS	F	P-value	F crit
TPR	Dataset scheme	0.093	2.000	0.046	0.903	0.443	4.459
	Algorithm	0.769	4.000	0.192	3.749	0.053	3.838
FPR	Dataset scheme	0.145	2.000	0.072	1.973	0.201	4.459
	Algorithm	1.408	4.000	0.352	9.579	0.004	3.838
Prec	Dataset scheme	0.007	2.000	0.004	3.480	0.082	4.459
	Algorithm	0.125	4.000	0.031	30.132	0.000	3.838

(B) BOX ML CLASSIFIERS

Measure	Source of Variation	SS	df	MS	F	P-value	F crit
TPR	Dataset scheme	0.026	2.000	0.013	0.509	0.616	4.103
	Algorithm	1.131	5.000	0.226	8.894	0.002	3.326
FPR	Dataset scheme	0.056	2.000	0.028	4.814	0.034	4.103
	Algorithm	1.508	5.000	0.302	52.046	0.000	3.326
Prec	Dataset scheme	0.007	2.000	0.003	3.063	0.092	4.103
	Algorithm	0.143	5.000	0.029	26.848	0.000	3.326

(C) KEY: *SS*: sum of squares; *df*: degrees of freedom, *MS*: mean square; *F*: F-value of experimental data; *P-val*: probability of samples of from same population despite difference in variance; *F crit*: minimum F value for accepting null hypothesis at α=0.05; *Dataset scheme*: the dataset scheme for the experiments; *Algorithm*: sample groups based on ML classifier algorithm.

To investigate whether the use of different train-test dataset scheme and different ML classifiers resulted in a statistically significant difference in the TPR, FPR and PRC measures for botnet applications, we carried out a two-way analysis of variance (ANOVA). The results of this analysis are summarised in Part (A) of Table 4-7 and demonstrate that the statistically significant differences were only the FPR and Prec differences across the different atomic ML algorithms ($F(1,5)=9.579$, $p=.0004$ for FPR and $F(1,7)= 30.132$, $p=.000$ for Prec). The two-way analysis of variance (ANOVA) for box ML algorithms as shown in Part (B) of Table 4-7: (a) only statistically significant differences were the FPR differences across the different dataset scheme ($F(1,4)=4.814$, $p=.0034$); (b) the algorithms has a significant effect on all measures.

4.5.2.5 Summary of main results of the first set of experiments

The main observations are drawn from an experiment I are:

1. The algorithms KNN and Naïve Bayesian have had the best performance in terms of precision and FPR amongst the atomic algorithms in the 90-10 and 50-50 scheme. However, recall performance of Naïve Bayesian was bad (0.064 and 0.096 for two cross-validation dataset scheme). In terms of recall, the best performers amongst single algorithms were SVM and MNN. However, both these algorithms had low precision and high FRP rates.
2. ML-BOX (AND) and ML-BOX+(AND) have had the best performance in terms of precision and FPR amongst the aggregate (box) algorithms in the 90-10 and 50-50 schemes. However, performance in terms of recall for ML-BOX+(AND) was so poor (0.053and 0.036 in the 90-10 and 50-50 schemes). In terms of recall, the best performers amongst box algorithms were ML-BOX(OR) and ML-BOX+(OR). However, only ML-BOX+(OR) appeared to have acceptable precision and FPR rate.
3. ML-BOX+(HALF) has the best AUC with 0.919, and the J48 and KNN have relatively acceptable AUC. The worst one is SVM which is only 0.516.

4.5.2.6 Threats to validity

The main threats to the validity of Experiment I are as follows:

1. Even though the number of packets is large, the number of the stream is less than 2000. Therefore the size of the dataset may not be enough to prove the feasibility of the system.
2. The slightly difference normal application operation during the experiment maybe infect the analyser result.

4.5.3 Experiment II

4.5.3.1 Purpose

The second experiment focused on assessing the capability of classifiers to detect totally unknown mobile botnet malware.

4.5.3.2 Set up

To reach the purpose, we partitioned the infected stream dataset into different subsets containing only data from the individual mobile botnet malware families. This produced nine sets of infected data coming from all families in Table 4-1 except from family eight which did not produce any infected data. The nine sets of infected data were mixed with a random selection of 10% of normal stream data to formulate an infected family data set. Subsequently, we used ~90-10% 10-fold validation by selecting data streams from 8 families and testing it on the remaining one family and ~50-50% 2-fold validation by selecting data streams from 5 families and testing it on the remaining four families.

4.5.3.3 Results

The results of experiment 2 in terms of recall, TPR and precision are shown in Table F-2 which shows the results for each of the individual malware families as produced in the 90-10 scheme. The corresponding bar charts are presented in Figure 4-11, Figure 4-12 and Figure 4-13. The different performance among ML algorithms is similar with the result of Experiment I which describe in Section 4.5.2. Regarding the performance across the different malware family, the recall of malware family 3 is obviously superior to others by using NB and ML-BOX(AND) algorithms in Figure 4-11. Meanwhile the Precision measure of malware family 3 is higher in these two algorithms than all other algorithms that can be found in Figure 4-13. As can be seen from the J48, ML-

BOX+(AND), ML-BOX+(OR) and ML-BOX+(HALF) algorithms in Figure 4-11, the recall of malware family 5 is significantly lower than other families over 50%. In terms of FPR, there is no remarkable difference between different malware family especially for NB SVM and ML-BOX(OR) algorithms.

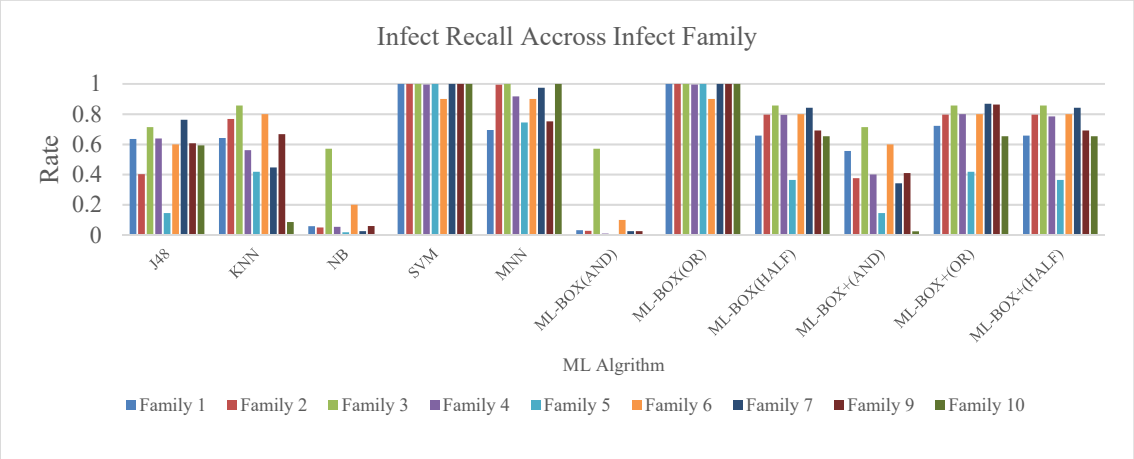


Figure 4-11 - Infect recall across infect malware family

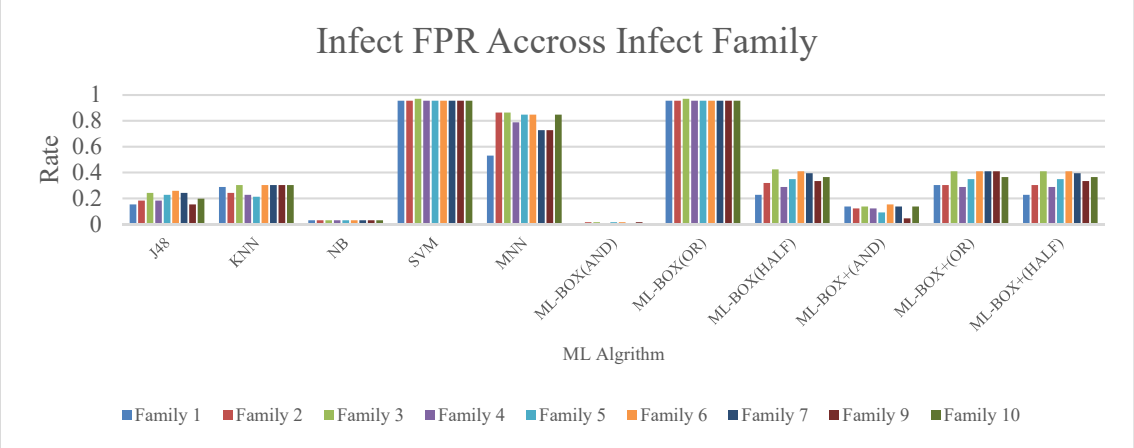


Figure 4-12 - Infect FPR across infect malware family

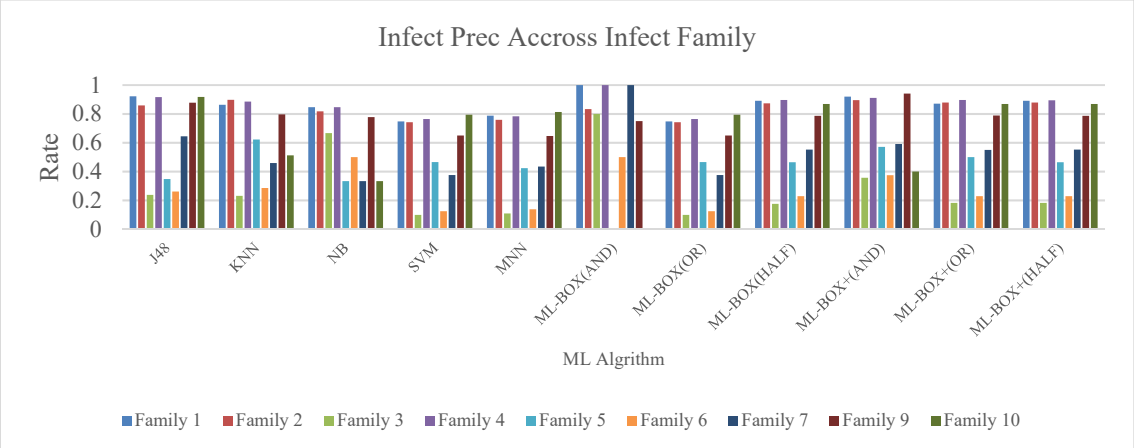


Figure 4-13 - Infect Prec across infect malware family

4.5.3.4 Analysis

Table 4-8 shows the average recall, TPR and precision across all families for the two splits and the relative ranking of each algorithm given the relevant measure (in the case of J48 for example the table shows ".665 /1" under precision for the 90-10 scheme meaning that the precision of J48 was .665 and that this algorithm was ranked 1st amongst the atomic algorithms).

Table 4-8 - Ranking of algorithms for infected stream traffic in experiment II

	Precision (ave)		FPR (ave)		Recall (ave)	
Classifier	**90-10**	**50-50**	**90-10**	**50-50**	**90-10**	**50-50**
Naïve Bayesian	.606 /3	.693 /2	.030 /1	.158 /1	.116 /5	.190 /5
J48 Tree	.665 /1	.756 /1	.204 /2	.222 /2	.567 /4	.530 /3
MNN	.544 /4	.544 /5	.783 /4	.680 /4	.886 /2	.727 /2
KNN	.617 /2	.656 /3	.276 /3	.336 /3	.583 /3	.511 /4
SVM	.529 /5	.555 /4	.957 /5	.900 /5	.988 /1	.927 /1
ML-BOX(AND)	.735 /1	.741 /1	.008 /1	.031 /1	.088 /6	.089 /6
ML-BOX(OR)	.529 /6	.582 /6	.957 /6	.935 /5	.988 /1	.976 /1
ML-BOX(HALF)	.637 /5	.674 /5	.345 /4	.398 /4	.718 /3	.613 /3
ML-BOX+(AND)	.662 /2	.735 /2	.119 /2	.148 /2	.396 /5	.388 /5
ML-BOX+(OR)	.640 /3	.691 /4	.360 /5	.410 /6	.753 /2	.704 /2
ML-BOX+ (HALF)	.638 /4	.694 /3	.342 /3	.344 /3	.716 /4	.602 /4

From the Figure 4-14, the column chart of Table 4-8, we can see clearly that both of recall and FPR are higher than others for the SVM, MNN and ML-BOX(OR) algorithms. In contrast, recall and FPR of NB and ML-BOX(AND) are lowest among all the algorithms. Comparing with Experiment I, the recall of other six algorithms is slightly inferior to the corresponding performance data in Figure 4-8. The reason for decrement is the unknown malware family data in the testing dataset.

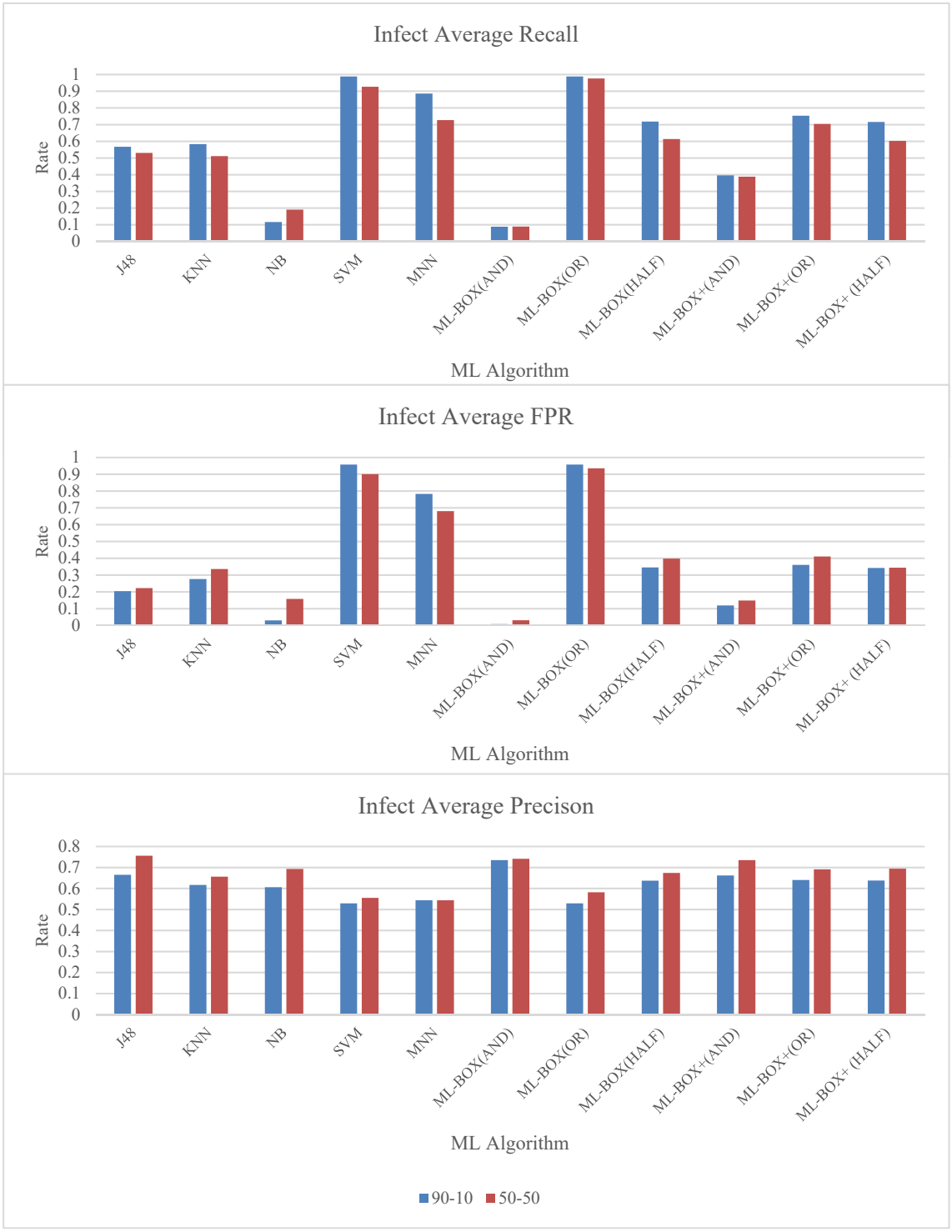


Figure 4-14 - Comparison of ML algorithms

To explore whether the observed differences were statistically significant, we carried a two-way analysis of variance (ANOVA) for atomic and box algorithm separately. Table 4-9 Part (A) shows that the all measures are statistically significant across different atomic ML algorithm ($F(1,3)=45.633$, $p=.000$ TPR, $F(1,5)=507.734$, $p=.000$ for FPR, $F(1,3)=45.633$, $p=.000$ for Prec). However, the none of them are statistically significant across the data scheme. The box algorithms results are shown in Table 4-9 Part (B) which demonstrates statistically significant differences in all measures (i.e., in TPR, FPR, PRC) across the different atomic and box ML classifiers, and across the different data scheme at $\alpha=0.05$.

Table 4-9 - Outcome of analysis of variance for experiment II

(A) ATOMIC ML CLASSIFIERS

Measure	Source of Variation	SS	df	MS	F	P-value	F crit
TPR	Dataset scheme	0.394	8.000	0.049	2.148	0.060	2.244
	Algorithm	4.183	4.000	1.046	45.633	0.000	2.668
FPR	Dataset scheme	0.030	8.000	0.004	1.356	0.253	2.244
	Algorithm	5.712	4.000	1.428	507.734	0.000	2.668
Prec	Dataset scheme	0.394	8.000	0.049	2.148	0.060	2.244
	Algorithm	4.183	4.000	1.046	45.633	0.000	2.668

(B) BOX ML CLASSIFIERS

Measure	Source of Variation	SS	df	MS	F	P-value	F crit
TPR	Dataset scheme	0.663	8.000	0.083	6.294	0.000	2.180
	Algorithm	4.540	5.000	0.908	69.011	0.000	2.449
FPR	Dataset scheme	0.044	8.000	0.005	4.510	0.001	2.180
	Algorithm	4.842	5.000	0.968	802.994	0.000	2.449
Prec	Dataset scheme	0.663	8.000	0.083	6.294	0.000	2.180
	Algorithm	4.540	5.000	0.908	69.011	0.000	2.449

(C) KEY: as in Table 4-7

4.5.3.5 Summary of key results of second experiment set

The main observations drawn from experiment II are:

1. The precision, recall and FPR of all classifiers (both the atomic and the aggregated ones) dropped w.r.t experiment 1, as it can be seen by contrasting the recall and precision figures for the 90-10 and 50-50 cross validation column for stream data in Table F-1 with the corresponding figures in Table F-2.

2. The drop was more significant in the case of a recall.
3. The cross validation with the 50-50 slit generated better outcomes than the 90-10 split in terms of precision, recall and FPR for all algorithms. This was probably due to overfitting, as in the 90-10 scheme we found recall to correlate positively with the training-to-test data set size (TTTS) ratio and precision to correlate with TTTS negatively: the correlation coefficients were 0.41 for TTTS/Recall, and – 0.90 for TTTS/Precision.
4. Results were poor for all families with a low number (<50) of infected streams (i.e., families 3, 5, 6, 7).
5. The algorithms J48 and Naïve Bayesian have had the best performance in terms of precision and FPR amongst the atomic algorithms in the 90-10 and 50-50 scheme. However, their performance in terms of recall was not so good (0.567 and 0.116, respectively). In terms of recall, the best performers amongst single algorithms were SVM and MNN. However, both these algorithms had low precision and high FRP rates.
6. ML-BOX (AND) and ML-BOX+(AND) have had the best performance in terms of precision and FPR amongst the aggregate (box) algorithms in the 90-10 and 50-50 schemes. However, their performance in terms of recall was not poor (0.088 and 0.396, respectively). In terms of recall, the best performers amongst box algorithms were ML-BOX(OR) and ML-BOX+(OR). However, only ML-BOX+(OR) appeared to have acceptable precision and FPR rate.
7. Recall and FPR were found to correlate positively with the size of the infected data set of a family and precision was found to correlate negatively with it.

4.5.3.6 Threats to validity

1. We divided the network traffic into different malware families based on the feature of IP addresses. There is a small set of network traffic that classified wrongly

possibly. This error can cause deviations for performance and result of machine learning classification.

2. The number of malware families used in the experiment is not enough to prove the system can be used for detection for unknown mobile botnet malware. The 10 botnet malware families we selected in the experiment could not predict the behaviour of other unknown mobile botnet malware behaviour.

4.5.4 Experiment III

4.5.4.1 Purpose

Although the performance of mobile devices has improved significantly in recent years, their computing and energy capabilities are still limited. Therefore, a system deployed on a mobile device should be designed to minimise the demand for such resources. Hence, in our experiments, we should also evaluate the execution time and battery consumption of MBotCS.

4.5.4.2 Set up

The mobile device used in this evaluation was a GT-I9228 with 1440 MHz CPU clock, 1 GB of RAM and battery of 2500 mAh. The specification of the mobile device used in this evaluation is shown in Table 4-10. MBotCS, tPacketCapture, and Gsam Battery Monitor had been installed on it. Then we made a random selection of 10 botnet applications and ten normal applications of those indicated in Section 4.5.1.1 and ran the evaluation experiment for 12 hours. The set up of the experiment involved the following sequence of steps:

(1) Charged the battery of the mobile device fully and installed all the applications.

(2) Launched the Gsam Battery Monitor, tPacketCapture and MBotCS applications.

(3) Launched the normal and infected applications mentioned above and run 5 minutes for each application to simulate user behaviours, and the remaining experiment time keep the mobile device on standby.

(4) Gathered and analysed results.

(5) A comparison experiment was performed for a time period of the same length on the following day. The set up was identical to the initial experiment (i.e., we went through steps (1) - (4) except that we did not deploy MBotCS.

Table 4-10 - The specs of GT-I9228

Parameters	**Value**
CPU Clock	1400 MHz
RAM	1 GB
Battery	2500 mAh

4.5.4.3 Results

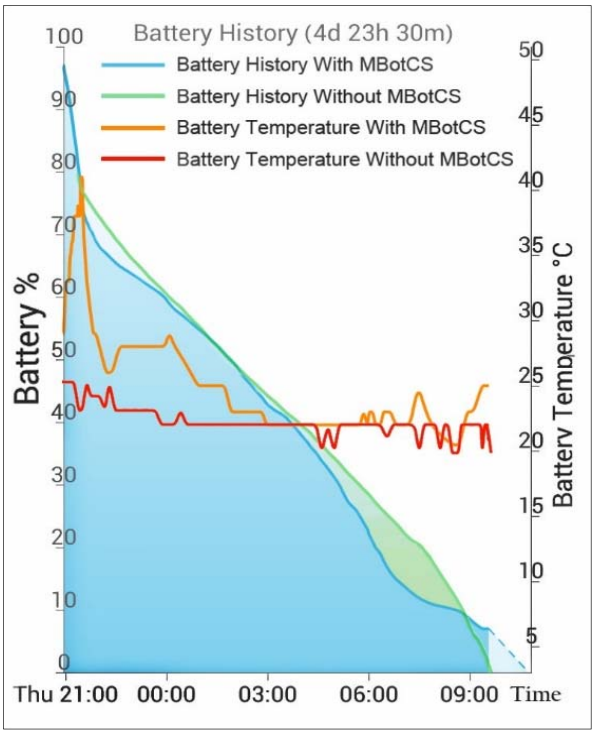


Figure 4-15 - Battery consumption

Results - Battery Consumption: The graphs of battery consumption in percentage terms and battery temperature during the experiment is shown in Figure 4-15. According to the figure, the battery consumption was not affected significantly by the use of MBotCS. In particular, the use of MBotCS consumed 0.5% of the total battery usage of the device during the period of its deployment. Of this, 0.2% was the battery usage caused by tPacketCapture.

Execution Time: When activated, MBotCS checks the *pcap* file every 3 seconds, and if new traffic is captured, it scans and analyses it. In these scans, the scan sequence number (sq) is recorded. Using the J48, KNN and ML-BOX+(HALF) classifiers in the ML-Analyser, we recorded the number of streams (N^{sq}) in the new traffic and the total execution time (T^{sq}) for analysing the new traffic. Figure 4-16 shows the average

execution times for the three classifiers, computed by the formula $T_{ave}^{sq} = \sum_{i=0}^{sq} T^{sq} / \sum_{i=0}^{sq} N^{sq}$ (the average for sequence number 100, for instance, is the average of execution time of a classifier over all stream instances from 1 to 100). The figure also shows the fitted curves for the average execution times of these algorithms.

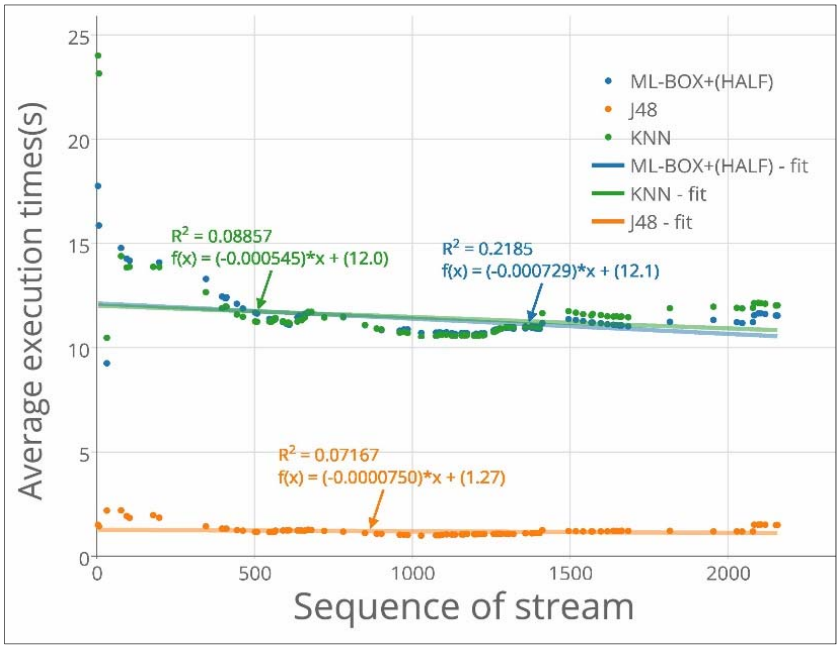


Figure 4-16 - The ML-analyser execution time

4.5.4.4 Analysis

These Figure 4-15 show that MBotCS has had a very low energy effect on the battery consumption of the device.

The results show that the average execution time of J48 across all executions was 1.216 seconds with a standard deviation of 0.228 and the average of KNN across all executions was 11.562 seconds with a standard deviation of 1.779. The average of ML-BOX+ (HALF) across all executions was 11.387 seconds with a standard deviation of 1.087. A t-test check showed the statistical significance of the observed differences between the

average execution times of J48 and KNN at $\alpha = 0.05$ ($p-value = 2.701E{-}91 << 0.05$), confirming that J48 have had better performance than KNN.

Also, the average execution time of different classifiers remained almost constant with respect to the processed number of streams performance, as shown by the curves fitted on execution times in Figure 4-16. This indicates the capability of MBotCS to produce a reasonably fast detection/response once the ML-Analyser has been trained.

4.5.4.5 Threats to validity

There are other factors that cannot be controlled for affecting the battery consumption.

1. The signal of WIFI on the mobile device will affect the battery consumption. A weak signal will lead to high battery consumption.
2. The temperature of the environment will affect the battery consumption. The high temperature will lead to high battery consumption.

4.6 Experiments for system call analysis

4.6.1 Overview

Apart from analysing the network traffic to detect botnets on Android device, we also performed an experimental study on the use of ML algorithms for the detection of mobile botnets, based on the analysis of system (i.e., Android OS) calls. In particular, the main contributions of our experimental study with respect to previous work are that it has investigated:

(a) The use of not only atomic but also box ML classifiers using supervised learning.

(b) The performance of ML classifiers a wider set of detection scenarios than existing work including KBKN scenario, UBKN scenario and UNUB scenario which introduced in Section 4.4.

(c) A comprehensive set of Android mobile botnets, which had not been considered previously, without relying on any form of synthetic training data.

(d) The statistical significance of differences in detection performance measures with respect to ML algorithms, system call aggregation periods, normal and botnet applications, and different types of botnet families.

The following paragraph will introduce our approach and the methodological setup of the experiments and gives an analysis of the results obtained from them.

4.6.2 Methodological setup of the experiments

In the following, we describe the methodological set up for the experimental analysis of system logs. Figure 4-17 shows an overview of this set up, which involved three main steps: (1) the capture of mobile device system calls to txt file which contains full system call sequence in-formation for further analysis (system call capture); (2) the selection of feature and generation of the dataset for training and testing the system call analyser (dataset generation); and (3) the experimental use of different ML classifier algorithms as the basis for training the system call analyser (classifier analysis).

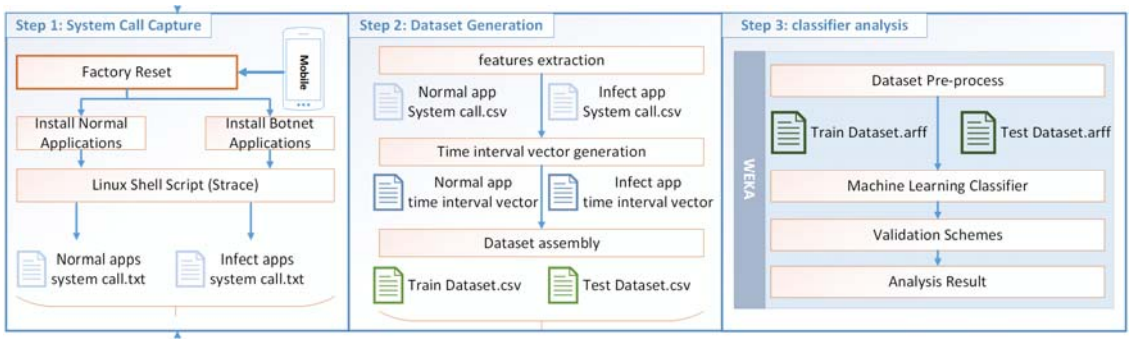


Figure 4-17 - Experimental set up

4.6.2.1 System call capture

The mobile device used in the experiments was the OnePlus One generation (A0001) running Android v. 5.2 (*Lollipop*). To avoid interference with other applications, the mobile device used for the experiment was reset to the default Android OS settings before the start of the experiments.

To capture system calls, we deployed 12 normal applications and 10 mobile botnet applications. The normal applications that we selected due to their popularity were: *Chrome, Gmail, Maps, Facebook, YouTube, Messenger, Twitter, PlayNewsstand, Flipboard, Feedly, Skype, and Mail-Droid*. Also, to be certain about their genuineness, all of them were downloaded from Google Play. The botnet applications were selected from the MalGenome project [339]. MalGenome has collected more than 1200 Android malware applications. The vast majority of these applications (i.e., more than 90%) are botnets. From the whole set of MalGenome applications, we selected botnet applications from 10 different families, shown in Table 4-1. These families were selected to ensure coverage of different types of attacks, namely device control, hidden SMS dispatching, stealing and forwarding device information to remote servers (e.g., Botmasters), UI control and execution of commands, and hidden unauthorised downloads, contact of premium services.

As the kernel of Android OS is Linux, there are more than 250 Linux system calls that could be made by an Android application. To capture system calls, we used *strace*, i.e., a debugging and monitoring Linux utility. *strace* was used on the Android device (i.e., A0001) through the *Android Debug Bridge* (ADB), i.e., a command line tool providing a Unix shell that can be used to run commands on Android connected devices or Android device emulators. Although *strace* can get all the system call for a process with a specific id PID, due to the limited resources of mobile devices, when applications are paused or put into the background, Android stops the corresponding process. This process is forked again with a new PID when the application is resumed. To address this issue, we

implemented a Linux Shell Script, called *System Call Monitor* (*SCM*) see Appendix B. Within it; we used the process name, which doesn't change in the Android operating system to get the corresponding PID dynamically. SCM monitored the output of *ps* utility command continually, and checked whether the PID changed for same process name. If the PID had been changed, SCM used strace to capture the system calls of a new process for the corresponding application.

4.6.2.2 Datasets Generation

In the experiments, we used two data sets: (a) primitive system call data captured during the trial operation and (b) derived data aggregating system call profiles of different applications into different time periods. These data sets were generated as described below.

Primitive data set: In the first stage, we installed the 12 normal applications and ten malware applications and deployed the SCM on the device. Then we launched these applications and the capture script over a 24-hour trial period. To simulate the activities of the mobile device, a table of frequently used actions of the normal applications, listed in Appendix E, was generated and executed. These actions were executed every one hour over the trial period. During this period, we collected 12 distinct data sets for the normal applications (one data set per application) and ten distinct data sets for the mobile botnet applications (one data set per botnet).

Table 4-11 - Structure of primitive system call data set

Timestamp	Call Name	Call Return	Time Spent	Label
1.43E+12	clock_gettime	0	0	B
1.44E+12	epoll_pwait	1	0.01	B
1.46E+12	recvfrom	104	0	B

From these raw datasets, we extracted four features that we considered potentially useful as indicators of variability between normal and botnet applications (see Table G-2). These were: (1) the timestamp of the system call, (2) the system call name, (3) return

value of the call, and (4) the time spent on the call (i.e., the call's total execution time). System call instances were also labelled as "normal" and "infect" based on the application they were generated from. This process generated a data set of system call records of the form shown in Table 4-11. Altogether, we recorded 2,666,619 calls of 61 different system operations.

To establish the potential utility of the descriptive features (1) - (4) in the different records we conducted a preliminary statistical analysis. The purpose of this analysis was to establish whether botnet and normal applications had any statistically significant variability with regards to the features. The outcomes of this analysis are discussed in detail in next Section. In summary, however, this analysis indicated that the only statistically significant difference between botnets and normal applications was related to the frequency of calls of different system functions. Due to this, we decided to aggregate the data in the primitive datasets into aggregate call profiles and use them, instead of the primitive data, to train the classifier algorithms. The dataset derived from this process is described next.

Table 4-12 - Structure of derived dataset

App Package Name	Recv from	Futex	epoll_ pwait	clock_ gettime	write	Get uid32	…	label
com_android_chrome	.0879	.0887	.0980	.4460	.0756	.0766	…	N
flipboard_app	.0690	.0169	0.1119	0.4462	.0644	.0945	…	N
com_km_installer	.0121	.0032	.0720	.3349	.0263	.0528	…	B
greenrobt	.0954	.0031	.0827	.3226	.0277	.0612	…	B

Derived data set: To derive aggregated system call profiles for different normal and botnet applications, we sliced the total 24-hour period over time intervals (τ) of 10, 30, 60, 300, 600 seconds and for each of these intervals, we produced a vector for each of the 22 applications showing the relative frequency of calls to different system functions (i.e., the number of calls made to the specific system function divided by the total number of calls made to any system function by the relevant application within the period). This

process resulted in the generation of relative system call frequency tuples of the form shown in Table 4-12. In total for the intervals of 10, 30, 60, 300 and 600 seconds we obtained 19008 (B: 8640), 6336 (B: 2880), 3168 (B: 1440), 630 (B: 288), 315 (B: 144) system call vectors, respectively (B call vectors number in parenthesis).

4.6.2.3 Analysed ML Algorithms

In the second set of experiments, we used the same atomic ML algorithms and ML-box algorithms that we used in the first set (see Section 4.1). The use of ML box algorithms based on the results of the best two atomic classifiers in each experiment according to the AUC measure (see below). In the following, we will refer to the outcomes of these box classifiers as "ML-BOX+ (.)" where "ML-BOX (.)" is the underpinning basic box algorithm. In the case of ML-BOX+(HALF), if the best two algorithms classified an instance of dataset in the same class, ML-BOX+(HALF) generated the same common classification but if the best two algorithms were in disagreement, ML-BOX+(HALF) generated a classification based on the outcome of the 3 remaining classifiers by taking a vote over them.

4.6.3 Basic statistical analysis

The initial analysis that we performed on the primitive system call log data was statistical and was carried out with the aim to identify whether: (a) the frequency of calls to different system functions, (b) the size of the return value of calls, or (c) the time spent on the call system calls varied significantly, in a statistical sense, across normal and botnet applications. The purpose of this analysis was to identify descriptors with a potentially high classification effect for the detailed classifier-training phase.

This analysis showed that only (a) varied in a statistically significant manner across botnet and normal applications. More specifically, for each of the sixty-one (61), different system calls that were recorded, we measured the relative frequency of the call for normal

(FN) and botnet applications (FB). The statistical significance of the observed differences between FN and FB ratios was tested using the *z-score test* [356]. The test showed that in 42 out of 61 different system calls the FN and FB ratios varied in a statistically significant manner at p=.01. Table 4-13 shows the 10 system calls with the highest z-score. This analysis also indicated that botnet applications made significantly fewer calls to all system operations but two, i.e., *epoll_pwait* and *recvfrom*. The FB of *recvfrom* was 37.32 percentage points higher than its FN, and the FB of *epoll_pwait* was 14.79 percentage points higher than its FN.

Table 4-13 - Statistical significance of system call frequency differences

Call Name	**# Calls by Botnets**	**B-RF**	**# Calls by N apps**	**N-RF**	**Z-score**
recvfrom	462576	.445	117073	.072	72.54
futex	2797	.003	192404	.118	353.24
epoll_pwait	236368	.227	129500	.080	342.27
clock_gettime	268049	.258	680662	.418	266.67
write	3149	.003	81215	.050	213.27
getuid32	9750	.009	96814	.059	203.72
gettid	0	.000	44263	.027	169.53
ioctl	24655	.024	111877	.069	162.66
mprotect	1151	.001	29329	.018	126.71
read	4838	.005	35666	.022	112.38

The functions implemented by these two calls are relevant to network connectivity, and hence they can be the reason for the observed differences in their relative call frequencies. More specifically, *recvfrom* is used to receive data (messages) from a socket (whether or not it is connection-oriented), and *epoll_pwait* waits for events of monitoring multiple file descriptors to see if I/O operations occurred on them. Thus, both *recvfrom* and *epoll_pwait* relate to an application's connection to networks outside the mobile device, an activity that is necessary for botnet applications, as such applications need to communicate with their botmaster regularly in order to retrieve the commands to execute

on the local device and report information obtained from this device back to the botmaster.

4.6.4 First experiment: KBKN scenario

In the first experiment (Exp 1), we used training sets including both normal and botnet application call vectors (of the form shown in Table 4-12) to train the different atomic classifiers. We also *90–10 percent cross-validation scheme* of WEKA to evaluate the dataset. More specifically, we executed a total of 10 training evaluation dataset pairs. In each of these pairs, 90% of the full set of call vectors summarised in Table 4-12, consisting of both normal (N) and botnet (B) application vectors, was selected as the training set and the remaining 10% was used as the evaluation set. The training and test sets were selected randomly, but each pair of them used in the experiment was, by virtue of its selection, guaranteed to include both normal and botnet application vectors, albeit in different proportions. Due to this set up, it was possible for classifiers to have been trained with call vectors of a botnet application before being asked to detect whether a previously unseen vector of the same application belongs to it. Thus, this experiment realised a KBKN scenario. The experiment was repeated using call vectors aggregated over five different time periods, i.e., 10, 30, 60, 300 and 600 seconds. Hence, in total the set included 500 executions (100 per each aggregation period).

The results of Exp 1 for different atomic ML classifiers are shown in part (a) of Table F-3. The table shows the TPR, FPR, PRC and AUC measures for normal and botnet applications separately grouped also by the aggregation period of the underlying data set (i.e., the 10, 30, 60, 300 and 600 second periods). The measures in the table were generated as an average measure computed across all the ten 90–10% splits of the call vector set generated by WEKA. The results in the table have also been coloured according to the VR criteria introduced in Section 4.1, using green colour to indicate the cases “very good” and red colour to indicate the cases of “weak” performance.

As shown in Table F-3, the classifier, which had the best average performance in terms of AUC for botnet applications across all the call vector aggregation periods, was the multi-layer perceptron (NN) (AUC=.995). SVM, however, have had the best performance in terms of average TPR (.975), FPR (.013) and PRC (.966) in botnets. Overall, SVM and NN demonstrated "very good" performance in terms of all metrics for botnets based on the VR criteria. KNN and J48 also showed "very good" performance in terms of FPR and PRC in all cases but missed the top performance range in terms of TPR (J48 in the 30s and 300s datasets, and KNN in the case of the 30s dataset). Note, however, that in none of these cases the TPR of J48 and KNN fell below .8 (i.e., in the "weak" performance range). The NB classifier has had the weakest performance of all the five classifiers with regards to FPR and PRC (.037 and .914, respectively), without however its performance being "weak" according to the VR criteria.

The bar charts are corresponding to the Table F-3 is shown in Figure 4-18 and Figure 4-19 that group the different time interval dataset into ML algorithm series. There are 8 charts that divided by 2 types of application (normal and botnet) and 4 types of performance measurements (TPR, FPR, Prec and AUC). Overall, the performance is good with relatively high TPR, Prec, AUC and low FPR in both normal and botnet measures. Regarding the comparison between the different time interval dataset, in Figure 4-18 we can find that the normal TPR of 300s time interval dataset in the BOX-AND+ algorithm has the lowest performance with less than 0.85. The FPR of 300s and 600s time interval datasets are higher than others in J48 and NB algorithms respectively with values that are greater than 0.15. The difference of Prec and AUC is unapparent among various datasets across ML algorithms and all with values more than 0.9. In respect of Botnet measure in Figure 4-19, the 300s and 600s time interval dataset have relatively poor TPR (less than 0.8) in J48 and NB algorithm respectively. The 300s time interval dataset also has high FPR (more than 0.15) in the BOX-AND+ algorithm. The FPR of 10s time interval dataset is relatively higher than others in NB and BOX-OR algorithms. Meanwhile, in NB and BOX-OR algorithms, the Prec of 10s time interval dataset is only around 0.8 that lower

than others. Similar to the normal result, the AUC of the botnets still has the balanced performance for all type datasets across various ML algorithms.

The Figure 4-20 reveals the average performance of various type dataset between ML algorithms. As can be seen from the first two charts, all the average TPRs are more than 0.9, and average FPRs are less than 0.1 for both of normal and botnet. Especially, the FPR in normal application of BOX-OR is less than 0.005 and the FPR of all algorithms except NB, BOX-OR and BOX-AND+ are around 0.01 in botnet application. The performance of average Prec and AUC across the ML algorithms are similar to the TPR that higher than 90 percent.

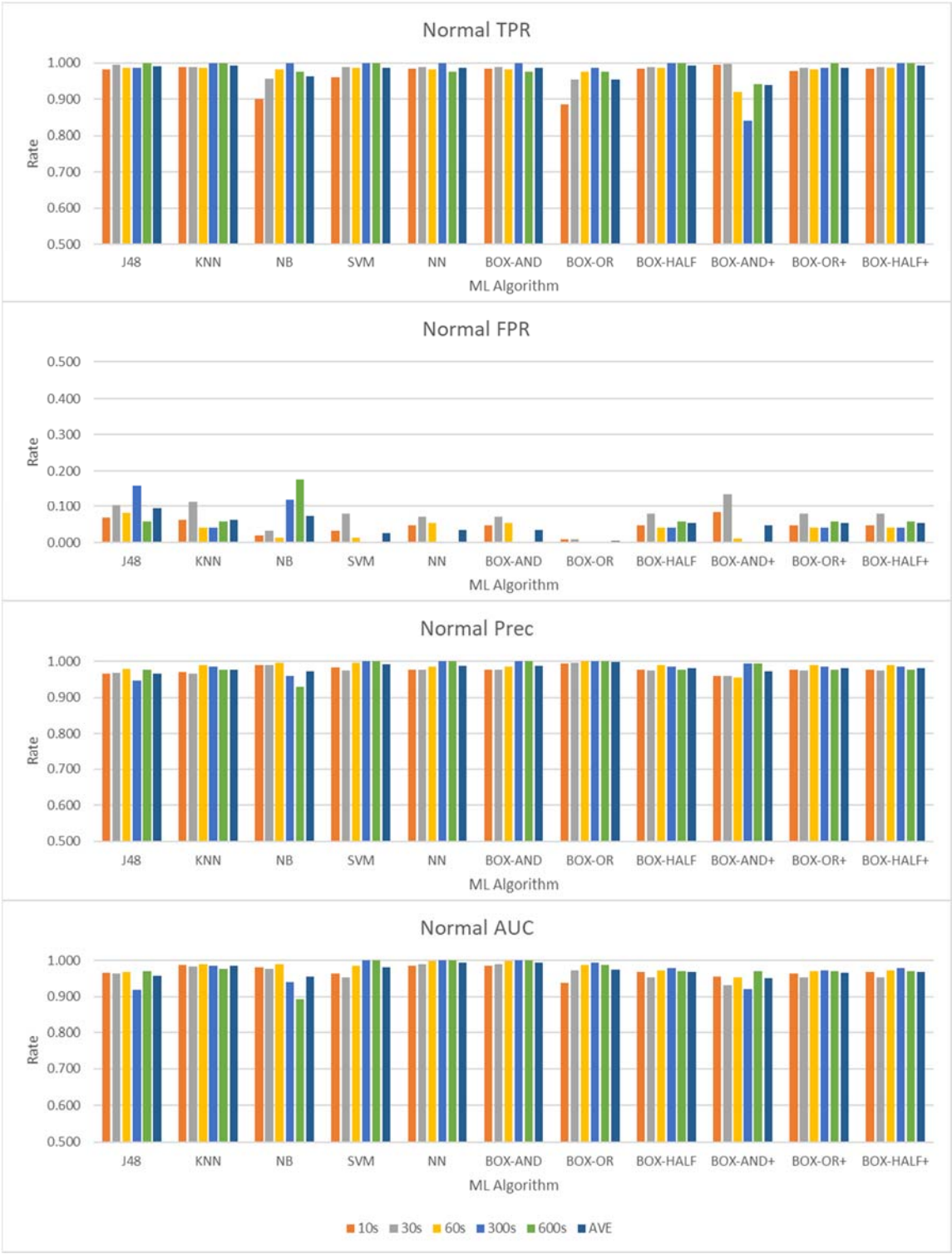


Figure 4-18 - Performance of normal across different time interval dataset

Figure 4-19 - Performance of botnet across different time interval dataset

Figure 4-20 - Average performance of time interval dataset

Table 4-14 - Outcomes of analysis of variance for experiment 1

(A) ATOMIC ML CLASSIFIERS

	VarSr	*SS*	*df*	*MS*	*F*	*P-val*	*F crit*
TPR	A.Per	0.004	4	0.001	0.555	0.697	3.006
	Alg	0.016	4	0.004	1.844	0.169	3.006
FPR	A.Per	0.003	4	0.000	3.070	0.047	3.006
	Alg	0.002	4	0.000	2.575	0.077	3.006
PRC	A.Per	0.012	4	0.003	2.998	0.050	3.006
	Alg	0.012	4	0.003	2.954	0.052	3.006
AUC	A.Per	0.000	4	0.000	0.329	0.854	3.006
	Alg	0.005	4	0.001	2.892	0.056	3.006

(B) BOX ML CLASSIFIERS

	VarSr	*SS*	*df*	*MS*	*F*	*P-val*	*F crit*
TPR	A.Per	0.011	4	0.002	5.134	0.005	2.866
	Alg	0.009	5	0.001	3.453	0.020	2.710
FPR	A.Per	0.001	4	0.000	0.288	0.881	2.866
	Alg	0.012	5	0.002	2.186	0.096	2.710
PRC	A.Per	0.008	4	0.002	2.282	0.096	2.866
	Alg	0.021	5	0.004	4.851	0.004	2.710
AUC	A.Per	0.001	4	0.000	2.813	0.053	2.866
	Alg	0.006	5	0.001	7.689	0.000	2.710

(C) KEY: *VarSr*: source of variance; *SS*: sum of squares; *df*: degrees of freedom, *MS*: mean square; *F*: F-value of experimental data; *P-val*: probability of samples of from same population despite difference in variance; *F crit*: minimum F value for accepting null hypothesis at α=0.05; *A.Per*: sample groups based on call vectors aggregation period; *Alg*: sample groups based on ML classifier algorithm.

To explore whether the use of different aggregation periods and different ML classifiers resulted in a statistically significant difference in the TPR, FPR and PRC measures for botnet applications, we carried out a two-way analysis of variance (ANOVA). The results of this analysis are summarised in Part (A) of Table 4-14 and demonstrate that the only statistically significant differences were the FPR differences across the different aggregation periods ($F(1,4)=3.0702$, $p=.047$). It should be noted, however, that even this difference was statistically significant at $\alpha=.05$ but not for lower α levels (e.g., for $\alpha=.025$).

In Exp 1, we also evaluated the effect of using box ML algorithms. The performance of these algorithms with respect to the used evaluation metrics is summarised in Part (B)

of Table F-3. As shown in it, BOX-AND has had the best performance for botnet applications with regards to the average AUC for botnet applications across all data aggregation periods (.995). It was, however, outperformed by BOX-OR with respect to the average TPR for botnet applications across all data aggregation periods (.997 vs. .965); by BOX-HALF and BOX-HALF+ in terms of average FPR for botnet applications across all data aggregation periods; and by BOX-HALF, BOX-AND+ and BOX-HALF+ in terms of average PRC for botnet applications across all data aggregation periods. However, none of these differences were statistically significant as the two-way ANOVA indicated. In particular, as shown in Part (B) of Table 4-14: (a) the aggregation period and the box algorithm had a significant effect on TPR ($F(1,4)=5.1340$, $p=.0052$ for aggregation period; $F(1,5)=3.4536$, $p=.0207$ for algorithms); (b) the algorithms had a significant effect on PRC ($F(1,5)=4.8513$, $p=.0046$); and (c) the algorithms had a significant effect on AUC ($F(1,5)=7.6892$, $p=.0004$).

Overall, the use of box ML classifiers did not improve the average AUC measure with respect to the best atomic ML classifier (i.e., NN). Considering TPR, BOX-OR performed marginally better than the best atomic ML classifier (i.e., SVM) with respect to TPR (TPR of .997 vs. .975) but at the expense of a lower accuracy, i.e. a higher FPR (.045 vs. .013) and a lower PRC (.899 vs. .981).

In summary, the first set of experiments indicated that:

- It is possible to detect botnet applications by collecting aggregate system call vectors of the mobile botnet and normal applications and analysing them through atomic and/or box ML classifiers that have been previously trained on such applications.
- The use of box ML classifiers led to better results than atomic classifiers (the use of the BOX-AND classifier led to recall and precision as high as 96% and an FPR of about 1.5%). However, atomic ML classifier also demonstrated

strong performance (recall and precision rates in excess of 90%, FPR of less than 3%).

- The aggregating period did not cause any significant statistical differences in the detection TPR and PRC for botnet applications in either atomic or box ML classifiers. It did, however, affected FPR in the case of atomic classifiers: the 600s aggregation period gave the lowest average FPR (.009) and the 10s-aggregation period gave the highest average FPR (.037) in this case.

4.6.5 Second experiment: UBKN scenario

In the second set of experiments (Exp 2) our objective was to investigate the capability of ML classifiers to detect new botnet applications, i.e., applications whose system call profiles have not been used to train classifiers prior to detection in the presence of known normal applications (UBKN scenario). These experiments were based on the call profiles that we collected through the process discussed in Section 4.6.2.2, except that the classifier training and test sets were formed differently than in the first experiment set. More specifically, each pair of test/training datasets was formed as follows:

Test set: A test set included the full set of system call vectors of one botnet application plus a subset of the system call vectors of normal applications, including 10% of them. The selected subset of normal application call vectors was one of the 10 subsets of equal size of all the N call vectors that were formed through the random partition.

Training set: The training set paired with a test set included the full set of system call vectors of the remaining botnet applications plus the remaining nine subsets of the N system call vectors (i.e., 90% of the N call vectors).

For each botnet application, we formed 10 different test/training set pairs as described above and run 10 different experiments. Hence, we used 90 different test/training data set

pairs in total, i.e., 10 test/training data sets for each of the 9 botnet applications B1–B9 (*magicshop* was excluded from this experiment as it produced only 13 calls during the entire data collection period). This design enabled us to test the ML classifiers for each of the botnet applications separately.

Table F-4 summarises the results of this experiment, showing the average TPR, FPR, PRC and AUC measures computed for different ML classifiers and botnet applications (Part (A) shows the results for atomic ML classifiers and Part (B) shows the results for box ML classifiers). Each of the measures in the table is an average measure computed from 10 different test/training sets formed for each of the different botnet applications (B1 – B9). Also, the performance measures are also shown separately for normal and botnet datasets for each of the different botnet applications (B1–B9) and coloured as in Exp 1. Based on the AUC for botnet applications, the best atomic ML classifier was J48 with average AUC for botnet applications of .936.

The Figure 4-21, Figure 4-22 and Figure 4-23 are visualisation for the Table F-4. The performance of NN BOX-AND and BOX-OR algorithms exhibits the obvious difference between botnet families, especially for TPR and FPR. The botnet family 1, 4 and 7 has higher TPR than other families in normal applications in NN BOX-AND and BOX-OR algorithms. Meanwhile, they have lower FPR in botnet applications. In term of the comprehensive measure AUC, the botnet family 4 has a very low value that less than 0.6 in SVM algorithm.

J48 had an average TPR of .848, a very low FPR of .046 and PRC of.833. J48 was outperformed by Naïve Bayes (NB) in terms of TPR (NB: .908), but the latter algorithm performed worse than it in terms of FPR (.123 vs. .046) and PRC (.733 vs. .833). Hence, although J48 was able to detect a lower percentage of botnet activity, its results were more accurate. In terms of accuracy (FPR and PRC) for botnet applications, the best algorithm

was KNN with an average FPR of .032 and an average PRC of .836. However, KNN showed a lower TPR rate for botnet applications (.694).

According to the VR criteria introduced in Section 4.1, J48 had a very good TPR in four botnets (B4, B5, B8 and B9) and a weak one in three botnets (B2, B6 and B7). Its FPR was very good in 6 out of the 9 botnets and weak in no botnets. Finally, its precision was very good in two botnets (B4 and B5) and average PRC of .836. However, KNN showed a considerably lower recall (TPR) rate for botnet applications (.694).

NB had a very good TPR in five botnets (B2–B5 and B8) and a weak one in two botnets (B1 and B9). Its FPR, however, was weak in all but one botnets (B7). Similarly, its precision was weak in 6 botnets (B1, B3, B5 and B7–B9) and very good in two (B2 and B4). KNN (i.e., the third best performing algorithm in terms of AUC) had a very good TPR in 3 botnets (B4, B5 and B9) and a weak one in 5 botnets (B1, B2 and B6–B8); very good FPR in all botnets and very good precision in two botnets (B4 and B6); and weak precision in three (B2, B5 and B8). The worst of all the algorithms were NN as it had a weak TPR in 6 botnets (B1 and B3–B7), a weak FPR in all botnets except B7 and weak precision in 6 botnets (B1, B3, B5, B6, B8 and B9).

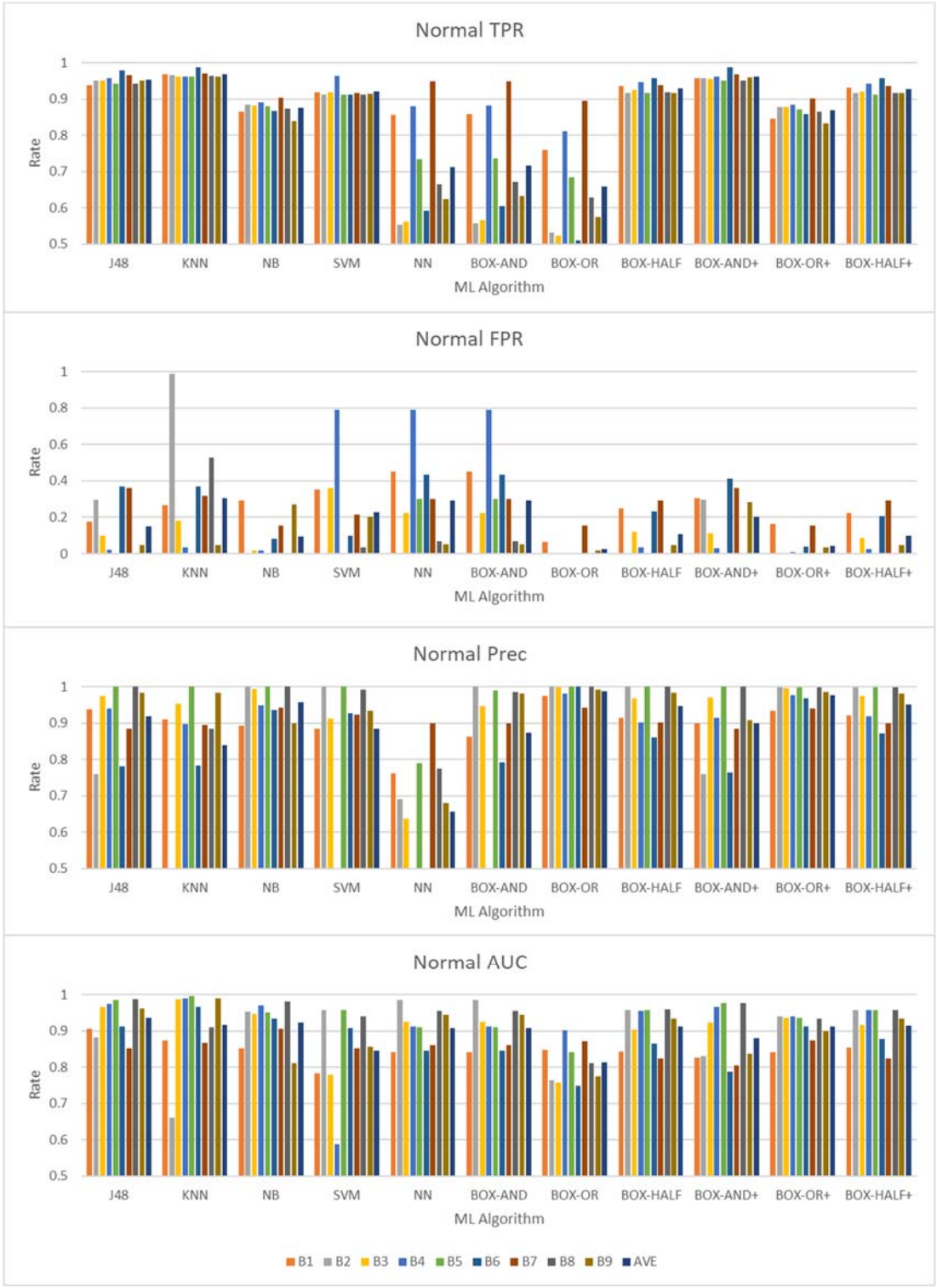


Figure 4-21 - Performance of normal across different malware family dataset

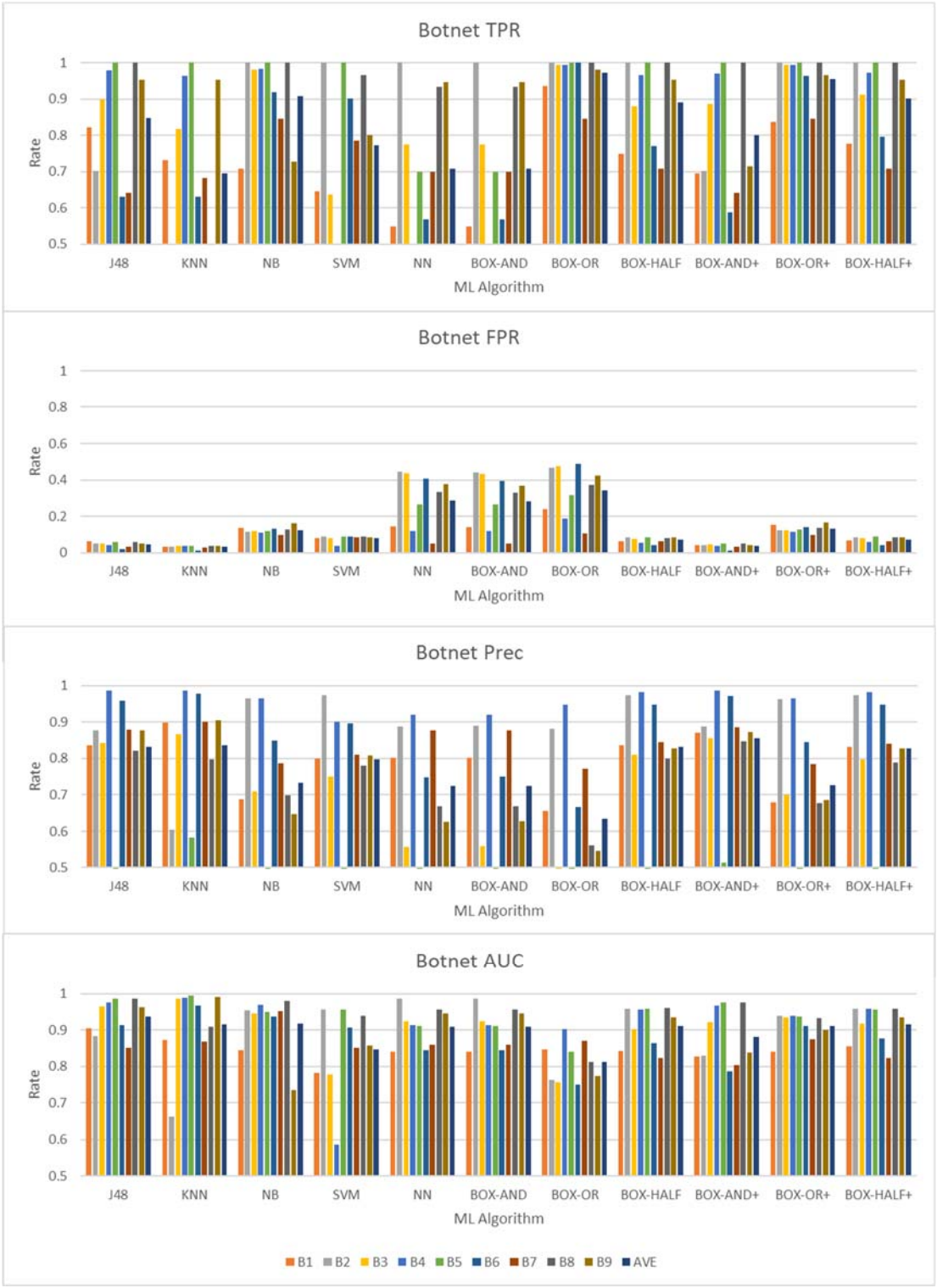


Figure 4-22 - Performance of botnet across different malware family dataset

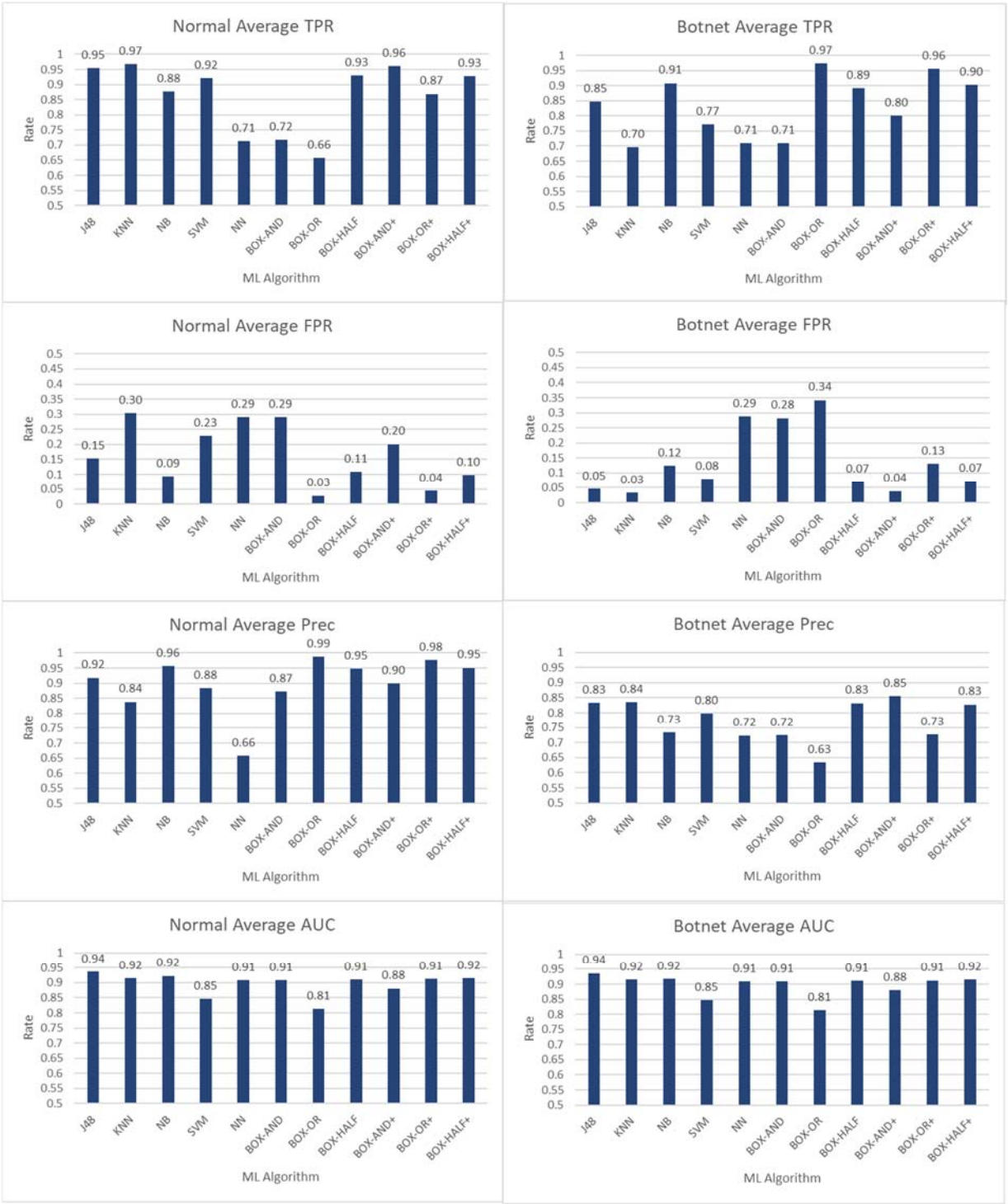


Figure 4-23 - Average performance of malware family dataset

Table 4-15 - Outcomes of analysis of variance for experiment 2

(A) ATOMIC ML CLASSIFIERS

	VarSr	*SS*	*df*	*MS*	*F*	*P-val*	*F crit*
TPR	Bot	0.442	8	0.055	1.483	0.161	1.961
	Alg	3.832	4	0.958	25.710	0.000	2.394
	Int	1.403	32	0.043	1.176	0.237	1.472
FRP	Bot	0.442	8	0.055	1.483	0.161	1.961
	Alg	3.832	4	0.958	25.710	0.000	2.394
	Int	1.403	32	0.043	1.176	0.237	1.472
PRC	Bot	8.767	8	1.096	33.353	0.000	1.961
	Alg	1.037	4	0.259	7.893	0.000	2.394
	Int	2.516	32	0.078	2.393	0.000	1.472
AUC	Bot	0.516	8	0.064	12.718	0.000	1.961
	Alg	0.425	4	0.106	20.970	0.000	2.394
	Int	2.453	32	0.076	15.105	0.000	1.472

(B) BOX ML CLASSIFIERS

	VarSr	*SS*	*df*	*MS*	*F*	*P-val*	*F crit*
TPR	Bot	4.093	8	0.511	33.242	0.000	1.957
	Alg	4.504	5	0.900	58.528	0.000	2.232
	Int	5.671	40	0.141	9.211	0.000	1.419
FRP	Bot	1.230	8	0.153	2.899	0.003	1.957
	Alg	7.142	5	1.428	26.926	0.000	2.232
	Int	2.096	40	0.052	0.9881	0.494	1.419
PRC	Bot	14.036	8	1.754	51.539	0.000	1.957
	Alg	3.310	5	0.662	19.449	0.000	2.232
	Int	1.551	40	0.038	1.139	0.262	1.419
AUC	Bot	0.791	8	0.099	10.851	0.000	1.957
	Alg	0.717	5	0.143	15.727	0.000	2.232
	Int	0.700	40	0.017	1.920	0.000	1.419

(C) KEY: *VarSr*: source of variance; *SS*: sum of squares; *df*: degrees of freedom, *MS*: mean square; *F*: F-value of experimental data; *P-val*: probability of samples of from same population despite difference in variance; *F crit*: minimum F value for accepting null hypothesis at α=0.05; *Bot*: sample groups based on botnet application; *Alg*: sample groups based on ML classifier algorithm; *Int*: interaction between *Alg* and *Bot* groups.

As indicated by the performance measures of Table F-4 none of the algorithms demonstrated fully consistent performance across all different botnet applications. To investigate, further, whether the observed differences were statistically significant, we carried a two-way analysis of variance (ANOVA). More specifically, this analysis explored whether the differences in the average TPR, FPR, PRC and AUC measures computed for the different algorithms and for the different botnet applications (B1-B9) were statistically significant. Table 4-15 (Part (A)) summarises the outcomes of this

analysis and highlighting (in green) the observed differences that were statistically significant. More specifically, as shown in the table, the differences observed in the average PRC were statistically significant both across the different botnet applications ($F(1,8)=33.353$, $p=.000$) and across the different ML algorithms ($F(1,4)=7.8935$, $p=.000$). Also, the differences observed in the average AUC were statistically significant both across the different botnet applications ($F(1,8)=12.7187$, $p=.0000$) and across the different ML algorithms ($F(1,4)=2.9705$, $p=.0000$). The interaction between the ML algorithm and the botnet application also led to statistically significant differences in the case of PRC ($F(1,32)=2.3933$, $p=.0001$) and AUC ($F(1,32)=15.1059$, $p=.0000$). For TPR and FPR, whilst the differences observed across the different algorithms for both measures were statistically significant ($F(1,4)=25.71$ and $p=.0000$ in both cases), their differences observed across different botnet families were not ($F(1,8)=1.4835$ and $p=.1611$ in both cases). The latter finding is of particular importance as it indicates that the performance of our approach in terms of TPR and FPR is not affected by specific botnet applications in a statistically significant manner.

In Exp 2, we also evaluated the effect of using box ML algorithms. The performance of these algorithms with respect to the used evaluation metrics is summarised in Part (B) of Table F-4. As shown in the table, BOX-HALF+ had the best performance for botnet applications with regards to AUC (.915). With respect to TPR, however, BOX-HALF+ was outperformed by BOX-OR and BOX-OR+ (.902 vs. .972 and .955, respectively). It was also outperformed by BOX-AND+ in terms of FPR (.084 vs. .04) and precision (.828 vs. .872).

The use of box ML algorithms did not improve AUC over atomic algorithms: the AUC of BOX-HALF+ was less than the AUC of the J48 algorithm (.915 vs. .936). In terms of TPR, box algorithms showed improved performance over the atomic ones since BOX-OR and BOX-OR+ have had better average TPR than NB, and very good performance in

7 out of the 9 botnet applications according to the VC criterion. However, this came at the expense of substantially reduced FPR (BOX-OR's FPR was .342, and BOX-OR+'s FPR was .181). As in the case of atomic algorithms, we also checked whether the differences in the TPR, FPR and PRC measure across the different box algorithms and botnet applications were statistically significant, using two-way ANOVA. The outcomes of this analysis are shown in Part (B) of Table 4-15. As shown in the table, the differences observed in the average TPR were statistically significant both across the different botnet applications ($F(1,8)=33.2426$, $p=.000$), across the different ML algorithms ($F(1,5)=58.5256$, $p=.000$), and when considering the interaction of these two factors ($F(1,40)=9.2114$, $p=.000$).

The differences observed in the average FPR were also statistically significant across the different botnet applications ($F(1,8)=2.8996$, $p=.0036$) and across the different ML algorithms ($F(1,5)=26.9284$, $p=.000$), but not when considering the interaction of these two factors ($F(1,40)=.9881$, $p=.4943$). Similarly, the differences observed in the average PRC were statistically significant across the different botnet applications ($F(1,8)=51.539$, $p=.0000$) and across the different ML algorithms ($F(1,5)=19.4495$, $p=.000$), but not when considering the interaction of these two factors ($F(1,40)=1.1395$, $p=.2622$). Finally, the differences observed in the average AUC were statistically significant across the different botnet applications ($F(1,8)=1.851$, $p=.0000$), across the different ML algorithms ($F(1,4)=15.7279$, $p=.0000$), and when considering the interaction between the ML algorithm and the botnet application ($F(1,40)=1.9201$, $p=.0008$).

In conclusion, the second set of experiments indicated that,

- Overall, the average TPR, FPR, PRC and AUC for botnet applications of all the atomic and box algorithms in this experiment dropped with respect to the corresponding measures for the same algorithm in Exp 1. This was expected since, in Exp 1, the classifiers had been trained in the botnet applications that

were tested on in the test phase, whereas in Exp 2 the classifiers were not trained in the botnet applications that they were tested in the test phase.

- Despite the performance drop, however, we believe that the performance shown by both individual and box ML algorithms in Exp 2 indicates the merit of our approach in detecting mobile botnets based on device system calls.

4.6.6 Third experiment: UBUN scenario

In the third experiment (Exp 3), our objective was to test classifiers using totally unknown system call profiles of not only botnet applications but also normal applications (UBUN scenario). To test this, we formed the classifier training and test datasets as follows.

- *Test set:* A test set included the full set of system call vectors of one botnet application plus the full set of system call vectors of one normal application.
- *Training set*: The training set paired with a test set was formed by including the full set of system call vectors of the remaining eight botnet applications plus the full set of the system call vectors of the remaining nine normal applications.

According to this scheme, we generated 108 (9 × 12) pairs of test and training sets.

The results of this experiment for the atomic and box ML algorithms are shown in Part (A) and Part (B) of Table F-5, respectively. These results are grouped by the unknown botnet application used to form the test set and include the average results computed across all the 12 test data sets in which call vectors of the particular botnet were combined with each of the 12 normal applications.

The corresponding graphs for Table F-5 shown in Figure 4-24, Figure 4-25 and Figure 4-26. Figure 4-24 and Figure 4-25 present the performance that is grouped by the botnet families for normal and botnet application respectively. Moreover, the Figure 4-26 illustrates the average performance of different ML algorithms.

Based on the AUC for botnet applications, the best atomic ML classifier in this experiment was J48 with an average AUC for botnet applications of 0.89. J48 had an average TPR of 0.75 and a low average FPR of 0.08. Its average PRC for botnets was 0.78. As in Exp 2, in terms of TPR, J48 was outperformed by Naïve Bayes (NB), which had a TPR of 0.92. NB, however, performed substantially worse than J48 in terms of average FPR (0.36 vs. 0.08) and precision (0.61 vs. 0.78) for botnet applications. Given the VR criteria set in Section 4.1, none of the atomic classifiers showed very good performance across all the three performance measures of TPR, FPR and PRC, for either botnet or normal applications.

The use of box ML algorithms in Exp 3 improved the outcomes of atomic algorithms. In particular, BOX-HALF and BOX-HALF+ achieved higher average TPR and PRC rates for botnet applications than the best atomic classifier (i.e., 0.82 in both cases) and the same average best FPR rate as atomic classifiers (i.e., 0.08). The performance of BOX-HALF and BOX-HALF+ was also better than the performance of all atomic classifiers in terms of TPR and PRC for normal applications.

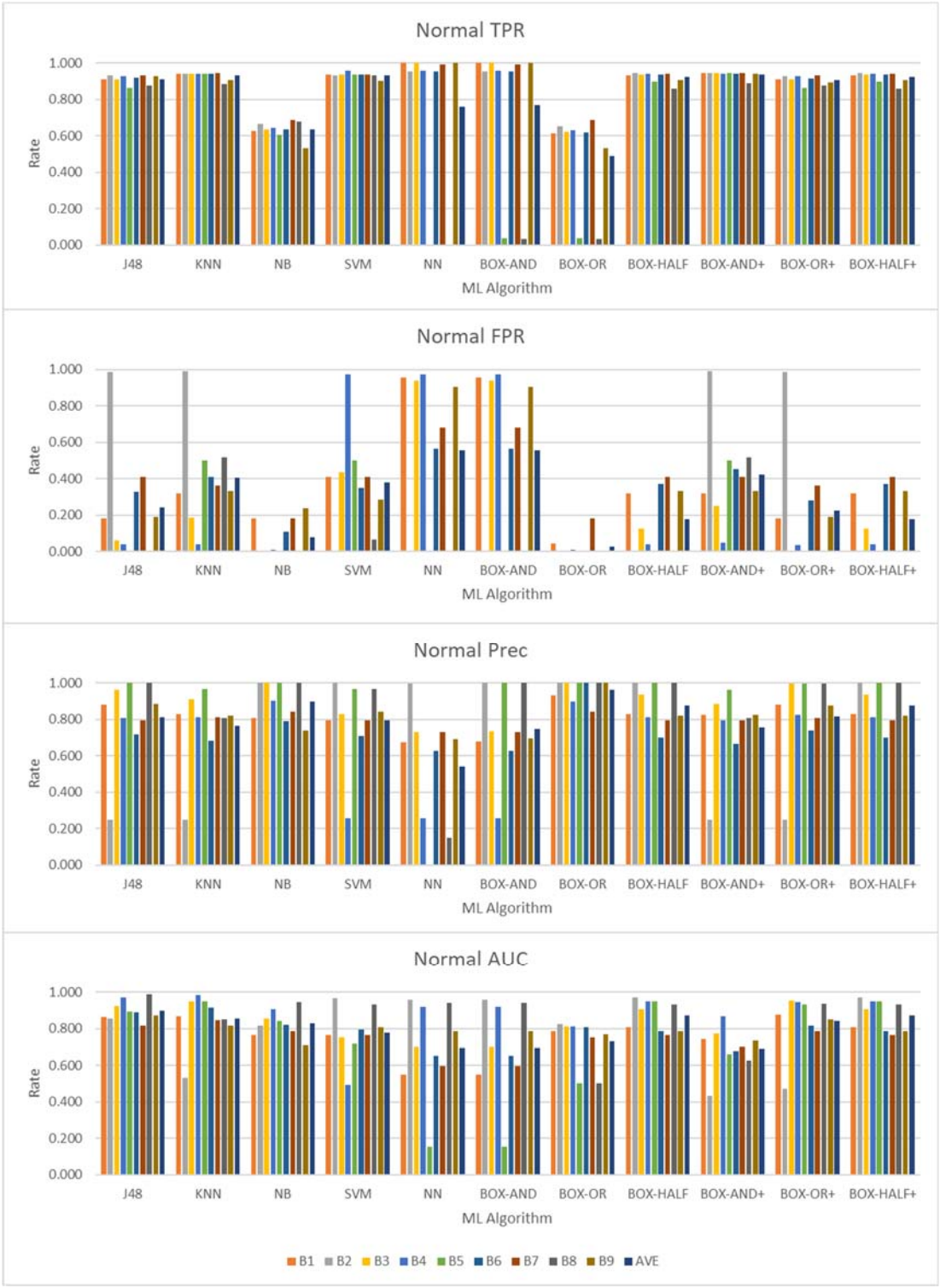


Figure 4-24 - Performance of normal across different malware family dataset

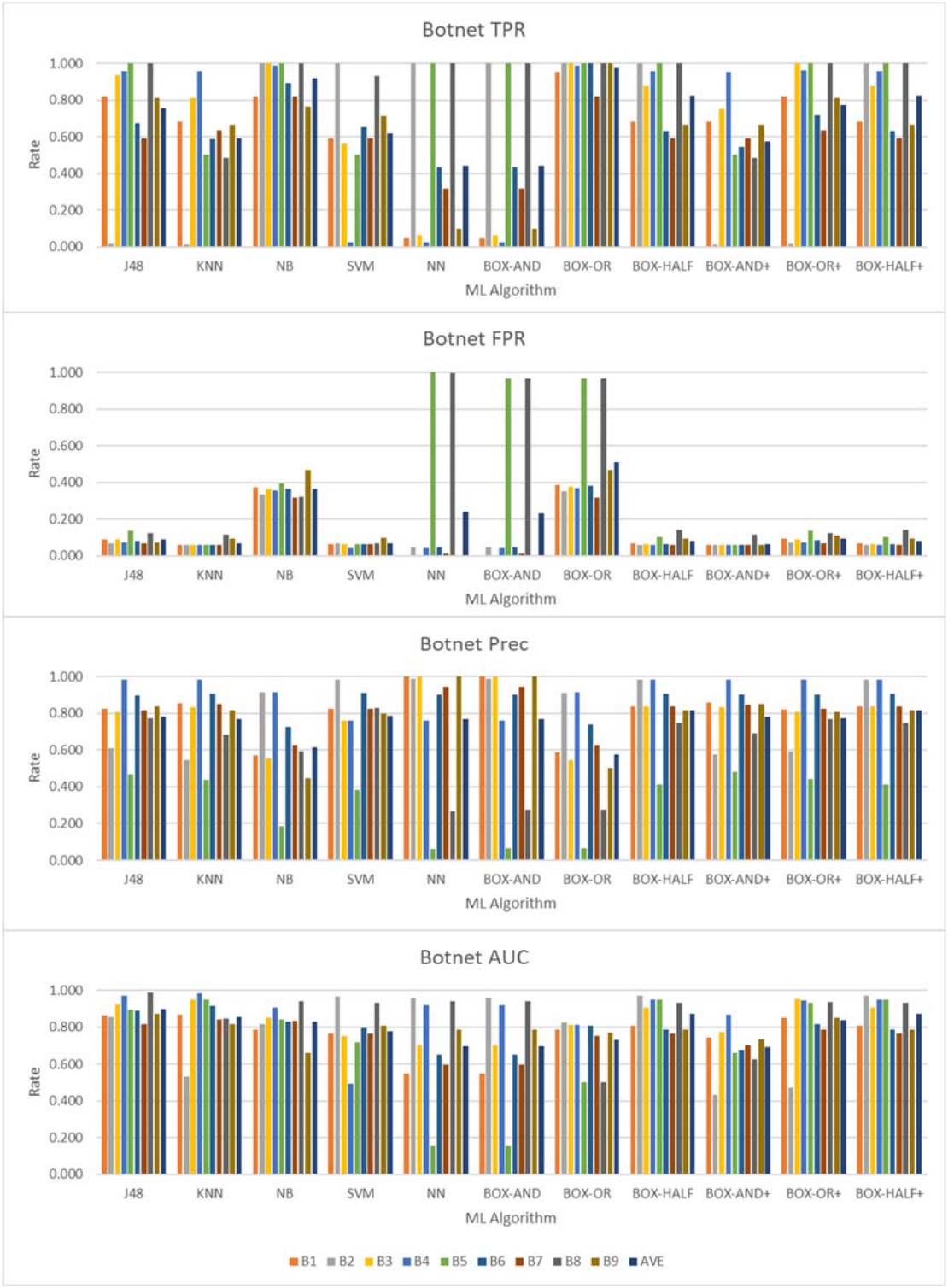


Figure 4-25 - Performance of botnet across different malware family dataset

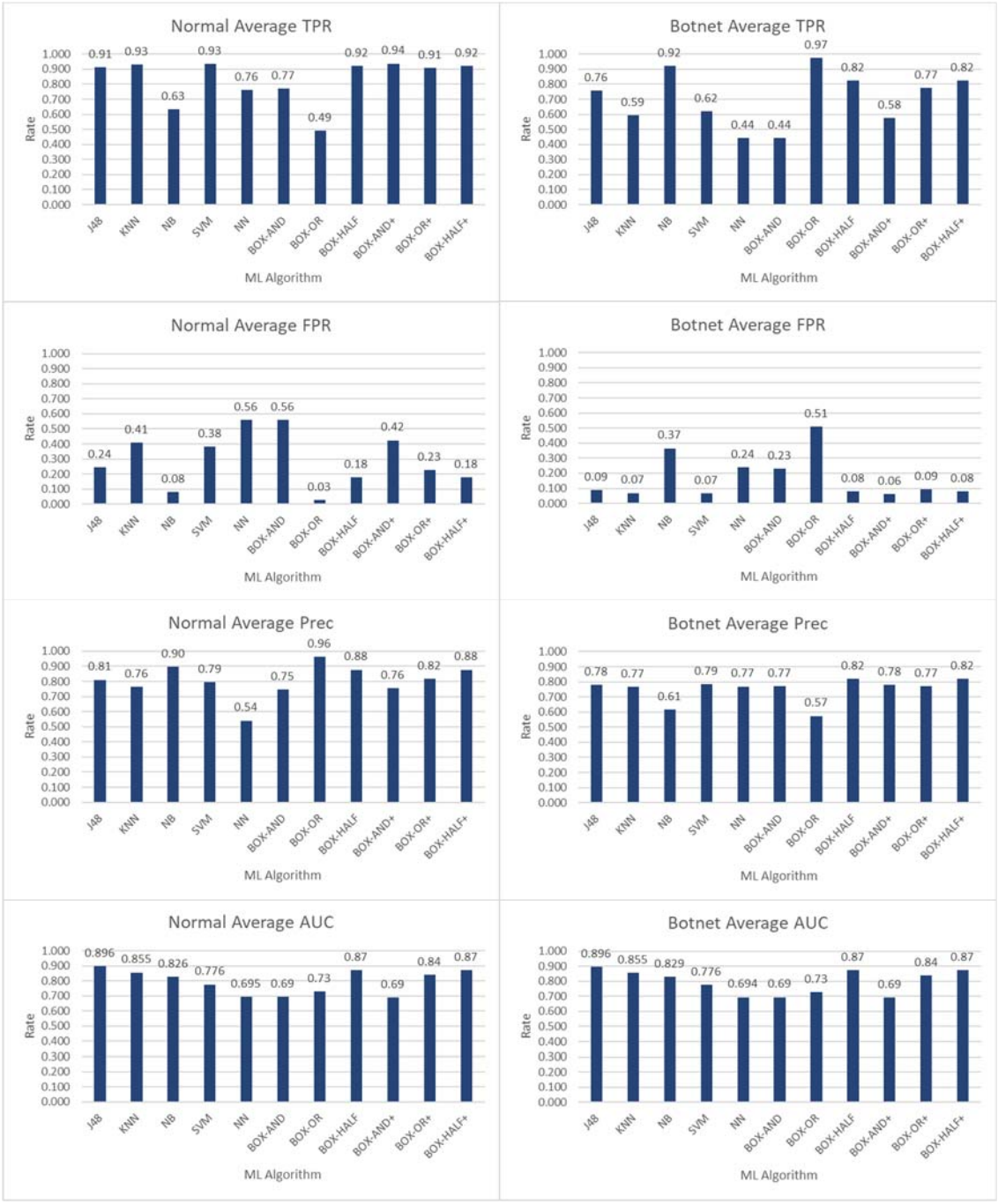


Figure 4-26 - Average performance of malware family dataset

The analysis of the variance of the performance measures in Exp 3 indicated significant statistical differences in all measures (i.e., in TPR, FPR, PRC and AUC) across the different atomic and box ML classifiers, and across the different botnet applications at α=0.05 (see Table 4-16). Also, the interaction between the algorithm used and the botnet family caused statistically significant differences in TPR and AUC but not in FPR and PRC in the case of BOX classifiers.

Table 4-16 - Outcomes of analysis of variance for experiment 3

(A) ATOMIC ML CLASSIFIERS

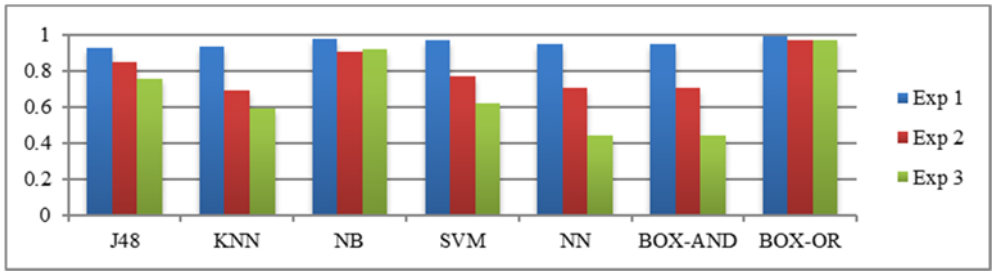
	VarSr	*SS*	*df*	*MS*	*F*	*P-val*	*F crit*
TPR	Bot	5.357	8	0.670	8934.9	0.000	1.957
	Alg	14.045	4	3.511	46849.3	0.000	2.390
	Int	37.654	32	1.177	15699.8	0.000	1.467
FRP	Bot	4.107	8	0.513	22.8	0.000	1.957
	Alg	7.586	4	1.896	84.4	0.000	2.390
	Int	14.106	32	0.441	19.6	0.000	1.467
PRC	Bot	15.396	8	1.924	59.2	0.000	1.957
	Alg	2.280	4	0.570	17.5	0.000	2.390
	Int	9.629	32	0.301	9.2	0.000	1.467
AUC	Bot	1.856	8	0.232	27.9	0.000	1.957
	Alg	2.628	4	0.657	79.0	0.000	2.390
	Int	8.693	32	0.272	32.7	0.000	1.467

(B) BOX ML CLASSIFIERS

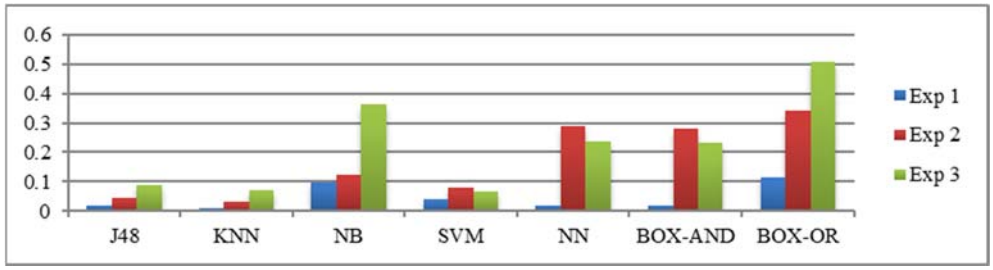
	VarSr	*SS*	*df*	*MS*	*F*	*P-val*	*F crit*
TPR	Bot	7.291	8	0.911	14424.2	0.000	1.954
	Alg	20.622	5	4.124	65272.0	0.000	2.229
	Int	32.269	40	0.807	12767.2	0.000	1.415
FRP	Bot	8.985	8	1.123	62.3	0.000	1.954
	Alg	16.970	5	3.394	188.4	0.000	2.229
	Int	14.009	40	0.350	19.4	0.000	1.415
PRC	Bot	21.860	8	2.732	94.9	0.000	1.954
	Alg	4.340	5	0.868	30.2	0.000	2.229
	Int	8.790	40	0.220	7.6	0.000	1.415
AUC	Bot	2.244	8	0.280	42.5	0.000	1.954
	Alg	3.135	5	0.627	95.1	0.000	2.229
	Int	10.578	40	0.264	40.1	0.000	1.415

(C) KEY: *VarSr*: source of variance; *SS*: sum of squares; *df*: degrees of freedom, *MS*: mean square; *F*: F-value of experimental data; *P-val*: probability of samples of from same population despite difference in variance; *F crit*: minimum F value for accepting null hypothesis at α=0.05; *Bot*: sample groups based on botnet application; *Alg*: sample groups based on ML classifier algorithm; *Int*: interaction between *Alg* and *Bot* groups.

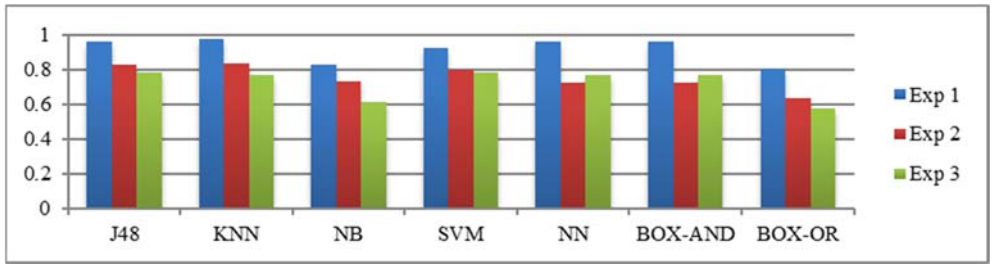
4.6.7 Overall discussion & threats to validity



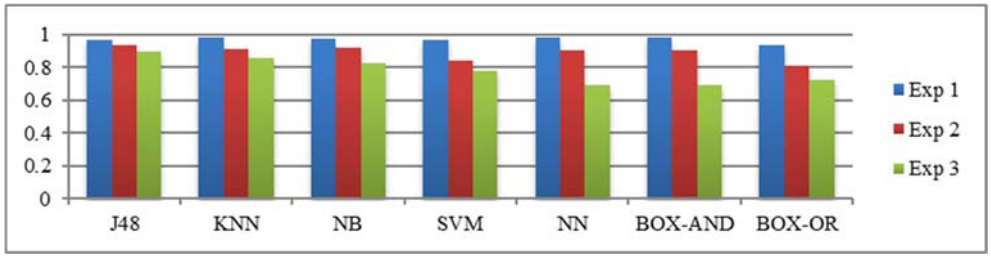
TPR



FPR



PRC



AUC

Figure 4-27 - Average TPR, FPR, PRC and AUC measures for botnet applications in all three experiments.

(Note: The measures for the first experiment (Exp 1) are those obtained for the 10-second period, as this was the aggregation period used in the other two experiments (Exp 2 and Exp 3). Only BOX-AND and BOX-OR are shown for box classifiers the algorithms used for BOX-HALF, BOX-AND+, BOX-OR+ and BOX-HALF+ were not the same across the different experiments.)

The performance of atomic and box ML classifiers varied across the different experiments for all performance measures. Figure 4-27 shows these differences graphically, plotting the average TPR, FPR, PRC and AUC measures for each the atomic ML classifiers in the three experiments. As shown in the figure, the performance of all classifiers (atomic and box) deteriorated from Exp 1 to Exp 2 and from Exp 2 to Exp 3. This was in line with expectations, as the three experiments increased the amount of unknown information: KBKN to UBKN and UBUN.

Considering AUC, the atomic classifier that was more robust to the increasing extent of "unknowns" in the experiments was J48: its AUC dropped by less than 1 percentage point across the different experiments. In terms of TPR, NB showed the less deterioration (i.e., 7 percentage points from Exp 1 to 2). NN resulted in the worst drop of TPR (51 percentage points when contrasting Exp 1 and Exp 3). In terms of FPR, SVM showed the minimum deterioration between Exp 1 and Exp 3 (only 3 percentage points) and NB showed the maximum deterioration (27 percentage points). In terms of PRC, SVM showed the minimum deterioration between Exp 1 and Exp 3 (14 percentage points), and NB and NN the maximum (21 percentage points each). Of the BOX classifiers, whilst BOX-OR's performance was robust in terms of TPR (drop of 2 percentage points between Exp 1 and Exp 3) the performance of both BOX-AND and BOX-OR dropped significantly in terms of FPR (increase of 21 percentage points between Exp 1 and Exp 3) and PRC (drop of 19 percentage points between Exp 1 and Exp 3).

Overall, we believe that the most appropriate indicator regarding the merit of the ML-based analysis of system calls is the outcome of Exp 2. This is because system call profiles on new normal applications may be required prior to making such applications available

on legitimate marketplaces to enable fast offline re-training of ML classifiers in them. With regards to botnets, the set up of Exp 2 was also more realistic than that of Exp 1 as new botnets can appear at any time and there will always be some period before they are detected and training ML classifiers in their traffic. Based on the outcomes of Exp 2, we think that it is fair to say that OS calls based detection has merit since, even with one atomic ML classifier (J48), we were able to reach a TPR of .85, and FPR of 0.05 and PRC of 0.83. Also, TPR and FPR did not appear to vary across different botnet families in a statistically significant way.

Nevertheless, there are some potential threats to the validity of the outcomes of our experiments:

- In general, there is an active period for every botnet. Some botnets may change the botmaster server or go through updates of the malicious code in the infected applications. None of these was reflected in the system call dataset that we considered.
- The availability of a botmaster for every botnet malware could not be guaranteed in our experiments. Therefore, the considered botnets may have further system activity that was not captured.
- The size of the system call dataset for different applications used in the experiments was not same. Some of them were relatively small which was the main reason for having low performance in some botnets.

4.7 Conclusion of experiments

1. Experiment I indicated that our system could be used for distinction of normal and infected network traffic. The performance of J48 and KNN machine learning algorithms is better than other machine learning algorithms which are relatively feasible to classify the network traffic.
2. Experiment II indicated that our system could be used for detecting unknown mobile botnets. Though the performance in the case of unknown botnets is worse than in the case of known ones (see Experiment I), it still can disclose the difference between the malware and benign network traffic.
3. Experiment III indicated the capability of MBotCS to produce a reasonably fast detection/response once the ML-Analyser has been trained.

Chapter 5 Conclusion

In this final chapter, an overview of the main insights will be presented. Meanwhile, we will also show advances gained and state how these have addressed the research challenges and objectives. It is clear that mobile botnet detection is new research field, many unresolved issues are however still apparent, for which we will provide pointers to future research.

5.1 Discussion

In this section, we revisit the objectives set at the start of this research and discuss the extent to which they were achieved.

1. **Objective 1** (To undertake and produce a comprehensive survey of the botnet and mobile botnet research) -- **Finish**: A comprehensive survey of the botnet and the mobile botnet is presented in Chapter 2. There are five parts of the survey including the basic knowledge, conventional botnet, network traffic based anomaly detection, machine learning and mobile botnet. Because mobile botnet is the primary target for our research, we introduced the mobile botnet separately. Both for the conventional botnet and mobile botnet, four aspects are presented which cover the accidents, creation techniques, detection technique and the comparison. The detection techniques are the most important part of the survey. According to

the Chapter 2, we can get an overall understanding of the current state of the botnet and mobile botnet.

2. **Objective 2** (To design a botnet detection system that can operate on mobile devices, to detect unknown mobile botnet with network traffic and system call based on the use of machine learning techniques.) – **Finish:** The design of mobile botnet detection system MBotCS is presented in Chapter 3. We introduced the architecture of MBotCS and explained the meaning and mechanism of every part of the system. In order to evaluate the feasibility of the MBotCS system, we perform a set of experiments in Section 4.1 to verify the machine learning can be used for mobile detection botnet based on the network traffic and system call. According to the result of the experiment, we can find that the system not only can be used for classification of the normal and abnormal traffic and sequence of the system call but also used for the unknown mobile botnet.
3. **Objective 3** (To implement the new mobile botnet detection system on Android devices, addressing the open issues identified in Section 1.2) – **Finish:** The implementation of mobile botnet detection system MBotCS is also presented in Chapter 3. The implementation of MBotCS system includes the user interface and some key programme code for the specific importance functions. The experiment in Section 4.5.4 also proves that the usability of the system on the mobile device. We also deployed the source code of the MBotCS system the GitHub which is a web-based Git repository hosting service.
4. **Objective 4** (To provide an experimental evaluation of the approach.) – **Finish:** There are six experiments described in Section 4.1 that include a very detailed evaluation of the result. We use both tables and graphs to illustrate the performance of every approach.

The hypotheses in Chapter 1 should also be reviewed:

1. **Research hypotheses I – Validity**: According to the result of experiments in Section 4.5.2 and 4.5.3, the features packets/stream frame duration, packets/stream packet size, and arguments number in HTTP request of network traffic can be used for distinguishing normal and botnet Android application by using machine learning.
2. **Research hypotheses II – Invalidity**: According to the basic statistical analysis of system call dataset in Section 4.6.3, the feature of system call cannot distinguish the botnet and normal Android application directly.
3. **Research hypotheses III – Validity**: The analysis in Section 4.6.2.2 indicated that the only statistically significant difference between the botnets and normal applications was related to the frequency of calls of different system functions. Moreover, the result of experiments in Section 4.6.4 to 4.6.6 all prove that frequency of system calls in different time interval can be used for distinguishing normal and botnet Android application by using machine learning.
4. **Research hypotheses IV – Validity**: Based on the result of experiments in 4.1 which contains a detailed comparison of atomic algorithms and Box algorithms present the advantage of aggregated machine learning algorithm. Especially for UBUN scenario, BOX-HALF and BOX-HALF+ have better performance.
5. **Research hypotheses V – Validity**: According to the result of an experiment in Section 4.5.4, the machine learning-based detection system has low energy effect on the battery consumption of the device. Meanwhile it has acceptable execute time.

5.2 Summary of contributions

Base on the comprehensive insight into botnet and survey for the mobile botnet, the contributions of this research can be summarised as follows:

Contribution 1: A central contribution of this thesis is the design and implementation of the mobile botnet detection system MBotCS. This system analyses the traffic data passing through the mobile botnet and system call invoked by applications which are based on machine learning. We described the architecture of MBotCS in Chapter 3 which contains four parts (traffic data pre-processor, machine learning analyser, user interface and training dataset). Based on the architecture, we also implement the system on Android mobile device. According to the experiment, the system has a very low energy effect on the battery consumption of the device with only 0.5% of the total battery during the period of the experiment. Moreover,the J48 algorithm has fast average execution time with only 1.216 seconds.

Contribution 2: We performs a series of experiments that are superior to existing research as follows: (a) The use of not only atomic but also box ML classifiers using supervised learning. (b) The investigation of the performance of ML classifiers a wider set of detection scenarios than existing work, namely detection of known botnets and known normal applications (KBKN scenario), unknown botnets and known normal applications (UBKN scenario), and unknown botnets and normal applications (UNUB scenario). (c) The use of a comprehensive set of Android mobile botnets, which had not been considered previously, without relying on any form of synthetic training data. (d) The conduct of a thorough sensitivity analysis in which the statistical significance of differences in detection performance measures on ML algorithms, system call aggregation periods, normal and botnet applications, and different types of botnet families have been explored.

Our publication also represents the outcomes and contribution to our research. We can also collect feedback from other researchers by attending the conference and publish papers:

MBotCS: A mobile Botnet detection system based on machine learning (Xin Meng and George Spanoudakis)

5.3 Further research

Currently, we are investigating the possibility of analysing traffic across networks of mobile devices (as opposed to single devices) and traffic between botnets and the system software of the device to see if we get any performance gains. We are also planning more extensive experimental evaluations with larger data sets. Finally, we want to explore the use of unsupervised classification and contrast its outcomes with supervised classification and investigate the reasons underpinning the differences in the performance of the basic ML algorithms.

5.3.1 Future research directions

1. Improvement the performance of machine learning: We have used five atomic machine learning algorithms and six aggregation algorithms in our experiments. The evaluations have performed for these algorithms to find that the performance is different from these algorithms. So we can know that it is still possible to improve the performance of the detection through improving or changing machining learning algorithms. Therefore, one of the future research directions to try other machine learning algorithms for the classification and improved current machine learning algorithms. More specifically, the unsupervised and deep learning machine learning algorithms should be considered in the future research.
2. Improvement of the training dataset: We have used ten existing mobile botnet malware families (nearly 170 applications) as a sample for training dataset. There is a part of the *MalGenome* project of a mobile botnet. So to make sure that the method can be used for a general mobile botnet, we need to choose more type of

mobile botnet malware application to perform the experiments in the future. The other reason to add more types of malware family is to increase the robustness of machine learning analysis. As we know that the malicious code on bots could be updated by the botmaster periodically. However more types of botnet malware can cover more patterns of malicious behaviours. Then the experiments result could be more stable even thinking about the scalability of the application.

3. Improvement of the attributes selection of network traffic for the machine learning: According to the Section 4.5.1.1, we select some features for training which include Packets/Stream Frame Duration, Packets/Stream Packet Size, and Arguments Number in HTTP Request URL. These features are part of attributes of the network traffic. In order to improve the performance of the machine learning algorithms, we can study the distinction between normal and abnormal traffic to dig more features for machine learning.
4. Further study to investigate detection based on both system calls (internal activity) and network traffic (external communications) and explore potential performances gains. Meanwhile perform extensive experimental evaluations Android devices in order to evaluate the implications of the integrated approach to the CPU, memory and battery of the device.

5.3.2 Planned and related work within MBotCS

1. Improvement of the system on the Android mobile device.

The rudimentary implementation of the MBotCS is presented in this thesis. We use the tPacketCapture to realise the network traffic monitor and WEKA to realise the machine learning algorithms. There are also some improving points for the implementation of MBotCS.

- Through studying the network traffic capture techniques, implementation for the function for the real-time network traffic on a mobile device. Under the current knowledge, we can create a connection of VPN on the mobile device and let all the traffic pass through this connection to record.
- Reimplementation the machine learning algorithm by using native programme language to accelerate the computational speed which can take full advantage of most out the limited resources on the mobile device.
- Adding the programme configuration system and the warning system, which can improve the user experience.

2. The open standards for mobile botnet detection

According to analysis about the previous botnet detection, we can find that comparison of the different techniques is one challenge. The most important reason is that there is not open standards dataset for testing these techniques. Therefore, the extension of the MBotCS can make contributions for the open standards for mobile botnet detection. These are several points should be paid attention in the future.

- Development the API to visit the training dataset and create access standard for a different programming language.
- Establishment of a mechanism for collection training dataset to expand the size of training dataset and update the training dataset.
- Thinking about the privacy and confidentiality in the datasets captured from the mobile devices.

References

1. Brian, J.L. *DEFENSE DEPARTMENT CYBER EFFORTS: DOD Faces Challenges In Its Cyber Activities*. 2011 [cited 2015 24 August]; Available from: http://www.gao.gov/products/GAO-11-75.
2. Weber, T. *Criminals 'may overwhelm the web*. 2007 [cited 2015 24 August]; Available from: http://news.bbc.co.uk/1/hi/business/6298641.stm.
3. AsSadhan, B., et al. *Detecting botnets using command and control traffic*. in *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*. 2009. IEEE.
4. Miller, R. *Botnet with 10,000 Machines Shut Down*. 2004 [cited 2015 24 August]; Available from: http://news.netcraft.com/archives/2004/09/08/botnet_with_10000_machines_shut_down.html.
5. BBC. *Zombie plague sweeps the internet*. 2008 [cited 2015 24 August]; Available from: http://news.bbc.co.uk/1/hi/technology/7596676.stm.
6. BBC. *Spam 'produces 17m tons of CO2'*. 2009 [cited 2015 24 August]; Available from: http://news.bbc.co.uk/1/hi/technology/8001749.stm.
7. Liu, F. *MDK: The Largest Mobile Botnet in China*. 2013 [cited 2015 24 August]; Available from: http://www.symantec.com/connect/ko/blogs/mdk-largest-mobile-botnet-china.
8. Muncaster, P. *Citadel botnet resurges to storm Japanese PCs*. 2013 [cited 2015 24 August]; Available from: http://www.theregister.co.uk/2013/09/04/citadel_wreaks_havoc_in_japan/.
9. Bisson, D. *Attackers Launched 124,000 DDoS Events Per Week Over Past 18 Months, Finds Report*. 2016 [cited 2017 June 18]; Available from: https://www.tripwire.com/state-of-security/incident-detection/attackers-launched-124000-ddos-events-per-week-over-past-18-months-finds-report/.

10. Bisson, D. *The 5 Most Significant DDoS Attacks of 2016*. 2016 [cited 2017 June 18]; Available from: https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/5-significant-ddos-attacks-2016/.
11. Woolf, N. *DDoS attack that disrupted internet was largest of its kind in history, experts say*. 2016 [cited 2017 June 18]; Available from: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet.
12. Mimoso, M. *MIRAI BOTS MORE THAN DOUBLE SINCE SOURCE CODE RELEASE*. 2016 [cited 2017 June 18]; Available from: https://threatpost.com/mirai-bots-more-than-double-since-source-code-release/121368/.
13. Labs, L.T.R. *How the Grinch Stole IoT*. 2016 [cited 2017 June 18]; Available from: http://www.netformation.com/level-3-pov/how-the-grinch-stole-iot.
14. Christiaan Beek, D.F., Paula Greve,Yashashree Gund, Francisca Moreno, Eric Peterson, Craig Schmugar, Rick Simon, Dan Sommer, Bing Sun, Ravikant Tiwari, Vincent Weafer, *McAfee Labs Threats Report*. 2017, McAfee Labs.
15. Bisson, D. *100,000 Bots Infected with Mirai Malware Behind Dyn DDoS Attack*. 2016 [cited 2017 June 18]; Available from: https://www.tripwire.com/state-of-security/latest-security-news/100000-bots-infected-mirai-malware-caused-dyn-ddos-attack/.
16. Paganini, P. *For the first time massive DDoS attacks hit Russian banks in 2016*. 2016 [cited 2017 June 18]; Available from: http://securityaffairs.co/wordpress/53312/cyber-crime/ddos-attacks-russia-banks.html.
17. Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019*. 2015.
18. Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper*. 2017.
19. FRAMINGHAM, M., *Worldwide Smartphone Market Will See the First Single-Digit Growth Year on Record, According to IDC*. 2015.
20. Christian Funk, M.G., *KASPERSKY SECURITY BULLETIN 2013*. 2013, Kaspersky Lab Global Research and analysis team.
21. Unuchek, R., *Mobile malware evolution 2016*. 2017, securelist.
22. Porras, P., H. Saidi, and V. Yegneswaran, *An analysis of the ikee. b iphone botnet*, in *Security and Privacy in Mobile Information and Communication Systems*. 2010, Springer. p. 141-152.

23. Zorz, Z., *Android Trojan with botnet capabilities found in the wild*. 2014.

24. Strazzere, T., *The new NotCompatible: Sophisticated and evasive threat harbors the potential to compromise enterprise networks*. 2014.

25. Kalige, E., D. Burkey, and I. Director, *A case study of Eurograbber: How 36 million euros was stolen via malware.* Versafe (White paper), 2012.

26. Seo, S.-H., et al., *Detecting mobile malware threats to homeland security through static analysis.* Journal of Network and Computer Applications, 2014. **38**: p. 43-53.

27. Eslahi, M., R. Salleh, and N.B. Anuar. *MoBots: A new generation of botnets on mobile devices and networks*. in *Computer Applications and Industrial Electronics (ISCAIE), 2012 IEEE Symposium on*. 2012. IEEE.

28. Xiang, C., et al. *Andbot: towards advanced mobile botnets*. in *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*. 2011. USENIX Association.

29. Batyuk, L., et al. *Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications*. in *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*. 2011. IEEE.

30. Anwar, S., et al. *A static approach towards mobile botnet detection*. in *Electronic Design (ICED), 2016 3rd International Conference on*. 2016. IEEE.

31. Zhou, W., et al. *Fast, scalable detection of piggybacked mobile applications*. in *Proceedings of the third ACM conference on Data and application security and privacy*. 2013. ACM.

32. Ham, Y.J., et al. *Activation pattern analysis on malicious android mobile applications*. in *Proc. 2013 Firnst International Conference on Artificial Intelligence, Modelling \& Simulation*. 2013.

33. Ham, Y.J., et al., *Android mobile application system call event pattern analysis for determination of malicious attack.* International Journal of Security and Its Applications, 2014. **8**(1): p. 231-246.

34. Shabtai, A., U. Kanonov, and Y. Elovici. *Detection, Alert and Response to Malicious Behavior in Mobile Devices: Knowledge-Based Approach*. in *Recent Advances in Intrusion Detection*. 2009. Springer.

35. Vural, I. and H. Venter, *Mobile botnet detection using network forensics*, in *Future Internet-FIS 2010*. 2010, Springer. p. 57-67.

36. Johnson, E. and I. Traore. *SMS Botnet Detection for Android Devices through Intent Capture and Modeling*. in *Reliable Distributed Systems Workshop (SRDSW), 2015 IEEE 34th Symposium on*. 2015. IEEE.

37. Schmidt, A.-D., et al., *Monitoring smartphones for anomaly detection.* Mobile Networks and Applications, 2009. **14**(1): p. 92-106.

38. Alzahrani, A.J. and A.A. Ghorbani. *SMS-Based Mobile Botnet Detection Module*. in *IT Convergence and Security (ICITCS), 2016 6th International Conference on*. 2016. IEEE.

39. Feizollah, A., et al., *A study of machine learning classifiers for anomaly-based mobile botnet detection.* Malaysian Journal of Computer Science, 2014. **26**(4).

40. Meng Xin, S.G., *MBotCS: A Mobile Botnet Detection System Based on Machine Learning*, in *10th International Conference on Risks and Security of Internet and Systems (CRiSIS 2015)*. July 2015: Greece.

41. Karim, A., R. Salleh, and M.K. Khan, *SMARTbot: A Behavioral Analysis Framework Augmented with Machine Learning to Identify Mobile Botnet Applications.* PLoS ONE, 2016. **11**(3): p. e0150077.

42. Singh, K., et al., *Evaluating bluetooth as a medium for botnet command and control*, in *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2010, Springer. p. 61-80.

43. Zhou, Y. and X. Jiang, *An analysis of the anserverbot trojan*. 2011.

44. Hamandi, K., et al. *Android SMS botnet: a new perspective*. in *Proceedings of the 10th ACM international symposium on Mobility management and wireless access*. 2012. ACM.

45. Faghani, M.R. and U.T. Nguyen. *Socellbot: A new botnet design to infect smartphones via online social networking*. in *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*. 2012. IEEE.

46. Geng, G., et al., *The design of sms based heterogeneous mobile botnet.* Journal of Computers, 2012. **7**(1): p. 235-243.

47. Geng, G., et al. *An improved sms based heterogeneous mobile botnet model*. in *Information and Automation (ICIA), 2011 IEEE International Conference on*. 2011. IEEE.

48. Hasan, R., et al. *Sensing-enabled channels for hard-to-detect command and control of mobile devices*. in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 2013. ACM.

49. Hua, J. and K. Sakurai, *A sms-based mobile botnet using flooding algorithm*, in *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*. 2011, Springer. p. 264-279.

50. Hua, J. and K. Sakurai, *Botnet command and control based on Short Message Service and human mobility.* Computer Networks, 2013. **57**(2): p. 579-597.

51. Jiang, R.M., et al. *JokerBot–An Android-Based Botnet*. in *Applied Mechanics and Materials*. 2013. Trans Tech Publ.

52. Li, Y., et al., *Control Method of Twitter-and SMS-Based Mobile Botnet*, in *Trustworthy Computing and Services*. 2013, Springer. p. 644-650.

53. Mulliner, C. and J.-P. Seifert. *Rise of the iBots: Owning a telco network*. in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*. 2010. IEEE.

54. Pieterse, H. and M. Olivier. *Design of a hybrid command and control mobile botnet*. in *Proceedings of the 8th International Conference on Information Warfare and Security: ICIW 2013*. 2013. Academic Conferences Limited.

55. Shuai, W., et al., *S-URL Flux: A Novel C&C Protocol for Mobile Botnets*, in *Trustworthy Computing and Services*. 2013, Springer. p. 412-419.

56. Zeng, Y., *On detection of current and next-generation botnets*. 2012, The University of Michigan.

57. Zeng, Y., K.G. Shin, and X. Hu. *Design of SMS commanded-and-controlled and P2P-structured mobile botnets*. in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. 2012. ACM.

58. Zhao, S., et al. *Cloud-based push-styled mobile botnets: a case study of exploiting the cloud to device messaging service*. in *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012. ACM.

59. Weidman, G., *Transparent botnet command and control for smartphones over sms.* Shmoocon 2011, 2011.

60. McCarty, B., *Botnets: Big and bigger.* Security & Privacy, IEEE, 2003. **1**(4): p. 87-90.

61. Puri, R., *Bots & botnet: An overview.* SANS Institute, 2003. **3**: p. 58.

62. Abu Rajab, M., et al. *A multifaceted approach to understanding the botnet phenomenon*. in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006. ACM.

63. Zhu, Z., et al. *Botnet research survey*. in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*. 2008. IEEE.

64. Liu, J., et al. *Botnet: classification, attacks, detection, tracing, and preventive measures*. in *EURASIP journal on wireless communications and networking*. 2009. IEEE Computer Society.

65. Li, C., W. Jiang, and X. Zou. *Botnet: Survey and case study*. in *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*. 2009. IEEE.

66. Symantec, *Spybot worm.* Symantec, 2003.

67. Shin, Y.-H. and E.-G. Im. *A survey of botnet: consequences, defenses and challenges*. in *Joint Workshop on Internet Security*. 2009.

68. Zhang, L., et al. *A survey on latest botnet attack and defense*. in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. 2011. IEEE.

69. Silva, S.S., et al., *Botnets: A survey.* Computer Networks, 2013. **57**(2): p. 378-403.

70. Dagon, D., et al. *A taxonomy of botnet structures*. in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. 2007. IEEE.

71. Zeidanloo, H.R., et al. *A taxonomy of botnet detection techniques*. in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. 2010. IEEE.

72. Zeidanloo, H.R. and A.A. Manaf. *Botnet command and control mechanisms*. in *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on*. 2009. IEEE.

73. Czosseck, C. and K. Podins. *A Usage-Centric Botnet Taxonomy*. in *Proceedings of the 10th European Conference on Information Warfare and Security*. 2011.

74. Hachem, N., et al. *Botnets: lifecycle and taxonomy*. in *Network and Information Systems Security (SAR-SSI), 2011 Conference on*. 2011. IEEE.

75. Pagliery, J. *Nearly one million Android phones infected by hackers*. 2016 [cited 2017; Available from: http://money.cnn.com/2016/11/30/technology/android-phones-infected/index.html.

76. wikipedia. *Botnet*. 2015 [cited 2015 24 August]; Available from: https://en.wikipedia.org/wiki/Botnet.

77. Genachowski, F.C.J., *Final Report US Anti-Bot Code of Conduct (ABCs) for Internet Service Providers (ISPs).* 2012.

78. Ianelli, N. and A. Hackworth, *Botnets as a vehicle for online crime.* FORENSIC COMPUTER SCIENCE IJoFCS, 2005: p. 19.

79. OECD, *Proactive Policy Measures by Internet Service Providers against Botnets*. 2012: OECD Publishing.

80. OECD, *Malicious software(Malware):A Security Threat to the Internet Economy*. 2008.

81. Feily, M., A. Shahrestani, and S. Ramadass. *A survey of botnet and botnet detection*. in *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*. 2009. IEEE.

82. Bailey, M., et al. *A survey of botnet technology and defenses*. in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*. 2009. IEEE.

83. Oikarinen, J. and D. Reed, *Internet relay chat protocol.* 1993.

84. Berners-Lee, T., R. Fielding, and H. Frystyk, *Hypertext transfer protocol--HTTP/1.0*. 1996.

85. Wilde, E. and A. Vaha-Sipila, *Uri scheme for global system for mobile communications (gsm) short message service (sms)*. 2010.

86. Barford, P. and V. Yegneswaran, *An inside look at botnets*, in *Malware Detection*. 2007, Springer. p. 171-191.

87. Cooke, E., F. Jahanian, and D. McPherson. *The zombie roundup: Understanding, detecting, and disrupting botnets*. in *Proceedings of the USENIX SRUTI Workshop*. 2005.

88. Wang, P., S. Sparks, and C.C. Zou, *An advanced hybrid peer-to-peer botnet.* Dependable and Secure Computing, IEEE Transactions on, 2010. **7**(2): p. 113-127.

89. Ollmann, G., *Botnet communication topologies.* Retrieved September, 2009. **30**: p. 2009.

90. Leyden, J. *IRC botnets dying off*. 2010 [cited 2015 Symantec]; Available from: http://www.theregister.co.uk/2010/11/16/irc_botnets_dying_off/.

91. Dittrich, D. and S. Dietrich. *P2P as botnet command and control: a deeper insight*. in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*. 2008. IEEE.

92. Wikipedia. *Push technology*. 2015 [cited 2015 24 August]; Available from: https://en.wikipedia.org/wiki/Push_technology.

93. Apple. *Apple's Push Notification Service*. 2015 [cited 2015 24 August]; Available from: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html.

94. BlackBerry. *Blackberry's Push Service*. 2013 2013-04-14T19:12:54Z [cited 2015 24 August]; Available from: http://developer.blackberry.com/services/push/.

95. Google. *Google Cloud Messaging*. 2014 [cited 2014 24 August]; Available from: https://developers.google.com/cloud-messaging/.

96. Microsoft. *Push Notifications (Windows Phone)*. 2015 [cited 2015 24 August]; Available from: https://msdn.microsoft.com/en-us/library/hh221549.aspx.

97. Nokia. *Nokia's Notifications API (NNA)*. 2013 [cited 2014 August]; Available from: https://projects.developer.nokia.com/notificationsapi/wiki.

98. Google. *Android Cloud to Device Messaging Framework*. 2012 [cited 2013 10 September]; Available from: Android Cloud to Device Messaging Framework.

99. Sharma, R.K. and G.S. Chandel, *Botnet detection and resolution challenges: A survey paper.* Int. J. Comput. Inf. Technol. Bioinforma, 2009. **1**: p. 10-15.

100. Schiller, C. and J.R. Binkley, *Botnets: The killer web applications*. 2011: Syngress.

101. Cisco. *What Is the Difference: Viruses, Worms, Trojans, and Bots?* 2015 [cited 2018 14 April]; Available from: https://www.cisco.com/c/en/us/about/security-center/virus-differences.html.

102. Symantec. *PrettyPark.Worm*. 2007 [cited 2018 14 April]; Available from: https://www.symantec.com/security_response/writeup.jsp?docid=2000-121508-3334-99.

103. Symantec. *What is the difference between viruses, worms, and Trojans?* 2016 [cited 2018 14 April]; Available from: https://support.symantec.com/en_US/article.TECH98539.html.

104. Symantec. *JS.Debeski.Trojan*. 2007 [cited 2018 14 April]; Available from: https://www.symantec.com/security_response/writeup.jsp?docid=2003-100216-0712-99.

105. Tiwari, A. *What Is The Difference: Viruses, Worms, Ransomware, Trojans, Bots, Malware, Spyware, Etc?* 2015 [cited 2018 14 April]; Available from: https://fossbytes.com/difference-viruses-worms-ransomware-trojans-bots-malware-spyware-etc/.

106. Rouse, M. *Zeus Trojan (Zbot)*. 2012 [cited 2018 14 April]; Available from: https://searchsecurity.techtarget.com/definition/Zeus-Trojan-Zbot.

107. Rumen, S., R. Jonathan, and S. Thomas. *Eggheads*. 2010 [cited 2015 24 August]; Available from: http://www.eggheads.org/.

108. Wikipedia. *Agobot*. 2015 [cited 2015 24 August]; Available from: https://en.wikipedia.org/wiki/Agobot.

109. Kharouni, L., *SDBOT IRC botnet continues to make waves.* A Trend Micro White Paper, 2009: p. 1-20.

110. infectionvectors.com. *Agobot and the Kitchen Sink*. 2014 [cited 2015 24 August]; Available from: http://www.itk.ilstu.edu/faculty/ytang/botnet/Agobot_&_the_Kit-chen_Sink.pdf.

111. McAfee. *W32/Sdbot.worm*. 2013 [cited 2015 24 August]; Available from: http://home.mcafee.com/VirusInfo/VirusProfile.aspx?key=3948010.

112. Yen, T.-F. and M.K. Reiter, *Traffic aggregation for malware detection*, in *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2008, Springer. p. 207-227.

113. Yury, M. *The Bagle botnet*. 2005 [cited 2015 24 August]; Available from: https://securelist.com/analysis/36046/the-bagle-botnet/.

114. Miller, C., *The Rustock botnet spams again.* SC Magazine, July, 2008. **25**.

115. Microsoft. *Win32/Rustock*. 2007 [cited 2015 24 August]; Available from: http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32/Rustock.

116. John, E.D. *Srizbi Becomes World's Largest Botnet*. 2008 [cited 2015 24 August]; Available from: http://www.pcworld.com/article/146017/article.html.

117. Brian, P. *IT Security & Network Security News & Reviews: The Rise and Fall of the Srizbi Botnet*. 2009 [cited 2015 24 August]; Available from: http://www.eweek.com/c/a/Security/The-Rise-and-Fall-of-the-Srizbi-Botnet.

118. McAfee. *W32/Akbot*. 2006 [cited 2015 24 August]; Available from: http://www.mcafee.com/threat-intelligence/malware/default.aspx?id=138050.

119. McAfee. *Virus Profile: W32/Akbot!d*. 2010 [cited 2015 24 August]; Available from: http://home.mcafee.com/virusinfo/virusprofile.aspx?key=260585.

120. TrendMirco. *CUTWAIL*. 2009 [cited 2015 24 August]; Available from: http://about-threats.trendmicro.com/malware.aspx?language=au&name=CUTWAIL.

121. Sousa, R., et al. *Analyzing the behavior of top spam botnets*. in *Communications (ICC), 2012 IEEE International Conference on*. 2012. IEEE.

122. Stone-Gross, B., et al. *The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns*. in *USENIX workshop on large-scale exploits and emergent threats (LEET)*. 2011.

123. Stevens, K. and D. Jackson, *Zeus banking trojan report.* Atlanta, DELL Secureworks. http://www. secureworks. com/research/threats/zeus, 2010.

124. Falliere, N. and E. Chien, *Zeus: King of the bots.* Symantec Security Respons e (http://bit. ly/3VyFV1), 2009.

125. Gary, W. *UAB Computer Forensics Links Fake Online Postcards to Most Prevalent U.S. Computer Virus*. 2009 [cited 2015 24 August]; Available from: https://www.uab.edu/newsarchive/66204-uab-computer-forensics-links-fake-online-postcards-to-most-prevalent-u-s-computer-virus.

126. Steven, M. *Microsoft identifies two Zeus botnet crime ring suspects*. 2012 [cited 2015 24 August]; Available from: http://www.cnet.com/news/microsoft-identifies-two-zeus-botnet-crime-ring-suspects/.

127. Wikipedia. *Zeus (trojan horse)*. 2015 [cited 2015 24 August]; Available from: https://en.wikipedia.org/wiki/Zeus_(trojan_horse.

128. Larkin, E. *Storm Worm's virulence may change tactics*. Network World (August 2, 2007) 2007 [cited 2015 24 August].

129. Stewart, J., *Inside the storm: Protocols and encryption of the storm botnet.* Black Hat USA, 2008.

130. Stover, S., et al., *Analysis of the Storm and Nugache Trojans: P2P is here.* USENIX; login, 2007. **32**(6): p. 18-27.

131. Holz, T., et al., *Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm.* LEET, 2008. **8**(1): p. 1-9.

132. Sinclair, G., C. Nunnery, and B.B. Kang. *The waledac protocol: The how and why*. in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. 2009. IEEE.

133. Baltazar, J., J. Costoya, and R. Flores, *Infiltrating WALEDAC Botnet's Covert Operations.* TREND MICRO, 2009.

134. Stock, B., et al. *Walowdac-analysis of a peer-to-peer botnet*. in *Computer Network Defense (EC2ND), 2009 European Conference on*. 2009. IEEE.

135. Nicolas, F. *The Sality Botnet*. 2010 [cited 2015 24 August]; Available from: http://www.symantec.com/connect/blogs/sality-botnet.

136. Falliere, N., *Sality: Story of a peer-to-peer viral network.* Rapport technique, Symantec Corporation, 2011.

137. wikipedia. *Conficker*. 2015 [cited 2015 24 August]; Available from: https://en.wikipedia.org/wiki/Conficker.

138. Robert, M. *Conficker worm gets an evil twin*. 2009 [cited 2015 24 August]; Available from: http://www.computerworld.com/article/2531360/network-security/conficker-worm-gets-an-evil-twin.html.

139. McMillan, R., *Experts bicker over conficker numbers.* Techworld, April, 2009. **15**.

140. AmCham, V., *Computing Tips: "Amazing" Downadup worm (Conficker) has infected 10 million PCs.* 2012. **2015**.

141. Porras, P., *Inside risks reflections on Conficker.* Communications of the ACM, 2009. **52**(10): p. 23-24.

142. Nahorney, B., *The Downadup Codex a comprehensive guide to the threat's mechanics.* Symantec| Security Response, 2009.

143. Villeneuve, N., J. dela Torre, and D. Sancho, *Asprox Reborn.* 2013.

144. Shin, Y., S. Myers, and M. Gupta, *A case study on asprox infection dynamics*, in *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2009, Springer. p. 1-20.

145. Borgaonkar, R. *An analysis of the asprox botnet*. in *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*. 2010. IEEE.

146. Symantec. *Trojan.Asprox*. 2007 [cited 2015 24 August]; Available from: http://www.symantec.com/security_response/writeup.jsp?docid=2007-060812-4603-99.

147. Brian, K. *Who Is the 'Festi' Botmaster?* 2012 [cited 2015 24 August]; Available from: http://krebsonsecurity.com/2012/06/who-is-the-festi-botmaster/.

148. Eugene, R. and M. Aleksandr. *King of Spam: Festi Botnet Analysis*. 2012 [cited 2015 24 August]; Available from: http://www.welivesecurity.com/wp-content/media_files/king-of-spam-festi-botnet-analysis.pdf.

149. Jeremy, K. *Spamhaus Declares Grum Botnet Dead, but Festi Surges*. 2012 [cited 2015 24 August]; Available from: http://www.pcworld.com/article/260984/spamhaus_declares_grum_botnet_dead_but_festi_surges.html.

150. Matrosov, A. and E. Rodionov, *Festi Botnet Analysis & Investigation.* 2011.

151. Brian, K. *Inside the Grum Botnet*. 2012 [cited 2015 24 August]; Available from: http://krebsonsecurity.com/2012/08/inside-the-grum-botnet/.

152. Mushtaq, A. *Grum, World¡¯s Third-Largest Botnet, Knocked Down*. 2012 [cited 2015 24 August]; Available from: https://www.fireeye.com/blog/threat-research/2012/07/grum-botnet-no-longer-safe-havens.html.

153. Golovanov, S. and I. Soumenkov, *TDL4 top bot.* Kaspersky Lab Analysis, 2011.

154. Greengard, S., *The war against botnets.* Communications of the ACM, 2012. **55**(2): p. 16-18.

155. Luo, Y., *Efficiency Study of Sybil Attack on P2P Botnets*. 2012, Concordia University Montreal, Quebec, Canada.

156. Elinor, M. *Microsoft halts another botnet: Kelihos*. 2011 [cited 2015 24 August]; Available from: http://news.cnet.com/8301-1009_3-20112289-83/microsoft-halts-another-botnet-kelihos/.

157. Werner, T., *Botnet Shutdown Success Story: How Kaspersky Lab Disabled the Hlux/Kelihos Botnet, 2011.* Technical Repo rt: http://www. securelist. com/en/blog/208193137, 2011.

158. Don, R. *'Chameleon Botnet' takes $6-million-a-month in ad money*. 2013 [cited 2015 24 August]; Available from: http://www.cnet.com/news/chameleon-botnet-takes-6-million-a-month-in-ad-money/.

159. Smith, M.L., *Prosecuting the Undead: Federal Criminal Law in a World of Zombies.* 2013.

160. Spider.io. *Discovered: Botnet Costing Display Advertisers over Six Million Dollars per Month*. 2013 [cited 2015 24 August]; Available from: http://www.spider.io/blog/2013/03/chameleon-botnet/.

161. Starnberger, G., C. Kruegel, and E. Kirda. *Overbot: a botnet protocol based on Kademlia*. in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*. 2008. ACM.

162. Liu, C., et al. *A recoverable hybrid C&C botnet*. in *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*. 2011. IEEE.

163. Tyagi, A.K. and G. Aghila, *A wide scale survey on botnet.* International Journal of Computer Applications, 2011. **34**(9): p. 9-22.

164. Les, C. *Passive vs. Active Monitoring*. 2001 [cited 2015 24 August]; Available from: http://www.slac.stanford.edu/comp/net/wan-mon/passive-vs-active.html.

165. Vrable, M., et al., *Scalability, fidelity, and containment in the potemkin virtual honeyfarm.* ACM SIGOPS Operating Systems Review, 2005. **39**(5): p. 148-162.

166. Roesch, M. *Snort: Lightweight Intrusion Detection for Networks*. in *LISA*. 1999.

167. Goebel, J. and T. Holz. *Rishi: Identify bot contaminated hosts by irc nickname evaluation*. in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. 2007. Cambridge, MA.

168. Kugisaki, Y., et al. *Bot detection based on traffic analysis*. in *Intelligent Pervasive Computing, 2007. IPC. The 2007 International Conference on*. 2007. IEEE.

169. Wurzinger, P., et al., *Automatically generating models for botnet detection*, in *Computer Security–ESORICS 2009*. 2009, Springer. p. 232-249.

170. Aviv, A.J. and A. Haeberlen. *Challenges in Experimenting with Botnet Detection Systems*. in *CSET*. 2011.

171. Brezo, F., et al. *Challenges and limitations in current botnet detection*. in *Database and Expert Systems Applications (DEXA), 2011 22nd International Workshop on*. 2011. IEEE.

172. Lin, B., et al., *Botnet Emulation: Challenges and Techniques*, in *Emerging Technologies for Information Systems, Computing, and Management*. 2013, Springer. p. 897-908.

173. Barsamian, A.V., *Network characterization for botnet detection using statistical-behavioral methods*. 2009, Dartmouth College.

174. François, J., S. Wang, and T. Engel, *BotTrack: tracking botnets using NetFlow and PageRank*, in *NETWORKING 2011*. 2011, Springer. p. 1-14.

175. Gu, G., J. Zhang, and W. Lee, *BotSniffer: Detecting botnet command and control channels in network traffic.* 2008.

176. Houmansadr, A. and N. Borisov, *Botmosaic: Collaborative network watermark for botnet detection.* arXiv preprint arXiv:1203.1568, 2012.

177. Liu, D., et al. *A P2P-botnet detection model and algorithms based on network streams analysis*. in *Future Information Technology and Management Engineering (FITME), 2010 International Conference on*. 2010. IEEE.

178. Strayer, W.T., et al., *Botnet detection based on network behavior*, in *Botnet Detection*. 2008, Springer. p. 1-24.

179. Gu, G., et al. *BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection*. in *USENIX Security Symposium*. 2008.

180. Stoll, C., *The cuckoo's egg: tracking a spy through the maze of computer espionage*. 2005: Simon and Schuster.

181. Hongyan, Z., *Research and design of Botnet detection system*. 2011, Beijing University of Posts and Telecommunications.

182. Patil, E., *Analysis of rxbot.* 2009.

183. Paxson, V., *Bro: a system for detecting network intruders in real-time.* Computer networks, 1999. **31**(23): p. 2435-2463.

184. Al-Hammadi, Y. and U. Aickelin, *Detecting botnets through log correlation.* arXiv preprint arXiv:1001.2665, 2010.

185. Al-Hammadi, Y.A.A., *Behavioural correlation for malicious bot detection*. 2010, University of Nottingham.

186. Davis, N., *Botnet detection using correlated anomalies.* Technical University of Denmark Informatics and Mathematical Modelling, 2012.

187. Felix, J., C. Joseph, and A.A. Ghorbani, *Group behavior metrics for p2p botnet detection*, in *Information and Communications Security*. 2012, Springer. p. 93-104.

188. Thakur, M.R., *A Distributed and Cooperative Approach to Botnet Detection Using Gossip Protocol.* arXiv preprint arXiv:1207.0122, 2012.

189. Zeidanloo, H.R. and A.B.A. Manaf, *Botnet detection by monitoring similar communication patterns.* arXiv preprint arXiv:1004.1232, 2010.

190. Binkley, J.R. and S. Singh. *An algorithm for anomaly-based botnet detection*. in *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*. 2006.

191. Choi, H., et al. *Botnet detection by monitoring group activities in DNS traffic*. in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*. 2007. IEEE.

192. Gu, G., et al. *BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation*. in *Usenix Security*. 2007.

193. Gu, G., et al. *Active botnet probing to identify obscure command and control channels*. in *Computer Security Applications Conference, 2009. ACSAC'09. Annual*. 2009. IEEE.

194. Xu, K., et al. *Detecting infection onset with behavior-based policies*. in *Network and System Security (NSS), 2011 5th International Conference on*. 2011. IEEE.

195. Bilge, L., et al. *Disclosure: detecting botnet command and control servers through large-scale netflow analysis*. in *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012. ACM.

196. javvin. *Information, Computer and Network Security Terms Glossary and Dictionary*. 2013 [cited 2014 24 August]; Available from: http://www.javvin.com/networksecurity/CER.html.

197. Elovici, Y., et al., *Applying machine learning techniques for detection of malicious code in network traffic*, in *KI 2007: Advances in Artificial Intelligence*. 2007, Springer. p. 44-50.

198. Liu, L., et al., *Bottracer: Execution-based bot-like malware detection*, in *Information Security*. 2008, Springer. p. 97-113.

199. Livadas, C., et al. *Usilng machine learning technliques to identify botnet traffic*. in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*. 2006. IEEE.

200. Fawcett, T., *An introduction to ROC analysis.* Pattern recognition letters, 2006. **27**(8): p. 861-874.

201. Francois, J., et al. *BotCloud: detecting botnets using MapReduce*. in *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*. 2011. IEEE.

202. Wang, K. and S.J. Stolfo. *Anomalous payload-based network intrusion detection*. in *Recent Advances in Intrusion Detection*. 2004. Springer.

203. Anderson, J.P., *Computer security threat monitoring and surveillance*. 1980, Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

204. Chandola, V., A. Banerjee, and V. Kumar, *Anomaly detection: A survey.* ACM computing surveys (CSUR), 2009. **41**(3): p. 15.

205. Duda, R.O., P.E. Hart, and D.G. Stork, *Pattern classification*. 2012: John Wiley & Sons.

206. Pang-Ning, T., M. Steinbach, and V. Kumar. *Introduction to data mining*. in *Library of Congress*. 2006.

207. Danielsson, P.-E., *Euclidean distance mapping.* Computer Graphics and image processing, 1980. **14**(3): p. 227-248.

208. Segaran, T., *Programming collective intelligence: building smart web 2.0 applications*. 2007: " O'Reilly Media, Inc.".

209. Chandola, V., S. Boriah, and V. Kumar, *Understanding categorical similarity measures for outlier detection.* Technology Report, University of Minnesota, 2008.

210. Munson, B.R., D.F. Young, and T.H. Okiishi, *Fundamentals of fluid mechanics*. 1990: New York.

211. Breunig, M.M., et al. *LOF: identifying density-based local outliers*. in *ACM sigmod record*. 2000. ACM.

212. Tang, J., et al., *Enhancing effectiveness of outlier detections for low density patterns*, in *Advances in Knowledge Discovery and Data Mining*. 2002, Springer. p. 535-548.

213. Sun, P. and S. Chawla. *On local spatial outliers*. in *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. 2004. IEEE.

214. Yu, J.X., et al., *Finding centric local outliers in categorical/numerical spaces.* Knowledge and Information Systems, 2006. **9**(3): p. 309-338.

215. Ertöz, L., M. Steinbach, and V. Kumar, *Finding topics in collections of documents: A shared nearest neighbor approach*. 2004: Springer.

216. Guha, S., R. Rastogi, and K. Shim. *ROCK: A robust clustering algorithm for categorical attributes*. in *Data Engineering, 1999. Proceedings., 15th International Conference on*. 1999. IEEE.

217. Ester, M., et al. *A density-based algorithm for discovering clusters in large spatial databases with noise*. in *Kdd*. 1996.

218. Smith, R., et al., *Clustering approaches for anomaly based intrusion detection.* Proceedings of intelligent engineering systems through artificial neural networks, 2002: p. 579-584.

219. He, Z., X. Xu, and S. Deng, *Discovering cluster-based local outliers.* Pattern Recognition Letters, 2003. **24**(9): p. 1641-1650.

220. Motulsky, H., *Intuitive biostatistics: a nonmathematical guide to statistical thinking*. 2013: Oxford University Press, USA.

221. Simon, P., *Too Big to Ignore: The Business Case for Big Data*. 2013: John Wiley & Sons.

222. Mitchell, T.M., *The discipline of machine learning*. Vol. 17. 2006: Carnegie Mellon University, School of Computer Science, Machine Learning Department.

223. Bishop, C.M., *Pattern recognition and machine learning*. 2006: springer.

224. Kleinbaum, D.G. and M. Klein, *Analysis of Matched Data Using Logistic Regression*. 2010: Springer.

225. Goh, A., *Back-propagation neural networks for modeling complex systems.* Artificial Intelligence in Engineering, 1995. **9**(3): p. 143-151.

226. Ye, Y. and C.-C. Chiang. *A parallel apriori algorithm for frequent itemsets mining*. in *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*. 2006. IEEE.

227. Ahmad, A. and L. Dey, *A k-mean clustering algorithm for mixed numeric and categorical data.* Data & Knowledge Engineering, 2007. **63**(2): p. 503-527.

228. Vert, J.-P. and Y. Yamanishi. *Supervised graph inference*. in *Advances in Neural Information Processing Systems*. 2004.

229. Gómez-Chova, L., et al. *Semi-supervised cloud screening with Laplacian SVM*. in *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*. 2007. IEEE.

230. Watkins, C.J. and P. Dayan, *Q-learning.* Machine learning, 1992. **8**(3-4): p. 279-292.

231. Tesauro, G., *Temporal difference learning and TD-Gammon.* Communications of the ACM, 1995. **38**(3): p. 58-68.

232. Goodfellow, I., et al., *Deep learning*. Vol. 1. 2016: MIT press Cambridge.

233. Deng, L. and D. Yu, *Deep learning: methods and applications.* Foundations and Trends® in Signal Processing, 2014. **7**(3–4): p. 197-387.

234. Murphy, K.P., *Naive bayes classifiers.* University of British Columbia, 2006.

235. Quinlan, J.R., *Simplifying decision trees.* International journal of man-machine studies, 1987. **27**(3): p. 221-234.

236. Mitsa, T., *Temporal data mining.* 2010.

237. Shannon, C., *A mathematical theory of distribution.* Bell Syst Techn, 1948. **27**: p. 623.

238. Quinlan, J.R., *C4. 5: programs for machine learning*. 2014: Elsevier.

239. Rokach, L. and O. Maimon, *Data mining with decision trees: theory and applications*. 2014: World scientific.

240. Deng, H., G. Runger, and E. Tuv, *Bias of importance measures for multi-valued attributes and solutions.* Artificial neural networks and machine Learning–ICANN 2011, 2011: p. 293-300.

241. Bhargava, N., et al., *Decision tree analysis on j48 algorithm for data mining.* Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering, 2013. **3**(6).

242. Cunningham, P. and S.J. Delany, *k-Nearest neighbour classifiers.* Multiple Classifier Systems, 2007: p. 1-17.

243. Altman, N.S., *An introduction to kernel and nearest-neighbor nonparametric regression.* The American Statistician, 1992. **46**(3): p. 175-185.

244. Cover, T. and P. Hart, *Nearest neighbor pattern classification.* IEEE transactions on information theory, 1967. **13**(1): p. 21-27.

245. Kuramochi, M. and G. Karypis, *Gene classification using expression profiles: a feasibility study.* International Journal on Artificial Intelligence Tools, 2005. **14**(04): p. 641-660.

246. GRT. *KNN*. 2014 [cited 2017 June]; Available from: http://www.nickgillian.com/wiki/pmwiki.php/GRT/KNN.

247. Boland, M.V. and R.F. Murphy, *A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of HeLa cells.* Bioinformatics, 2001. **17**(12): p. 1213-1223.

248. Auer, P., H. Burgsteiner, and W. Maass, *A learning rule for very simple universal approximators consisting of a single layer of perceptrons.* Neural Networks, 2008. **21**(5): p. 786-795.

249. Tu, J.V., *Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes.* Journal of clinical epidemiology, 1996. **49**(11): p. 1225-1231.

250. Kotsiantis, S.B., I.D. Zaharakis, and P.E. Pintelas, *Machine learning: a review of classification and combining techniques.* Artificial Intelligence Review, 2006. **26**(3): p. 159-190.

251. Kotsiantis, S.B., I. Zaharakis, and P. Pintelas, *Supervised machine learning: A review of classification techniques*. 2007.

252. Hearst, M.A., et al., *Support vector machines.* Intelligent Systems and their Applications, IEEE, 1998. **13**(4): p. 18-28.

253. Hall, M., et al., *The WEKA data mining software: an update.* ACM SIGKDD explorations newsletter, 2009. **11**(1): p. 10-18.

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

255. Powers, D.M., *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.* 2011.

256. Hsu, L.M. and R. Field, *Interrater agreement measures: Comments on Kappan, Cohen's Kappa, Scott's π, and Aickin's α.* Understanding Statistics, 2003. **2**(3): p. 205-219.

257. Viner, J., *Cost curves and supply curves*. 1932: Springer.

258. Levinson, N., *The Wiener RMS (root mean square) error criterion in filter design and prediction.* 1947.

259. Iversen, G.R. and H. Norpoth, *Analysis of variance*. 1987: Sage.

260. Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018*. 2013.

261. Sanou, B., *The world in 2014 ICT Facts and figures*. 2014.

262. Mark, P. *IPHONE VIRUSES: IKEE.B WORM*. 2012 [cited 2015 24 August]; Available from: http://letsunlockiphone.guru/ios-viruses-iphone-ikee-b-worm/.

263. McNamee, K. *Malware Analysis Report-Trojan: AndroidOS/DroidDeluxe*. 2011 [cited 2015 24 August].

264. Bradley, T. *DroidDream becomes Android market nightmare*. PC World, Mar. 2011 [cited 2015 24 August]; Available from: http://www.pcworld.com/businesscenter/article/221247/droiddream_becomes_android_market_nightmare.html.

265. Perez, S., *More DroidDream Details Emerge: It was Building a Mobile Botnet*. 2011.

266. Xuxian, J. *DroidKungFu3*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu3/.

267. Xuxian, J. *DroidKungFu2*. 2011 [cited 2015 24 August]; Available from: http://www.cs.ncsu.edu/faculty/jiang/DroidKungFu2/.

268. Xuxian, J. *DroidKungFu: New Sophisticated Android Malware Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.cs.ncsu.edu/faculty/jiang/DroidKungFu.html.

269. Xuxian, J. *YZHCSMS: New Android SMS Trojan Found in Official and Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/YZHCSMS/.

270. Micro, T. *ANDROIDOS_PLANKTON.P*. 2011 [cited 2015 24 August]; Available from: http://about-threats.trendmicro.com/uk/malware/ANDROIDOS_PLANKTON.P.

271. Xuxian, J. *Plankton: New Stealthy Android Spyware Found in Official Android Market*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/Plankton/.

272. Symantec. *Android.Golddream*. 2011 [cited 2015 24 August]; Available from: http://www.symantec.com/security_response/writeup.jsp?docid=2011-070608-4139-99.

273. Xuxian, J. *GoldDream: New Android Malware Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/GoldDream/.

274. McAfee. *Virus Profile: Android/HippoSMS.A*. 2011 [cited 2015 24 August]; Available from: http://home.mcafee.com/virusinfo/virusprofile.aspx?key=544065#none.

275. Xuxian, J. *HippoSMS: New Android Malware Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/HippoSMS/.

276. F-secure. *On Android threats Spyware:Android/SndApps.A and Trojan:Android/SmsSpy.D*. 2011 [cited 2015 24 August]; Available from: http://www.f-secure.com/weblog/archives/00002202.html.

277. Xuxian, J. *SndApps: New Questionable Android Apps Found and Removed from the Official Android Market*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/SndApps/.

278. Xuxian, J. *NickiBot: New Android Spyware Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/NickiBot/.

279. FortiGuard. *Android/RogueSPPush.A*. 2011 [cited 2015 24 August]; Available from: http://www.fortiguard.com/search.php?action=detail_by_virus_name&data=Android/RogueSPPush.A!tr.

280. Xuxian, J. *RogueSPPush: New Android SMS-related Malware Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/RogueSPPush/.

281. Tim, W. *GingerMaster Is First Malware To Utilize A Root Exploit On Android 2.3*. 2011 [cited 2014 24 August]; Available from: http://www.darkreading.com/mobile/gingermaster-is-first-malware-to-utilize/231500422.

282. Xuxian, J. *GingerMaster: First Android Malware Utilizing a Root Exploit on Android 2.3 (Gingerbread)*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/GingerMaster/.

283. Xuxian, J. *DroidDeluxe: New Root-Capable Android Malware Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/DroidDeluxe/.

284. Xuxian, J. *AnserverBot: Highly Sophisticated Android Trojan Using Public, Encrypted Blog Entries for Command and Control (C&C)*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/AnserverBot.

285. Zhou, Y. and X. Jiang, *An analysis of the anserverbot trojan*. 2011, Technical report, NQ Mobile Security Research Center.

286. Xuxian, J. *DroidCoupon: New Root-Capable Android Malware Masquerades as Coupon App*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/DroidCoupon.

287. F-secure. *Trojan:Android/BeanBot.A*. 2011 [cited 2015 24 August]; Available from: http://www.f-secure.com/v-descs/trojan_android_beanbot.shtml.

288. Xuxian, J. *BeanBot: New Android SMS Trojan Found in Alternative Android Markets*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/BeanBot/.

289. Mike, L. *New Android Trojan Masquerades as Google Library, Taps Device Administration API*. 2011 [cited 2015 24 August]; Available from: http://www.securityweek.com/new-android-trojan-masquerades-google-library-taps-device-administration-api.

290. Xuxian, J. *DroidLive:New Android SMS Trojan Disguised as a Google Library*. 2011 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/DroidLive/.

291. Cathal, M. *Android.Bmaster: A Million-Dollar Mobile Botnet*. 2012 [cited 2015 24 August]; Available from: http://www.symantec.com/connect/blogs/androidbmaster-million-dollar-mobile-botnet.

292. Cathal, M. *Symantec Security Response Android.Bmaster*. 2012 [cited 2015 24 August]; Available from: http://www.symantec.com/security_response/writeup.jsp?docid=2012-020609-3003-99.

293. Awais, I. *"RootSmart" Malware Infecting 10,000+ Android Smartphones On Daily Basis, Turns Your Device Into A Zombie*. 2012 [cited 2015 24 August]; Available from: http://www.redmondpie.com/rootsmart-malware-infecting-10000-android-smartphones-on-daily-basis-turns-your-device-into-a-zombie/.

294. Xuxian, J. *RootSmart New Android Malware Utilizes the GingerBreak Root Exploit*. 2012 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/RootSmart/.

295. Xuxian, J. *PushBot A Push-Based App Delivery Model Identified in the Wild*. 2012 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/PushBot/.

296. Xuxian, J. *DKFBootKit New DroidKungFu Variant Moves Towards the First Android BootKit*. 2012 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/DKFBootKit/.

297. Bob, P. *A Closer Look at ANDROIDOS_TIGERBOT.EVL*. 2012 [cited 2015 24 August]; Available from: http://blog.trendmicro.com/trendlabs-security-intelligence/a-closer-look-at-androidos_tigerbot-evl/?utm_content=Google+Reader.

298. Xuxian, J. *TigerBot New Android Malware Identified in Alternative Android Markets*. 2012 [cited 2015 24 August]; Available from: http://www.csc.ncsu.edu/faculty/jiang/TigerBot/.

299. F-secure. *Trojan:Android/UpdtKiller.A*. 2012 [cited 2015 24 August]; Available from: http://www.f-secure.com/v-descs/trojan_android_updtkiller.shtml.

300. Xuxian, J. *UpdtKiller New Android Malware Removes Antivirus Software*. 2012 [cited 2015 24 August]; Available from: http://research.nq.com/?p=454/.

301. Denis. *New ZitMo for Android and Blackberry*. 2012 [cited 2014 24 August]; Available from: http://www.securelist.com/en/blog/208193760.

302. Michael, M. *Zitmo Trojan Variant Eurograbber Beats Two-Factor Authentication to Steal Millions*. 2012 [cited 2015 24 August]; Available from: http://threatpost.com/zitmo-trojan-variant-eurograbber-beats-two-factor-authentication-steal-millions-120612/.

303. Vural, I. and H.S. Venter, *Combating Mobile Spam through Botnet Detection using Artificial Immune Systems.* J. UCS, 2012. **18**(6): p. 750-774.

304. Reina, A., A. Fattori, and L. Cavallaro, *A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors.* EuroSec, April, 2013.

305. Maggi, F., A. Valdi, and S. Zanero. *AndroTotal: a flexible, scalable toolbox and service for testing mobile malware detectors*. in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*. 2013. ACM.

306. Team, B.R., *SandDroid: An APK Analysis Sandbox. Xi'an Jiaotong University*. 2014.

307. app360scan. *App360scan*. 2013 [cited 2015 24 August]; Available from: http://www.app360scan.com/.

308. Spreitzenbarth, M., et al. *MobileSandbox: looking deeper into android applications*. in *Proc. the 28th ACM Symposium on Applied Computing (SAC)*. 2013.

309. Bläsing, T., et al. *An android application sandbox system for suspicious software detection*. in *Malicious and unwanted software (MALWARE), 2010 5th international conference on*. 2010. IEEE.

310. Shabtai, A., et al., *"Andromaly": a behavioral malware detection framework for android devices.* Journal of Intelligent Information Systems, 2012. **38**(1): p. 161-190.

311. Shabtai, A., et al., *Using the KBTA method for inferring computer and network security alerts from time-stamped, raw system metrics.* Journal in computer virology, 2010. **6**(3): p. 239-259.

312. Abela, K.J., et al., *An automated malware detection system for android using behavior-based analysis AMDA.* International Journal of Cyber-Security and Digital Forensics (IJCSDF), 2013. **2**(2): p. 1-11.

313. Burguera, I., U. Zurutuza, and S. Nadjm-Tehrani. *Crowdroid: behavior-based malware detection system for android*. in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. 2011. ACM.

314. Mas' ud, M.Z., et al. *Analysis of features selection and machine learning classifier in android malware detection*. in *Information Science and Applications (ICISA), 2014 International Conference on*. 2014. IEEE.

315. Zhao, D., et al., *Botnet detection based on traffic behavior analysis and flow intervals.* Computers & Security, 2013. **39**: p. 2-16.

316. Amos, B., H. Turner, and J. White. *Applying machine learning classifiers to dynamic android malware detection at scale*. in *Wireless communications and mobile computing conference (iwcmc), 2013 9th international*. 2013. IEEE.

317. Aung, Z. and W. Zaw, *Permission-based android malware detection.* International Journal of Scientific and Technology Research, 2013. **2**(3): p. 228-234.

318. McNeil, P., et al., *SCREDENT: Scalable Real-time Anomalies Detection and Notification of Targeted Malware in Mobile Devices.* Procedia Computer Science, 2016. **83**: p. 1219-1225.

319. Dini, G., et al. *MADAM: a multi-level anomaly detector for android malware*. in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. 2012. Springer.

320. Feizollah, A., et al., *A review on feature selection in mobile malware detection.* Digital Investigation, 2015. **13**: p. 22-37.

321. Spanoudakis, G., C. Kloukinas, and K. Androutsopoulos. *Dynamic Verification and Control of Mobile Peer-to-Peer Systems*. in *Internet Monitoring and Protection, 2008. ICIMP'08. The Third International Conference on*. 2008. IEEE.

322. Google. *VpnService*. 2017 [cited 2018 14 April]; Available from: https://developer.android.com/reference/android/net/VpnService.html.

323. Taosoftware. *tpacketcapture*. 2015 [cited 2015 25 June]; Available from: https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapture.

324. Linux. *strace(1) - Linux man page*. 2015 [cited 2018 14 April]; Available from: https://linux.die.net/man/1/strace.

325. Firebase. *Cloud Firestore*. 2017 [cited 2018 14 April]; Available from: https://firebase.google.com/docs/firestore/.

326. Alejandro, B., *Real-Time Databases*, in *Encyclopedia of Database Technologies and Applications*, C.R. Laura, D. Jorge Horacio, and E.F. Viviana, Editors. 2005, IGI Global: Hershey, PA, USA. p. 524-529.

327. Mozilla. *Using server-sent events*. 2018 [cited 2018 14 April]; Available from: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events.

328. Weka. *WEKA: Vote Class Java Doc*. 2015 [cited 2018 14 April]; Available from: http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/Vote.html.

329. Weka. *WEKA: EvaluationClass Java Doc*. 2015 [cited 2018 14 April]; Available from: http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html.

330. Dierks, T., *The transport layer security (TLS) protocol version 1.2.* 2008.

331. Tuecke, S., et al., *Internet X. 509 public key infrastructure (PKI) proxy certificate profile*. 2004.

332. Zhou, X. and X. Tang. *Research and implementation of RSA algorithm for encryption and decryption*. in *Strategic Technology (IFOST), 2011 6th International Forum on*. 2011. IEEE.

333. Rivest, R., *The MD5 message-digest algorithm.* 1992.

334. Google. *Shrink Your Code and Resources*. 2017 [cited 2018 14 April]; Available from: https://developer.android.com/studio/build/shrink-code.html.

335. Google. *Android NDK*. 2017 [cited 2018 14 April]; Available from: https://developer.android.com/ndk/index.html.

336. Rish, I. *An empirical study of the naive Bayes classifier*. in *IJCAI 2001 workshop on empirical methods in artificial intelligence*. 2001. IBM New York.

337. Cortes, C. and V. Vapnik, *Support-vector networks.* Machine learning, 1995. **20**(3): p. 273-297.

338. Mark, B. *JNetPcap OpenSource User Guide*. 2014 [cited 2015 9 September]; Available from: http://jnetpcap.com/?q=userguide.

339. Zhou, Y. and X. Jiang. *Dissecting android malware: Characterization and evolution*. in *Security and Privacy (SP), 2012 IEEE Symposium on*. 2012. IEEE.

340. F-secure. *Trojan:Android/BaseBridge.A Threat description*. 2011 [cited 2017; Available from: https://www.f-secure.com/v-descs/trojan_android_basebridge.shtml.

341. F-secure. *Trojan:Android/DroidDream.A Threat description*. 2011 [cited 2017; Available from: https://www.f-secure.com/v-descs/trojan_android_droiddream_a.shtml.

342. maggarwal. *More variants of DroidKungFu 3 found*. 2011 [cited 2017; Available from: http://forums.juniper.net/t5/Security-Now/More-variants-of-DroidKungFu-3-found/ba-p/132757.

343. Lab, M. *Virus Profile: Android/DroidKungFu.D*. 2011 [cited 2017; Available from: https://origin-home.mcafee.com/virusinfo/VirusProfile.aspx?key=555498#none.

344. F-Secure, *MOBILE THREAT REPORT Q4 2011* 2012.

345. Katsuki, T. *Android.Adrd Versus Android.Geinimi*. 2011 [cited 2017; Available from: https://www.symantec.com/connect/blogs/androidadrd-versus-androidgeinimi.

346. Symantec. *Android.Golddream*. 2011 [cited 2017; Available from: https://www.symantec.com/security_response/writeup.jsp?docid=2011-070608-4139-99&tabid=2.

347. Microsoft. *Trojan: AndroidOS/Kmin.A*. 2011 [cited 2017; Available from: https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Trojan%3AAndroidOS%2FKmin.A.

348. Symantec. *Android.Pjapps*. 2011 [cited 2017; Available from: https://www.symantec.com/security_response/writeup.jsp?docid=2011-022303-3344-99&tabid=2.

349. f-secure. *Trojan:Android/Plankton Threat description*. 2011 [cited 2017; Available from: https://www.f-secure.com/v-descs/trojan_android_plankton.shtml.

350. Google. *Dashboards*. 2014 [cited 2015 24 August]; Available from: https://developer.android.com/about/dashboards/index.html.

351. Chappell, L.A. and G. Combs, *Wireshark 101: Essential Skills for Network Analysis*. 2013: Protocol Analysis Institute, Chappell University.

352. Wireshark. *The wireshark network analyzer 1.12.2*. 2014 [cited 2015 9 June]; Available from: https://www.wireshark.org/docs/man-pages/tshark.html.

353. Google. *Google IP address ranges*. 2014 [cited 2015 24 August]; Available from: https://support.google.com/a/answer/60764?hl=en.

354. Braun, L., G. Munz, and G. Carle. *Packet sampling for worm and botnet detection in TCP connections*. in *Network Operations and Management Symposium (NOMS), 2010 IEEE*. 2010. IEEE.

355. Refaeilzadeh, P., L. Tang, and H. Liu, *Cross-validation*, in *Encyclopedia of database systems*. 2009, Springer. p. 532-538.

356. Shiffler, R.E., *Maximum Z scores and outliers.* The American Statistician, 1988. **42**(1): p. 79-80.

Appendix A Key Implementation Code of Android Application

We list all the important key implementation code of the MBotCS on the Android devices. This code can be compiled by Gradle and run any devices with Android OS version > 4.0.

A.1 PCAP file parse

Use *readPcapToTemp* to read the PCAP file to the temporary file.

```
    private void readPcapToTemp() throws Exception {
        Log.d("readPcapToTemp", "Start to new alarm work");
        File testDatasetFile = new File(pcapPathParam);
        FileInputStream testPcapFile = null;
        try {
            testPcapFile = new FileInputStream(testDatasetFile);
        } catch (IOException e) {
            e.printStackTrace();
        }
        assert testPcapFile != null;
        long available = 0;
        Log.d("readPcapToTemp", "readCounter: "+readCounter);
        Log.d("readPcapToTemp", "locationByte: "+locationByte);
        if (locationByte == 0) {
            int m = testPcapFile.read(pcapHeaderTemp);
            if (m == 24) {
                locationByte += 24;
            }else{
                Log.d("ERROR:readPcapToTemp", "Not enough byte in PCAP
error, there are only: "+m);
            }
            available = testPcapFile.available();
            Log.d("readPcapToTemp", "available: "+available);
        } else {
            long actualSkip = testPcapFile.skip(locationByte);
            Log.d("readPcapToTemp", "actualSkip: "+actualSkip);
            available = testPcapFile.available();
            Log.d("readPcapToTemp", "available: "+available);
        }
        if (available !=0 ){
```

```
            Log.d("readPcapToTemp", "locationByte: "+locationByte);
            String tempPathFull;
            tempPathFull = pcapTempParam + "/pcap_temp_" +
String.valueOf(readCounter) + ".pcap";
            File tempPcap = new File(tempPathFull);
            FileOutputStream tempPcapFile = null;
            MyPcap pcap = null;
            try {
                pcap = PcapParser.unpackSimpleWithoutHeader(testPcapFile);
            } catch (IOException e) {
                e.printStackTrace();
            }
            tempPcapFile = new FileOutputStream(tempPcap);
            tempPcapFile.write(pcapHeaderTemp);
            assert pcap != null;
            List<PcapData> dataList = pcap.getData();
            byte[] bytesHeader;
            byte[] bytesContent;
            for (int i = 0; i < dataList.size(); i++) {
                bytesHeader = dataList.get(i).getInfoHeaderByte();
                bytesContent = dataList.get(i).getContent();
                locationByte += (bytesHeader.length + bytesContent.length);
                Log.d("readPcapToTemp", "locationByte: " + locationByte);
                tempPcapFile.write(bytesHeader);
                tempPcapFile.write(bytesContent);
            }
            tempPcapFile.close();
            pcapTcpToArff(tempPathFull);
            Log.d("readPcapToTemp", "Finish write File: "+locationByte);
            String arffPathFull;
            arffPathFull = pcapTempParam + "/arff_temp_" +
String.valueOf(readCounter) + ".arff";
            mlAnalyser(arffPathFull,ML_ALGORITH_BOX_HALF1);
            mlAnalyser(arffPathFull,ML_ALGORITH_J48);
            mlAnalyser(arffPathFull,ML_ALGORITH_KNN);
            readCounter++;
        }
    }
```

Read the PCAP file repeatedly:

```
    private void scanPcapRepeatedly() {
        scanPcap = new TimerTask() {
            @Override
            public void run() {
                Log.d("alarm", "The task start,  location:" + locationByte
+ "count:" + readCounter);
                try {
                    readPcapToTemp();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
```

```
        timer.scheduleAtFixedRate(scanPcap,0,UPDATE_INTERVAL);
    }
```

Use *pcapTcpToArff* to realise the PCAP file parser to convert the format of PCAP to ARFF format which can be recognised by the WEKA machine learning system.

```
private void pcapTcpToArff(String pcapOnePath) throws IOException {
        final StringBuilder errbuf = new StringBuilder();
        Log.d("pcapTcpToArff","Opening pcapOnePath for reading:
"+pcapOnePath);
        Pcap pcap = Pcap.openOffline(pcapOnePath, errbuf);
        if (pcap == null) {
            Log.d("pcapTcpToArff","Error while opening device for capture:
"+errbuf.toString());
            return;
        }
        JFlowMap superFlowMap = new JFlowMap();
        pcap.loop(Pcap.LOOP_INFINITE, superFlowMap, null);
        Iterator iterator = superFlowMap.entrySet().iterator();
        //Write the ARFF file to the disk
        String arffPathFull;
        arffPathFull = pcapTempParam + "/arff_temp_" +
String.valueOf(readCounter) + ".arff";
        File tempArff = new File(arffPathFull);
        FileOutputStream tempArffFile = null;
        tempArffFile = new FileOutputStream(tempArff);
        String arffRelation = "@relation monitor_traffic\n\n";
        String arffAttribute1 = "@attribute 'Frame duration' numeric\n";
        String arffAttribute2 = "@attribute 'TCP size' numeric\n";
        String arffAttribute3 = "@attribute 'Argument Count' numeric\n";
        String arffAttribute4 = "@attribute Lable {infect,normal}\n\n";
        String arffData = "@data\n";
        assert tempArffFile != null;
        tempArffFile.write(arffRelation.getBytes());
        tempArffFile.write(arffAttribute1.getBytes());
        tempArffFile.write(arffAttribute2.getBytes());
        tempArffFile.write(arffAttribute3.getBytes());
        tempArffFile.write(arffAttribute4.getBytes());
        tempArffFile.write(arffData.getBytes());
        double flowDeltaTime = 0.0;
        long flowLength = 0;
        long flowArgument = 0;
        long time_start_seconds;
        long time_start_nano;
        long time_end_seconds;
        long time_end_nano;
        double time_start = 0;
        double time_end;
        double time_delta = 0;
        String interval = ",";
        String classType = "normal";
        String lineBreak = "\n";
        Ip4 ip4 = new Ip4(); // Preallocat IP version 4 header
        Tcp tcp = new Tcp();
        Ethernet eth = new Ethernet();
```

```
        Http http = new Http();
        while (iterator.hasNext()) {
            Map.Entry entry = (Map.Entry) iterator.next();
            Object key = entry.getKey();
            JFlow oneFlow  = (JFlow) entry.getValue();
            List<JPacket> allPacket = oneFlow.getAll();
            time_start_seconds =0;
            time_start_nano = 0;
            time_end_seconds = 0;
            time_end_nano = 0;
            time_start = 0.0;
            time_end =0.0;
            time_delta = 0.0;
            for(int i=0; i<allPacket.size() ;i++){
                PcapPacket packet = (PcapPacket) allPacket.get(i);
                flowLength += allPacket.get(i).getTotalSize();
                JCaptureHeader captureHeader =packet.getCaptureHeader();
                if (allPacket.get(i).hasHeader(tcp) &&
allPacket.get(i).hasHeader(http)) {
                    allPacket.get(i).getHeader(eth);
                    allPacket.get(i).getHeader(tcp);
                    allPacket.get(i).getHeader(ip4);
                    if(tcp.destination() == 80) {
                        if(http.hasField(Http.Request.Accept) &&
http.fieldValue(Http.Request.Accept).contains("text/html")) {
                            String host =
http.fieldValue(Http.Request.Host);
                            String url = host +
http.fieldValue(Http.Request.RequestUrl);
                            Log.d("packet","url"+url);
                            int count = url.length() -
url.replaceAll("\\=","").length();
                            flowArgument += (long)count;
                        }
                    }
                }
                if(i == 0){
                    time_start_seconds = captureHeader.seconds();
                    time_start_nano = captureHeader.nanos();
                    String
timeStr=String.valueOf(time_start_seconds)+"."+String.valueOf(time_start_na
no);
                    Log.d("packet","time_start\n"+timeStr);
                    time_start =Double.parseDouble(timeStr);
                }
                if(i == allPacket.size()-1){
                    time_end_seconds = captureHeader.seconds();
                    time_end_nano = captureHeader.nanos();
                    String
timeStr=String.valueOf(time_end_seconds)+"."+String.valueOf(time_end_nano);
                    Log.d("packet","time_end\n"+timeStr);
                    time_end =Double.parseDouble(timeStr);
                    time_delta = time_end - time_start;
                }
            }
            flowDeltaTime = time_delta;
```

```
            tempArffFile.write(String.valueOf(flowDeltaTime).getBytes());
            tempArffFile.write(interval.getBytes());
            tempArffFile.write(String.valueOf(flowLength).getBytes());
            tempArffFile.write(interval.getBytes());
            tempArffFile.write(String.valueOf(flowArgument).getBytes());
            tempArffFile.write(interval.getBytes());
            tempArffFile.write(classType.getBytes());
            tempArffFile.write(lineBreak.getBytes());
            flowDeltaTime = 0.0;
            flowLength = 0;
            flowArgument = 0;
            time_start_seconds = 0;
            time_start_nano = 0;
            time_end_seconds = 0;
            time_end_nano = 0;
            time_start = 0.0;
            time_end = 0.0;
            time_delta = 0.0;
        }
        tempArffFile.close();
        pcap.close();
    }
```

A.2 Machine learning analyser

Use the WEKA java library for Android platform. The component is realised by the class ML Analyser:

```
public class MLAnalyser {
    NaiveBayes m_classifier_NB = new NaiveBayes();
    J48 m_classifier_j48 = new J48();
    MultilayerPerceptron m_classifier_MNN = new MultilayerPerceptron();
    IBk m_classifier_KNN = new IBk();
    SMO m_classifier_SVM = new SMO();
    int[] NB,J48,MNN,KNN,SVM;
    int[] AND,OR,HALF;
    int[] AND1,OR1,HALF1;
    private void counterSingleProcess(Double trainValue, Double testValue,
int[] counter){
        if(trainValue==0.0&&testValue==0.0){
            counter[0]++;
        }else if(trainValue==0.0&&testValue==1.0) {
            counter[1]++;
        }else if(trainValue==1.0&&testValue==0.0) {
            counter[2]++;
        }else {
            counter[3]++;
        }
    }
    private void counterAND(Double sumValue, Double testValue){
        if(sumValue==0.0&&testValue==0.0){
            AND[0]++;
```

```
        }else if(sumValue==0.0&&testValue==1.0) {
            AND[1]++;
        }else if(sumValue>0.0&&testValue==0.0) {
            AND[2]++;
        }else if(sumValue>0.0&&testValue==1.0) {
            AND[3]++;
        }
    }
    private void counterAND1(Double sumValueBetter, Double testValue){
        if(sumValueBetter==0.0&&testValue==0.0){
            AND1[0]++;
            HALF1[0]++;
        }else if(sumValueBetter==0.0&&testValue==1.0) {
            AND1[1]++;
            HALF1[1]++;
        }else if(sumValueBetter>0.0&&testValue==0.0) {
            AND1[2]++;
        }else if(sumValueBetter>0.0&&testValue==1.0){
            AND1[3]++;
        }
    }
    private void counterOR(Double sumValue, Double testValue){
        if(sumValue<5.0&&testValue==0.0){
            OR[0]++;
        }else if(sumValue<5.0&&testValue==1.0) {
            OR[1]++;
        }else if(sumValue==5.0&&testValue==0.0) {
            OR[2]++;
        }else if(sumValue==5.0&&testValue==1.0) {
            OR[3]++;
        }
    }
    private void counterOR1(Double sumValueBetter, Double testValue){
        if(sumValueBetter<2.0&&testValue==0.0){
            OR1[0]++;
        }else if(sumValueBetter<2.0&&testValue==1.0) {
            OR1[1]++;
        }else if(sumValueBetter==2.0&&testValue==0.0) {
            OR1[2]++;
            HALF1[2]++;
        }else if(sumValueBetter==2.0&&testValue==1.0){
            OR1[3]++;
            HALF1[3]++;
        }
    }
    private void counterHALF(Double sumValue, Double testValue){
        if(sumValue<3.0&&testValue==0.0){
            HALF[0]++;
        }else if(sumValue<3.0&&testValue==1.0) {
            HALF[1]++;
        }else if(sumValue>2.0&&testValue==0.0) {
            HALF[2]++;
        }else if(sumValue>2.0&&testValue==1.0) {
            HALF[3]++;
        }
    }
```

```
    private void counterHALF1(Double sumValueWorse, Double
sumValueBetter,Double testValue){
        if(sumValueBetter==1.0&&sumValueWorse<2.0&&testValue==0.0){
            HALF1[0]++;
        }else if(sumValueBetter==1.0&&sumValueWorse<2.0&&testValue==1.0) {
            HALF1[1]++;
        }else if(sumValueBetter==1.0&&sumValueWorse>1.0&&testValue==0.0) {
            HALF1[2]++;
        }else if(sumValueBetter==1.0&&sumValueWorse>1.0&&testValue==0.0){
            HALF1[3]++;
        }
    }
    private double computeRecall(int TP, int FN) {
        return (double)TP/(TP+FN);
    }
    private double computeFPR(int FP, int TN) {
        return (double)FP/(FP+TN);
    }
    private double computePrecision(int TP, int FP) {
        return (double)TP/(TP+FP);
    }
    public MLAnalyser(){
        m_classifier_MNN.setLearningRate(0.9);
        m_classifier_MNN.setHiddenLayers("t");
        m_classifier_MNN.setSeed(5);
        m_classifier_MNN.setReset(false);
        m_classifier_MNN.setMomentum(0.2);
        NB = new int[4];
        J48 = new int[4];
        MNN = new int[4];
        KNN = new int[4];
        SVM = new int[4];
        AND = new int[4];
        OR = new int[4];
        HALF = new int[4];
        AND1 = new int[4];
        OR1 = new int[4];
        HALF1 = new int[4];
        for (int i=0; i<4; i++){
            NB[i] = 0;
            J48[i] = 0;
            MNN[i] = 0;
            KNN[i] = 0;
            SVM[i] = 0;
            AND[i] = 0; // all 0 -> 0 sum of the classify value =0
            OR[i] = 0;  // any 0 -> 0 sum of the classify value <5
            HALF[i] = 0;// more than half 0-> sum of the classify value<3
            AND1[i] = 0; // all 0 -> 0 sum of the classify value =0
            OR1[i] = 0;  // any 0 -> 0 sum of the classify value <5
            HALF1[i] = 0;// more than half 0-> sum of the classify value<3
        }
    }
    public void trainClassifier(String trainDataset) throws Exception {
        String datasetFilename = trainDataset;
        File trainDatasetFile = new File(datasetFilename);
        ArffLoader datasetArff = new ArffLoader();
```

```
        datasetArff.setFile(trainDatasetFile);
        Instances trainData = datasetArff.getDataSet();
        trainData.setClassIndex(3);
        m_classifier_NB.buildClassifier(trainData);
        m_classifier_j48.buildClassifier(trainData);
        m_classifier_KNN.buildClassifier(trainData);
        m_classifier_SVM.buildClassifier(trainData);
    }
    public ArrayList<String> datasetAnalyser(String testDataset, int
algorithm) throws Exception {
        File testDatasetFile = new File(testDataset);
        ArffLoader datasetArff = new ArffLoader();
        datasetArff.setFile(testDatasetFile);
        Instances testData = datasetArff.getDataSet();
        testData.setClassIndex(3);
        double sum = testData.numInstances();
        ArrayList<String> result=new ArrayList<String>();
        double NBClassValue = 0;
        double j48ClassValue = 0;
        double MNNClassValue = 0;
        double KNNClassValue = 0;
        double SVMClassValue = 0;
        double BoxClassValue = 0;
        for(int i=0; i<sum; i++){
            if(algorithm == 0){
                NBClassValue =
m_classifier_NB.classifyInstance(testData.instance(i));
result.add(testData.instance(i).toString(0)+","+testData.instance(i).toStri
ng(1)+","+testData.instance(i).toString(2)+","+String.valueOf(NBClassValue)
);
                Log.d("MLAnalyser","NBClassValue"+NBClassValue);
            }else if (algorithm == 1){
                j48ClassValue =
m_classifier_j48.classifyInstance(testData.instance(i));

result.add(testData.instance(i).toString(0)+","+testData.instance(i).toStri
ng(1)+","+testData.instance(i).toString(2)+","+String.valueOf(j48ClassValue
));
                Log.d("MLAnalyser","j48ClassValue"+j48ClassValue);
            }else if (algorithm == 2){
                MNNClassValue = 0;
result.add(testData.instance(i).toString(0)+","+testData.instance(i).toStri
ng(1)+","+testData.instance(i).toString(2)+","+String.valueOf(MNNClassValue
));
                Log.d("MLAnalyser","MNNClassValue"+MNNClassValue);
            }else if (algorithm == 3){
                KNNClassValue =
m_classifier_KNN.classifyInstance(testData.instance(i));
result.add(testData.instance(i).toString(0)+","+testData.instance(i).toStri
ng(1)+","+testData.instance(i).toString(2)+","+String.valueOf(KNNClassValue
));
                Log.d("MLAnalyser","KNNClassValue"+KNNClassValue);
            }else if (algorithm == 4){
                SVMClassValue =
m_classifier_SVM.classifyInstance(testData.instance(i));
```

```
result.add(testData.instance(i).toString(0)+","+testData.instance(i).toStri
ng(1)+","+testData.instance(i).toString(2)+","+String.valueOf(SVMClassValue
));
                Log.d("MLAnalyser","SVMClassValue"+SVMClassValue);
            }else if (algorithm == 5){
                KNNClassValue =
m_classifier_KNN.classifyInstance(testData.instance(i));
                j48ClassValue =
m_classifier_j48.classifyInstance(testData.instance(i));
                if(KNNClassValue == j48ClassValue) {
                    BoxClassValue = j48ClassValue;
                }else{
                    MNNClassValue = 0;
                    NBClassValue =
m_classifier_NB.classifyInstance(testData.instance(i));
                    SVMClassValue =
m_classifier_SVM.classifyInstance(testData.instance(i));
                    double sumWorst =
MNNClassValue+NBClassValue+SVMClassValue;
                    if(sumWorst > 1.0){
                        BoxClassValue = 1.0;
                    }else{
                        BoxClassValue = 0.0;
                    }
                }
result.add(testData.instance(i).toString(0)+","+testData.instance(i).toStri
ng(1)+","+testData.instance(i).toString(2)+","+String.valueOf(BoxClassValue
));
            }
        }
        return  result;
    }
    public String[] datasetAnalyser(String fullDataset,int validationType,
int[] args ) throws Exception {
        String[] result = new String[15];
        File datasetFile = new File(fullDataset);
        try{
           File file = new File(fullDataset);
            FileInputStream fis = new FileInputStream(file);
            byte[] buffer = new byte[fis.available()];
            fis.read(buffer);
            fis.close();
            String res = EncodingUtils.getString(buffer, "UTF-8");
            Log.i("file"," file read ok: " + res);
        }catch(Exception ex){
            Log.i("file"," file read fail : ");
        }
        ArffLoader datasetArff = new ArffLoader();
        datasetArff.setFile(datasetFile);
        Instances fullData = datasetArff.getDataSet();
        if(validationType == 0){
            int seed = args[0];
            int folds = args[1];
            Random rand = new Random(seed);
```

```
            Instances randData = new Instances(fullData);
            randData.randomize(rand);
            for (int n = 0; n < folds; n++) {
                Instances train = randData.trainCV(folds, n);
                Instances test = randData.testCV(folds, n);
                train.setClassIndex(3);
                test.setClassIndex(3);
                double sum = test.numInstances(),
                        right = 0.0f;
                m_classifier_NB.buildClassifier(train);
                m_classifier_j48.buildClassifier(train);
                m_classifier_KNN.buildClassifier(train);
                m_classifier_SVM.buildClassifier(train);
                for(int i=0; i<sum; i++){
                    double testClassValue = test.instance(i).classValue();
                    double NBClassValue =
m_classifier_NB.classifyInstance(test.instance(i));
                    double j48ClassValue =
m_classifier_j48.classifyInstance(test.instance(i));
                    double KNNClassValue =
m_classifier_KNN.classifyInstance(test.instance(i));
                    double SVMClassValue =
m_classifier_SVM.classifyInstance(test.instance(i));
                    double MNNClassValue = 0.0;
                    double sumClassValue =
NBClassValue+j48ClassValue+MNNClassValue+KNNClassValue+SVMClassValue;
                    double sumClassValueBetter =
j48ClassValue+KNNClassValue;
                    double sumClassValueWorse =
NBClassValue+MNNClassValue+SVMClassValue;
                    counterSingleProcess(NBClassValue,testClassValue,NB);
                    counterSingleProcess(j48ClassValue,testClassValue,J48);
                    counterSingleProcess(MNNClassValue,testClassValue,MNN);
                    counterSingleProcess(KNNClassValue,testClassValue,KNN);
                    counterSingleProcess(SVMClassValue,testClassValue,SVM);
                    counterAND(sumClassValue,testClassValue);
                    counterOR(sumClassValue, testClassValue);
                    counterHALF(sumClassValue, testClassValue);
                    counterAND1(sumClassValueBetter, testClassValue);
                    counterOR1(sumClassValueBetter, testClassValue);
                    counterHALF1(sumClassValueBetter, sumClassValueWorse,
testClassValue);
                }
            }
        }
        result[0] = "Naive Bayesian
Recall:"+String.valueOf(computeRecall(NB[0],NB[2]));
        result[1] = "Naive Bayesian FPR:"+String.valueOf(computeFPR(NB[1],
NB[3]));
        result[2] = "Naive Bayesian
Precision:"+String.valueOf(computePrecision(NB[0], NB[1]));
        result[3] = "J48
Recall:"+String.valueOf(computeRecall(J48[0],J48[2]));
        result[4] = "J48 FPR:"+String.valueOf(computeFPR(J48[1],J48[3]));
        result[5] = "J48
Precision:"+String.valueOf(computePrecision(J48[0],J48[1]));
```

```
        result[6] = "MNN
Recall:"+String.valueOf(computeRecall(MNN[0],MNN[2]));
        result[7] = "MNN FPR:"+String.valueOf(computeFPR(MNN[1],MNN[3]));
        result[8] = "MNN
Precision:"+String.valueOf(computePrecision(MNN[0],MNN[1]));
        result[9] = "KNN
Recall:"+String.valueOf(computeRecall(KNN[0],KNN[2]));
        result[10] = "KNN FPR:"+String.valueOf(computeFPR(KNN[1],KNN[3]));
        result[11] = "KNN
Precision:"+String.valueOf(computePrecision(KNN[0],KNN[1]));
        result[12] = "SVM
Recall:"+String.valueOf(computeRecall(SVM[0],SVM[2]));
        result[13] = "SVM FPR:"+String.valueOf(computeFPR(SVM[1],SVM[3]));
        result[14] = "SVM
Precision:"+String.valueOf(computePrecision(SVM[0],SVM[1]));
        return result;
    }
    public ArrayList<String> datasetAnalyser(String trainDataset, String
testDataset) throws Exception {
        ArrayList<String> result=new ArrayList<String>();
        String trainDatasetFilename;
        trainDatasetFilename = trainDataset;
        File trainDatasetFile = new File(trainDatasetFilename);
        String testDatasetFilename;
        testDatasetFilename = testDataset;
        File testDatasetFile = new File(testDatasetFilename);
        ArffLoader trainDatasetArff = new ArffLoader();
        trainDatasetArff.setFile(trainDatasetFile);
        ArffLoader testDatasetArff = new ArffLoader();
        testDatasetArff.setFile(testDatasetFile);
        Instances trainData = trainDatasetArff.getDataSet();
        Instances testData = testDatasetArff.getDataSet();
        trainData.setClassIndex(3);
        testData.setClassIndex(3);
        m_classifier_NB.buildClassifier(trainData);
        m_classifier_j48.buildClassifier(trainData);
        m_classifier_KNN.buildClassifier(trainData);
        m_classifier_SVM.buildClassifier(trainData);
        double sum = testData.numInstances(),
                right = 0.0f;
        for(int i=0; i<sum; i++){
            double testClassValue = testData.instance(i).classValue();
            double NBClassValue =
m_classifier_NB.classifyInstance(testData.instance(i));
            double j48ClassValue =
m_classifier_j48.classifyInstance(testData.instance(i));
            double KNNClassValue =
m_classifier_KNN.classifyInstance(testData.instance(i));
            double SVMClassValue =
m_classifier_SVM.classifyInstance(testData.instance(i));
            double MNNClassValue = 0.0;
            String testClassStr;
            if(testClassValue==0.0){
                testClassStr = "infect";
            }else{
                testClassStr = "normal";
```

```
            }
            result.add(testData.instance(i).toString()+testClassStr);
            double sumClassValue =
NBClassValue+j48ClassValue+MNNClassValue+KNNClassValue+SVMClassValue;
            double sumClassValueBetter = j48ClassValue+KNNClassValue;
            double sumClassValueWorse =
NBClassValue+MNNClassValue+SVMClassValue;
            counterSingleProcess(NBClassValue,testClassValue,NB);
            counterSingleProcess(j48ClassValue,testClassValue,J48);
            counterSingleProcess(MNNClassValue,testClassValue,MNN);
            counterSingleProcess(KNNClassValue,testClassValue,KNN);
            counterSingleProcess(SVMClassValue,testClassValue,SVM);
            counterAND(sumClassValue,testClassValue);
            counterOR(sumClassValue, testClassValue);
            counterHALF(sumClassValue, testClassValue);
            counterAND1(sumClassValueBetter, testClassValue);
            counterOR1(sumClassValueBetter, testClassValue);
            counterHALF1(sumClassValueBetter, sumClassValueWorse,
testClassValue);
        }
        return result;
    }
}
```

A.3 Android intent service

In order to run the monitor service asynchronously, we make use of the intent service on Android platform to keep the user interface separate with the monitor service.

```
    public static void startActionScanPcap(Context context, String
pcapPath, String tempPath, String trainPath) {
        Intent intent = new Intent(context,
PcapMonitorIntentService.class);
        intent.setAction(ACTION_SCAN_PCAP);
        intent.putExtra(EXTRA_PARAM_PCAP_PATH, pcapPath);
        intent.putExtra(EXTRA_PARAM_TEMP_PATH, tempPath);
        intent.putExtra(EXTRA_PARAM_TRAIN_PATH, trainPath);
        context.startService(intent);
    }
    public static void startActionScanStop(Context context) {
        Intent intent = new Intent(context,
PcapMonitorIntentService.class);
        intent.setAction(ACTION_SCAN_STOP);
        context.startService(intent);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            final String action = intent.getAction();
            if (ACTION_SCAN_PCAP.equals(action)) {
```

```
                final String pcapPath =
intent.getStringExtra(EXTRA_PARAM_PCAP_PATH);
                final String tempPath =
intent.getStringExtra(EXTRA_PARAM_TEMP_PATH);
                final String trainPath =
intent.getStringExtra(EXTRA_PARAM_TRAIN_PATH);
                try {
                    handleActionScanPcap(pcapPath, tempPath, trainPath);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else if (ACTION_BAZ.equals(action)) {
                final String param1 = intent.getStringExtra(EXTRA_PARAM1);
                final String param2 = intent.getStringExtra(EXTRA_PARAM2);
                handleActionBaz(param1, param2);
            } else if (ACTION_SCAN_STOP.equals(action)) {
                handleActionScanStop();
            }
        }
    }

    private void handleActionScanStop() {
        scanPcap.cancel();
        stopSelf();
    }
    private void handleActionScanPcap(String pcapPath, String tempPath,
String trainPath) throws Exception {
        pcapPathParam = pcapPath;
        pcapTempParam = tempPath;
        trainDatasetParam = trainPath;

        mlAnalyser.trainClassifier(trainPath);
        scanPcapRepeatedly();
    }
```

A.4 User interface

The layout of the user interface is show as follows:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="1">
    <TextView
        android:layout_width="273dp"
        android:layout_height="wrap_content"
        android:text="@string/train_title"
        android:layout_gravity="left" />
    <Button
        android:id="@+id/train_dataset_select_btn"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/btn_select_file"
        android:onClick="selectFile"/>
    <EditText android:id="@+id/train_dataset"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/train_dataset_hint"
        android:focusable="false"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/test_title"
        android:layout_gravity="left" />
    <Button
        android:id="@+id/test_dataset_select_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/btn_select_file"
        android:onClick="selectFile"/>
    <EditText android:id="@+id/test_dataset"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/test_dataset_hint"
        android:focusable="false"/>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:weightSum="1">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/btn_validation"
            android:id="@+id/btn_validation"
            android:layout_weight="0.25"
            android:enabled="false"/>

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/btn_simulation"
            android:id="@+id/btn_simulation"
            android:layout_weight="0.25"
            android:enabled="false"/>

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/btn_monitor"
            android:id="@+id/btn_monitor"
            android:layout_weight="0.25" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/btn_monitor_stop"
        android:id="@+id/btn_monitor_stop"
        android:layout_weight="0.25" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/output_title"
        android:id="@+id/textView"
        android:layout_gravity="left"
        android:background="#ff66ffd0" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fillViewport="false"
        android:id="@+id/scrollView"
        android:layout_gravity="center_horizontal"
        android:background="#ffe8f2ff">
        <LinearLayout
            android:id="@+id/output_layout"
            android:layout_height="match_parent"
            android:layout_width="wrap_content"
            android:orientation="vertical"/>
    </ScrollView>
</LinearLayout>
</LinearLayout>
```

Appendix B System Call Monitor Bash Script

This script can run on the mobile device to capture system calls of specified applications. The script is based on the Bash which is supported on the Linux kernel on Android OS.

```
#!/bin/bash
#-----------------------------------------------------
# Read the packagesName.dat file to the variables
#-----------------------------------------------------
TAG_READ="Read PackagesName:"
#We store the information from the sub index 1
index=1
while read line
do
    #use two 1-dimention array to present the 2-dimentions array
    arrPackagesName[$index]="$line"
    #Intial the pid of the every monitored packages as 0
    arrPackagesNamePid[$index]=0
    #LOGGER
    log="$TAG_READ: arrPackagesName[$index]=${arrPackagesName[$index]} and
arrPackagesNamePid[$index]=${arrPackagesNamePid[$index]}"
    echo "$log" >> sh_log.log
    #LOGGER_END
    index=`expr $index + 1`
done < packagesName.dat
#TODO_FINISH: We need to check whether we read all the lines from the
files.
#ANS: We find that this is the problem of the file packagesName.dat add one
line and delete it, it work well.
sumPackagesName=`expr $index - 1`
#LOGGER
log="$TAG_READ read finish there are $sumPackagesName line in
packagesName.dat"
echo "$log" >> sh_log.log
#LOGGER_END
#-----------------------------------------------------
# Start to process the current processes and run strace
#-----------------------------------------------------
dirDATE=`date "+%Y-%m-%d_%H-%M-%S"`
mkdir $dirDATE
ps_index=1
TAG_PS="PS in While:"
index_ps_log=0
while true; do
```

```
    ps_index=`expr $ps_index + 1`
    start_time=`date "+%Y-%m-%d %H-%M-%S"`
    #echo "$TAG_PS While $ps_index start: $start_time" >> sh_log.log

    #Get the output of the `ps` command
    #TODO: BUG we need to generate the current process situation with one
time ps. or the two files will be not consistent.
    ps | awk '{print $2, $9}' > currentProcessInfo.tmp
    awk '{print $1}' currentProcessInfo.tmp > pidList.dat
    awk '{print $2}' currentProcessInfo.tmp > nameList_pre.dat

    # ps | awk '{print $2}' > pidList.dat
    # ps | awk '{print $9}' > nameList_pre.dat
    sed 's/[./:]/_/g' nameList_pre.dat > nameList.dat

    #Put the files into the variables
    #read the pidList.dat -> arrPidList
    index=0
    while read line
    do
        if [ $index -gt 0 ];then
            arrPidList[$index]="$line"
        fi
        index=`expr $index + 1`
    done < pidList.dat
    sumpid=`expr $index - 1`
    #read the nameList.dat -> arrNameList
    index=0
    while read line
    do
        if [ $index -gt 0 ];then
            arrNameList[$index]="$line"
        fi
        index=`expr $index + 1`
    done < nameList.dat
    sumname=`expr $index - 1`
    #TODO We need to check 1: all the process information has been stored
in the variables. 2: whether the sumname is equal to sumpid
    echo "$TAG_PS sumpid=$sumpid sumname=$sumname" >> sh_log.log
    #We start to visit all the processes to compare with the process in the
packagesName.dat
    index=0
    null="null"
    TAG_KEY="KEY:"
#    index_ps_log=0
    #think about the current process list is record from index=0 and so we
need to compare with the sum+1. the index=0 is the header.
    sumname1=`expr $sumname + 1`
    while [ $index -le $sumname1 ]
    do
        lineNo=$(sed -n -e /^${arrNameList[$index]}$/= packagesName.dat|sed
-n '1p')
        #TODO: we neen to make sure that we select right
#        echo "$TAG_KEY The lineNo is $lineNo" >> sh_log.log
        if [ ${lineNo:-"null"} = $null ];then
```

```
            echo "Cannot find this process in the packagesName.dat" >
useless.log
        else
            echo "$TAG_KEY Find the process in line $lineNo and name:
${arrNameList[$index]}" >> sh_log.log
            echo "$TAG_KEY previous pid: ${arrPackagesNamePid[$lineNo]} and
current pid: ${arrPidList[$index]}" >> sh_log.log
            if [ ${arrPidList[$index]} -ne
${arrPackagesNamePid[$lineNo]} ];then

filename=$dirDATE"/"${arrNameList[$index]}"_"${arrPidList[$index]}".txt"
                arrPackagesNamePid[$lineNo]=${arrPidList[$index]}
                strace_time=`date "+%Y-%m-%d %H-%M-%S"`
                # echo "CP: cp nameList.dat
ps/${arrPidList[$index]}_${index_ps_log}_nameList.dat"
                # echo "CP: cp pidList.dat
ps/${arrPidList[$index]}_${index_ps_log}_pid_list.dat"
                cp nameList.dat
ps/${index_ps_log}_${arrPidList[$index]}_nameList.dat
                cp pidList.dat
ps/${index_ps_log}_${arrPidList[$index]}_pid_list.dat
                index_ps_log=`expr $index_ps_log + 1`
                echo "$TAG_KEY ${strace_time}:Start to execute strace:
${arrPidList[$index]}, ${arrNameList[$index]}">> sh_log.log
                echo "$TAG_KEY strace -tt -T -p ${arrPidList[$index]} -o
$filename" >> sh_log.log
                strace -tt -T -p ${arrPidList[$index]} -o $filename &
            fi
        fi
        ##########################################################
        index=`expr $index + 1`
    done
    end_time=`date "+%Y-%m-%d_%H-%M-%S"`
    echo "$TAG_PS $ps_index\\t$start_time\\t$end_time" >> sh_log.log
done
```

Appendix C Key Implementation Code of Broker

The project is managed by Maven and use command: *mvn build* to download the dependencies and compile.

The Publisher implementation:

```
final class PublisherImpl implements Publisher {
  PublisherImpl(Builder builder) throws IOException {
    topic = builder.topic;

    maxBatchMessages = builder.maxBatchMessages;
    maxBatchBytes = builder.maxBatchBytes;
    maxBatchDuration = builder.maxBatchDuration;
    hasBatchingBytes = maxBatchBytes > 0;

    maxOutstandingMessages = builder.maxOutstandingMessages;
    maxOutstandingBytes = builder.maxOutstandingBytes;
    failOnFlowControlLimits = builder.failOnFlowControlLimits;
    this.flowController =
        new FlowController(maxOutstandingMessages, maxOutstandingBytes,
failOnFlowControlLimits);

    sendBatchDeadline = builder.sendBatchDeadline;

    requestTimeout = builder.requestTimeout;

    messagesBatch = new LinkedList<>();
    messagesBatchLock = new ReentrantLock();
    activeAlarm = new AtomicBoolean(false);
    int numCores = Math.max(1, Runtime.getRuntime().availableProcessors());
    executor =
        builder.executor.isPresent()
            ? builder.executor.get()
            : Executors.newScheduledThreadPool(
                numCores * DEFAULT_MIN_THREAD_POOL_SIZE,
                new ThreadFactoryBuilder()
                    .setDaemon(true)
                    .setNameFormat("cloud-pubsub-publisher-thread-%d")
                    .build());
    channels = new Channel[numCores];
```

```
    channelIndex = new AtomicLong(0);
    for (int i = 0; i < numCores; i++) {
      channels[i] =
          builder.channelBuilder.isPresent()
              ? builder.channelBuilder.get().build()
              : NettyChannelBuilder.forAddress(PUBSUB_API_ADDRESS, 443)
                  .negotiationType(NegotiationType.TLS)
                  .sslContext(GrpcSslContexts.forClient().ciphers(null).bui
ld())
                  .executor(executor)
                  .build();
    }
    credentials =
        MoreCallCredentials.from(
            builder.userCredentials.isPresent()
                ? builder.userCredentials.get()
                : GoogleCredentials.getApplicationDefault()
                    .createScoped(Collections.singletonList(PUBSUB_API_SCOP
E)));
    shutdown = new AtomicBoolean(false);
    messagesWaiter = new MessagesWaiter();
  }

  @Override
  public ListenableFuture<String> publish(PubsubMessage message) {
    if (shutdown.get()) {
      throw new IllegalStateException("Cannot publish on a shut-down
publisher.");
    }

    final int messageSize = message.getSerializedSize();
    try {
      flowController.reserve(1, messageSize);
    } catch (CloudPubsubFlowControlException e) {
      return Futures.immediateFailedFuture(e);
    }
    OutstandingBatch batchToSend = null;
    SettableFuture<String> publishResult = SettableFuture.create();
    final OutstandingPublish outstandingPublish = new
OutstandingPublish(publishResult, message);
    messagesBatchLock.lock();
    try {
      if (!messagesBatch.isEmpty()
          && hasBatchingBytes
          && batchedBytes + messageSize >= getMaxBatchBytes()) {
        batchToSend = new OutstandingBatch(messagesBatch, batchedBytes);
        messagesBatch = new LinkedList<>();
        batchedBytes = 0;
      }
      if (!hasBatchingBytes || messageSize < getMaxBatchBytes()) {
        batchedBytes += messageSize;
        messagesBatch.add(outstandingPublish);
      if (messagesBatch.size() == getMaxBatchMessages()) {
          batchToSend = new OutstandingBatch(messagesBatch, batchedBytes);
          messagesBatch = new LinkedList<>();
          batchedBytes = 0;
```

```
        }
      }
      if (!messagesBatch.isEmpty()) {
        setupDurationBasedPublishAlarm();
      } else if (currentAlarmFuture != null) {
        logger.debug("Cancelling alarm");
        if (activeAlarm.getAndSet(false)) {
          currentAlarmFuture.cancel(false);
        }
      }
    } finally {
      messagesBatchLock.unlock();
    }

    messagesWaiter.incrementPendingMessages(1);

    if (batchToSend != null) {
      logger.debug("Scheduling a batch for immediate sending.");
      final OutstandingBatch finalBatchToSend = batchToSend;
      executor.execute(
          new Runnable() {
            @Override
            public void run() {
              publishOutstandingBatch(finalBatchToSend);
            }
          });
    }
    if (hasBatchingBytes && messageSize >= getMaxBatchBytes()) {
      logger.debug("Message exceeds the max batch bytes, scheduling it for
immediate send.");
      executor.execute(
          new Runnable() {
            @Override
            public void run() {
              publishOutstandingBatch(
                  new
OutstandingBatch(ImmutableList.of(outstandingPublish), messageSize));
            }
          });
    }
    return publishResult;
  }
          @Override
          public void onFailure(Throwable t) {
            long nextBackoffDelay =
computeNextBackoffDelayMs(outstandingBatch);

            if (!isRetryable(t)
                || System.currentTimeMillis() + nextBackoffDelay
                    > outstandingBatch.creationTime
                        + PublisherImpl.this.sendBatchDeadline.getMillis())
{
              try {
                for (OutstandingPublish outstandingPublish :
                    outstandingBatch.outstandingPublishes) {
                  outstandingPublish.publishResult.setException(t);
```

```
                }
              } finally {
                messagesWaiter.incrementPendingMessages(-
outstandingBatch.size());
              }              return;
            }

            executor.schedule(
                new Runnable() {
                  @Override
                  public void run() {
                    publishOutstandingBatch(outstandingBatch);
                  }
                },
                nextBackoffDelay,
                TimeUnit.MILLISECONDS);
          }
        });
  }
```

The Subscriber implementation:

```
public class SubscriberImpl extends AbstractService implements Subscriber {
  public SubscriberImpl(SubscriberImpl.Builder builder) throws IOException
{
    receiver = builder.receiver;
    maxOutstandingBytes = builder.maxOutstandingBytes;
    maxOutstandingMessages = builder.maxOutstandingMessages;
    subscription = builder.subscription;
    ackExpirationPadding = builder.ackExpirationPadding;
    streamAckDeadlineSeconds =
        Math.max(
            INITIAL_ACK_DEADLINE_SECONDS,
            Ints.saturatedCast(ackExpirationPadding.getStandardSeconds()));

    flowController =
        new FlowController(builder.maxOutstandingBytes,
builder.maxOutstandingBytes, false);

    numChannels = Math.max(1, Runtime.getRuntime().availableProcessors()) *
CHANNELS_PER_CORE;
    executor =
        builder.executor.isPresent()
            ? builder.executor.get()
            : Executors.newScheduledThreadPool(
                numChannels * THREADS_PER_CHANNEL,
                new ThreadFactoryBuilder()
                    .setDaemon(true)
                    .setNameFormat("cloud-pubsub-subscriber-thread-%d")
                    .build());

    channelBuilder =
        builder.channelBuilder.isPresent()
            ? builder.channelBuilder.get()
            : NettyChannelBuilder.forAddress(PUBSUB_API_ADDRESS, 443)
                .maxMessageSize(MAX_INBOUND_MESSAGE_SIZE)
```

```
                .flowControlWindow(5000000) // 2.5 MB
                .negotiationType(NegotiationType.TLS)
                .sslContext(GrpcSslContexts.forClient().ciphers(null).build
())
                .executor(executor);

    credentials =
        builder.credentials.isPresent()
            ? builder.credentials.get()
            : GoogleCredentials.getApplicationDefault()
                .createScoped(Collections.singletonList(PUBSUB_API_SCOPE));

    streamingSubscriberConnections = new
ArrayList<StreamingSubscriberConnection>(numChannels);
    pollingSubscriberConnections = new
ArrayList<PollingSubscriberConnection>(numChannels);
  }

  @Override
  protected void doStart() {
    logger.debug("Starting subscriber group.");
    startStreamingConnections();
    notifyStarted();
  }

  @Override
  protected void doStop() {
    stopAllStreamingConnections();
    stopAllPollingConnections();
    notifyStopped();
  }

  private void startStreamingConnections() {
    synchronized (streamingSubscriberConnections) {
      for (int i = 0; i < numChannels; i++) {
        streamingSubscriberConnections.add(
            new StreamingSubscriberConnection(
                subscription,
                credentials,
                receiver,
                ackExpirationPadding,
                streamAckDeadlineSeconds,
                ackLatencyDistribution,
                channelBuilder.build(),
                flowController,
                executor));
      }
      startConnections(
          streamingSubscriberConnections,
          new Listener() {
            @Override
            public void failed(State from, Throwable failure) {
              stopAllStreamingConnections();
              if (failure instanceof StatusRuntimeException
                  && ((StatusRuntimeException)
failure).getStatus().getCode()
```

```
                    == Status.Code.UNIMPLEMENTED) {
                logger.info("Unable to open streaming connections, falling
back to polling.");
                startPollingConnections();
                return;
              }
              notifyFailed(failure);
            }
          });
    }

    ackDeadlineUpdater =
        executor.scheduleAtFixedRate(
            new Runnable() {
              @Override
              public void run() {
                long ackLatency =

ackLatencyDistribution.getNthPercentile(PERCENTILE_FOR_ACK_DEADLINE_UPDATES
);
                if (ackLatency > 0) {
                  int possibleStreamAckDeadlineSeconds =
                      Math.max(
                          MIN_ACK_DEADLINE_SECONDS,
                          Ints.saturatedCast(
                              Math.max(ackLatency,
ackExpirationPadding.getStandardSeconds())));
                  if (streamAckDeadlineSeconds !=
possibleStreamAckDeadlineSeconds) {
                    streamAckDeadlineSeconds =
possibleStreamAckDeadlineSeconds;
                    logger.debug(
                        "Updating stream deadline to {} seconds.",
streamAckDeadlineSeconds);
                    for (StreamingSubscriberConnection
subscriberConnection :
                        streamingSubscriberConnections) {

subscriberConnection.updateStreamAckDeadline(streamAckDeadlineSeconds);
                    }
                  }
                }
              }
            },
            ACK_DEADLINE_UPDATE_PERIOD.getMillis(),
            ACK_DEADLINE_UPDATE_PERIOD.getMillis(),
            TimeUnit.MILLISECONDS);
  }

  private void stopAllStreamingConnections() {
    stopConnections(streamingSubscriberConnections);
    ackDeadlineUpdater.cancel(true);
  }

    stopConnections(pollingSubscriberConnections);
  }
```

```
  private void startConnections(
      List<? extends AbstractSubscriberConnection> connections,
      final Listener connectionsListener) {
    final CountDownLatch subscribersStarting = new
CountDownLatch(numChannels);
    for (final AbstractSubscriberConnection subscriber : connections) {
      executor.submit(
          new Runnable() {
            @Override
            public void run() {
              subscriber.startAsync().awaitRunning();
              subscribersStarting.countDown();
              subscriber.addListener(connectionsListener, executor);
            }
          });
    }
    try {
      subscribersStarting.await();
    } catch (InterruptedException e) {
      throw new RuntimeException(e);
    }
  }

  private void stopConnections(List<? extends AbstractSubscriberConnection>
connections) {
    ArrayList<AbstractSubscriberConnection> liveConnections;
    synchronized (connections) {
      liveConnections = new
ArrayList<AbstractSubscriberConnection>(connections);
      connections.clear();
    }
    final CountDownLatch connectionsStopping = new
CountDownLatch(liveConnections.size());
    for (final AbstractSubscriberConnection subscriberConnection :
liveConnections) {
      executor.submit(
          new Runnable() {
            @Override
            public void run() {
              try {
                subscriberConnection.stopAsync().awaitTerminated();
              } catch (IllegalStateException ignored) {
              }
              connectionsStopping.countDown();
            }
          });
    }
    try {
      connectionsStopping.await();
    } catch (InterruptedException e) {
      throw new RuntimeException(e);
    }
  }
}
```

Appendix D **Key Implementation Code of Analyser**

The project is managed by Maven and use command: *mvn build* to download the dependencies and compile.

```
public class WekaProcess {
    static Logger logger = Logger.getLogger(WekaProcess.class);
    public static void  main(String[] args) throws Exception{

        NaiveBayes classifierNB = new NaiveBayes();
        J48 classifierJ48 = new J48();
        MultilayerPerceptron classifierMNN = new MultilayerPerceptron();
        IBk classifierKNN = new IBk();
        SMO classifierSVM = new SMO();
        //Configure the classifier, others options is keeping default
        classifierMNN.setLearningRate(0.9);
        classifierMNN.setHiddenLayers("t");
        classifierMNN.setSeed(5);
        classifierMNN.setReset(false);
        classifierMNN.setMomentum(0.2);
        classifierKNN.setKNN(2);
        //mix
        //Load the mix csv file
        for (int aTimeIntervalInSecond : Config.timeIntervalInSecond) {
            // get the specific file by the time interval by using the rule
            CSVLoader mixLoad = new CSVLoader();
            try {
                mixLoad.setSource(
                        new
File(FileHelper.FilePathGen("mix",aTimeIntervalInSecond,0)));
            } catch (IOException e) {
                e.printStackTrace();
            }

            Instances mixData = mixLoad.getDataSet();

            mixData.setClassIndex(mixData.numAttributes() - 1);
            EvaluationEnhance eval = new EvaluationEnhance(mixData);
            List exportData = new ArrayList<Map<String, String>>();
            eval.crossValidateModel(classifierJ48, mixData, 10, new
Random(1));
            exportData = eval.exportList("J48", exportData);
            EvaluationEnhance eval2 = new EvaluationEnhance(mixData);
            eval2.crossValidateModel(classifierKNN, mixData, 10, new
Random(1));
```

```
            exportData = eval2.exportList("KNN", exportData);

            EvaluationEnhance eval3 = new EvaluationEnhance(mixData);
            eval3.crossValidateModel(classifierNB, mixData, 10, new
Random(1));
            exportData = eval3.exportList("NB", exportData);

            EvaluationEnhance eval4 = new EvaluationEnhance(mixData);
            eval4.crossValidateModel(classifierSVM, mixData, 10, new
Random(1));
            exportData = eval4.exportList("SVM", exportData);

            EvaluationEnhance eval5 = new EvaluationEnhance(mixData);
            eval5.crossValidateModel(classifierMNN, mixData, 10, new
Random(1));
            exportData = eval5.exportList("MNN", exportData);
            LinkedHashMap map = new LinkedHashMap();
            for (int i=1;i<16;i++) {
                map.put(String.valueOf(i), String.valueOf(i));
            }
            CsvUtil.createCSVFile(exportData, map,
FileDirectory.WORK_PATH_WIN,
                    String.valueOf(aTimeIntervalInSecond) +
"analysis.csv");

        }
    }
}
```

Appendix E Normal Application Actions

Chrome: Open settings and login/logout with valid credentials; Open new normal tab TAB1, type and go to URL; add it to favorites; Open a new normal tab TAB2 and search for city by Google; Close TAB1 and TAB2; Open a new incognito tab INTAB1 and type and access a URL; Add URL to the Favorites; Open a new incognito tab INTAB2 and search for URL by Google; Close INTAB1 and INTAB2; Open favourites and select last added favourite; Open history and select the initially visited URL
Gmail: Open the main interface and refresh; Send an email; Receive email; Open last email and Mark it; Move an email to the Social Folder; Delete an email from Primary; Receive email with attachment; Download the attachment
Maps: Get location in the main interface; Open explore around you; Search British Museum and find the direction to it from current location; Close application; Open settings and open map history; Open British Museum from the list; Switch the setting from Traffic→Public Transit→Bicycling→satellite→Terrain; Search British Library and open the description (photos and reviews), then star it; Save a small offline map and delete it; magnify the map and save it
Facebook: Login and logout with valid account; Search for someone and add her as a friend; Open main interface and refresh; Post status; Post Photo; Check in
YouTube: Search for British Museum and get a list of videos; Open a video and add it to watch later; View a full video and minimise it to the right bottom corner; Open history and a video; Open best of YouTube, open music to view a video; Upload a video and delete it.

Messenger: Search for a user and add her as friend; Send message to friend; Receive message from friend; Open main interface and refresh; Open news and the first article in it; Add it to the bookmarks and go back
Twitter: Open my library and add more to select featured; add Wall Street Journal and then remove it
PlayNewStand: Search for a user and add him/her as a friend; Open main interface and load a refresh; Post status; Post Photo; Send/Receive message
Flipboard: Open and refresh; Open a news item and mark it as Liked; Open classification; Open news to select an item from it; Add comment to news.
Feedly: Open main interface and refresh; Add content to search Tech and add Engadget to content; Refresh and open news items from Engadget and then mark is as Liked; Remove Engadget from content; Open Explore and add news item.
Skype: Search for contacts; Send message to a contact; Receive message from contact; Call a contact; Receive a call from a contact.
MailDroid: Open the main interface and refresh; Send/receive email; Open/flag/delete email; Receive email with attachment; Download attachment.

Appendix F Tables of Experiments Result

Table F-1 - Results of experiment I of network traffic

Validation / Algorithms	90%-10% 10-fold cross-validation		Recall	FPR	Precision	50%-50% 2-fold cross-validation		Recall	FPR	Precision	10%-90% split dataset		Recall	FPR	Precision
Naive Bayesian	Packet	Infect	0.031	0.006	0.760	Packet	Infect	0.030	0.006	0.750	Packet	Infect	0.609	0.006	0.994
		Normal	0.994	0.969	0.609		Normal	0.994	0.970	0.608		Normal	0.994	0.391	0.609
	Stream	Infect	0.064	0.028	0.788	Stream	Infect	0.096	0.043	0.781	Stream	Infect	0.938	0.827	0.645
		Normal	0.972	0.936	0.391		Normal	0.957	0.904	0.395		Normal	0.173	0.062	0.635
J48 Tree	Packet	Infect	0.335	0.066	0.769	Packet	Infect	0.344	0.077	0.746	Packet	Infect	0.198	0.041	0.763
		Normal	0.934	0.665	0.680		Normal	0.923	0.656	0.681		Normal	0.959	0.802	0.644
	Stream	Infect	0.908	0.276	0.842	Stream	Infect	0.870	0.307	0.821	Stream	Infect	0.881	0.398	0.780
		Normal	0.724	0.092	0.829		Normal	0.693	0.130	0.766		Normal	0.602	0.119	0.760
MNN	Packet	Infect	0.183	0.161	0.428	Packet	Infect	0.455	0.404	0.426	Packet	Infect	0.909	0.807	0.427
		Normal	0.839	0.817	0.609		Normal	0.596	0.545	0.624		Normal	0.193	0.091	0.761
	Stream	Infect	0.877	0.752	0.654	Stream	Infect	0.946	0.826	0.650	Stream	Infect	0.982	0.913	0.633
		Normal	0.248	0.123	0.556		Normal	0.174	0.054	0.667		Normal	0.087	0.018	0.750
KNN	Packet	Infect	0.455	0.154	0.660	Packet	Infect	0.461	0.177	0.632	Packet	Infect	0.479	0.210	0.602
		Normal	0.846	0.545	0.702		Normal	0.823	0.539	0.698		Normal	0.790	0.521	0.696
	Stream	Infect	0.893	0.216	0.870	Stream	Infect	0.887	0.248	0.853	Stream	Infect	0.800	0.230	0.848
		Normal	0.784	0.107	0.818		Normal	0.752	0.113	0.804		Normal	0.770	0.200	0.706
SVM	Packet	Infect	0.012	0.003	0.726	Packet	Infect	0.012	0.003	0.726	Packet	Infect	0.013	0.003	0.733
		Normal	0.997	0.988	0.605		Normal	0.997	0.988	0.605		Normal	0.997	0.987	0.604
	Stream	Infect	0.998	0.966	0.626	Stream	Infect	0.997	0.969	0.625	Stream	Infect	0.999	0.983	0.620
		Normal	0.034	0.002	0.917		Normal	0.031	0.003	0.870		Normal	0.017	0.001	0.909
ML-BOX(AND)	Stream	Infect	0.053	0.011	0.887	Stream	Infect	0.036	0.008	0.884	Stream	Infect	0.679	0.150	0.884
		Normal	0.946	0.487	0.751		Normal	0.992	0.964	0.389		Normal	0.850	0.321	0.612
ML-BOX(OR)	Stream	Infect	0.996	0.969	0.625	Stream	Infect	0.996	0.958	0.627	Stream	Infect	0.998	0.954	0.637
		Normal	0.053	0.011	0.887		Normal	0.042	0.004	0.871		Normal	0.046	0.002	0.929
ML-BOX(HALF)	Stream	Infect	0.941	0.349	0.813	Stream	Infect	0.936	0.340	0.817	Stream	Infect	0.945	0.670	0.703
		Normal	0.941	0.349	0.813		Normal	0.660	0.064	0.864		Normal	0.330	0.055	0.782
ML-BOX+(AND)	Stream	Infect	0.845	0.129	0.914	Stream	Infect	0.835	0.148	0.902	Stream	Infect	0.759	0.173	0.880
		Normal	0.947	0.382	0.801		Normal	0.852	0.165	0.761		Normal	0.827	0.241	0.672
ML-BOX+(OR)	Stream	Infect	0.947	0.382	0.801	Stream	Infect	0.945	0.410	0.789	Stream	Infect	0.891	0.460	0.764
		Normal	0.845	0.129	0.914		Normal	0.590	0.055	0.870		Normal	0.540	0.109	0.746
ML-BOX+(HALF)	Stream	Infect	0.939	0.359	0.814	Stream	Infect	0.847	0.205	0.900	Stream	Infect	0.856	0.426	0.782
		Normal	0.939	0.165	0.922		Normal	0.795	0.153	0.704		Normal	0.574	0.144	0.691

Table F-2 - Results of experiment II of network traffic

Malware Family	1 (187 infect streams)			2 (181 infect streams)			3 (7 infect streams)		
Measures	Recall	FPR	Precision	Recall	FPR	Precision	Recall	FPR	Precision
Naive Bayesian	0.059	0.03	0.846	0.05	0.03	0.818	0.571	0.03	0.667
J48 Tree	0.636	0.152	0.922	0.403	0.182	0.859	0.714	0.242	0.238
MNN	0.695	0.53	0.788	0.994	0.864	0.759	1	0.864	0.109
KNN	0.642	0.288	0.863	0.768	0.242	0.897	0.857	0.303	0.231
SVM	1	0.955	0.748	1	0.955	0.742	1	0.97	0.099
ML-BOX(AND)	0.032	0	1	0.028	0.015	0.833	0.571	0.015	0.8
ML-BOX(OR)	1	0.955	0.748	1	0.955	0.742	1	0.97	0.099
ML-BOX(HALF)	0.658	0.227	0.891	0.796	0.318	0.873	0.857	0.424	0.176
ML-BOX+(AND)	0.556	0.136	0.92	0.376	0.121	0.895	0.714	0.136	0.357
ML-BOX+(OR)	0.722	0.303	0.871	0.796	0.303	0.878	0.857	0.409	0.182
ML-BOX+(HALF)	0.658	0.227	0.891	0.796	0.303	0.878	0.857	0.409	0.182
Malware Family	**4 (205 infect streams)**			**5 (55 infect streams)**			**6 (10 infect streams)**		
Measures	**Recall**	**FPR**	**Precision**	**Recall**	**FPR**	**Precision**	**Recall**	**FPR**	**Precision**
Naive Bayesian	0.054	0.03	0.846	0.018	0.03	0.333	0.2	0.03	0.5
J48 Tree	0.639	0.182	0.916	0.145	0.227	0.348	0.6	0.258	0.261
MNN	0.917	0.788	0.783	0.745	0.848	0.423	0.9	0.848	0.138
KNN	0.561	0.227	0.885	0.418	0.212	0.622	0.8	0.303	0.286
SVM	0.995	0.955	0.764	1	0.955	0.466	0.9	0.955	0.125
ML-BOX(AND)	0.01	0	1	0	0.015	0	0.1	0.015	0.5
ML-BOX(OR)	0.995	0.955	0.764	1	0.955	0.466	0.9	0.955	0.125
ML-BOX(HALF)	0.795	0.288	0.896	0.364	0.348	0.465	0.8	0.409	0.229
ML-BOX+(AND)	0.4	0.121	0.911	0.145	0.091	0.571	0.6	0.152	0.375
ML-BOX+(OR)	0.8	0.288	0.896	0.418	0.348	0.5	0.8	0.409	0.229
ML-BOX+(HALF)	0.785	0.288	0.894	0.364	0.348	0.465	0.8	0.409	0.229
Malware Family	**7 (38 infect streams)**			**9 (117 infect streams)**			**10 (243 infect streams)**		
Measures	**Recall**	**FPR**	**Precision**	**Recall**	**FPR**	**Precision**	**Recall**	**FPR**	**Precision**
Naive Bayesian	0.026	0.03	0.333	0.06	0.03	0.778	0.004	0.03	0.333
J48 Tree	0.763	0.242	0.644	0.607	0.152	0.877	0.593	0.197	0.917
MNN	0.974	0.727	0.435	0.752	0.727	0.647	1	0.848	0.813
KNN	0.447	0.303	0.459	0.667	0.303	0.796	0.086	0.303	0.512
SVM	1	0.955	0.376	1	0.955	0.65	1	0.955	0.794
ML-BOX(AND)	0.026	0	1	0.026	0.015	0.75	0	0	0
ML-BOX(OR)	1	0.955	0.376	1	0.955	0.65	1	0.955	0.794
ML-BOX(HALF)	0.842	0.394	0.552	0.692	0.333	0.786	0.654	0.364	0.869
ML-BOX+(AND)	0.342	0.136	0.591	0.41	0.045	0.941	0.025	0.136	0.4
ML-BOX+(OR)	0.868	0.409	0.55	0.863	0.409	0.789	0.654	0.364	0.869
ML-BOX+(HALF)	0.842	0.394	0.552	0.692	0.333	0.786	0.654	0.364	0.869

Table F-3 - Performance measures in experiment 1 of system call

(A) ATOMIC ML CLASSIFIERS

		J48				KNN				NB				SVM				NN			
I	D	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC
10s	N	.98	.07	.97	.97	.99	.06	.97	.99	.90	.02	.99	.98	.96	.03	.98	.96	.98	.05	.98	.98
	B	.93	.02	.96	.97	.94	.01	.98	.99	.98	.10	.83	.97	.97	.04	.92	.96	.95	.02	.96	.98
30s	N	1.0	.10	.97	.97	.99	.11	.97	.98	.96	.03	.99	.98	.99	.08	.98	.95	.99	.07	.98	.99
	B	.90	.00	.98	.97	.89	.01	.96	.98	.97	.04	.87	.98	.92	.01	.96	.95	.93	.01	.96	.99
60s	N	.99	.08	.98	.97	.99	.04	.99	.99	.98	.01	1.0	.99	.99	.01	1.0	.99	.98	.05	.99	1.0
	B	.92	.01	.94	.97	.96	.01	.95	.99	.99	.02	.94	.98	.99	.01	.95	.99	.95	.02	.93	1.0
300s	N	.99	.16	.95	.92	1.0	.04	.99	.99	1.0	.12	.96	.94	1.0	.00	1.0	1.0	1.0	.00	1.0	1.0
	B	.84	.01	.95	.92	.96	.00	1.0	.99	.88	.00	1.0	.96	1.0	.00	1.0	1.0	1.0	.00	1.0	1.0
600s	N	1.0	.06	.98	.97	1.0	.06	.98	.98	.98	.18	.93	.89	1.0	.00	1.0	1.0	.98	.00	1.0	1.0
	B	.94	.00	1.0	.97	.94	.00	1.0	.98	.82	.02	.93	.91	1.0	.00	1.0	1.0	1.0	.02	.94	1.0
AV	N	0.99	0.09	0.97	0.96	0.99	0.06	0.98	0.98	0.96	0.07	0.97	0.96	0.99	0.02	0.99	0.98	0.99	0.03	0.99	0.99
	B	0.91	0.01	0.97	0.96	0.94	0.01	0.98	0.98	0.93	0.04	0.91	0.96	0.98	0.01	0.97	0.98	0.97	0.01	0.96	0.99

(B) BOX ML CLASSIFIERS

		BOX-AND				BOX-OR				BOX-HALF				BOX-AND+				BOX-OR+				BOX-HALF+			
I	D	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC
10s	N	.98	.05	.98	.98	.88	.01	.99	.94	.98	.05	.98	.97	.99	.08	.96	.95	.98	.05	.98	.96	.98	.05	.98	.97
	B	.95	.02	.96	.98	.99	.12	.81	.94	.95	.02	.97	.97	.92	.01	.99	.95	.95	.02	.95	.96	.95	.02	.97	.97
30s	N	.99	.07	.98	.99	.95	.01	1.0	.97	.99	.08	.98	.95	1.0	.14	.96	.93	.99	.08	.98	.95	.99	.08	.98	.95
	B	.93	.01	.96	.99	.99	.05	.87	.97	.92	.01	.96	.95	.86	.00	.99	.93	.92	.01	.95	.95	.92	.01	.96	.95
60s	N	.98	.05	.99	1.0	.97	.00	1.0	.99	.99	.04	.99	.97	.92	.01	.96	.95	.98	.04	.99	.97	.99	.04	.99	.97
	B	.95	.02	.93	1.0	1.0	.03	.91	.99	.96	.01	.95	.97	.99	.08	.98	.95	.96	.02	.93	.97	.96	.01	.95	.97
300s	N	1.0	.00	1.0	1.0	.99	.00	1.0	.99	1.0	.04	.99	.98	.84	.00	1.0	.92	.99	.04	.99	.97	1.0	.04	.99	.98
	B	1.0	.00	1.0	1.0	1.0	.01	.96	.99	.96	.00	1.0	.98	1.0	.16	.95	.92	.96	.01	.96	.97	.96	.00	1.0	.98
600s	N	.98	.00	1.0	1.0	.98	.00	1.0	.99	1.0	.06	.98	.97	.94	.00	.99	.97	1.0	.06	.98	.97	1.0	.06	.98	.97
	B	1.0	.02	.94	1.0	1.0	.02	.94	.99	.94	.00	1.0	.97	1.0	.06	.98	.97	.94	.00	.99	.97	.94	.00	1.0	.97
AV	N	.99	.03	.99	.99	.95	.00	1.0	.98	.99	.05	.98	.97	.94	.05	.97	.95	.99	.05	.98	.97	.99	.05	.98	.97
	B	.97	.01	.96	.99	1.0	.05	.90	.98	.95	.01	.97	.97	.95	.06	.98	.95	.95	.01	.96	.97	.95	.01	.97	.97

Table F-4 - Performance measures in experiment 2 of system call

(A) ATOMIC ML CLASSIFIERS

		J48				KNN				NB				SVM				NN			
B	D	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC
B1	N	.94	.18	.94	.90	.97	.27	.91	.87	.86	.29	.89	.85	.92	.35	.88	.78	.86	.45	.76	.84
	B	.82	.06	.84	.90	.73	.03	.90	.87	.71	.14	.69	.84	.65	.08	.80	.78	.55	.14	.80	.84
B2	N	.95	.30	.76	.88	.97	.99	.24	.66	.89	.00	1.0	.95	.91	.00	1.0	.96	.55	.00	.69	.99
	B	.70	.05	.88	.88	.01	.03	.60	.66	1.0	.11	.97	.95	1.0	.09	.97	.96	1.0	.45	.89	.99
B3	N	.95	.10	.97	.96	.96	.18	.95	.99	.88	.02	.99	.95	.92	.36	.91	.78	.56	.23	.64	.92
	B	.90	.05	.84	.96	.82	.04	.87	.99	.98	.12	.71	.95	.64	.08	.75	.78	.78	.44	.56	.92
B4	N	.96	.02	.94	.97	.96	.04	.90	.99	.89	.02	.95	.97	.96	.79	.38	.59	.88	.79	.29	.91
	B	.98	.04	.99	.97	.96	.04	.99	.99	.98	.11	.97	.97	.21	.04	.90	.59	.21	.12	.92	.91
B5	N	.94	.00	1.0	.99	.96	.00	1.0	1.0	.88	.00	1.0	.95	.91	.00	1.0	.96	.73	.30	.79	.91
	B	1.0	.06	.42	.99	1.0	.04	.58	1.0	1.0	.12	.28	.95	1.0	.09	.46	.96	.70	.27	.42	.91
B6	N	.98	.37	.78	.91	.99	.37	.78	.97	.87	.08	.94	.93	.91	.10	.93	.91	.59	.43	.39	.85
	B	.63	.02	.96	.91	.63	.01	.98	.97	.92	.13	.85	.94	.90	.09	.90	.91	.57	.41	.75	.85
B7	N	.97	.36	.88	.85	.97	.32	.90	.87	.90	.15	.94	.91	.92	.21	.92	.85	.95	.30	.90	.86
	B	.64	.03	.88	.85	.68	.03	.90	.87	.85	.10	.79	.95	.79	.08	.81	.85	.70	.05	.88	.86
B8	N	.94	.00	1.0	.99	.96	.53	.88	.91	.87	.00	1.0	.98	.91	.03	.99	.94	.66	.07	.77	.96
	B	1.0	.06	.82	.99	.47	.04	.80	.91	1.0	.13	.70	.98	.97	.09	.78	.94	.93	.34	.67	.96
B9	N	.95	.05	.98	.96	.96	.05	.98	.99	.84	.27	.90	.81	.91	.20	.93	.86	.62	.05	.68	.94
	B	.95	.05	.88	.96	.95	.04	.90	.99	.73	.16	.65	.74	.80	.09	.81	.86	.95	.38	.63	.94
AV	N	.95	.15	.92	.94	.97	.30	.84	.92	.88	.09	.96	.92	.92	.23	.88	.85	.71	.29	.66	.91
	B	.85	.05	.83	.94	.70	.03	.84	.92	.91	.12	.73	.92	.77	.08	.80	.85	.71	.29	.72	.91

(B) BOX ML CLASSIFIERS

		BOX-AND				BOX-OR				BOX-HALF				BOX-AND+				BOX-OR+				BOX-HALF+			
B	D	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU
B1	N	.86	.45	.86	.84	.76	.06	.97	.85	.94	.25	.91	.84	.96	.30	.90	.83	.85	.16	.93	.84	.93	.22	.92	.85
	B	.55	.14	.80	.84	.94	.24	.66	.85	.75	.06	.84	.84	.70	.04	.87	.83	.84	.15	.68	.84	.78	.07	.83	.85
B2	N	.56	.00	1.0	.99	.53	.00	1.0	.76	.92	.00	1.0	.96	.96	.30	.76	.83	.88	.00	1.0	.94	.92	.00	1.0	.96
	B	1.0	.44	.89	.99	1.0	.47	.88	.76	1.0	.08	.97	.96	.70	.04	.89	.83	1.0	.12	.96	.94	1.0	.08	.97	.96
B3	N	.57	.23	.95	.92	.52	.01	1.0	.76	.93	.12	.97	.90	.96	.11	.97	.92	.88	.01	1.0	.94	.92	.09	.97	.92
	B	.78	.43	.56	.92	.99	.48	.49	.76	.88	.07	.81	.90	.89	.04	.86	.92	.99	.12	.70	.94	.91	.08	.80	.92
B4	N	.88	.79	.39	.91	.81	.01	.98	.90	.95	.03	.90	.96	.96	.03	.91	.97	.89	.01	.98	.94	.94	.03	.92	.96
	B	.21	.12	.92	.91	.99	.19	.95	.90	.97	.05	.98	.96	.97	.04	.99	.97	.99	.11	.96	.94	.97	.06	.98	.96
B5	N	.74	.30	.99	.91	.69	.00	1.0	.84	.92	.00	1.0	.96	.95	.00	1.0	.98	.87	.00	1.0	.94	.91	.00	1.0	.96
	B	.70	.26	.43	.91	1.0	.31	.18	.84	1.0	.08	.46	.96	1.0	.05	.51	.98	1.0	.13	.25	.94	1.0	.09	.45	.96
B6	N	.60	.43	.79	.85	.51	.00	1.0	.75	.96	.23	.86	.86	.99	.41	.76	.79	.86	.04	.97	.91	.96	.20	.87	.88
	B	.57	.40	.75	.85	1.0	.49	.67	.75	.77	.04	.95	.86	.59	.01	.97	.79	.96	.14	.85	.91	.80	.04	.95	.88
B7	N	.95	.30	.90	.86	.90	.15	.94	.87	.94	.29	.90	.82	.97	.36	.88	.80	.90	.15	.94	.87	.94	.29	.90	.82
	B	.70	.05	.88	.86	.85	.10	.77	.87	.71	.06	.84	.82	.64	.03	.88	.80	.85	.10	.78	.87	.71	.06	.84	.82
B8	N	.67	.07	.98	.96	.63	.00	1.0	.81	.92	.00	1.0	.96	.95	.00	1.0	.98	.87	.00	1.0	.93	.92	.00	1.0	.96
	B	.93	.33	.67	.96	1.0	.37	.56	.81	1.0	.08	.80	.96	1.0	.05	.85	.98	1.0	.13	.68	.93	1.0	.08	.79	.96
B9	N	.63	.05	.98	.94	.58	.02	.99	.77	.92	.05	.98	.93	.96	.29	.91	.84	.83	.03	.98	.90	.92	.05	.98	.93
	B	.95	.37	.63	.94	.98	.42	.55	.77	.95	.08	.83	.93	.71	.04	.87	.84	.97	.17	.69	.90	.95	.08	.83	.93
AV	N	.72	.29	.87	.91	.66	.03	.99	.81	.93	.11	.95	.91	.96	.20	.90	.88	.87	.04	.98	.91	.93	.10	.95	.92
	B	.71	.28	.72	.91	.97	.34	.63	.81	.89	.07	.83	.91	.80	.04	.85	.88	.96	.13	.73	.91	.90	.07	.83	.92

Table F-5 - Performance measures in experiment 3 of system call

(A) ATOMIC ML CLASSIFIERS

B	D	J48				KNN				NB				SVM				NN			
		TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC	TPR	FPR	PRC	AUC
B1	N	.91	.18	.88	.86	.94	.32	.83	.86	.63	.18	.81	.76	.94	.41	.80	.76	1.0	.95	.68	.55
	B	.82	.09	.83	.86	.68	.06	.85	.86	.82	.37	.57	.78	.59	.06	.82	.76	.05	.00	1.0	.55
B2	N	.93	.98	.25	.85	.94	.99	.25	.53	.66	.00	1.0	.81	.93	.00	1.0	.96	.96	.00	1.0	.95
	B	.02	.07	.61	.85	.01	.06	.55	.53	1.0	.34	.91	.81	1.0	.07	.98	.96	1.0	.04	.99	.95
B3	N	.91	.06	.96	.92	.94	.19	.91	.94	.63	.00	1.0	.85	.93	.44	.83	.74	1.0	.94	.73	.70
	B	.94	.09	.81	.92	.81	.06	.83	.94	1.0	.37	.55	.85	.56	.07	.76	.74	.06	.00	1.0	.70
B4	N	.93	.04	.81	.97	.94	.04	.81	.98	.64	.01	.90	.90	.96	.97	.26	.49	.96	.97	.26	.91
	B	.96	.07	.98	.97	.96	.06	.98	.98	.99	.36	.91	.90	.03	.04	.76	.49	.03	.04	.76	.91
B5	N	.86	.00	1.0	.89	.94	.50	.96	.94	.61	.00	1.0	.84	.94	.50	.96	.71	.00	.00	.00	.15
	B	1.0	.14	.47	.89	.50	.06	.44	.94	1.0	.39	.18	.84	.50	.06	.38	.71	1.0	1.0	.06	.15
B6	N	.92	.33	.72	.89	.94	.41	.68	.91	.64	.11	.79	.82	.94	.35	.71	.79	.95	.57	.63	.65
	B	.67	.08	.90	.89	.59	.06	.91	.91	.89	.36	.73	.83	.65	.06	.91	.79	.43	.05	.90	.65
B7	N	.93	.41	.80	.81	.94	.36	.81	.84	.69	.18	.84	.78	.94	.41	.80	.76	.99	.68	.73	.59
	B	.59	.07	.82	.81	.64	.06	.85	.84	.82	.31	.63	.83	.59	.06	.82	.76	.32	.01	.94	.59
B8	N	.88	.00	1.0	.98	.89	.52	.81	.84	.68	.00	1.0	.94	.93	.07	.96	.93	.00	.00	.15	.93
	B	1.0	.12	.77	.98	.48	.11	.68	.84	1.0	.32	.59	.94	.93	.07	.83	.93	1.0	1.0	.27	.93
B9	N	.93	.19	.88	.87	.91	.33	.82	.81	.53	.24	.74	.70	.90	.29	.84	.80	1.0	.90	.69	.78
	B	.81	.07	.84	.87	.67	.09	.82	.81	.76	.47	.45	.66	.71	.10	.80	.80	.10	.00	1.0	.78
AV	N	.91	.24	.81	.89	.93	.40	.76	.85	.63	.08	.89	.82	.93	.38	.79	.77	.76	.55	.54	.69
	B	.75	.08	.78	.89	.59	.06	.76	.80	.92	.36	.61	.82	.61	.06	.78	.77	.44	.23	.76	.69

(B) BOX ML CLASSIFIERS

B	D	BOX-AND				BOX-OR				BOX-HALF				BOX-AND+				BOX-OR+				BOX-HALF+			
		TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU	TPR	FPR	PRC	AU
B1	N	1.0	.95	.68	.55	.61	.05	.93	.78	.93	.32	.83	.81	.94	.32	.83	.74	.91	.18	.88	.88	.93	.32	.83	.81
	B	.05	.00	1.0	.55	.95	.39	.59	.78	.68	.07	.84	.81	.68	.06	.86	.74	.82	.09	.82	.85	.68	.07	.84	.81
B2	N	.96	.00	1.0	.96	.65	.00	1.0	.83	.94	.00	1.0	.97	.94	.99	.25	.43	.93	.98	.25	.47	.94	.00	1.0	.97
	B	1.0	.04	.99	.96	1.0	.35	.91	.83	1.0	.06	.99	.97	.01	.06	.58	.43	.02	.07	.59	.47	1.0	.06	.99	.97
B3	N	1.0	.94	.73	.70	.62	.00	1.0	.81	.94	.13	.93	.91	.94	.25	.88	.77	.91	.00	1.0	.95	.94	.13	.93	.91
	B	.06	.00	1.0	.70	1.0	.38	.54	.81	.88	.06	.84	.91	.75	.06	.84	.77	1.0	.09	.81	.95	.88	.06	.84	.91
B4	N	.96	.97	.26	.92	.63	.01	.90	.81	.94	.04	.81	.95	.94	.05	.79	.87	.93	.04	.82	.94	.94	.04	.81	.95
	B	.03	.04	.76	.92	.99	.37	.91	.81	.96	.06	.98	.95	.95	.06	.98	.87	.96	.07	.98	.94	.96	.06	.98	.95
B5	N	.04	.00	1.0	.15	.04	.00	1.0	.50	.90	.00	1.0	.95	.94	.50	.96	.66	.86	.00	1.0	.93	.90	.00	1.0	.95
	B	1.0	.96	.07	.15	1.0	.96	.07	.50	1.0	.10	.41	.95	.50	.06	.48	.66	1.0	.14	.44	.93	1.0	.10	.41	.95
B6	N	.95	.57	.63	.65	.62	.00	1.0	.81	.94	.37	.70	.78	.94	.45	.66	.68	.92	.28	.74	.82	.94	.37	.70	.78
	B	.43	.05	.90	.65	1.0	.38	.74	.81	.63	.06	.91	.78	.55	.06	.90	.68	.72	.08	.90	.82	.63	.06	.91	.78
B7	N	.99	.68	.73	.60	.69	.18	.84	.75	.94	.41	.80	.77	.94	.41	.80	.70	.93	.36	.81	.78	.94	.41	.80	.77
	B	.32	.01	.94	.60	.82	.31	.63	.75	.59	.06	.84	.77	.59	.06	.84	.70	.64	.07	.83	.78	.59	.06	.84	.77
B8	N	.04	.00	1.0	.94	.04	.00	1.0	.50	.86	.00	1.0	.93	.89	.52	.81	.62	.88	.00	1.0	.94	.86	.00	1.0	.93
	B	1.0	.96	.27	.94	1.0	.96	.27	.50	1.0	.14	.75	.93	.48	.11	.69	.62	1.0	.12	.77	.94	1.0	.14	.75	.93
B9	N	1.0	.90	.69	.79	.53	.00	1.0	.77	.91	.33	.82	.79	.94	.33	.83	.74	.89	.19	.88	.85	.91	.33	.82	.79
	B	.10	.00	1.0	.78	1.0	.47	.50	.77	.67	.09	.82	.79	.67	.06	.85	.74	.81	.11	.81	.85	.67	.09	.82	.79
AV	N	.77	.56	.75	.69	.49	.03	.96	.73	.92	.18	.88	.87	.94	.42	.76	.69	.91	.23	.82	.84	.92	.18	.88	.87
	B	.44	.23	.77	.69	.97	.51	.57	.73	.82	.08	.82	.87	.58	.06	.78	.69	.77	.09	.77	.84	.82	.08	.82	.87

Appendix G Training Dataset Feature Selection

Table G-1 - Network Traffic Feature Selection

Packets/Stream Frame Duration
Packets/Stream Packet Size
Arguments Number in HTTP Request URL

Table G-2 - System Calls Dataset Feature Selection

The timestamp of the system call
The system call name
Return value of the call
The time spent on the call
Frequency of calls over different time intervals (10,30,60,300,600)