CHID: CONDITIONAL HYBRID INTRUSION DETECTION SYSTEM FOR REDUCING FALSE POSITIVES AND RESOURCE CONSUMPTION ON MALICOUS DATASETS

HASHEM MOHAMMED ALAIDAROS

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2017

Awang Had Salleh
Graduate School
of Arts And Sciences

Universiti Utara Malaysia

PERAKUAN KERJA TESIS / DISERTASI
(Certification of thesis / dissertation)

Kami, yang bertandatangan, memperakukan bahawa
(We, the undersigned, certify that)

HASHEM MOHAMMED ABDULRAHMAN AL-AIDAROS

calon untuk Ijazah PhD
(candidate for the degree of)

telah mengemukakan tesis / disertasi yang bertajuk:
(has presented his/her thesis / dissertation of the following title):

"CHID: CONDITIONAL HYBRID INSTRUSION DETECTION SYSTEM FOR REDUCING FALSE POSITIVES AND RESOURCE CONSUMPTION ON MALICOUS DATASETS"

seperti yang tercatat di muka surat tajuk dan kulit tesis / disertasi.
(as it appears on the title page and front cover of the thesis / dissertation).

Bahawa tesis/disertasi tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan, sebagaimana yang ditunjukkan oleh calon dalam ujian lisan yang diadakan pada : *27 November 2016.*
That the said thesis/dissertation is acceptable in form and content and displays a satisfactory knowledge of the field of study as demonstrated by the candidate through an oral examination held on: *November 27, 2016.*

Pengerusi Viva: *(Chairman for VIVA)*	Assoc. Prof. Dr. Wan Rozaini Sheik Osman	Tandatangan *(Signature)*
Pemeriksa Luar: *(External Examiner)*	Prof. Dr. Omar Zakaria	Tandatangan *(Signature)*
Pemeriksa Dalam: *(Internal Examiner)*	Dr. Mohd Nizam Omar	Tandatangan *(Signature)*
Nama Penyelia/Penyelia-penyelia: *(Name of Supervisor/Supervisors)*	Dr. Massudi Mahmuddin	Tandatangan *(Signature)*

Tarikh:
(Date) November 27, 2016

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to :

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Memeriksa paket untuk mengesan pencerobohan berhadapan cabaran apabila berlakunya jumlah trafik rangkaian yang tinggi. Proses pengesanan berdasarkan paket bagi setiap muat beban pada wayar mengurangkan prestasi sistem pengesanan pencerobohan rangkaian (NIDS). Isu ini memerlukan kepada satu pengenalan NIDS berasaskan aliran untuk mengurangkan jumlah data yang akan diproses dengan memeriksa agregat maklumat dari paket yang berkaitan. Walau bagaimanapun, pengesanan berdasarkan aliran masih mengalami penjanaan amaran positif palsu kerana input data yang tidak lengkap. Kajian ini mencadangkan Pengesanan Pencerobohan Hibrid Bersyarat (CHID) dengan mencantumkan pengesanan berasaskan aliran dengan pengesanan berasaskan paket. Tambahan lagi, ia juga bertujuan untuk memperbaiki penggunaan sumber pendekatan pengesanan berasaskan paket. CHID menggunakan algoritma penilaian ciri pembalut atribut yang menandakan aliran hasad untuk analisis selanjutnya oleh pengesanan berasaskan paket. Pendekatan Rangka Kerja Input telah digunakan untuk mencetus aliran paket diantara pengesanan berasaskan paket dan berasaskan aliran. Eksperimen tapak ujiterkawal telah dijalankan untuk menilai prestasi mekanisme pengesanan CHID menggunakan set data yang diperolehi daripada pada kadar trafik yang berbeza. Hasil penilaian didapati CHID memperoleh peningkatan prestasi yang ketara dari segi penggunaan sumber dan kadar paket susut, berbanding pelaksanaan pengesanan berasaskan paket lalai. Pada kelajuan 200 Mbps, CHID dalam senario IRC-bot, boleh mengurangkan 50.6% dari penggunaan memori dan menyusut 18.1% penggunaan CPU tanpa paket susut. Pendekatan CHID boleh mengurangkan kadar positif palsu berdasarkan pengesanan berasaskan aliran dan mengurangkan penggunaan sumber pengesanan berasaskan paket disamping memelihara ketepatan pengesanan. Pendekatan CHID boleh dianggap sebagai sistem generik untuk diaplikasikan untuk sistem pemantauan pengesanan pencerobohan.

Kata Kunci: Pengesanan berasaskan aliran, Pengesanan berasaskan paket, Bro-NIDS, Rangka kerja input.

Abstract

Inspecting packets to detect intrusions faces challenges when coping with a high volume of network traffic. Packet-based detection processes every payload on the wire, which degrades the performance of network intrusion detection system (NIDS). This issue requires an introduction of a flow-based NIDS that reduces the amount of data to be processed by examining aggregated information of related packets. However, flow-based detection still suffers from the generation of the false positive alerts due to incomplete data input. This study proposed a Conditional Hybrid Intrusion Detection (CHID) by combining the flow-based with packet-based detection. In addition, it is also aimed to improve the resource consumption of the packet-based detection approach. CHID applied attribute wrapper features evaluation algorithms that marked malicious flows for further analysis by the packet-based detection. Input Framework approach was employed for triggering packet flows between the packet-based and flow-based detections. A controlled testbed experiment was conducted to evaluate the performance of detection mechanism's CHID using datasets obtained from on different traffic rates. The result of the evaluation showed that CHID gains a significant performance improvement in terms of resource consumption and packet drop rate, compared to the default packet-based detection implementation. At a 200 Mbps, CHID in IRC-bot scenario, can reduce 50.6% of memory usage and decreases 18.1% of the CPU utilization without packets drop. CHID approach can mitigate the false positive rate of flow-based detection and reduce the resource consumption of packet-based detection while preserving detection accuracy. CHID approach can be considered as generic system to be applied for monitoring of intrusion detection systems.

Keywords: Flow-based detection, Packet-based detection, Input Framework approach.

Acknowledgement

First of all, I would like to express my sincere and deep gratitude to my supervisor Dr. Massudi Mahmuddin who provided considerable and invaluable insights and comments to help me on this journey. Without his patient support, enlightened guidance, it is impossible for me to complete and enhance the quality of my work.

I would like to thank the InterNetWork Lab team for their co-operation, kindness, and sharing useful discussions for my research.

Thank you goes to Dr. Shakeel Habeeb, the dean of Prince Sultan College (PSCJ), Al-Faisal University, for providing me the testbed resources for my research.

Special thanks go to Johanna Amann and Robin Sommer, the developers of the Bro-IDS. Their supports with the many problems encountered during experiments were of great help.

Finally, my heartiest gratitude goes to my beloved family, in particular, my parents my wife without their love, support and encouragement, it would not be possible for me end this journey.

Table of Contents

Permission to Use i
Abstrak ii
Abstract iii
Acknowledgement iv
Table of Contents v
List of Tables x
List of Figures xii
List of Appendices xiv
List of Abbreviations xv
List of Publications xvii

CHAPTER ONE INTRODUCTION 1
1.1 Background 1
1.2 Intrusion Detection System 2
1.3 Motivation 3
1.4 Problem Statement 5
1.5 Research Questions 8
1.6 Research Objectives 8
1.7 Research Contribution 9
1.8 Scope of the Study 9
1.9 Research Steps 10
1.10 Organization of the Thesis 12

CHAPTER TWO LITERATURE REVIEW 13
2.1 Growth of Traffic and Threats 13
2.1.1 Internet Attacks 15
2.1.2 Growth of Threats 16
2.2 Intrusion Detection Systems 17
2.2.1 Security Tools and Techniques 17
2.2.2 IDS Structure 19
2.2.3 Software-based IDS 20

2.2.4 IDS Types 22
2.2.4.1 Location-based IDS 23
2.2.4.2 Detection-based IDS 25
2.2.4.3 Data-processed-based NIDS 26
2.2.5 NIDS Requirements 28
2.2.6 NIDS Challenges 30
2.3 Packet-based NIDS 31
2.3.1 Scalability 32
2.3.2 Detection Accuracy 35
2.3.3 Botnet Detections Related Works 37
2.4 Flow-based NIDS 42
2.4.1 Flow-based Detection Overview 42
2.4.2 Structure 44
2.4.2.1 Exporter 45
2.4.2.2 Collector 46
2.4.2.3 Analyser 47
2.4.3 Scalability 47
2.4.4 Detection Accuracy 48
2.4.5 Botnet Detection Related Works 50
2.5 Packet-based and Flow-based Detection Comparison 54
2.5.1 Comparison 54
2.5.2 Trade-offs 54
2.6 False Positive Reduction 56
2.6.1 Scope of Attacks 56
2.6.2 Network Awareness 57
2.6.3 Traffic Cleanness 58
2.6.4 Alert Correlation 59
2.6.5 Hybrid Signature-based and Anomaly-based 62
2.6.6 Hybrid Flow-based and Packet-based 66
2.7 Chapter Summary 68
CHAPTER THREE METHODOLOGY 70

3.1 Proposed Mechanism Design 70
3.2 Component Identification 76
3.2.1 Traffic Capture 76
3.2.2 Flow Aggregation 77
3.2.3 Packet-based Detection 78
3.2.4 Flow-based Detection 80
3.3 Implementation 81
3.4 Evaluation 83
3.4.1 Experimental Environment 83
3.4.2 Experiment Setup 85
3.4.3 Measurement Procedures 89
3.4.4 Dataset 92
3.4.4.1 Malicious Datasets 93
3.4.4.2 Background Traces 96
3.4.5 Evaluation Metrics 97
3.5 Chapter Summary 99
CHAPTER FOUR TWO STAGES FLOW-BASED DETECTION 100
4.1 Introduction 100
4.2 Design 104
4.3 Attack Selection 104
4.3.1 IRC-bot Behaviour 105
4.3.2 P2P-bot Behaviour 107
4.4 Detection Scripts Derivation 108
4.4.1 Packet and Flow Analysis 109
4.4.1.1 Flows Labelling 112
4.4.1.2 Attribute Classifications 112
4.4.2 Detection Policy Scripts 115
4.5 Detection Implementations 116
4.5.1 Flow-based Detection 116
4.5.1.1 Threshold-based Mechanism 119
4.5.1.2 Proof of Concept 121

4.5.2 Packet-based Detection 123
4.6 Evaluation Environment 126
4.7 Chapter Summary 127
CHAPTER FIVE CONDITIONAL HYBRID INTRUSION DETECTION 128
5.1 Introduction 128
5.2 Proposed Mechanism 131
5.2.1 Design and Theory 131
5.2.2 Combination Approach Scenario 133
5.3 Implementation 135
5.3.1 Traffic Recording Strategy 135
5.3.2 Subsequent-Packet Strategy 138
5.3.3 PH and FL Communicating Process Implementation 140
5.3.3.1 BPF-only Method 141
5.3.3.2 Input Framework (IF) Method 142
5.3.4 IF Method Integration 143
5.3.4.1 Reading Files 144
5.3.4.2 Updating Table 146
5.3.4.3 BPF Filtering 147
5.3.4.4 Proof of Concept 148
5.3.5 Partial Payload Inspection Approach 150
5.3.6 Switching Approach based on Traffic Rate 151
5.4 Evaluation 155
5.4.1 Attack Scenarios 155
5.4.2 Experimental Environments 156
5.4.3 Measurement Procedures 158
5.4.4 Traffic Data for CHID Mechanism 159
5.5 Chapter Summary 160
CHAPTER SIX RESULT AND DISCUSSION 161
6.1 Flow-based Detection Scripts 161
6.1.1 Dataset Correctness 161
6.1.2 Most Significant Attributes 162

6.1.3 Detection Accuracy 164
6.1.4 False Positive Test 165
6.1.5 Resource Consumption 166
6.2 CHID Mechanism 171
6.2.1 Detection Accuracy 171
6.2.2 Resource Consumption 173
6.2.3 Filtered Hosts and IF Method 177
6.2.4 Packet Drop Rate 179
6.2.5 Partial Payload in PH 181
6.3 Chapter Summary 183

CHAPTER SEVEN CONCLUSION 186
7.1 Summary of Research 186
7.2 Objectives Achievements 187
7.2.1 First Objective 187
7.2.2 Second Objective 188
7.2.3 Third Objective 189
7.3 Main Contribution 191
7.4 Limitations and Future Works 193
a. Multi-thread Approach 193
b. PF_RING Packet Capturing 193
c. Diverse Attacks Scenario 194
d. Tuning Flow Keys and Timeouts 194
7.5 Chapter Summary 195

REFERENCES 196

List of Tables

Table 2.1 Growing of Threats [29] 16
Table 2.2 Differences between IDS, Firewall, and IPS 18
Table 2.3 Examples of Botnets Attack [25] 39
Table 2.4 Botnet Detection Methods with Packet-based Approach 41
Table 2.5 Attacks Detectable by Flow-based NIDS only 50
Table 2.6 Botnet Detection Methods with Flow-based Approach 53
Table 2.7 Comparison between Packet-based and Flow-based NIDS 55
Table 2.8 Related Works for False Positive Reduction 62
Table 2.9 Related Works of Hybrid Detection Methods for False Positive Reduction 65
Table 3.1 Bro Advantages among Other NIDSs [10] 79
Table 3.2 System and Hardware Description Used in Testbed 86
Table 3.3 Software Applications Description Used in Testbed 86
Table 3.4 Loge Files Disabled 89
Table 3.5 Datasets and Statistics 94
Table 3.6 Notion Matrix [163] 97
Table 4.1 Fields Description of Wired.log file 111
Table 4.2 Features of Flow and Packet Generated from Logs 111
Table 4.3 Attributes Used for Classification 114
Table 4.4 Attribute Selection Setting and Classification Selection 115
Table 5.1 Two Combination Approaches 133
Table 5.2 Datasets for Detection Accuracy Measurements 160
Table 6.1 Best Three Important Attributes 163
Table 6.2 False Positive Rate (FPR) 164
Table 6.3 Precision Results 164
Table 6.4 Detection Results with P2P-bot Scenario 172
Table 6.5 Detection Results with IRC-bot Scenario 173
Table 6.6 Packet Drop Rate in P2P-bot Scenario 179
Table 6.7 Packet Drop Rate in IRC-bot Scenario 179

Table 6.8 Comparison between Full-Payload and Partial-Payload Inspection in PH for P2P-bot Scenario 182
Table 6.9 Comparison between Full-Payload and Partial-Payload Inspection in PH for IRC-bot Scenario 182
Table D.1 Fields Description of Connection.log file 225
Table D.2 Fields Description of Signatures.log file 226
Table D.3 Fields Description of Notice.log file 227

List of Figures

Figure 1.1: Challenges and Consequences for NIDS....4
Figure 2.1: Expansion of the Internet Users over Years [1] 14
Figure 2.2: IDS Main Components 19
Figure 2.3: Types of IDS (dash line indicates the research scope) 23
Figure 2.4: NIDS Location....24
Figure 2.5: Packet-based NIDS....32
Figure 2.6: Centralized Botnet Attack Methods 38
Figure 2.7: Decentralized Botnet Attack Methods....39
Figure 2.9: Flow-based Components [19]....45
Figure 3.1: Research Methodology 71
Figure 3.3: Conceptual Model of the CHID Approach....73
Figure 3.4: Proposed Flow Chart 74
Figure 3.5: Component Requirements for Proposed Mechanism 81
Figure 3.6: Experimental Testbed 86
Figure 3.7: Experimental Commands 92
Figure 4.1: Illustration of (a) Packet-based Detection System and (b) Flow-based Detection System 105
Figure 4.2: Workflow for Deriving Flow-based Detection Policy Scripts 109
Figure 4.3: Two Stages Flow-based Detection Mechanism....117
Figure 4.4: Live Experiment for Proof-of-Concept 122
Figure 4.5: Sample of SumStats Scripts....126
Figure 5.1: Flow-based and Packet-based detection with a) Scalability Level and b) Alert Verification Level (x-axis indicates the detection type) 132
Figure 5.2: Two Hosts with P2P Communications....133
Figure 5.3: Proposed Flow Chart with Traffic Recording Strategy 136
Figure 5.4: Proposed Flow Chart with CHID Mechanism....140
Figure 5.5: IF Method Integration into PH 144
Figure 5.6: Combination of IF and BPF Filter Approaches....147
Figure 5.7: Switching between CHID and PO Approaches based on Traffic Rate .151

Figure 6.1: Memory Usage with Different Traffic Rates – P2P-bot (FL: flow-based detection; PO: default packet-based only) 166
Figure 6.2: CPU Usage with Different Traffic Rates – P2P-bot (FL: flow-based detection; PO: default packet-based only) 167
Figure 6.3: Memory Usage with Different Traffic Rates – IRC-bot (FL: flow-based detection; PO: default packet-based only) 168
Figure 6.4: CPU Usage with Different Traffic Rates – IRC-bot (FL: flow-based detection; PO: default packet-based only) 168
Figure 6.5: Memory Usage over Time at 200 Mbps` – P2P-bot (FL: flow-based detection; PO: default packet-based only) 170
Figure 6.6: CPU Usage over Time at 200 Mbps – P2P-bot (FL: flow-based detection; PO: the default packet-based only) 171
Figure 6.7: Memory Usage over Time at 200 Mbps – IRC-bot.....174
Figure 6.8: CPU Usage over Time at 200 Mbps – IRC-bot.....174
Figure 6.9: Memory Usage with Different Traffic Rates – P2P-bot.....175
Figure 6.10: CPU Usage with Different Traffic Rates – P2P 176
Figure 6.11: Memory Usage with Different Traffic Rates –IRC-bot.....177
Figure 6.12: CPU Usage with Different Traffic Rates – IRC-bot.....177
Figure 6.13: Drop packet Rate with Different Traffic Rates – P2P-bot and IRC-bot 180
Figure C.1: Classes of Host Behaviour for Worm Detection.....223
Figure E.1: CPU Usage over Time at 100 Mbps – P2P-bot.....228
Figure E.2: Memory Usage over Time at 100 Mbps – P2P-bot.....228
Figure E.3: CPU Usage over Time at 200 Mbps- P2P-bot 229
Figure E.4: Memory Usage over Time at 200 Mbps – P2P-bot.....229
Figure E.5: CPU Usage over Time at 500 Mbps – P2P-bot.....230
Figure E.6: Memory Usage over Time at 500 Mbps – P2P-bot.....230
Figure E.7: CPU Usage over Time at 1000 Mbps – P2P-bot.....231
Figure E.8: Memory Usage over Time at 1000 Mbps – P2P-bot.....231

List of Appendices

Appendix A Attack Classification 210
Appendix B NIDS Requirements.....215
Appendix C Attacks Detectable by Flow-based Approach.....218
Appendix D Main Bro Log Files 225
Appendix E Resource Consumptions Results.....228
Appendix F Samples of Detection Code.....232

List of Abbreviations

API	Application Programming Interface
BPF	Berkeley Packet Filtering
Broccoli	Bro Client Communications Library
C&C	Command and Control
CHID	Conditional Hybrid Intrusion Detection
CTU	Czech Technical University
DARPA	Defence Advanced Research Project Agency
DPI	Deep Packet Inspection
DoS	Denial of Service
FL	FLow-based-detection
FPA	Front Payload Aggregation
FPR	False Positive Rate
HIDS	Host-based Intrusion Detection System
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IF	Input Framework
IP	Internet Protocol
IPFIX	IP Flow Information Export
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
ISOT	Information Security and Object Technology

ISP	Internet Service Provider
KDD	Knowledge Discovery in Dataset
LAN	Local Area Networks
NAT	Network Address Translation
NIDS	Network Intrusion Detection System
OSI	Open Systems Interconnection
RP	Received Packets
P2P	Peer to Peer
PCAP	Packet Capturing
PH	Packet-based in Hybrid
PO	Packet-based Only
PSCJ	Prince Sultan College Jeddah
PYL	Payload
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
TPR	True Positive Rate
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
WAN	Wide Area Networks

List of Publications

[1] H. Alaidaros and M. Mahmuddin, "Conditional hybrid approach for intrusion detection," *Research Journal Information Technology*, vol. 8, pp. 55-65, 2016.

[2] H. Alaidaros, and M. Mahmuddin, "Flow-based Approach on Bro Intrusion Detection," *in Advancement on Information Technology International Conference*, 2015

[2] H. Alaidaros, M. Mahmuddin, A. Al Mazari, "From Packet-based Towards Hybrid Packet-based and Flow-based Monitoring for Efficient Intrusion Detection: An Overview" *in 1st Taibah University International Conference on Computing and Information Technology (ICCIT12)*, 2012

[4] H. Alaidaros, M. Mahmuddin, A. Al Mazari, "An Overview of Flow-based and Packet-based Intrusion Detection Performance in High Speed Network," *in 12th International Arab Conference on Information Technology (ACIT12)*, 2011.

CHAPTER ONE
INTRODUCTION

1.1 Background

The number of Internet clients and services is growing more and more [1]. New Internet applications give users benefits for either their businesses or future life. The Internet is a powerful medium that has changed how people communicate and do businesses with the partners. These universal applications let companies achieve things that never been imagined before.

In addition to growing of the Internet users, networks become bigger and bigger. Although the Internet gives users' bright life and good businesses, it also has its unknown dark face. Since many new Internet services, devices, and hosts are developing, the number of vulnerabilities either in user smartphones, computers or servers is also increasing [2]. The more computers connected to the Internet the more possibility that the attacks take place. Many security gaps are exposed and misused by attacks. Unfortunately, attacks are growing with the Internet almost in parallel, and the race between them is continuing.

The number and the damage cost by those attacks are rising continuously. The security threats can exploit all types of the network, including LAN-based clusters, intranet, large-scale computational grids, and peer-to-peer service networks. These threats also exploit all exposed protocols and operating systems (OS) threatening different kinds of their applications such as database and web servers. Considering the damage cost originated from the attacks, it is important to detect an attack as soon as possible. The

main security tools for attack detection are Intrusion Detection System (IDS), Firewall, and Intrusion Prevention System (IPS). IDS can be defined as "the process of identifying and responding to malicious activity targeted at computing and networking resources" [3]. It detects unwanted exploitation to a computer system, both through the Internet and intranet.

Firewall is able to prevent some communication forbidden by the pre-defined security policy. However, unlike IDS, it does not usually have the capability to search for anomalies or specific content patterns. IPS is the next security layer that combines the protection of firewalls with the monitoring ability of IDSs to protect networks. IPSs is designed to sit inline with traffic flows and prevent attacks in real-time. However, they block traffic independently without human interaction. Therefore, the main disadvantages of IPSs are the serious consequences when blocking useful traffic beside its bad performance in high volume networks. On the other hand, IDS needs human interaction to do further actions. In addition, it sends commands to other security devices of the network infrastructure that can filter the traffic. Based on literature, IDS promises better performance in high speed networks compared with IPS since it inspect the traffic offline instead of inline mode [4].

1.2 Intrusion Detection System

In general, IDSs are divided into two basic classes based on their position in the network or audit source location: host-based IDS (HIDS) and network-based IDS (NIDS). HIDS monitors a single machine and audit data, such as resource usage and system logs, traced by the hosting operating system. On the other hand, NIDS monitors

a network and analyses the traffic which flows through the link such as routers. This research focus on NIDS since it is mostly used today and promised high detection rate [5]. In addition, in NIDS, deploying a new host in a network does not need any configuration to that host to be protected from intrusions. However, HIDS does not have the capability to evaluate the network traffic.

There are two methods based on the data type to be analysed in NIDS, packet-based and flow-based. Packet-based, also named Deep Packet Inspection (DPI), has to inspect the whole packet. The flow-based technique is widely deployed as a data source in applications like network monitoring, traffic analysis, and security. In flow-based NIDS, rather than looking at all packets going through a network link, NIDS looks at aggregated information of related packets of network traffic in the form of flow. Such information includes the duration of flow and number of packets and bytes sent and received in a particular connection.

1.3 Motivation

For NIDS to be efficient, the main requirements as mentioned in [6] and [7] are:

- Scalability: all potential packets are examined without resource overload in NIDS;
- Detection accuracy: detecting wide scope of intrusions with high detection rate and fewer false alarms.

These two requirements are currently attracted by researchers [6, 7]. Weng, et al. [7] stated the key requirements of an NIDS are scalability and detection accuracy since

they are the most requirements that influenced by increasing line-rates of network traffic and growing number of attack patterns. Another reason for selecting these requirements is that with all progresses that NIDS have made over years, researchers struggle to make NIDS meet these requirements [8-10]. Figure 1.1 provides an overview of the challenges for NIDS in general. It is obvious from the Figure that NIDSs should be able to handle the growth in Internet bandwidth as well as the increased number of attacks. Network traffic continues to grow which presents many challenges in the domain of security. With an increasing data volume in the traffic, the challenges of packet-based NIDS increase [8, 11]. However, given a high volume network, they are not able to analyse all traffic and to cope with the current environments.

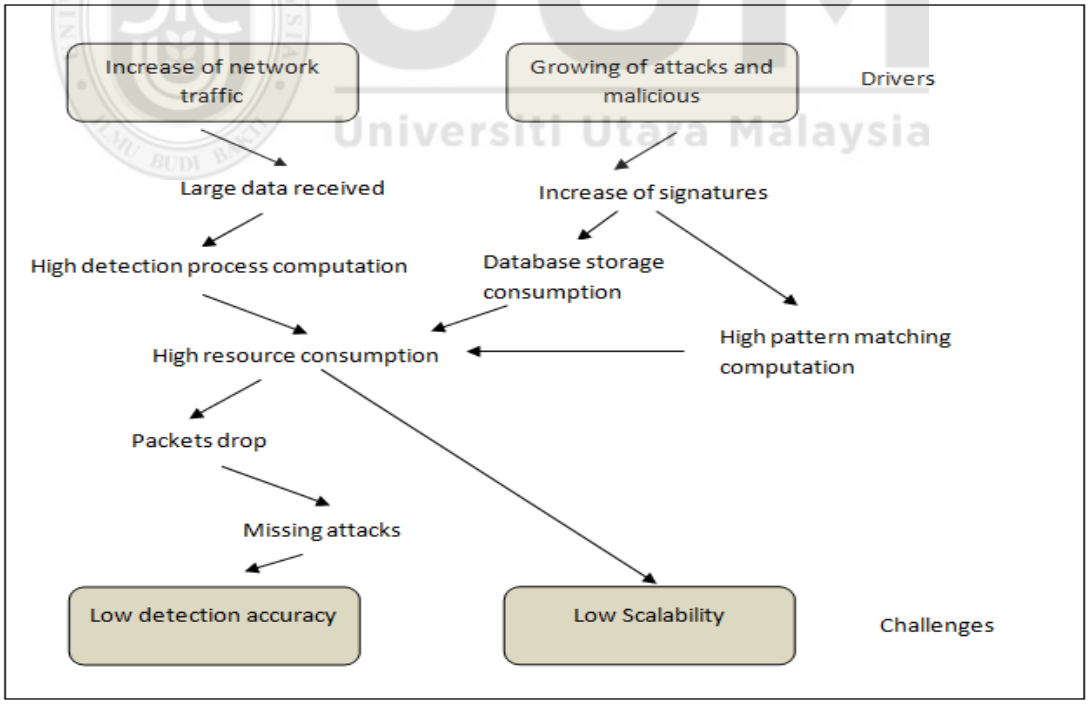

Figure 1.1. Challenges and Consequences for NIDS

When the whole payload of a packet is captured and inspected by NIDS, many challenges are faced. Capturing, scanning, and analysing the whole payload are very time-consuming. In addition, a large amount of data received from high volume network requires much computational performance and resources regarding hardware, capturing traffic, and detection process. Moreover, a drop in packets, resources consumption, and missing potential intrusions will occur if the NIDS is not able to let the analysis process be done [12, 13]. The lost packets may contain malicious data and threats [13]. Thus, false negatives rate increase, on the other hand, negative impact occurs on NIDS detection accuracy.

Growth and fast emergence of attacks and new threats are continued to increase. As the number of attacks increase, the number of signatures and profiles also increase. Hence, they have to be updated and stored in the NIDS database periodically. In this case, database storage must be able to accommodate these patterns. Thus, resources such as database size are very critical issues. Matching streaming payloads against thousands of rules is computationally intensive since it requires more effort for scanning with the huge size of the database, which leads to high resource consumption. NIDS that is based on payload inspection show that up to 80% of the total processing is spent on pattern matching [14]. Thus, there is an urgent need for higher processability of NIDSs.

1.4 Problem Statement

Although some improvements have been made for NIDS over the last couple of years, it still has some limitations. Researchers still struggle to make NIDSs process as little

amount of data as possible wi These two requirements are currently attracted by researchers thout compromising detection accuracy [15] . Also, there are lacks of the current computational frameworks, and identifying the right portion of the payload to be processed that can handle the growing of network traffic, the attacks, and threats [16]. In payload inspection, it implies that it “pays load” and overhead to the system. However, a vast amount of data requires a vast amount of computational performance, particularly complex algorithms. In other words, systems that are capable of monitoring every packet on a high-speed network are very expensive and high resource consumption. Although sampling techniques are used to reduce the load of NIDS, it may ignore potential packets that carry malicious traffic.

When the packet-based NIDSs have to inspect the whole payload of every incoming packet, it is difficult to cope with large volume traffic. As it is very time-consuming, therefore it is hard, or even impossible, to perform packet-based approach in this environment [9, 15, 17]. In other words, packet-based NIDS performance does not meet with the first feature of the efficient NIDS which is “scalability”. Packet-based NIDSs have attracted intensive research efforts [10, 18]. Although these systems demonstrated promising detection results, they suffer from poor scalability when they analyse a large volume of network traffic in high-speed networks.

Although the future approach, flow-based NIDS, has peace with modern high volume networks, it suffers from producing false positive alarms [9, 16]. The false positive generation occurs because the limitation of information in the flow-based approach and inability to access the raw data packets that might help for further investigation.

Also, the complexity of malicious and non-malicious (benign) network traffic characteristics allows flow-based NIDS to face challenges when distinguishing between these traffic [19]. In other words, the flow-based approach does not meet with the second feature of the efficient NIDS which is “detection accuracy”.

As a result when false positive alerts occur, the network operator has to receive a lot of irritating alarms which are not intrusions. However, when overloading occurs with these false alarms, system resources become exhausted, and the network infrastructure protection system becomes weaker. In addition, the value of the intrusion alert will be diminished and may not be noticed by this huge number of alerts. If the shortcomings of both intrusion detection techniques are examined, a trade-off if found between the full data available for detection in packet-based that leads to high resource consumption and the limited information available in flow-based for scalability that leads to high false alarms.

For detection accuracy in flow-based detection approach, several works showed the ability to detect malicious traffic. However, these researchers reported a significant number of false positive alerts [20, 21]. Although, a few works combine flow-based and packet-based detection for enhancing NIDS scalability and reducing false positive rates [9, 22, 23], their works were challenged since all full-payload packets, regardless they are suspicious or non-suspicious, are processed by packet-based NIDS that leads to a significant overhead consumption. Also, in their work, high-speed volume measurements were not considered.

1.5 Research Questions

The main research question of this research is: how to improve NIDS scalability while preserving detection accuracy and the following sub-questions are raised up:

i. How to improve NIDS efficiency based on flow-based detection approach?

ii. How to mitigate the false positive alerts generated by flow-based NIDS approach?

iii. How to evaluate the efficiency of a new proposed NIDS mechanism?

1.6 Research Objectives

The main objective of this research is to develop a mechanism named Conditional Hybrid Intrusion Detection (CHID) to improve NIDS scalability while preserving detection accuracy. To achieve this objective, the following three sub-objectives are identified:

i. To investigate the researches that can improve NIDS efficiency.

ii. To develop appreciate NIDS mechanism by reducing the rate of false positive.

iii. To evaluate the developed NIDS mechanism by measuring its efficiency through experiments.

1.7 Research Contribution

The contribution of this research is to add to the body of knowledge a new hybrid mechanism named CHID, that integrates and combines the two NIDSs approaches: packet-based and flow-based, to enhances the NIDS scalability and detection accuracy. Moreover, the mechanism. This mechanism utilizes the advantages of both approaches and overcomes their drawbacks. In other words, NIDSs process is scalable without compromising detection accuracy. The main contribution of this research can be stated as follows:

- An efficient flow-based detection mechanism that analyses the flows for suspicious identification.
- CHID mechanism that combines packet-based and flow-based detection approaches and provides scalability while preserving detection accuracy.

1.8 Scope of the Study

The proposed mechanism may not detect all ranges of attacks. Thus, specific types of attacks are considered. IRC and Peer to Peer (P2P) botnet attacks are considered to be the most strong–threat to the security of Internet-connected users and systems [24]. These attacks are selected in this research since they have frequent sequential patterns that involve several connections to or from the particular host in a short time [25, 26]. However, this characteristic fits the proposed NIDS mechanism since flow-based detection yields promising results when detecting botnet activities that perform repetitive traffic patterns.

Since this research deals with network traffic which flows through the segment, NIDS is considered in this research while HIDS is neglected. In addition, HIDS alone seems to be ineffective in current botnet attacks [27]. On the other hand, NIDS is mostly used today and promised high detection rate for botnet attacks [5]. Also, NIDSs have the following advantages over HIDS: in contrast to HIDSs, deploying a new host in a network does not need any configuration to that host to be protected from intrusions. For updating and maintaining IDS in a single network, it is easier to update one device such as NIDS, rather than updating multiple IDS devices for each host. HIDS does not have the capability to evaluate the network traffic.

NIDS efficiency depends on traffic amounts to be processed, detection algorithms, packet-capturing system, and the hardware. However, this research focuses on the amount of traffic fed to NIDSs for analysis. For detection method, signature-based for both flow-based and packet-based is used. Finally, in this work, the focus of detection accuracy improvement concerns the false positive rate.

1.9 Research Steps

To accomplish the goal of this research, Figure 1.2 shows research steps carried out in this thesis. The first step is to perform an in-depth study on the existing packet-based and flow-based NIDS approaches to identify their advantages and disadvantages and to explore the areas that should be eliminated or enhanced for scalable intrusion detection. Then the flow-based NIDS mechanism of IRC and P2P botnets should be designed and implemented. The third step is to perform performance evaluation of the

flow-based and packet-based NIDS mechanisms to analyze their effects on the accuracy of detection and resource consumptions.

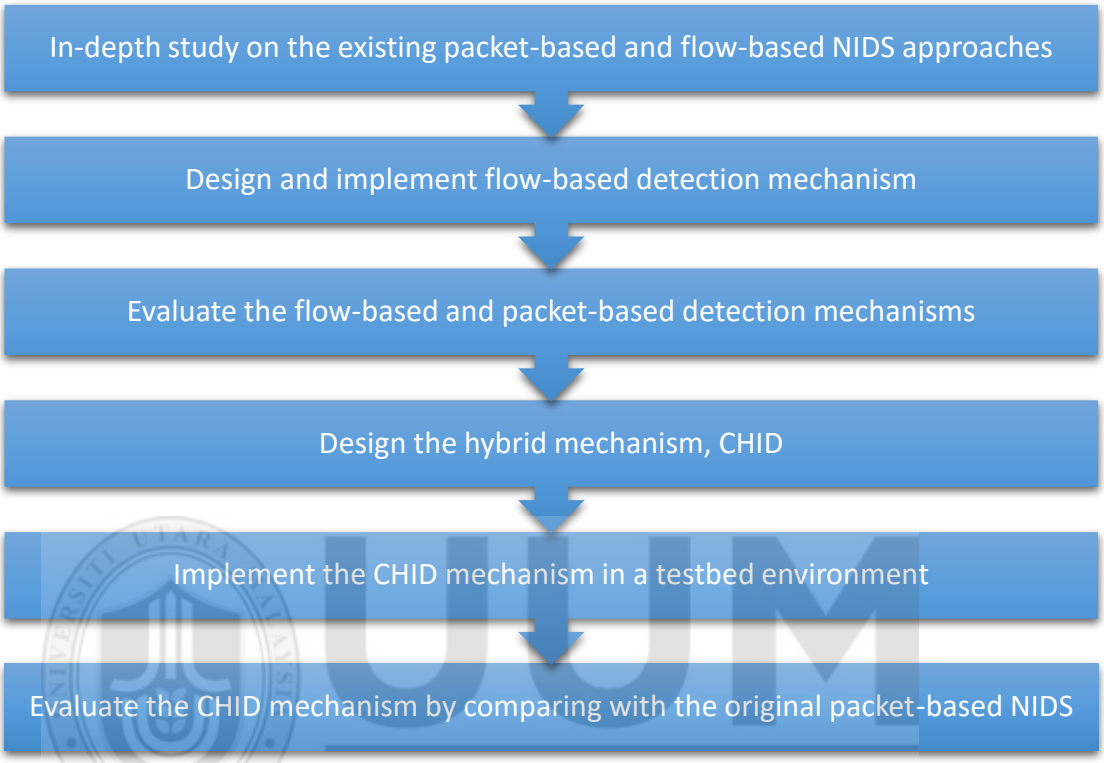

Figure 1.2. *Research Steps*

The next step is to develop a conceptual model by taking the advantages of flow-based and packet-based NIDS approaches and overcoming their drawbacks. Then the CHID mechanism that combines flow-based and packet-based detection mechanisms is designed and implemented in a testbed environment. The final step is to perform performance evaluation of the CHID mechanism by comparing with the original packet-based NIDS mechanism in term of accuracy of detection and resource consumptions.

1.10 Organization of the Thesis

This work is organized as follows: Chapter 1 presents the research motivation, problem statement, questions, objective, expected contributions and scope. Chapter 2 provides an overview of packet-based and flow-based NIDS approaches in term of scalability and detection accuracy. The chapter also presents related works that tackle the issues associated these approaches. More emphasis in this chapter is given the impacts and related works of false positive issues. In Chapter 3, research methodology of the study is explained in details. It presents the design of the proposed CHID mechanism. Then, implementation, evaluation, and validation of this research are discussed.

Chapter 4 studies how the flow-based NIDS approach can detect botnet malicious activities by implementing flow-based detection mechanism. Chapter 5 proposes CHID mechanism that reduces the false positive rate of flow-based detection and by combining flow-based with packet-based detection mechanisms. For Chapter 6, detection accuracy rate and resource consumption impacts on the flow-based detection mechanism compared with packet-based detection mechanism are presented. The combination approach (CHID) is then evaluated in term of resource consumption and accuracy detection level. Finally, the conclusion is presented in Chapter 7.

CHAPTER TWO
LITERATURE REVIEW

This chapter presents the importance packet-based and flow-based NIDS approaches for detecting attacks. Their advantages and challenges in high-speed networks are discussed in details. For operational definition, the combination of flow-based and packet-based detection approaches is defined as a mechanism. The term mechanism was also found in related studies that combine the two data sources [9, 23]. However, to distinguish the mechanism from the term "system", this mechanism is proposed to enhance the detection system as a central component.

This chapter is organized as follows: Section 2.1 explains how the Internet infrastructure, users, and threats are growing rapidly. Then types and challenges of IDS are presented in Section 2.2. Packet-based and flow-based NIDS and their related works regarding scalability and detection accuracy are discussed in details in Section 2.3 and 2.4 respectively. These sections also present related works in botnet detection. Section 2.5 present comparisons between flow-based and packet-based detection. In Section 2.6, related works on false positive reduction are discussed.

2.1 Growth of Traffic and Threats

The evolution of the Internet users is incredible as shown in Figure 2.1. In 2005, the number of users was about 1 million while it was about 3.4 million users by the end of 2015 [1]. Fast increase of computers connected to the Internet has resulted in the growth of Internet traffic and bandwidth in wide area networks (WAN). An access speed of 10 Gigabits per second (Gbps) becomes normal. Most large corporations,

universities, and government networks are moving toward higher speeds of up to 10 Gbps. Optical fiber infrastructures, which were restricted only to large businesses and ISPs connection, have been installed widely in the backbone network to meet the demand for the bandwidth.

In addition to the increasing of computers, increases of Internet services and multimedia contents need high bandwidth. As a result, ISPs and network components upgrade their network capabilities to accommodate the future traffic. Equipment performance grows required to support high-bandwidth traffic. WAN devices that were operated at speed 2.5 Gbps in 2000, they are operated at more than 40 Gbps nowadays.

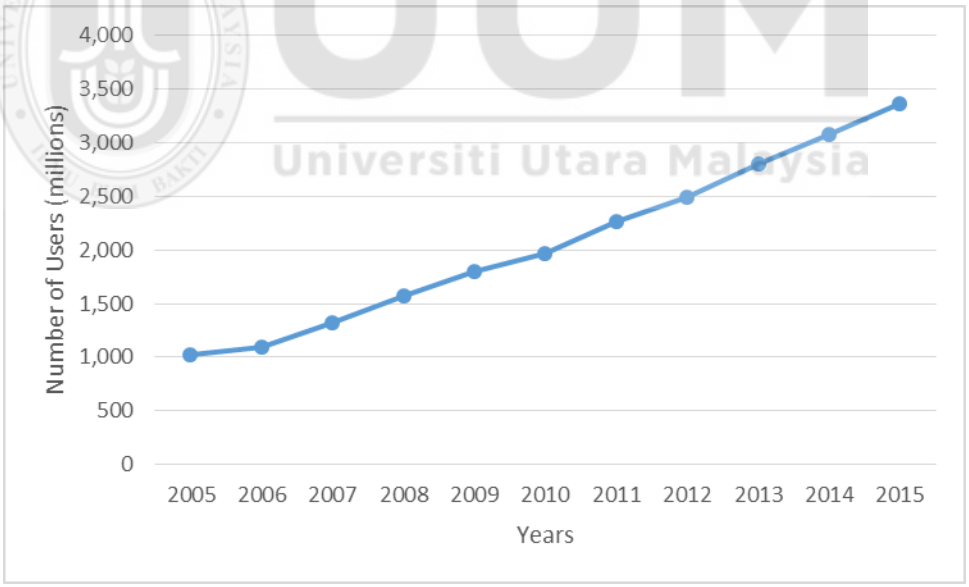

Figure 2.1. Expansion of the Internet Users over Years [1]

2.1.1 Internet Attacks

The Internet has its unknown dark face although it provides users comfortable life and business. When new Internet services, hosts, and devices are deployed, the number of vulnerabilities in the network components is growing; hence hosts are targeted by potential cybercriminals [8]. The vulnerability is a weakness, such as design flaws, or coding errors which an attacker uses it to compromise availability, confidentiality, or integrity of a host. Any threat that discovers and abuses the software and hardware vulnerability is called an attack. Unfortunately, attacks cause high damage cost to the organization. Resources affected by these attacks may have different symptoms such as denial of services (DoS) or data lost. Appendix A presents brief taxonomy and classifications of the Internet and computer attacks.

All types of network, including intranet, LANs, and large-scale computational grids, can be negatively affected by these attacks. Besides the hardware components, all exposed operating systems and protocols are also exploited by the crime activities which results in compromising the applications such as databases and web services. Such activities are conducted by installing malicious software (malware) programs on the victim's host. The user then executes these programs unintentionally to be infected by the attacks. Also, one of the most challenging risks today is insider threats. It is performed by internal users either by infecting the systems with malicious codes, or exposing or misusing private information.

2.1.2 Growth of Threats

Both attacks and Internet services are developing, and the race between them never stop [28]. The scope of attackers and malicious software has been changed significantly in a previous couple of years. Also, complexity and fast emergence of new attacks on the Internet are also growing. However, no in-deep technical knowledge is needed to create new attacks and malicious signatures.

When the number of devices is increasing, the number of vulnerabilities is also increasing. Hence the chances of threats are higher. Based on reports from 2016 Internet Security Threat Report [29], Symantec discovered more than 430 million new unique pieces of malware in 2015, up 36 percent from the year 2014. Also, 1.1 million web attacks were blocked per day on 2015 compared with 493 thousand blocked on 2014. They also claimed that the new mobile vulnerabilities and total identities exposed are also rising as shown in Table 2.1. The attackers are the ones who take the first steps, by creating new attacks, and security professionals have only to response.

Table 2.1

Growing of Threats [29]

Items	2013	2014	2015
Number of web attacks blocked per day	569K	493K	1.1M
New mobile vulnerabilities	127	168	528
Total identities exposed	-	348M	429M

2.2 Intrusion Detection Systems

In this section, the IDS structure and types of IDS are presented in term of locations, detection methods, and data type inputs. Certainly, these are not an exhaustive list of known types. Instead, the most widespread and interesting approaches related to this research are selected. Requirements of NIDS to be efficient are also discussed. Finally, problems that are faced by NIDS are presented.

2.2.1 Security Tools and Techniques

Computer security device should provide confidentiality, integrity, and availability, where confidentiality ensures information access to the only authorized party, integrity ensures information remains unchanged, and availability ensures accessibility of information by authorized parties whenever needed. Considering the damages and consequences of the attacks, it is important to find tools to detect attacks as soon as possible. One of these tools known as IDS which refers to detecting attacks or any unauthorized activities performed on a computer or a network. It detects unwanted exploitation to a computer system, both through the Internet and Intranet.

How IDSs differ from firewalls? Firewall is defined as a piece of hardware or software program which functions in a networked environment to prevent some communication forbidden by the pre-defined security policy. Firewall differs in the sense that it does not usually have the capability to search for anomalies or specific content patterns, such as spamming and worms, to the same degree as IDSs do. For these reasons, IDSs must be at the first line of defence and work along with firewalls.

An Intrusion Prevention Systems (IPS) is also a tool that detects attacks similar to IDS. IPS is the next security layer that combines the protection of firewalls with the monitoring ability of IDSs to protect networks with analysis necessary to make the proper decision on the fly. IPSs are set in *reactive* mode and designed to sit inline with traffic flows and prevent attacks in real-time. Since IPSs are inline inspection, they block traffic independently without human interaction. Therefore, the main disadvantages of IPSs are the serious consequences when blocking useful traffic (when false alarms rise) beside its bad performance in high volume networks.

Unlike IPS, IDS is mostly set in *passive* mode since it only raises an alarm in case of an intrusion and then needs human interaction to do further actions. Although it cannot directly block malicious connections, and to mitigate detected attacks, it sends commands to other devices of the network infrastructure that can filter the traffic, e.g. firewalls or routers with access control lists. Table 2.2 summarizes the difference between IDS, Firewall, and IPS.

Table 2.2

Differences between IDS, Firewall, and IPS

Feature	**IDS**	**Firewall**	**IPS**
Main task	Detect	Allow/ Block	Detect and block
Anomaly inspection?	Yes	No	Yes
Slow down the network?	No	No	Yes
Able to block intrusion	Manually and automated	Automated	Automated
Behaviour in detection	Mostly passive	Reactive	Reactive

2.2.2 IDS Structure

IDSs provides security management system for computer and networks. It detects potential malicious activities by gathering, analysing, and identifying information from different points of a network or on a particular host. An IDS can be defined as "the process of identifying and responding to malicious activity targeted at computing and networking resources" [3]. Though there are several implementations of IDS that share the common components to detect intrusions, these components include data collection (eyes), detecting engine (brain) with database (memory), and response components (mouth) as shown in Figure 2.2.

Data collection: It is also called "sensors" which act as an agent that monitors the data and traffic in real time. They are responsible for collecting, decoding, and pre-processing packets and make them available to the next component to detect intrusion. The input of the sensors can be system logs, system resources usage, network packets, or network flows.

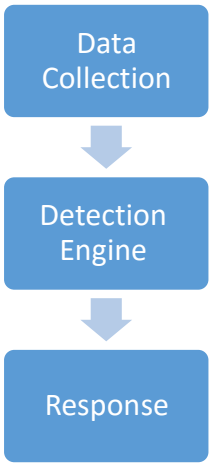

Figure 2.2. IDS Main Components

Detection Engine: It is the brain of the IDS and called “analysis module”, which is responsible for analysing, processing, and inspecting the input coming from data collection. In this module, IDS matches the input data with the database or threshold if needed, if a match is found, then it is an intrusion. Other analysis approaches are being introduced such as statistical analysis, pattern matching, artificial immune systems, and machine learning. Wu and Banzhaf [30] presented an overview and problems of different kinds of methods and algorithms that are used in IDS. The database stores all information about attack signatures, malicious patterns, and network profiles and behaviours.

Response: If an intrusion is detected, it sends an alarm or intrusion information to the operator to take further actions. The response can be active or passive. Most of the current IDSs are designed to be passive since they just send alarms when intrusions take place [8].

2.2.3 Software-based IDS

Having explained the structure of IDSs, this subsection presents the main IDS software in literature. The most software-based IDS used in the research community are Snort [31], Suricata [32] and Bro [33]. Snort is the most popular open-source IDS and used mostly for signature-based and packet-based detection [34]. It performs deep packet inspection using pattern matching in the form of rules. Rules describe network traffic data of internets in structured text files. When a security-related incident occurs, such as attack, these rules generate alerts. Suricata is also an open source application for intrusion detection [32]. Similar to Snort, Suricata uses rules for pattern matching.

These rules are compatible with Snort. Suricata is written in C language, and the modules have to be also writing in C which requires more expertise than the Bro language. Thus, Suricata may not be the best prototyping tool available.

Bro is open-source network IDS, and it monitors and inspects all traffic to detect suspicious activity [33]. It was developed by Vern Paxson of Lawrence Berkeley National Labs and the International Computer Science Institute. Bro is selected in this research since it has been primarily developed as a research platform for intrusion detection and traffic analysis. In addition, it was proved by Svoboda [10] that Bro is a useful tool for effective development of proof-of-concept and fully functional prototype. Also, it provides features through its script analysis engine and capability to extend the response via script; and remains best suited for high throughput research environments [4].

Bro is divided into three layers:

1. *libpcap*, packet capture.
2. *event engine* reduces the packet stream into a series of events.
3. *event handlers*, policy scripts.

It receives the captured traffic as raw network packets or flows, extract the meaning of them, and put them into context or stream. It represents this context through *events* "engine". In contrast to Suricata and Snort, Bro is not rule-driven in which it implements a scripting environment for creating rule-based detection. In other words, Bro detection rules are described by the scripts which use Bro programming language.

This language is an interpreted in a domain-specific type (e.g. `addr` type holds an IP address). Bro provides scripting language, and it comes with a large set of pre-built functions, yet the user can put Bro in novel ways by writing the own policy script. Bro policy script is the basic analyser used to specify what actions to take and how to report activities.

2.2.4 IDS Types

There are several types and taxonomies of IDS in the literature presented in [6, 15]. Figure 2.3 presents the main types of IDS concern in this research. Each type is explained in the following sections. Before IDS types are presented, several terms that are stated in this thesis should be defined.

False negative (undetected attacks): occurs when an IDS fails to identify an intrusion when one has taken place. This event should not be generated for high detection accuracy.

False positive (false alarms): occurs when an IDS incorrectly identified an intrusion when none had taken place. This event should not be generated for high detection accuracy.

True negative: occurs when no alert is raised, and no intrusion has taken place. This event should be generated for high detection accuracy.

True positive: occurs when an IDS correctly identified an intrusion when one had taken place. This event should be generated for high detection accuracy.

The following subsections present the types of IDSs based on location, detecting method, and process data.

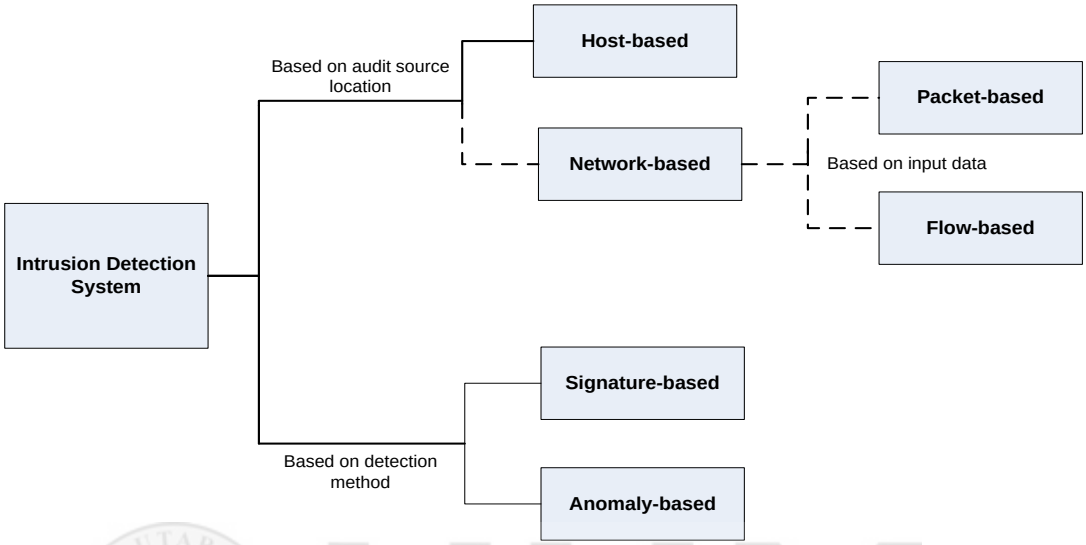

Figure 2.3. Types of IDS (dash line indicates the research scope)

2.2.4.1 Location-based IDS

IDSs can be divided into two classes based on their source audit locations: Host-based IDS (HIDS) and Network-based IDS (NIDS). HIDS resides locally on a single computer and protects this particular host from intrusions and attacks. It monitors data that resides on the host such as system logs, event logs, the local registry, and resource usage. Computers, where HIDS can be deployed, can be a server, workstation, or notebook. For NIDS, it captures and monitors traffic packets on the network and analyses them to discover if an attacker is attempting to hack the system (see Figure 2.4). Attacks such as DoS can be detected by NIDS. NIDS may be installed on a specific computer to watch its own incoming and outgoing traffic or can be installed

independently, such as in routers, to watch all network traffic; hence all machines in the network are protected.

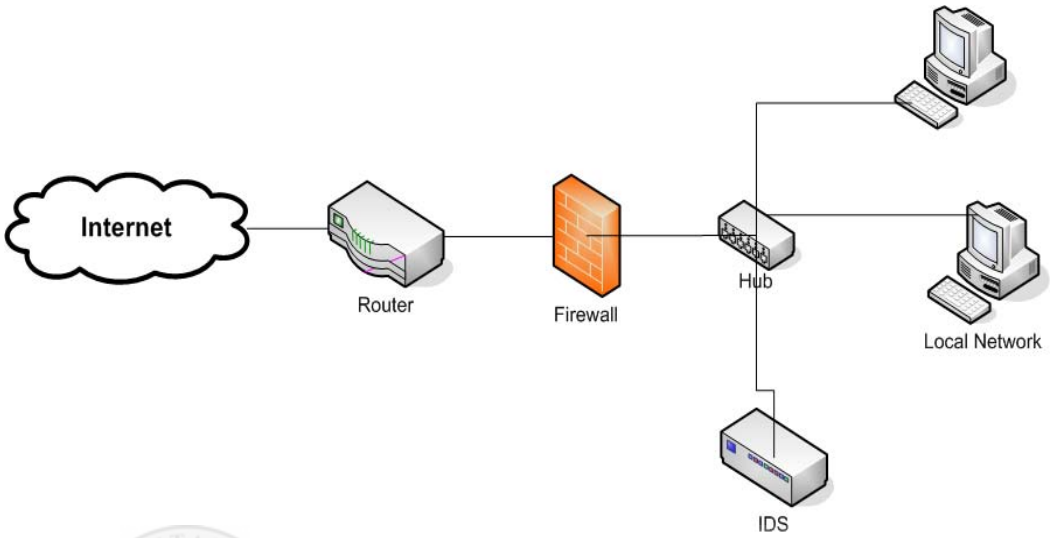

Figure 2.4. NIDS Location

Nowadays, a hybrid approach that combines HIDS and NIDS is implemented to provide a greater degree of security level. NIDSs have the following advantages over HIDS: in contrast to HIDSs, deploying a new host in a network does not need any configuration to that host to be protected from intrusions. For updating and maintaining IDS in a single network, it is easier to update one device such as NIDS, rather than updating multiple IDS devices for each host. This research adopted NIDS since it is mostly used today and promised high detection rate [5]. Also, NIDS has the capability to evaluate the network traffic on the main traffic gateway, while HIDS has the ability to analyse only its own (incoming/outgoing) network traffic.

2.2.4.2 Detection-based IDS

In term of detection method, the two primary methods are signature-based and anomaly-based IDS. Some IDSs combine the two approaches to providing more broad and accurate detection. A signature-based IDS also named “misuse-based IDS”, works similar to an anti-virus program. A signature is a pattern that represents a known threat. Every intrusion has its signatures and patterns. Examples of signatures may include an email with a subject named “Free Picture” with attached filed named Trojan.exe or with a certain message in payload content. These signatures are stored in a database. Signature-based IDS then simply compares what it analysed to the given list of signatures in the database. If a successful match is found, an alert is raised. Snort [31] and Bro [35] are two well-known examples of this type. They are open source and they monitor network traffic and then match each packet they observe against a set of rules and events.

For an anomaly-based IDS, also called “behaviour-based”, it does not deal with signatures; instead, it deals with behaviours of users, hosts, network connections, and services. It builds models of normal (benign) traffic profiles and patterns by collecting characteristics of system and network behaviours over a period of time and when the absence of intrusions. This stage is called training or learning stage and can be developed manually and automatically. When anomaly-based IDS take place, it uses statistical methods or neural network algorithms to compare what it is analysing (input events) to the related models. If any significant deviations or changes fall outside the predefined model is found, the traffic is marked as an anomaly (abnormal).

Anomaly-based IDS can detect new and unknown attacks. For example, when a host is infected by a new type of malware, the malware could initiate multiple connections, consume the network bandwidth more than expected, send a huge number of emails, or perform any significant behaviour that differs from the normal profile of the host. However, anomaly-based has some major drawbacks that include:

- Building and defining normal models are a critical issue. If the models are defined too narrow, some normal activities are marked as intrusion so that false positive rates increase. On the other hand, if the models are defined too broadly, some attack may not be detected; hence false negative rates increase [36].
- Since the network is dynamic, new services and maintenance and user habit keep change; the normal profile must be kept updated and regenerated periodically to avoid false negative [37].

As a summary, signature-based NIDS is considered in this research since anomaly NIDS approach faces challenges when building the normal profile of the network environments. Thus, it must continually adapt to the changing network environment. Otherwise, this approach may suffers from generating false negative and/or false positive rates.

2.2.4.3 Data-processed-based NIDS

This subsection presents the types of NIDSs based on the data processed. The process of NIDS requires data to be processed and analysed by the respective detection

methods and algorithms. Based on the type of data to be analysed in NIDS, there are two approaches: packet-based, flow-based NIDS. In packet-based, also named Deep Packet Inspection (DPI), a detecting engine has to scan and analyse the whole packet, header and payload, to determine whether the packet is an intrusion. Thus, it captures all traffic packets individually passing a certain monitoring point such as a router without ignoring any information. These packets are analysed by the detection engine either directly or remotely. These packets encompass layer 2 to 7 in Open Systems Interconnection (OSI). Signature-based NIDS, as discussed earlier, mostly uses packet-based for intrusion detection. All incoming packets are scanned and compared to every single rule of the database. In addition to monitoring all passing traffic, analysis based on certain protocols such as Internet Control Message Protocol (ICMP) that is based the headers only is possible; this is called "protocol-based NIDS". Further explanation about packet-based is presented in Section 2.3.

A flow-based NIDS does not look at the payload content for inspection and analysis. However, it relies on information and statistics of network flows. Such information includes a number of packets and bytes transferred over a particular time. Many routers and monitor probes can perform flow-based data collection and aggregation before NIDS analysis. A flow-based technique appears in the last ten years and widely used as data source not only for security purpose, but it was originally for network monitoring and analysis. Section 2.4 presents further details about this approach.

Anomaly-based and signature-based represent the detection methods and algorithms whereas flow-based and packet-based represent the data type fed into those detection

methods. However, not every anomaly-based NIDS uses flow-based approach. Some anomaly-based NIDSs use packet-based as input data [38, 39]. On the other hand, some signature-based NIDSs use flow-based for intrusion detection. This research focuses on this scenario when flow-based is used. Several researchers also work in this area [18, 23, 40]. In this case, the flow-based signatures describe network traffic by certain values, or ranges of values, of flow properties and statistics. The receiving flows are then compared with these signatures or thresholds for intrusion detection.

2.2.5 NIDS Requirements

There are many requirements for efficient NIDS mentioned in the literature. The two requirements that are focus in this research and consider the main requirements are scalability and detection accuracy [6, 7]. Weng, et al. [7] stated the key requirements of an NIDS that attracted researchers currently are scalability and detection accuracy. This is because they are the most requirements that influenced by increasing line-rates of network traffic and growing number of attack patterns. Another reason for selecting these two requirements is that with all progresses that NIDS have made over years, researchers struggle to make NIDS meet these requirements [8-10]. This is because of the trade-off between NIDS scalability and detection accuracy. In Sections 2.3 and 2.4, discussion on how the packet-based and flow-based NIDS impact on these two requirements is presented, respectively.

1. **Scalability:** NIDS should operate in large volume networks without resource consumption overhead, i.e. when all potential packets and traffic are analysed without packet loss. Thus, detection analysis should be performed smoothly in

a large data network as well as with increase traffic and network's size. Note that the term "potential packet" is used instead of "incoming packet"; this is because potential packets are extracted after sampling or filtering processes as will be discussed later.

2. **Detecting accuracy or detection rate**: beside all potential packets should be processed correctly; detection methods have to make the right decision, not to decide falsely. To achieve this requirement, the true-positive rate should be equal to the actual intrusions with fewer false positive and negative alarms.

Other requirements are also presented in [41, 42] (see Appendix B for more details) that include:

- **Detecting unknown attacks:** novel intrusion should be detected
- **Detecting encrypted traffic:** encrypted payloads should be readable and analysed for intrusion detection.
- **Early detection:** intrusion should be detected as soon as possible
- **Large data storage:** all potential signatures, profiles, alerts, and reports should be stored for long-term and further usage.
- **NIDS security:** NIDS should be secured enough against attackers who direct attacks into the NIDS itself.

- **Privacy:** NIDS should not violate privacy regulation of users by inspecting private information both in payload and header of the packets.

2.2.6 NIDS Challenges

The goal of this research is to propose an NIDS mechanism that meets the first two requirements mentioned in the previous section, which are scalability and detection accuracy. This section explains the reasons behind selecting these two requirements among others. There are several obstacles that hinder NIDSs to achieve and meet these requirements. Based on the survey conducted by [13, 19] these hindrances are as follows:

1. **The increase of volume traffic**: network traffic continues to grow which presents many challenges in the domain of security. NIDS have to receive a huge amount of data to be examined and analysed; hence negative impact occurs on NIDS scalability. In other words, storing and analysing of these data requires much computing for detection and resource consumption. Unfortunately, even if the NIDS processor speed increases, the increase of network speed is faster than the speed of NIDS techniques. Another impact of this issue is that when not all potential packets are inspected and analysed on time due to high resource consumption, some of these packets start to drop. The lost packets may contain malicious data and threats [13]. Thus, false negatives rate increase, on the other hand, negative impact occurs on NIDS detection accuracy.

2. **Growing of attacks and new malicious programs:** growth and fast emergence of attacks and new threats are continued to increase. As the number of attacks increase, the number of signatures or profiles also increase. Hence, they have to be updated and stored in the NIDS database periodically. In this case, database storage must be able to accommodate these patterns. On the other hand, matching process computation requires more effort for scanning with the huge size of the database, which leads to high resource consumption; hence negative impacts occur on both scalability and detection accuracy.

As a summary, the increase of network traffic and the growing of new attacks are considered to be the most challenges to NIDS, especially in packet-based detection system which will be discussed in the next section.

2.3 Packet-based NIDS

Having explained the structure, types, and challenges of IDS, this section discuss about packet-based NIDS. Most of the current NIDSs, whether signature-based or anomaly-based, use packet-based instead of flow-based approach [9]. The reason behind selecting packet-based approach is because it provides full information about the traffic which leads to better detection accuracy. This section emphasis on packet-based approach. Then the impacts of the packet-based NIDS on the two requirements: high scalability and detection accuracy, are presented. Filtering and sampling in packet and payload level are presented. Finally, attacks that are detectable by packet-based are discussed.

The components of packet-based NIDS are very similar to the traditional IDS structure described in Figure 2.2. Detailed view of packet-based NIDS is explained in Figure 2.5. In packet-based NIDS, the signatures or thresholds are stored in a database. Then NIDS simply compares the analysed packets to the given list of signatures or threshold in the database. If a successful match is found, an alert is raised.

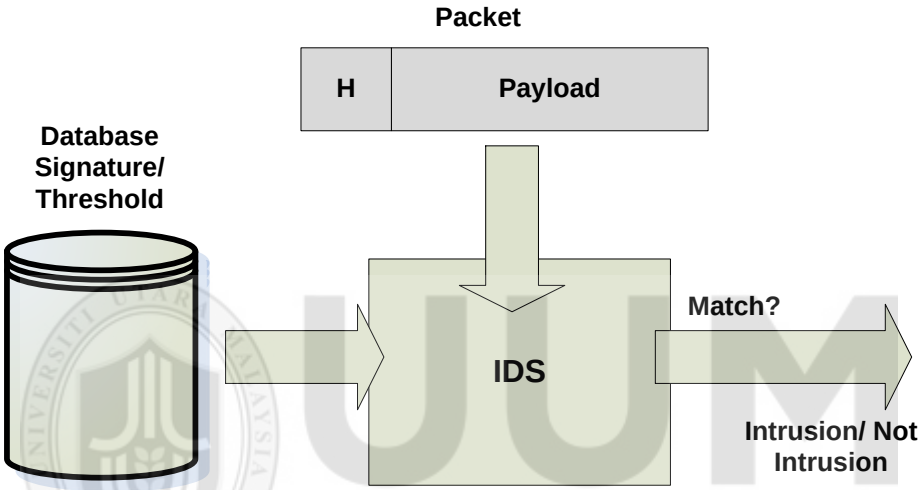

Figure 2.5. Packet-based NIDS

2.3.1 Scalability

With an increasing data volume in the traffic, the challenges of packet-based NIDS increase [8, 10, 11]. Many types of research attempted to improve the scalability of NIDS [9, 13, 43-46]. However, given a high volume network, they are not able to analyse all traffic and to cope with the current environments. The following points present how current environments influence the scalability of packet-based NIDS:

- **Computational Process:** When the whole payload of a packet is captured and inspected by NIDS, either signature-based or anomaly-based, many challenges

are faced. Capturing, scanning, and analysing the whole payload are very time-consuming. A Large amount of data received from high volume network requires much computational performance and resources regarding hardware, capturing traffic, and detection process. Researchers found that signature-based NIDS processing, which is based on payload, lies between 100 Mbps and 200 Mbps when commodity hardware is used and 1 Gbps when a dedicated server is used [47]. Korenek and Kobiersky [48] also stated that NIDS is difficult to keep up with traffic over 200 Mbps when full payload analysis is performed. Well-known packet-based NIDSs such as Snort show evidence of high resource consumption when dealing with a large amount of data found in high volume networks [9]. Matching streaming payloads against thousands of rules is computationally intensive. Payload inspection based NIDS such as Snort also shows that 80% of the total processing time of NIDS is left to string matching process [14].

- **Resource and Storage Capacity:** However, since the increase of signatures is still growing, resources such as database size are very critical issues. Dreger, et al. [12] analysed and showed the main drawback of the packet-based approach in term of resource and availability. These authors presented a comprehensive evaluation of packet-based NIDS performance in high volume network. They presented the key factors with respect to resource management and efficient packet processing. On the packet capturing side and far from the performance of NIDS itself, Braun, et al. [49] presented how the packet capture (headers and payloads) performance influence the hardware components such

memory and CPU in high volume networks. The metrics that are used in their investigation include the packets captured and CPU utilization. They analysed and evaluated the capture packet rates of different hardware and operating systems, thus, the valid conclusion can be drawn regarding significant impacts of packet capturing tasks.

Since the increase of computational process and storage capacity in packet-based NIDS depends on the amount of data to be processed, sampling algorithms are proposed to reduce the resource consumptions [50-53]. The sampling technique generally chooses a subset of packets which supposed to reflect the information of the entire all packets. The difference between filtering and sampling is that filtering has to know the traffic properties in advance. However, although filtering and sampling techniques reduce the overhead to NIDS, they require some processes and delay to achieve their job. For example, since filtering approach needs to classify incoming packets according to packet header fields, the comparison operation is required per packet. Also, filtering approach may ignore aggressive and potential packets that might have valuable information for intrusion detection [54]. Thus, much care is needed when choosing these approaches. In this research, packet filtering process is proposed based on suspicious flow-based IP addresses generated from another node.

Several approaches are also designed and proposed in recent years to reduce resource consumption of packet-based NIDS in hardware aspects [55]. Braun, et al. [49] designed detection process to enable string matching at several gigabits per second to improve the performance of NIDS. A mechanism that reduces the amount of packet

to be processed was also proposed by [56]. The authors introduced pre-filtering process in their approach by indexing the header information of all incoming packets. However, the cost of hardware-based NIDS is a big issue. Also, since some signatures contain regular expressions, approaches in [49, 55] have limited advantages for signature analysis [57].

2.3.2 Detection Accuracy

All common types of known attacks can be detected, provided that all potential traffic be analysed. Since packets contain all complete payload and headers up to application layer (layer 7 in OSI), packet-based NIDS can detect application intrusion easily. Header information is very useful to identify attacks targeting vulnerabilities of network stack implementation or scanning the networks to identify live services. On the other hand, payload information is useful to recognize attacks targeted vulnerabilities at application layer since the successful connection session can be established in a normal way. In other words, the attacks detected by packet-based NIDS only range from attacks which are more connection-oriented to attacks occur only in network payload.

Packet-based NIDS mostly provides signature-based NIDSs sufficient information to detect attacks. The main advantage of the packet-based technique is the low false positive rate. However, this advantage is difficult to be achieved in high volume networks. If the NIDS speed is not high enough to analyse all potential payloads, a drop of packets may occur. Thus, potential intrusions are missed and false negative

rate increases [13]. The following attacks can be detected using packet-based NIDS approach:

- DoS attack: The main objective of DoS attacks is to deny a legitimate user from using or accessing his/her system in a normal mode. When this attack uses a large number of attacked computers against target or targets, it is called distributed denial of service or (DDoS).
- Information Gathering and Scanning: These attacks try to gather information about the system and network for further attacks.
- Malicious Software: Malware includes Worms, Virus and Trojan horse, are malicious programs that are inserted into a host to corrupt a system, deny access to a service.
- IP Spoofing: This kind of attack is functioning on networks and TPC/IP protocols. Network spoofing is used when the attacker pretends himself as a legitimate user by spoofing who they are. Session Hijacking is the most popular attack in this kind of attack.
- Password attacks: This attack involves when the attacker is attempting to guess a password of a protected host. Password dictionary and brute force are the main example of this attack.
- Botnet attack: This attack infects a huge number of computers to be part of controlled botnet. These computers attack the target based on the command received such as DoS.

More details about these attacks and other attacks can be found in Appendix A. This research focus on botnet attack, specifically on IRC and P2P botnets. The reason behind this selection is that the botnet attacks are considered to be the most strong– threat to the security of Internet-connected users and systems, among other attacks [24]. Also, these attacks have frequent sequential patterns that involve several connections to or from the particular host in a short time [25, 26]. This characteristics are helpful for flow-based NIDS detection. Since this research focus on botnet attack, the following subsection present how this attack can be detected using packet-based NIDS.

2.3.3 Botnet Detections Related Works

This attack uses a large number of compromised computers (bots) to direct coordinated attack against target or targets. Figure 2.6 presents how botnet works and can be summarized in the following phases:

1. Propagation or Infection phase: The attacker (or so-called botmaster) infects a huge number of machines to be part of controlled botnet. The victim machines install bot (or backdoor) software downloaded from web or email link.

2. Command and Control (C&C) phase: These bots are controlled by a botmaster using C&C channels for communications. These channels can be centralized structure as shown in Figure 2.6 (e.g. when Internet Relay Chat (IRC) or Hyper Text Transfer Protocol (HTTP) are performed) or decentralized as shown in Figure 2.7 (e.g. when Peer to Peer (P2P) communications are performed among peers).

3. Behaviour or attack phase: Finally, the bots attack the target based on the command received from botmaster. Such attack may include DoS, phishing, and spam.

Since the first phase involves in executing bot software, it could be detected in host-based NIDS. However, this research focuses on other phases where NIDS plays a significant role in detecting botnet. Table 2.3 shows the main botnet attacks that are focused in this research and use command and control channels. Karim, et al. [58] presents a comprehensive review of the latest state-of-the-art techniques of botnet attacks.

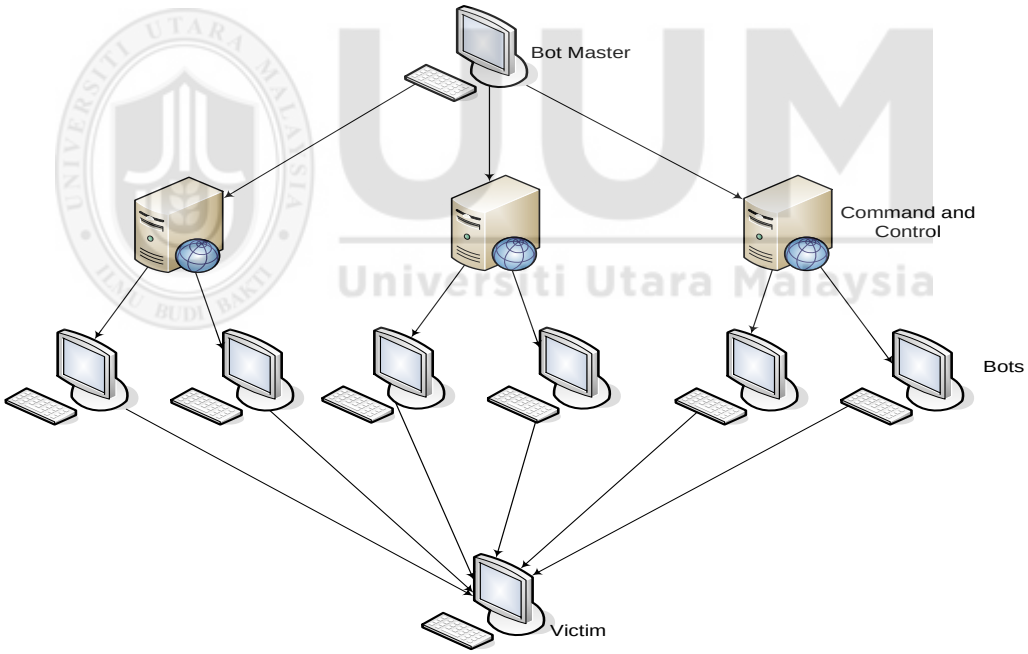

Figure 2.6. Centralized Botnet Attack Methods

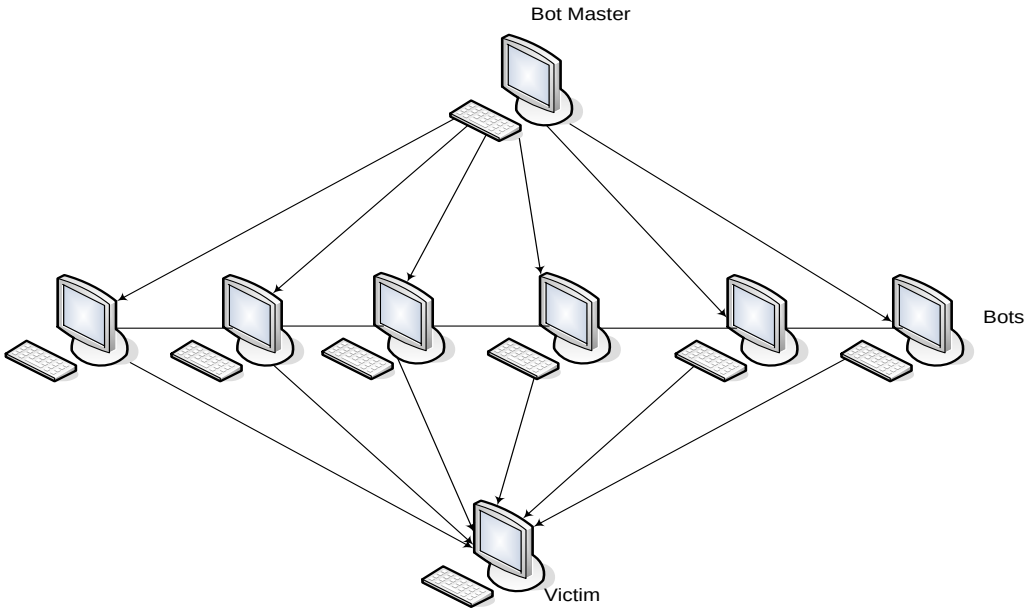

Figure 2.7. Decentralized Botnet Attack Methods

Table 2.3

Examples of Botnets Attack [25]

Name	**Architecture**	**Protocol**	**Types of Infections (propagation)**	**Communications Intervals**
Storm [59]	P2P	UDP	Spam, social engineering techniques	Frequently
Waledac [60]	P2P	TCP/UDP	Drive-by downloads, spam, social engineering	Frequently
Zues [61]	Centralized/ P2P	TCP/UDP /HTTP	Spam, Drive-by downloads	Frequently (every 2 minutes)
IRC-bot [62]	Centralized	TCP	Chat, spam	Frequently

Several detection systems were proposed to detect botnet attack. These systems include BotMiner [63], BotSniffer [64], Rishi [65], and Traffic Aggregation for Malware Detection (TAMD) [66]. These methods have the following characteristics: 1) performing full packet inspection to process packet content, and 2) observing attack patterns (such as scanning and sending spam emails) initiated by botnets. BotMiner, for example, can detect both centralized and P2P botnets by identifying groups of the hosts that share similar communication network characteristics and similar att-acks. It assumes that the bots belonging to the same botnet would share similar C&C communication patterns by responding similar commands to the bot master.

On the other hand, Rishi, TAMD, and BotSniffer, can detect C&C botnets by identifying synchronized traffic communications that share similar packet content. However, modern botnets start to use more stealthy ways, e.g. spam-bots that send spam via stolen webmail accounts like Gmail, to non-observable, thereby making these systems ineffective [18].

Bothunter [67] also was proposed to detect infected bots based on a pre-defined infection model. In other words, if the host behaviours are consistent with the infection model used by BotHunter, then the host is considered as a bot. This method detects both centralized and P2P structure. However, nowadays bots use a wide variety of approach for infection which may not be consistent with infection model of Bothunter [18]. Table 2.4 presents the main botnet detection methods using packet-based approach.

Table 2.4

Botnet Detection Methods with Packet-based Approach

Botnet Methods	P2P Detection	IRC Detection	HTTP Detection	Generic Detection
BotSniffer [64]		√	√	
BotMiner [63]	√	√	√	
TAMD [66]		√	√	
Rishi [65]		√		
BotHunter [67]		√		
Wurzinger, et al. [68]	√	√	√	
Markuf [69]				√
Synchronism [70]				√
Jian, et al. [71]	√			
Dan, et al. [72]	√			

√: means protocol is supported; P2P: Peer to Peer; IRC: Internet Relay Chat; HTTP: Hyper Text Transfer Protocol; Generic: detects any protocol

Since the huge volume of traffic requires great scalability for NIDS, detection methods need to process a large volume of traffic efficiently. However, as stated by Zhang, et al. [18], most of these detection methods rely on deep packet inspection (DPI) to process packet's payload which make their scalability is significantly constrained. In other words, these botnet detection methods may not be able to analyse all network traffic related to bot-infected hosts when these methods are deployed in high-speed network environments, and thus fail to detect these bots.

2.4 Flow-based NIDS

With the issues of packet-based NIDS, researchers had to find an alternative approach that receives a little amount of data while not compromising the accuracy. The candidate alternative that attracts the attention of researchers is flow-based NIDS technique. In this section, before discussing the scalability and detection accuracy of this approach, the state-of-the-art of this approach is presented. The main components of flow-based NIDS are also explained. Then the attacks that are detectable by flow-based only are presented.

2.4.1 Flow-based Detection Overview

Flow-based NIDS looks at aggregated information of related packets of network traffic in the form of flow record. Thus the amount of data to be processed is reduced. Moreover, it does not provide any payload to NIDS; it rather provides statistics and patterns about network connection to be processed by NIDS to identify malicious flows. Thus, flow-based gathers information of a group of packets while packet-based gathers information about individual packets.

However, to make the flow information, other network devices such as a router is usually used. Such information includes number of packets and bytes delivered during the flow, start and end time of the flow. This information is formed as “flow record”, and exported to NIDS (either signature-based or anomaly-based) for further analysis and detection. Currently, there are two protocols that aggregate packets and exports flow format: NetFlow and IP Flow Information Export (IPFIX).

- NetFlow protocol: is originally developed by Cisco. NetFlow version 9 is the latest version and version 5 is more popular [73].

- IPFIX protocol: is considered as the successor of NetFlow protocol and it was developed by Internet Engineering Task Force (IETF) working group [74].

These protocols are designed purposely for network statistics but later it was utilized for security advantages. However, any network device such as a router, probe, or switch with Flow-enabled can support these protocols. Wherever the term *flow* is mentioned in this thesis, the following definition is applied: a flow can be defined as a unidirectional data stream between two computer systems where all transmitted packets of this stream share the following characteristics: IP source and destination address, source and destination port number and protocol type [75]. These characteristics are called flow keys. However, these flow keys can be configured by the user in a flexible way. Several statistical information can be obtained from these flows using metering process include: 1) number of packets that are aggregated in a flow 2) Number of bytes in a flow 3) Start time when the first packet is received in a flow 4) End time when the last packet is received in a flow, and 5) TCP flags occurring in a flow.

These flow statistics, as well as flow key, are encapsulated and stored in the flow record datagram. Figure 2.8 shows how three packets, for example, corresponding to a flow record. The flow keys in this example are source and destination IP address, source and destination port number, and protocol. Where as shown in Figure 2.8, the flow record eliminates payload and calculate the length of all packets. The flow keys

and the statistical information are very valuable to detect intrusions with ignoring traffic payloads.

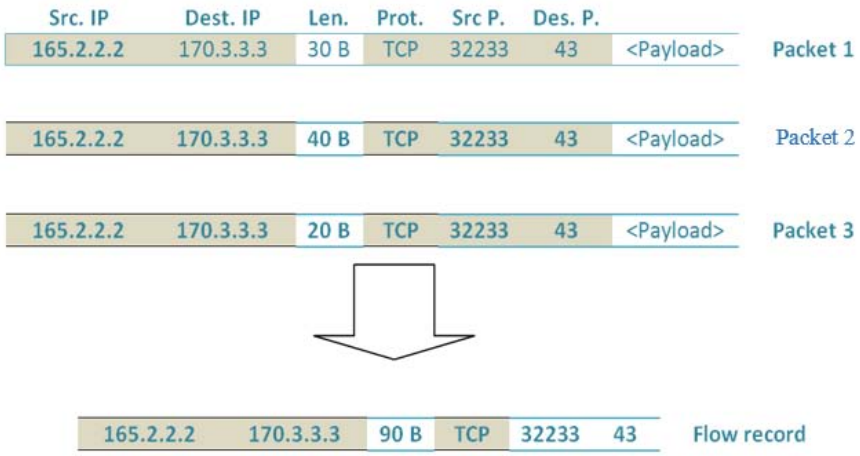

Figure 2.8. *Three Packets Corresponding to a Flow Record*

However, flow-based architecture is not as simple as packet-based NIDS. Also, packet capturing tools and packet sniffers such Wireshark [76] and tcpdump [77] differ from the flow-based approach. The only flow-based approach can support timeouts mechanism for exporting flow records [73]. Also, Wireshark and tcpdump do not have the ability to act as a collector so that the flows are readable by NIDS. While the only single packet is analysed in packet-based approach, flow-based converts a group of certain packets into flow record.

2.4.2 Structure

Flow-based NIDS consists of the following components: an exporter (flow aggregator), a collector, and the NIDS (or analyser) as shown in Figure 2.9. Each component is detailed in the following subsections.

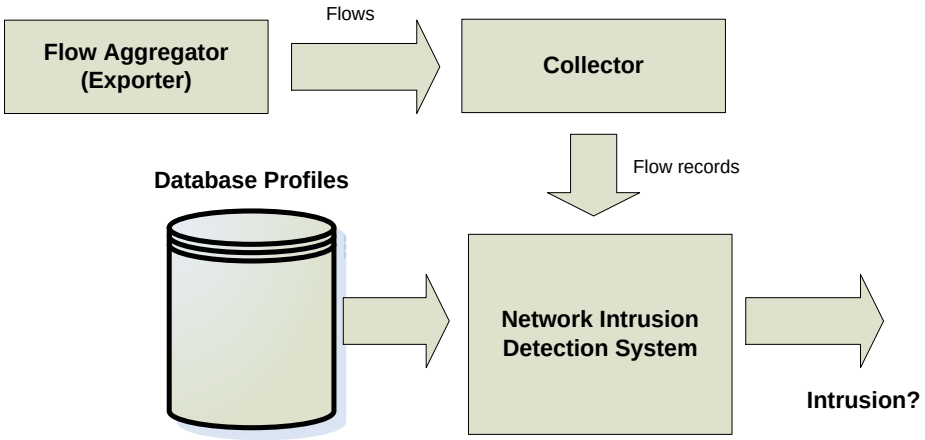

Figure 2.9. Flow-based Components [19]

2.4.2.1 Exporter

The exporter also named flow aggregator, can be considered as an observation point and it can be dedicated device or built-in router. The input of the exporter is the observed incoming potential packets. The main duty of the exporter is to create flow records by accounting traffic statistics from aggregating relevant packets that share certain flow keys. Flow aggregation means combining several data (from different packets) into a single composite (flow record) with discarding other data (such as payloads and some header fields of corresponding packets). The exporter starts creating flows when two hosts attempt to communicate even when no connection is established. Each header and timestamp of every potential packet are extracted when the exporter captures the packets.

However, these flow records are sent out to the collector when they expire. They expire in the following conditions (depends on the system configurations and can be changed according to security objectives):

- When the flow was idle for a predefined time (inactive timeout) when no new packet belongs to exist flows arrives.

- When the predefined maximum time is reached (active timeout). The active timeout ranges from 1 to 30 minutes.

- When TCP connection flow shutdown by Finish flag (FIN) or restart by Restart flag (RST).

- When the flow cache memory is full.

If any of the above cases occurs, corresponding flow records are exported to the collector. However, depending on the strategy of the exporter, a flow can be split into multiple flow records. The exporter can be located either inside the network for insider threats and malicious attacks or outside the network.

2.4.2.2 Collector

The collector's task is to retrieve the flow records created by the exporter and store, archive and organize them in a suitable format for NIDS for further analysis. It usually receives the flow record from multiple exporters using unreliable protocol UDP for the sake of performance. Also, the collector can be accessed through web-based end-frond software for getting information and statistics such as number of flow records and bytes.

2.4.2.3 Analyser

The analyser (detection system) then accesses to the collector to analyse and process the flows for intrusion detection. It then sends the decision to the reporter. Since flow-based NIDS processes a copy of the network traffic, although it can be near real-time, it is not an in-line device that can directly block malicious traffic. To mitigate the detected malicious, commands are sent to another device such as a firewall to filter or block this malicious traffic.

2.4.3 Scalability

NIDS in flow-based analyses small amount of data (flow records). Scalability issues, computational processes and storage capacity that appear in packet-based, are not that primary concerned. In large networks, however, storing flow records may be an issue, but much more affordable when comparing with the packet-based approach. Also, since the fact that more than one packets that have common properties are classified in one flow record, the number of flow records that has to be analysed is less than the number of corresponding packets. Thus, flow-based is the logical choice for NIDS in high volume networks [78].

However, it seems that the overhead and computational resources consumption in the detection analysis, whether by signature-based or anomaly-based, are disappeared but on the cost of flow exporting and collecting process. This is because that the exporter, or NetFlow device such as a router, offloads and absorbs the task from NIDS itself. It means that when the router is used to generate flow records, the router itself might be overloaded, hence a negative impact on the network, not on NIDS. This impact cannot

be seen in packet-based NIDS since no extra devices are involved. Also, the transmission of the exported flow records to the collector can consume a significant amount of bandwidth.

However, sampling approaches either in packet-sampling [79] or in flow-sampling level [73, 80] are proposed to solve these issues. However, several studies present the impact of sampling approaches on flow accounting [81] NIDS. As a conclusion, tradeoffs between reducing data (either in a packet or flow sampling) and supplying sufficient detailed measurement for intrusion detection are observed. As a consequence of packet sampling in flow-based NIDS, the properties and information of the original packets can be lost or at least estimated since the flows are shortened.

It is obvious that the scalability of flow-based NIDS depends mostly on the performance of the exporter [82]. If the amount of overhead in the network of flow-based NIDS is more or equivalent to the amount of overhead in the packet-based NIDS, then there is no advantage of using flow-based regarding scalability. Several works attempt to improve the performance of flow aggregation process such as [22, 73].

2.4.4 Detection Accuracy

The question remains whether flow-based provides enough information, compared to payload inspection, to be useful for intrusion detection. Since flow-based provides NIDS with only information about traffic statistics, patterns, and connections, attacks that are injected in payload might not be detected. Moreover, flow records contain aggregated data up to transport layer. Thus, some attacks, especially on the application

layer that carry malicious code, are only visible in the packet payloads. Researchers work very hard to detect these kinds of attacks. However, these kinds of attacks can be identified by flow-based NIDS if the attackers perform multiple attacks in series and cause many flows targeting the victim [19].

These issues encourage researchers to enhance flow-based detection accuracy and reducing the false negatives. Although this area of research is still young, promising results are achieved by researchers with focusing only on flow-based. When analysing the traffic statistics, patterns, and connections provided by flow-based aggregation, malicious and intrusion traffic patterns can be easily identified during attacks.

Many types of research in the literature proposed systems that detect malicious activities. For example, DoS attacks can be detected by using flow-based detection only [40, 57, 83, 84]. Abdulla, et al. [85] and Dressler, et al. [86] developed models that detect worm attacks by inspecting the flow of traffic only. In a recent study, several works proposed flow-based detection to detect SSH dictionary and brute force attacks [87, 88]. Also, flow-based detection only was also proposed to detect botnet traffic by [89-91].

Table 2.5 shows and classifies some of the attacks that can be detected by flow-based approach only. It also shows the type of detection methods used for each work. Appendix C presents how these works detect DoS, Worm, and SSH attacks. As a summary, flow-based NIDS is able to detect most of the attack mentioned in the literature. Several studies were proposed to enhance detection methods in this fields. The main disadvantage of these studies is the significant false alarm rate as will be

discussed later in this chapter. The following subsection presents how botnet attack can be detected using flow-based NIDS.

Table 2.5

Attacks Detectable by Flow-based NIDS only

Authors	Attack	Detection Method
David and Thomas [83], Yu, et al. [84]	DoS, and distributed DoS	Anomaly-based
Munz and Carle [57]	DoS	Hybrid-based
Kim, et al. [40]	DoS	Signature-based
Abdulla, et al. [85], Dressler, et al. [86]	Worm	Anomaly-based
Hellemons, et al. [87], Vizváry and Vykopal [88]	SSH and brute-force	Anomaly-based
Amini, et al. [89], Zhao, et al. [90]	Botnet	Not-specified

2.4.5 Botnet Detection Related Works

In this section, several studies on botnet detection using flow-based approach are presented. Several studies adopted machine learning techniques for detecting malicious activities. In machine learning, feature vectors are used to represent numeric characteristics, named features of an object in a mathematical and analysable way. On other words, it is a vector containing multiple elements about an object. These vectors are important for many different areas of machine learning because of the effectiveness and practicality of representing objects in a numerical way for the analysis process [92]. For example, to generate most significant attributes, Wrapper [93] subset evaluation is used to create all possible subsets from the feature vectors.

Wrapper approach basically finds appreciate features with a different set of features through a repetitive process of the classification algorithm [93].

Stevanovic and Pedersen [94] presents botnet detection method using flow-based data. They converted network traffic into network flows in which to be classified. Several classifiers were used for evaluating the validity of their method. They use real traffic traces of P2P botnets and background traffic for training and testing processes. The authors concluded that J48 [95], Random Tree [96], and Random Forest [97] algorithm were the most successful algorithms for their operations. Nogueira, et al. [98] also proposed the use of the flow-based approach to detect botnet activities among network traffic. The researchers employed a Neural Network model in conjunction with the flow-based approach. Their proposed model was evaluated by testing on traffic generated by P2P application, Skype. However, botnet activity was artificially generated. The detection of botnet activities using their method was quite successful but with the reasonable false positive rate.

Saad, et al. [99] implemented an approach to detect P2P botnet activity that is slightly different from the studies above. In addition, to using flow-based attributes, they used host-based attributes that are exhibited in communication between hosts. In their method, they use Support vector Machine (SVM) [100] as machine learning technique. The aim of their research was to meet three botnet detection requirements; adaptability, novelty detection, and early detection. They used traffic traces that represent real world scenarios for their experiments. However, the authors concluded that the proposed method did not adequately satisfy the three mentioned requirements

for effectively detecting botnets. For HTTP botnet detection, Haddadi, et al. [54] proposed a new framework based on botnet behaviour analysis. The authors employed machine learning algorithms on flow-based traffic using NetFlow software, Softflowd [101]. The implementation of proposed method used C4.5 and Naïve Bayes as learning algorithms. The results showed that the C4.5 algorithm obtained very promising performance on detecting HTTP botnet with a significant number of false positive alerts.

Zilong, et al. [62] developed a detection method for IRC botnet activity based on abnormal behaviour. They use NetFlow data as raw data. The main advantage of this approach is that it does not need application layer information for detection. However, this approach cannot detect real-time flow. BotFinder [102] proposed a detection method that senses HTTP and IRC bots using NetFlow dataset. This method leverages the discovery that C&C communication of certain bots trails specific regular patterns. Machine learning was used to identify the key features of these communications based on observing traffic that bots generated in a controlled environment. Based on these features, BotFinder creates models that can be deployed to identify bots. This approach achieves high detection rate but with generating false positive alerts.

Zhao, et al. [90] study the feasibility of detecting P2P botnet traffic without analysing a complete network flow. Machine learning was used for classifying behaviour based on time interval. The proposed method was based on Network flow while detecting real-time botnet by inspecting these flows in small windows. Using real datasets, they show that it is possible to identify the presence of botnet with high accuracy rate but

with the reasonable false positive rate. Finally, the method called Disclosure [103] proposed a large-scale botnet detection based on Netflow data and machine learning techniques. The method relied on flow attributes including source and destination IP address, source and destination port number, start and finish time of flow, and number of packets and bytes per flow. Also, flow size, and client access pattern have been used to distinguish C&C communication over normal traffic. As a result, the Disclosure can perform real-time detection of IRC and HTTP botnet with the reasonable false positive rate. Table 2.6 summarizes the related works discussed in this subsection.

As a summary, they use flow-based approach to detect various botnet attacks. However, they reported false positive rate in their studies. Also, it is stated by Wijesinghe, et al. [104] that high overheads were observed in their approaches in the aspect of dataset size, CPU utilized by machine learning techniques, and for IP flows capturing.

Table 2.6

Botnet Detection Methods with Flow-based Approach

Research	**P2P**	**IRC**	**HTTP**
Stevanovic and Pedersen [94]	√		
Nogueira, et al. [98]	√		
Saad, et al. [99]	√		
Haddadi, et al. [54]			√
Zilong, et al. [62]		√	
BotFinder [102]		√	√
Zhao, et al. [90]	√		
Disclosure [103]		√	√

2.5 Packet-based and Flow-based Detection Comparison

Comparisons between packet-based and flow-based approaches are presented in this section. Then the trade-offs between these approaches are discussed.

2.5.1 Comparison

Having explained the packet-based and flow-based NIDS and their related works regarding scalability and detection accuracy, Table 2.7 summarizes the differences between packet-based and flow-based NIDS. It shows some advantages of flow-based NIDS over packet-based detection. Unlike packet-based detection, since flow-based NIDS can provide valuable information to anomaly-based NIDS, flow-based NIDS can detect unknown attacks [41]. Also, encrypted payload does not influence the process of flow-based NIDS since no payload is analysed. In other words, encrypted malicious traffic can be detected using flow information without decryption [42]. However, the proposed approach used in this research has combination of flow-based and packet-based NIDS. In other words, the proposed approach attempts to take the advantages of flow-based NIDS and reduce the disadvantages of packet-based NIDS as discussed in Chapter 5.

2.5.2 Trade-offs

As mentioned earlier, the efficiency of NIDS depends mainly on the type of data to be processed: individual packet (with payload) or flows [105]. Packet-based provide full information to NIDS to detect attacks while flow-based provide limited and aggregated information to NIDS for intrusion detection. In term of scalability, since packet-based NIDS process large amount of data, it degrades its scalability, especially

in high volume network. On the other side, flow-based is a good choice for this issue since it deals with the small amount of data. In term of accuracy, Information gathered from packet-based is enough for NIDS to detect almost all kinds of attack, hence improving NIDS accuracy. On the other side, since the information gathered from flow-based is aggregated, NIDS accuracy suffers from false alarms.

Table 2.7

Comparison between Packet-based and Flow-based NIDS

Feature	**Flow-based NIDS**	**Packet-based NIDS**
Data to be analysed [106]	Flow records	Header and payload
Size of data to be analysed [43]	Small	Mostly large
Detection method mostly used [106]	Anomaly-based	Signature-based
Resource consumption [106]	Usually low	Usually high
Size of network preferred [43]	Small and Large network	Small network
Extra device needed[107]	Extra device is required to reform the traffic	No need extra device
Delay before detection analysis [107]	Has to delay since packets need to be aggregated	No delay
False positive rate [107]	High	Low
Allow to access raw packet data for further analysis [106]	Cannot access raw packet	Can access raw packet
Privacy [107]	Confidential data in payload is not compromized	Confidential data in payload is compromized

Flow-based detection promises to be able to process data in high volume network with limited data with a trade-off of a higher false positive rate, while in packet-based, it promises to be able to detect intrusion in low false alarms with a trade-off of higher resource consumption with additional data processing.

2.6 False Positive Reduction

Due to the complexity of characteristics of malicious and non-malicious traffic, NIDS based on flows faces challenges when distinguishing between this traffic especially when they have similar characteristics [105]. Such similar characteristics may include in flow duration, port number, and number of packets per flow.

It affects negatively on NIDS decision when considering normal traffic as intrusion, hence high false positive rate is a result [103]. However, enhancing the detection algorithm accuracy of NIDS has been a hot issue in research while relatively less for enhancing the false positive [108]. With the issues of the flow-based NIDS approach, researchers attempt to handle them with a variety of methods. This section discusses several approaches that aim to reduce the false positive rate.

2.6.1 Scope of Attacks

Approaches that reduce false positive rate are proposed by specializing NIDS to detect certain types of attacks, thus, make the signatures more specific. As discussed in Section 2.4.4, several types of research propose special NIDS that only focus on worms Abdulla, et al. [85], Dressler, et al. [86], botnet Amini, et al. [89], Zhao, et al. [90], DoS [83][84], and brute-force Hellemons, et al. [87], Vizváry and Vykopal [88]

attacks. By specializing and narrowing to certain types of attacks to be detected, the false positive rate can be reduced. This is because when the signatures are more general, any similar pattern can be identified as an intrusion. However, although these approaches show promising low false positive rate, they did not utilize the combination of flow-based and packet-based approaches for better results with increase the range of attacks.

2.6.2 Network Awareness

Another approach that reduces the false positive rate is when NIDS deeply understand the environment of the network behaviour [8, 106]. Information provided by legal vulnerability scanner, for example, helps NIDS to comprehend its environment. Shankar and Paxson [109] proposed a model that builds profiles of the environment using an active mapping that helps NIDS to understand its environment. These profiles store the current network architecture, topology, and policy.

Correlation between alerts and discovered vulnerabilities can also decrease false positive rate [110]. Also, Sourour, et al. [111] used further information about the network characteristics and provided them with NIDS analysis. These characteristics include security policy, network topology, and the types of software installed on the network and hosts. Adding these characteristics to NIDS can avoid false positive generation [18]. For example, when a security policy X is changed into Y and NIDS does not update with the policy Y, any event that matches with policy Y will be considered as an intrusion. Adding new device or host to a network should also inform NIDS to update the profile to avoid false positive. However, most of the network and

system environments are dynamic and not fixed. Thus, such approaches face challenges in modern environment, hence frequent building profiles is required to improve detection accuracy.

2.6.3 Traffic Cleanness

Some researchers focus on the pre-processing part of NIDS and machine learning to reduce the false positive rate. From the fact that the cause of high false positive is the raw data that enter into NIDS, they implement a pre-processing module to reduce ambiguity, irrelevant, duplicated, and noise data from traffic before entering into detection engine [17, 106]. Filtering technique for pre-processing using random forest [97] classification algorithms is proposed by [112]. Also, Bhatti and Virparia [113] proposed a pre-processing module that cleans the traffic and handles missing data of the traffic to be suitable and more accurate for further analysis.

In their module, four rounds were implemented. The first round was to remove noise and incomplete data. The second round was feature selection and extraction. In this round, it removed spurious and duplicate data to reduce the false positive rate. The third round used configuration based process where the network topology, existing host and services were stored. Since NIDS collected information from various sources for better detection accuracy, this raises a challenge with dealing with a different data format which might lead to a false positive. To solve this problem, the last round was proposed to generate unified data format to be analysed by NIDS. However, this approach might reach to exaggerative traffic cleaning. This is because when cleaning

the traffic, valuable information might be deleted and hinder an organization to access to them.

Also, Bhatti, et al. [114] proposed an approach that combines genetic and neural network techniques. Although the authors did not show any result, the proposed model divided into three stages to reduce false positive rate. Firstly, the preprocessing stage was used to clean the traffic and to reduce topological sensitive false positive. In the second stage, genetic algorithm and neural network identified intrusions by further processing. In the last stage, conflicting results were sent to NIDS to identify false positives and updated the system accordingly. Moreover, Spathoulas and Katsikas [115] proposed filtering method using fuzzy logic. The authors take some properties and information of generated alarms such as signature, timestamps, and IP addresses, as an input. Then filtering technique is used to reduce false alarm rates. However, although these alert analysis techniques are promising low false positive rate, scalability issues getting worse and the false negative rate become higher.

2.6.4 Alert Correlation

Alert correlations and post-processing techniques are also used for false positive reduction. Alerts that are generated by NIDS become inputs to a processor for further analysis using data mining or machine learning. Pietraszek and Tanner [116] and Abuadlla, et al. [17] used one layer as post-process for alert analysis after intrusion alarm was raised. The authors used either data mining (off-line) or machine learning (on-line).

To identify successful intrusion attempts and reduce the false positive rate, some approaches for alert verification are proposed based on host responses. Bolzoni, et al. [117] and Kaur [118] did not take only advantages of using alert correlation to provide a more accurate alert, but also analysing the outgoing traffic. The outgoing traffic is the output traffic from the compromised or the target system. They use three components in their approach: existing NIDS, anomaly detection output, and correlation system as shown in Figure 2.10. Their method checks if the traffic is raising an alert in the input NIDS generates an anomaly in outgoing traffic too.

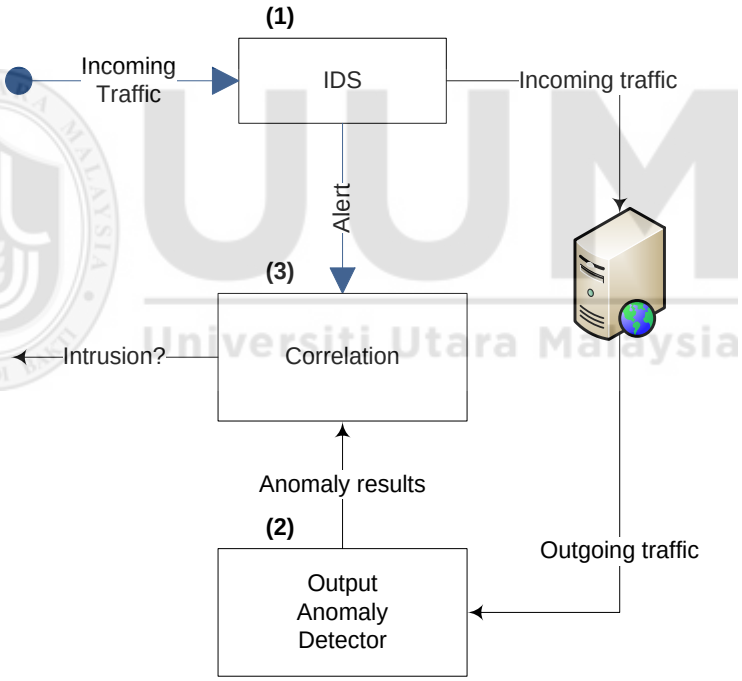

Figure 2.10. *Correlation Approach by Bolzoni et al. [117]*

The approach firstly takes the alert raised by NIDS (signature-based or anomaly-based) that monitors incoming traffic. Then anomaly detection output analyses the outgoing traffic of the victim host for further process. Correlation system used the alert

raised by NIDS and the result from anomaly detection output to remove false positive raised by input NIDS. If the anomaly detection output generates an anomaly, the alert is forwarded to the IT staff for desired action (true positive); otherwise, the traffic is discarded (false positive). However, this method is only designed for TCP-based network services such as HTTP. Also, they did not use packet-based NIDS for the second stage for decision confirmation.

The idea of utilizing the outgoing traffic of the victim host is also used before generating certain decision of worm detection. Gu, et al. [119] also introduced a model that scans the outgoing traffic for certain port numbers. When an anomaly attacks a certain host port number, its output traffic of that host is scanned. This output traffic is then used for checking other systems that have similar port service whether they have similar output traffic. If this is the case, it means that the attacker tries to contact multiple systems; hence, worm infection is probably performed. Wang, et al. [120] also proposed a model in a similar way. However, the outgoing traffic is cached when an anomaly is identified in a host to be compared to the neighboured output traffic. When this traffic is matched, this means that worms have infected the host, and these worms were trying to replicate themselves by spreading into another host.

As a summary, the approaches mentioned in this subsection have several limitations. Other types of attacks are not considered in these approaches. Also, since these approaches rely strongly on the output traffic or responses generated from the target system, forged output traffic generated by the attacker could trick the alert verification process and hide from detection. On other words, an attacker can modify the

anomalous output traffic, after the intrusion has taken place, to be legitimate traffic by crafting the attack payload. The target host then executes machine encoded instructions that are injected by the attacker to fake the outgoing traffic and to make the traffic normal. So, this vulnerability that accepts this code injection should be protected and the system output traffic sometimes cannot be used as a trusted method for reducing false positive rate. For more details on this issue, refer to [121]. Table 2.8 summarizes the related works from this section.

Table 2.8

Related Works for False Positive Reduction

Researches	**Approach**	**Limitation**
[83-85, 87-90]	Specializing attacks	Scalability issues were not studied in details.
[109-111]	Network and system environment awareness	Most of the network and system environments are dynamic
[112-115]	Traffic cleanness and machine learning	Exaggerative traffic cleaning may occur, and valuable information might be deleted
[116, 117, 119, 120]	Post-processing and alert correlation	Forged output traffic generated by the attacker could trick the alert verification.

2.6.5 Hybrid Signature-based and Anomaly-based

To reduce the NIDS false positive rates and to address the drawbacks of signature-based and anomaly-based detection, some researchers proposed approaches that combine both detection methods to reach accuracy (low false alarms) of the signature-based detection and have the ability to detect new attacks.

Aydın, et al. [122] proposed a hybrid NIDS by combining anomaly-based and signature-based detection. Signature-based detection is based on Snort. Their aim is to make NIDS to detect unknown attacks using anomaly-based approach and reduce false positive rates. These anomaly detections act as pre-processor of Snort and use statistical methods which are the most common method that detects intrusion by examining abnormal traffic. The pre-processor operation can give alarms, edit or filter the packets before they reach Snort detection engine. Although their approach has an improvement of detecting new attacks, they did not consider the high volume traffic issues. They use packet-based only as a data source for the engines which has negative impacts on the NIDS scalability. Also, their proposed hybrid forces all incoming packets to be inspected by the two engines, anomaly-based and signature-based, without sampling strategy. This means that delay and performance overheads occur in NIDS itself.

Tombini, et al. [123] proposed a hybrid architecture in which anomaly-based detection comes first, then feeding the signature-based detection with potential intrusions. They implement anomaly detection to list suspicious events. Then, signature detection is applied to classify these suspicious items into three groups. These groups are attacks, unknown attacks, and false alarms. The main limitation of this approach is that, in the case of suspicious items, corresponding packets have to be inspected twice, once by anomaly-based and the next by signature-based detection. Thus, scalability of the whole NIDS might be degraded.

Ding, et al. [124] and Hwang, et al. [125] proposed a technique that also combined signature-based and anomaly-based NIDS. Their hybrid consists of three sub-modules: signature detection, anomaly detection, and signature generation modules. The last module extracts the intrusion signatures that are detected by anomaly detection and map them into Snort rules. These signatures are then added to Snort database for future detection of similar attacks. By using the anomaly-based method, it shows an improved performance through mining anomalous traffic episodes from Internet connection. In their hybrid technique, placing signature-based engine at first stage means every per-packet must be inspected which degrades the scalability of NIDS. Thus, this hybrid approach is not suitable for high volume network.

Similar to [68][125], Yang, et al. [126] also proposed a hybrid approach of signature-based detection and then anomaly-based detection based on decision tree algorithms. Zhang and Zulkernine [127], Hussein, et al. [128], Day, et al. [122], and Kaur [111] proposed a hybrid approach that combines signature-based detection and anomaly-based. In their approaches, signature-based is followed by anomaly-based detection. Their evaluation showed that their signature detection produced a high detection rate with a low false positive rate, and the anomaly detection has the ability to detect novel intrusion. In [127], observed activities firstly are fed to the signature detection to detect known intrusion using random forest [97] algorithm. Then uncertain items that do not match any pattern in the previous phase are fed to the anomaly detection to detect unknown attacks using outlier detection that is provided by random forest algorithm. Signature detection can remove known intrusion signatures from the dataset, so that

the anomaly detection performance can be improved by only applying the rest signatures.

However, in their approach, some intrusions that are very similar to each other cannot be detected by the anomaly detection since this is a drawback of the outlier detection. Also, processing every packet on the wire effect negatively on NIDS scalability. Table 2.9 summarizes related works in this subsection along with the aspect of detection engine and data processed type. In the proposed mechanism in this research, both packet-based and flow-based are used as data source type as discussed in the next chapter.

Table 2.9

Related Works of Hybrid Detection Methods for False Positive Reduction

Authors	**Detection Engine**	**Data Source Type**
Aydın, et al. [122]	Anomaly-based (as pre-processor) then signature-based	Packet-based
Ding, et al. [124]	Signature-based then anomaly-based	Packet-based
Yang, et al. [126]	Signature-based and anomaly-based in parallel	Protocol-based and packet-based
Tombini, et al. [123]	Anomaly-based then signature-based	Packet-based
Zhang and Zulkernine [127]	Signature-based then anomaly-based	Packet-based
Hussein, et al. [128]	Signature-based then anomaly-based	Packet-based
Hwang, et al. [125]	Signature-based then anomaly-based	Packet-based
Day, et al. [129]	Signature-based then anomaly-based	Packet-based
Kaur [118]	Signature-based then anomaly-based	Packet-based

2.6.6 Hybrid Flow-based and Packet-based

Another approach proposed by researchers to reduce the false positive rate is by combining (hybrid) packet-based and flow-based techniques. However, the current research literature is rarely considering this approach. From the literature, this approach was conducted by [9], [22] and [23]. Limmer and Dressler [22] proposed a new monitoring technique called Front Payload Aggregation (FPA). Their technique combines the flow records and their corresponding (part of) payloads in the pre-processing stage to reduce the amount of data to be processed by NIDS.

The authors collected a certain amount of bytes from the payload and added it to the corresponding flow records before NIDS analyses it. They used VERsatile MONitoring Toolkit (Vermout) [22] for performance testing for their FPA. Vermout is an open-source software monitoring toolkit for creating and processing flow data based on the traffic data and producing flow records [22]. It also can accept and process raw packet by Packet Capturing (PCAP) library [130]. They successfully managed to integrate their FPA approach into the current monitoring protocol without any disruption.

Their work has several limitations. In their performance results, however, no performance on detection accuracy analysis in NIDS was presented in their work. In other words, false positive and negative rate were not studied in their analysis. Although flow aggregation goal was to reduce monitoring information, this goal cannot be accomplished when certain attacks are involved. Moreover, they proposed this approach with the context of enhancing the performance of signature-based

detection in high volume network, not from the context to reduce the false positive rate as the main mission of this research does. Also, packets of every flow regardless whether they are suspicious or non-suspicious, are processed by signature-base detection. This means more resource consumptions are needed for processing these packets.

Hensel [23] proposed a model that combines packet-based and flow-based. The author managed to reduce the number of packets to be processed by NIDS and enhance alert confidence. The most important part of combining the two approaches is the communicating mechanism between them. The author used the Bro client communications library (or so-called Broccoli) [131] to enable communicating process between flow-based and packet-based NIDS (e.g. exchange IP addresses). However, the author highlighted the significant overhead consumption of Broccoli method. Also, the full payload of the packets was captured and inspected, and high-speed volume measurements were not considered in his work. Overall, the author stated that no saving in resource consumption could be observed in his approach. Due to limited time, the research was not able to make a conclusive statement regarding detection performance

Golling, et al. [9] also attempt to take the advantages of combining flow-based and packet-based detection. However, although they considered high volume traffic (10 Gbps), no results were reported since the implementation is under deployment. As a summary, the work done by Limmer and Dressler [22] was focused on enhancing the scalability of signature-based detection, not on reducing false positive. On the other

side, although Hensel [23] focused on improving scalability and on false positive reduction, unfortunately, due to time limitation, the author could not do further evaluation to make a universal statement in term of detection performance.

2.7 Chapter Summary

At the beginning of this chapter, motivation of this research was presented. It showed the growing of network infrastructure, speed, and threats. There were lacks of the current computational frameworks, and identifying the right portion of the payload to be processed that could handle with the increasing of the network speed, growing of network traffic, and growing of the attacks and threats. State of the art of IDS, which is believed to protect from these threats, was explained. IDS types in term of location, detection methods, and data-processed were also presented.

The main problems and requirements of NIDS, which are scalability and detection rate, were discussed. The types of NIDS in term of data-processed, which are packet-based and flow-based, were explained in details. For each of these types, scalability and detection accuracy were discussed. Related works for botnet detection using both packet-based and flow-based were presented in details. Then, the comparison between these two types was explained and summarized. It showed that flow-based is the good choice in high volume network while the packet-based is more suitable for detection accuracy.

Also, future NIDS should be designed without the need of the payload for the sake of scalability. To handle the issue of high false positive in NIDS generally, several researchers had tried to reduce the rate of false positive using approaches such as

correlation, a hybrid of signature and anomaly methods, and a hybrid of packet-based and flow-based.

CHAPTER THREE
METHODOLOGY

In the previous chapter, discussion related flow-based, packet-based and its combination is presented. This chapter proposes the research methodology adopted in this research. The phases in research methodology include: proposed mechanism design, components identification, implementations, and evaluation, see Figure 3.1. For the first phase, Section 3.1 presents the design of the proposed mechanism. Each component in the proposed mechanism is explained in Section 3.2. These components include: traffic capture, flow aggregation, packet-based detection, and flow-based detection. Section 3.3 presents the implementation of the proposed mechanism. In this section, verification and validation of the proposed mechanism are also explained. Finally, Section 3.4 presents the evaluation of the implemented mechanisms in term of detection accuracy and resource consumption. In this section, the experiment testbed, datasets, and performance metrics used in this research are explained in details.

3.1 Proposed Mechanism Design

The outcome of this phase is the design of the CHID mechanism and it is based on the analysis of the literature. Where as shown in Figure 3.2, the idea of this design is to obtain the advantage of flow-based NIDS by having a small amount of data to be processed and the advantage of packet-based NIDS by having a low false positive rate (see the oval in Figure 3.2).

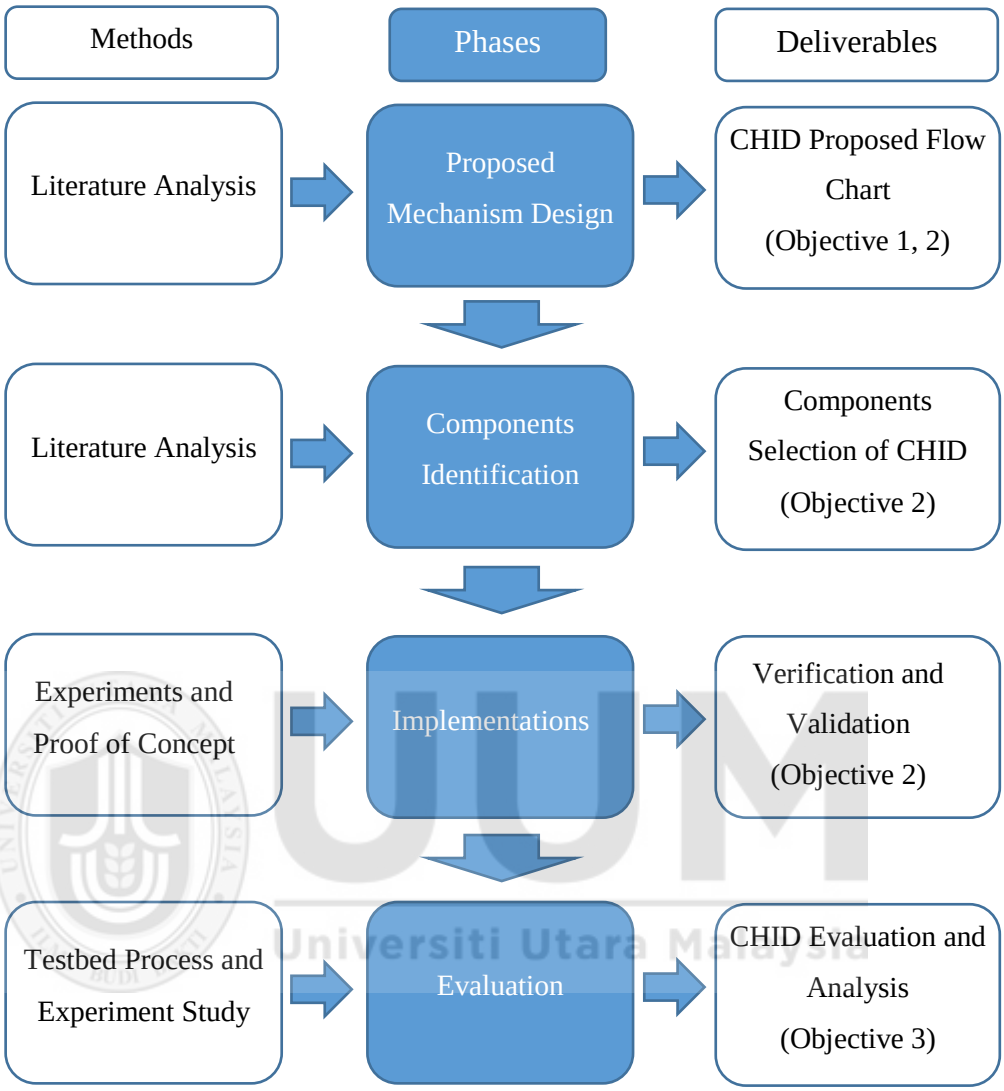

Figure 3.1. Research Methodology

Using this hybrid approach will overcome the bad scalability and packet drops of the packet-based NIDS. This is because the packet-based NIDS inspects only the packets marked by flow-based NIDS. On the other hand, using packet-based NIDS will overcome the high false positive rate of flow-based NIDS since the detected packets will be inspected by packet-based NIDS. To improve NIDS performance while basing on the critical review of many related works, false positive of flow-based detection and resource consumption of packet-based detection were determined as the key

factors to be addressed in this research. Furthermore, the overall ultimate goal for this research is the reduction of false positive and resource consumption of NIDS.

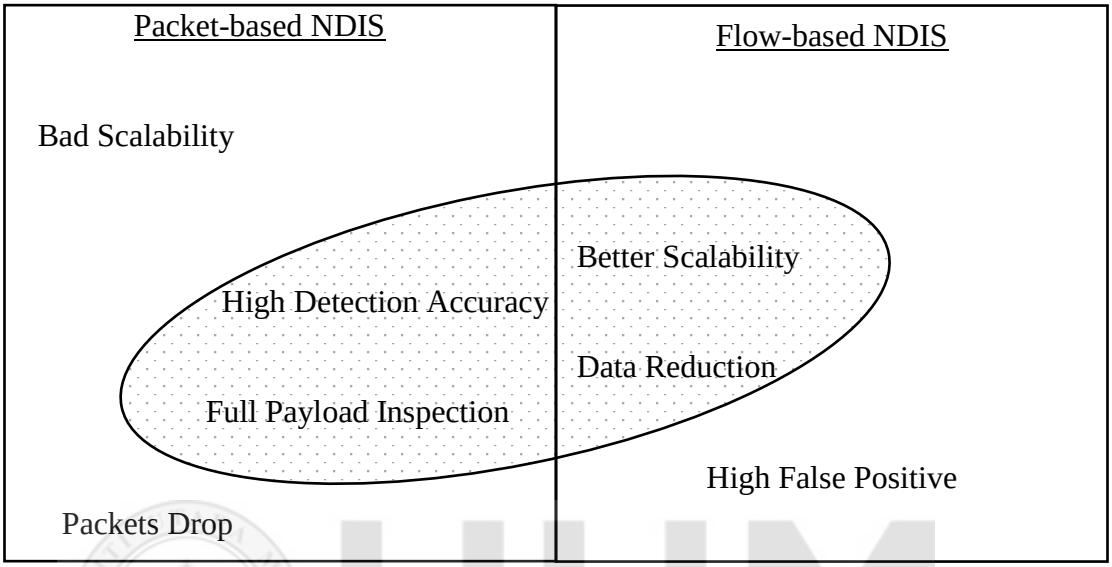

Figure 3.2. *The Idea of the Proposed Mechanism (the scope of the idea is covered by the oval)*

As a consequence of addressing these factors, a conceptual model of CHID was designed to describe the expected desired and improved situation using the hybrid approach. Figure 3.3 illustrates the conceptual model of the proposed solution named Conditional Hybrid Detection Approach (CHID). As shown in this Figure 3.3, false negative from flow-based NIDS is not considered since it is assumed in this research that the flow-based NIDS is able to detect all kinds of intrusions. In other words, the focus of detection accuracy improvement concerns the false positive rate only.

The flowchart is used in this thesis to present the proposed CHID mechanism as shown in Figure 3.4. As shown in the Figure, not every incoming packet goes through packet-based detection unless flow-based detection identifies it as suspicious traffic. The

proposed mechanism is conditional hybrid-based detection. It is conditional because the packets will be inspected by packet-based only if the condition occurs. When the flow-based detection identifies a suspicious flow, it will make the decision for these suspicious flows as an intrusion.

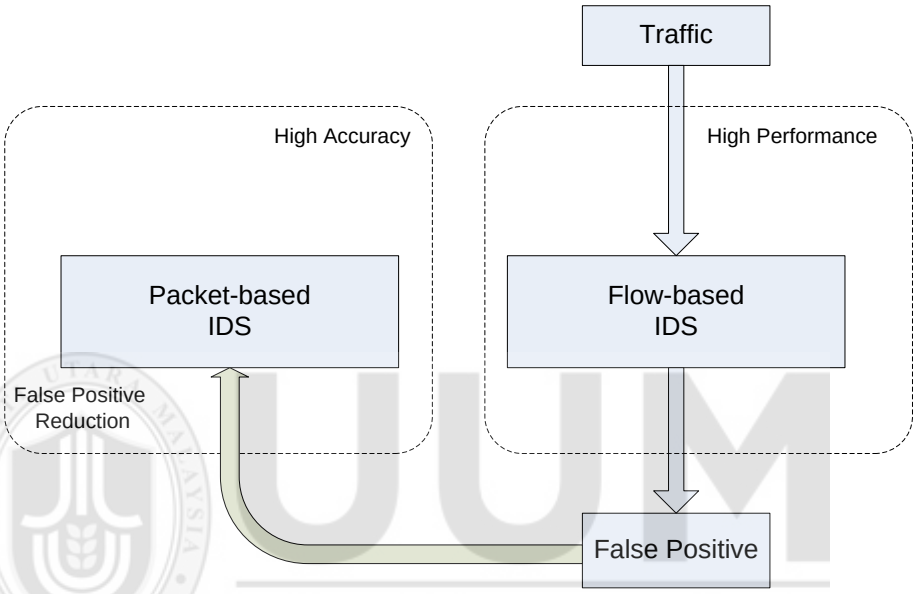

Figure 3.3. Conceptual Model of the CHID Approach

Marking these suspicious flows as intrusion will avoid false negative to take place. This step is important since the consequences and damages of a false negative are higher than false positive. Also, this mechanism can reduce the false positive rate of the flow-based detection. However, when the suspicious flows are marked as an intrusion, this decision is corrected to the operator later if it is found not an intrusion. This is achieved when packet-based detection analyses these packets that belong the suspicious connection and then makes a decision of this detection. In the case when no suspicious identified by flow-based detection, detection accuracy should be strong enough to avoid false negative.

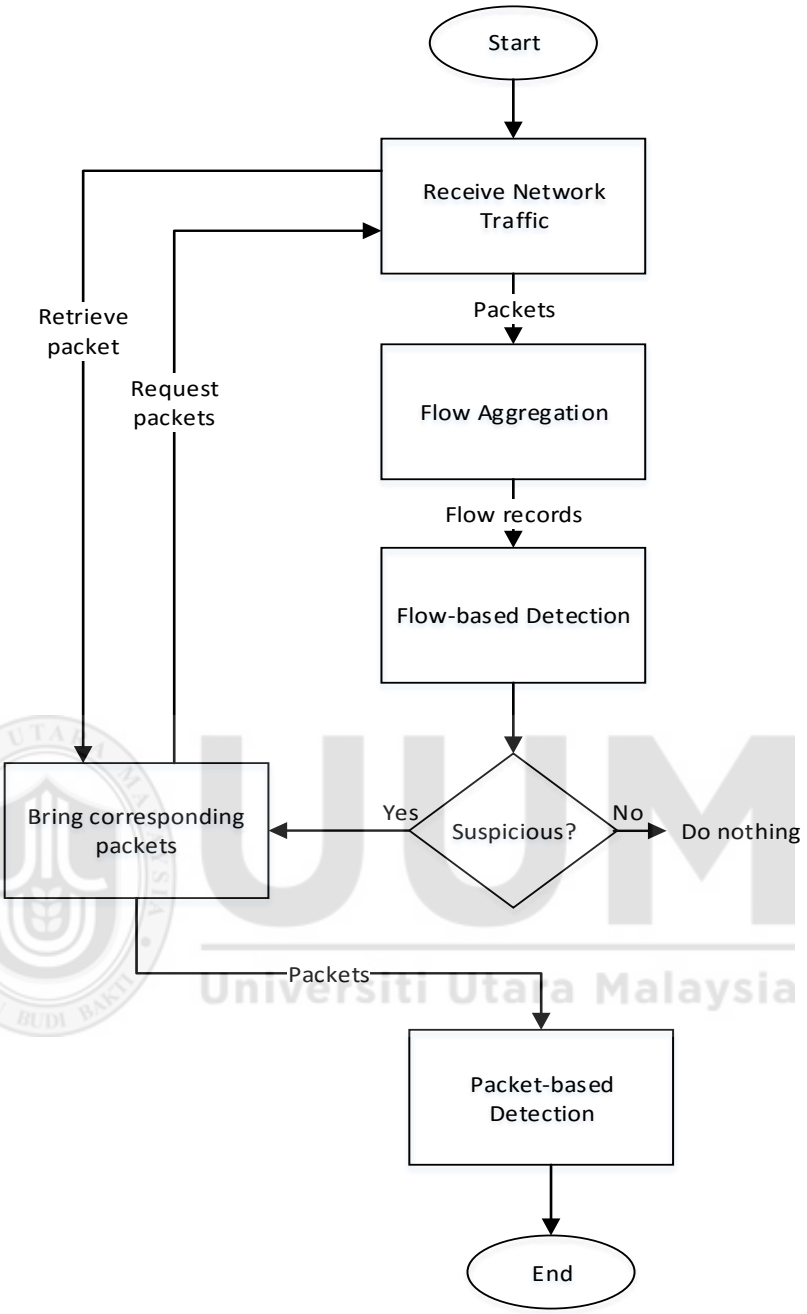

Figure 3.4. Proposed Flow Chart

Since packet-based NIDS has to import the suspicious IP addresses from flow-based NIDS, communicating process between the two detection should be designed. Input Framework (IF) method [132] will be proposed in this research as communicating

process method in CHID mechanism. It integrates external information in real-time into an NIDS source without negatively affecting the NIDS's main task, even in high-volume environments. However, this CHID approach is suitable for repetitive patterns in which the attack has a sequential frequent pattern that involves many connections to or from the particular host in short time.

A few existing researches used similar way in which they combine flow-based and packet-based NIDS [9], [22] and [23]. However, they differs from the proposed approach. Limmer and Dressler [22] proposed this approach with the context of enhancing the performance of signature-based detection, not from the context to reduce the false positive rate as the main mission of this research does. Also, unlike the proposed work, packets of every flow regardless whether they are suspicious or non-suspicious, are processed by signature-base detection. This means more resource consumptions are needed for processing these packets.

Instead of using IF method for communicating process between flow-based NIDS and packet-based NIDS in the proposed mechanism (see Chapter 5 for more details), Hensel [23] used the Bro client communications library (or so-called Broccoli) [131]. However, the author highlighted the significant overhead consumption of Broccoli method. Also, his approach captured and inspected the full payload of the packets instead of partial payload. In addition, [9] and [22], did not use machine learning technique for flow-based detection as the proposed mechanism does (see Chapter 4 for more details). Golling, et al. [9] reported that the implementation of their work was under deployment.

3.2 Component Identification

In this phase, the practical requirements to operate all the mechanism in this research are presented. Four main components in the proposed mechanisms include traffic capture, flow aggregation, packet-based detection, and flow-based detection. The tool for every component in the mechanism is explained and justified in this section. The deliverable of this phase is the component selections based on the proposed CHID mechanism.

3.2.1 Traffic Capture

Packet capturing is the practice of retrieving packets from the traffic link and the forward it to the respective device to do further process. This task can be performed by the very well-known software library named Packet Capturing (PCAP) [130]. PCAP (or libpcap for Windows) is an open source library suitable for off-the-shelf hardware and is used by many operating systems to capture the packet from network interface [130]. Linux system is considered to be the best choice since PCAP can optimize the Linux kernel for capturing the packets. Another library for traffic capture is PF_RING [133]. It directs the received packets to a ring buffer. However, PF_RING could not be implemented in this research since it requires cluster-mode in Bro NIDS.

PCAP has the ability for packet filtering. This filtering use mechanism called Berkeley Packet Filtering (BPF) [134]. This mechanism provides the ability to limit the monitoring of the incoming traffic to only predefined data. Before the packets are sent out to the next module, BPF has to be checked. For example, traffic might be filtered according to the protocol set (HTTP and/or FTP), IP addresses of the source and

destination, port numbers, or packet length. In normal flow aggregation, since flow aggregation needs data in each packet header up to transport layer, length filtering in PCAP is used to capture only this data instead of capturing the whole packet.

Also, PCAP can record packet not only with a full payload but with a predefined length of the payload. This is used when the monitoring system requires partial payload to be processed. This flexibility of size can be easily configured in PCAP. TCPDump tool [77] provides PCAP library and BPF filtering and storing the traffic on a disk for further investigation. From a performance perspective, monitoring system depends on the implementation of the traffic capturing mechanism. To achieve high scalability in high volume network (more than 1 Gb/s), every incoming packet must be retrieved and sent to next module. If the packet is not forwarded to the next module, drop packet occurs.

3.2.2 Flow Aggregation

Flow aggregation is the process that receives packets from PCAP and groups the packets that share flow keys into one flow records. Flow aggregation can be as a device (NetFlow router) or software running on a desktop computer [73]. In the implementation, software-based flow generation is selected for some reasons. Any configuration such as timeouts can be set easily. Parameters such as flow keys can be added or removed in a flexible way. In contrast, these features cannot be found in the hardware-based router device. This is because the device is mostly belonging to an organization that does not give users full control on it.

The most popular software-based flow aggregation is Softflowd [101]. It is open source software that captures packets at the network interface, aggregates the packets into flows, and exports the flow records to the collector. Softflowd supports NetFlow version 5 and 9 and IPFIX protocols. There are other software-based flow aggregations such as nProbe [135], and fprobe [136]. Softflowd is selected for this work for the following reasons:

- It provides efficiency and high performance in flow aggregation in software-based.
- Its source code is available, and it is Global Public License.
- It provides flexibility in configuring the parameters to produce flow records using command-line.

3.2.3 Packet-based Detection

The most software-based NIDS used in the research community are Snort [34], Suricata [32], and Bro [35]. To select software-type intrusion detection platform, the following criteria are evaluated and discussed for each system:

- User-friendly: Does NIDS allow development in an easy way?
- Prototyping: Is NIDS suitable for creating of method prototype?
- High-speed network support: Does NIDS support monitoring in high-speed traffic environments?
- Extensibility: Is the functionality of NIDS extendable in a reasonable way?

Among other NIDSs, Bro is user-friendly [10]. For example, in contrast to Suricata and Snort, Bro is not rule-driven in which it implements a scripting environment for creating rule-based detection. In other words, Bro detection rules are described by the scripts which use Bro programming language. This language is an interpreted in a domain-specific type. Bro policy script is the basic analyser used to specify what actions to take and how to report activities. Regarding prototyping, it was proved by Svoboda [10] that Bro is an essential tool for effective development of proof-of-concept and fully functional prototype. In addition, varies proof-of-concepts are required in this research in order to come out with the final prototype.

Also, Bro NIDS remains best suited for high throughput research environments as stated in [4]. For extending the functionality of the system, Bro offers features through its script analysis engine and capability to extend the response via script. Bro provides scripting language, and it comes with a large set of pre-built functions, yet the user can put Bro in novel ways by writing and extending the own policy script [10]. Table 3.1 presents the summary of the mentioned criteria related to each NIDS. Therefore, in this research, Bro is chosen as packet-based and flow-based detection.

Table 3.1

Bro Advantages among Other NIDSs [10]

NIDS	User-friendly	Prototyping	High-speed network support	Extendibility
Snort	No	No	Somewhat	Somewhat
Suricata	No	No	Somewhat	Somewhat
Bro	Yes	Yes	Yes	Yes

3.2.4 Flow-based Detection

In the flow-based detection stage, flow records are processed and analysed by the predefined algorithm and method. The output of the flow-based detection is a decision whether the flow is suspicious or non-suspicious. In the case of suspicious, the output contains details of the suspicious flow such as attacker's IP address and type of botnet activity. Flow-based analysis mostly depends on using statistical information and thresholds to differentiate between normal and abnormal behaviour. If the threshold is exceeded, then the suspicious state is identified. For example, if the number of flows that share the source and destination IP addresses reaches a specific number of flows within a specific time, then this flow is marked as suspicious.

Since Bro provides the flexibility of writing scripts and defining thresholds, this study intends to write codes in Bro and define a threshold for the certain type of attacks. However, since Bro do not provide flow-based detection, a mechanism is proposed to build this detection from scratch as presented in Chapter 4. To write the policy scripts to detect malicious activities, obtaining the rules and malicious flow features is required. To achieve this, machine learning algorithms are used to extract these rules and features from labelled datasets. Bro is an excellent choice for feature extraction [137]. This is because Bro provides real-time highly structured log files (can be used for digital forensic analysis or later research analysis) are generated and broken down by protocols and alerts that are written in plain text ASCII.

In this research, suspicious log file is created manually to be compatible with Bro logging format. To create this logging file, Bro comes with a logging framework [138]

that supports flexible key-value based logging interface that allows user to create own logging file in Bro logging format. The following abbreviations are defined to be used in the entire thesis: FL represents FLow-based detection; PH represents Packet-based detection in CHID approach; CHID is a combination of FL and PH, and PO represents the default Packet-based Only detection that inspects all packets. Figure 3.5 summarizes all the components with corresponding tools.

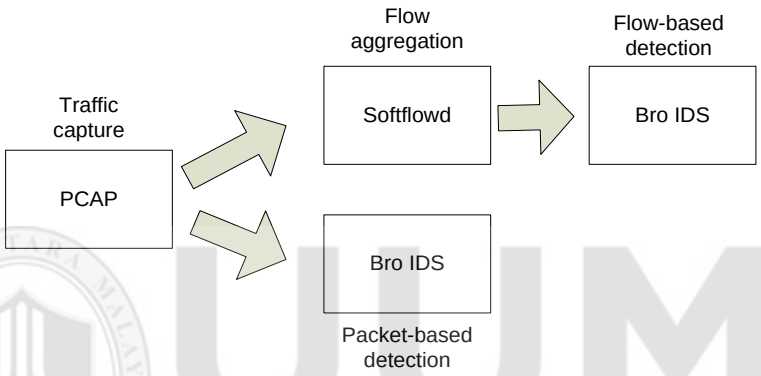

Figure 3.5. Component Requirements for Proposed Mechanism

3.3 Implementation

In this phase, the implementation of the components mentioned earlier is presented to operate the original and proposed detection mechanisms. The outcome of this phase is verification and validation of these mechanisms to make sure that all components are experimentally working together correctly and reflecting the conceptual model before evaluation process begins.

In this work, two mechanisms have been implemented:

1. Flow-based detection mechanism (Chapter 4).

2. CHID mechanism that consists of customized packet-based and flow-based detection (Chapter 5).

For high and accurate evaluation, all these mechanisms run in the same environment and platform and are injected with the same traffic (dataset). Verification and validation are performed in this phase. The verification measures the accuracy of transforming the proposed approach from a flow chart into a computer executable program [139]. All mechanisms that are used in this research are transformed into Bro script codes. Furthermore, these codes are then analysed and verified to ensure that they are free of errors and bugs. Also, each component is verified individually before implementing the overall mechanism.

For validation, it is the process when the approach, within its domain of applicability, behaves with satisfactory with the study of the research objectives [139]. For this, validation is required to be performed to ensure that the results obtained from the implementation of the proposed mechanism are meeting with its intended objectives. Furthermore, in the validating conceptual model, the research need to determine whether the assumptions in constructing the model are correct and reasonable. Also, proof of concept is mainly used in the research domain to demonstrate the feasibility of a new implemented model or idea. It is useful to verify and validate that the concept or theory of an approach is probably acceptable in a useful manner [140]. In this research, proof of concept is used to verify and validate the initial combination approach before finalizing the implementation of the proposed mechanism.

3.4 Evaluation

When the mechanisms are designed and implemented, their efficiencies must be evaluated. Also, in this research, performance evaluations should be run in different scenarios. In CHID approach, the full and partial payload for packet-based detection is implemented separately. Moreover, the implementation of the proposed approach should ensure that the model is capable of working with different type of attacks. These scenarios are implemented to evaluate the proposed approach in term of detection accuracy and resource consumption. In the evaluation, the resource consumption of Netflow aggregation is not considered because it is assumed to exist in a production network [9]. This section presents the experimental environments, testbed setup, measurement procedures, datasets that are injected into the detection systems, and the evaluation metrics.

3.4.1 Experimental Environment

Based on the literature, there is no common method for IDS evaluation [141]. However, in intrusion detection fields, comparing the new results with other's works were proven hard to accomplished [141]. When comparing new detection method that uses a new dataset with other methods, the other methods need to be run on the new dataset. However, to obtain the original implementation of other methods is difficult due to copyright issues. Also, most of the research in IDS field do not share their dataset due to privacy reasons [142].

Garcia, et al. [141] shows that majority of network-based detection methods were not able to compare their methods with third party methods. The authors also stated that

the difficulties behind this comparison are either their datasets tend to be private, or the description of their methods tends to be incomplete. Another reason of the difficulties of reproducing third party's method was explained by [143]. The authors stated that most of the detection proposals lack proper documentation of their methods and experiments.

As a consequence of the above issues, in this research, local comparisons are made between the proposed approaches with the default packet-based detection. For the experimental environment, sequential (one after another) experiments for both proposed approach and default packet-based with identical input data, require repeatable and reproducible environments. It is challenging to evaluate an IDS in such environments. The first option is to use live traffic. Since series of measurements are run in the experiments, running the experiments on live traffic approach is not possible because that will not yield fair comparisons when different configurations run in sequence. In this case, it is difficult to inject both approaches with the same traffic at the same time. In other words, live traffic has limitations, regarding the repeatability of experiments, for any systematic performance evaluation study.

The second option is to execute the experiments in offline-mode approach to receive datasets (traces). In this option, the traffic is captured from the network tap and record it into a file for future injection. Then the file is replayed and fed to both approaches, on the same machine, one after the other. This option is also not practical. This is because of Bro, in this case, would process the packets as quickly as possible at 100% CPU utilization. This observation was verified by the experiments when Bro packet-

based was run in offline mode to read dataset using "-r" command, the CPU usage reached and kept at 100% very quickly.

To address the aforementioned issues in those two options, a method that combines the best of both mentioned options is used. In this method, Bro reads the input from a pre-captured public dataset but in live mode, which can be achieved by replaying the traces into Bro through a switch (as discussed in next subsection). This approach results in a reproducible procedure that is comparable to using the data in the experiment on live traffic. The datasets used in the experiments are public and used for research community (see Section 3.4.4 for more details). Several researches used similar experimental environments including [144, 145], and [146]. Their NIDSs read the input from public and private datasets in live mode. They replay these datasets one after another for fair comparison purpose.

3.4.2 Experiment Setup

In this study, the experiments are run in a controlled environment, also called testbed. For more effective evaluation of the detection methods, testbed environment is preferred compared to simulating environments [141]. Simulation environment uses a computer-generated system to represent the behaviours of a real or proposed system. The experiment setup on testbed is depicted in Figure 3.6. The experimental environment run on two machines interconnected through a Gigabit switch. Both machines are running 12.04 Linux-based Ubuntu Desktop 64-bit with Intel i7 3.1 GHz with 32 GB of RAM. Table 3.2 summarizes the description of the hardware and system

for both machines while Table 3.3 presents the description of the software applications used in these machines.

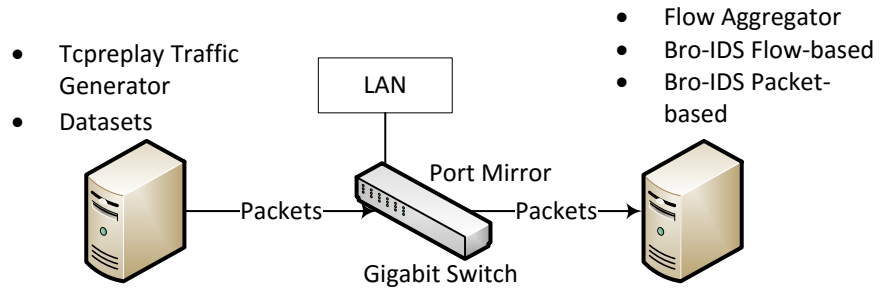

Figure 3.6. Experimental Testbed

Table 3.2

System and Hardware Description Used in Testbed

Requirements	**Description**
CPU	Intel i7 3.1 GHz
RAM	32 GB size
Operating System	Linux Ubuntu 64-bits
Switch	Linksys Smart Gigabit LGS318

Table 3.3

Software Applications Description Used in Testbed

Software Name	**Description**	**Software Version**
Bro	Intrusion Detection System	Bro 2.3 [33]
Tcpreplay	Traffic Replay	tcpreplay v4.0.5 [147]
Softflowd	Flow Aggregation	Softflowd v0.9.9 [101]

Network Interface Cards (NIC) of both machines support Gbps. Linux as the operating system is installed on these machines since Bro NIDS only support Unix-based systems. These machines are installed with default lippcap and BPF configurations for packet capturing and filtering from network interface card. The first machine is used for traffic generation. This machine replays real (previously captured) network traffic datasets on the wire using a packet generator tool that sends the packets back onto the network through other devices such as a switch, and then to the NIDS machine (second machine) for further analysis. Several packet generator tools are available [148], mainly including Tcpreplay [147], Bit-twist [149], and Tomahawk [150]. Tcpreplay is used in the experiments in this research. For efficient performance, tcpreplay is believed to be the best packet generator among others since it provides the ability to cache and store the traffic from hard drive to RAM [151]. This feature will avoid any disk I/O latency in order to increase performance. Another excellent feature of the tcpreplay tool is that different traffic speeds can be adjusted and controlled when injecting to the network interface.

The switch is selected since it supports Gigabit speed with port mirror enabled to forward all traffic to the analysing (second) machine. Since the NIDS is running in passive mode, it only receives a copy of the whole traffic for further analysis. The second machine is installed with Softflowd v0.9.9 [101] and Bro 2.3 [33]. Softflowd is used as the flow aggregator with default parameters to generate flow records from the dataset packets received and to export those records to the collector. These records are exported to the collector when they expire. Softflowd is also capable of generating Cisco Netflow export format. Also, to verify that Softflowd receives all packets from

the tcpreplay machine, the *Softflowctl* program is used to track the Softflowd process for statistical measurements.

Although Bro NIDS provide user-friendly interface for building policy script, it provides a command-line interface (CLI) for running and executing these scripts. Also, Bro is configured to collect the flow records by reading the flows from a UDP socket in the localhost with an approach suitable for Bro analysis. Localhost address is used since the flow aggregator and Bro flow-based NIDS are in the same machine. This scenario was also implemented in [23] and [144]. Moreover, Bro is installed on this machine with default configuration. To not to miss any received packets for Bro, Tcpreplay begins after Bro has been initiated. Then Bro instances are terminated immediately after capturing the last packet of the replayed dataset.

Because the Tcpreplay tool allows adjusting the transmission speed in Mbps, the experiments are repeated by replaying the datasets from 100 to 1000 Mbps. However, the experiments are not conducted beyond 1000 Mbps since the network interface cards of all machines support traffic volume up to 1000 Mbps. All of the tests were run three times to avoid any anomalies or noise in results. However, to relieve the load of Bro, generating irrelevant logs shown in Table 3.4 were disabled for all experiments. This is because *DNS.log, HTTP.log,* and *Connection.log* have nothing to do with intrusion detection and they are more related to general communication sessions.

Table 3.4

Loge Files Disabled

Log Files	Description
DNS.log	Generate DNS analysis and tracks DNS queries along with their responses.
HTTP.log	Generate HTTP analysis and tracks HTTP requests along with their responses.
Connection.log	Generate connection analysis and tracks TCP/UDP/ICMP connections

3.4.3 Measurement Procedures

To validate and evaluate each of detection mechanism in the proposed approach (flow-based and packet-based detections) and the default detection system, standard measurement procedures are defined. This subsection presents commands and their parameters for each instance used in the experiments. All these commands are in Linux-based command line platform.

A. Tcpreplay

Tcpreplay replays datasets in PCAP file format into the second machine to evaluate detection methods. The following command line is executed:

```
tcpreplay –i eth0 –M 200 –K dataset.pcap
```

The –i set the Ethernet interface where the tcpreplay send the dataset out. To adjust the traffic rate, –M 200 parameter is used, which means tcpreplay send the traffic in 200 Mbit per second. For efficient performance, –K is used to enable caching and

storing traffic to RAM. This is to avoid any disk I/O latency in order to increase performance. Since the RAM size should have enough capacity for this purpose, the RAM was increased to 32 GB.

B. Flow aggregation

For converting the received packets into flows the following command line is used:

```
softflowd -i eth0 –d –n 127.1.1.0:12345
```

The Ethernet interface that is listening for packets is specified with –i parameter. The –d is used for debugging. The softflowctl dump-flows command is used in order print all logged exported flows. Default maximum timeout is used. –n parameter is added to the command to tell Softflowd where to export flows on flow expiry timeouts. Since Bro is launched on the same machine, localhost IP address with an arbitrary port number is set.

C. FL

The following command is used for evaluating flow-based detection:

```
bro –Y 127.1.1.0:12345 flow-based-detection.bro
```

-Y parameter indicates the source where the flow-based detection receives flow records. In the experiments, this should be identical with Softflowd exporting location, hence localhost with the same port number. It is possible to add a parameter here to define the local host's subnet to be monitored. However, they are already defined them

in the script internally. The last parameter is the Bro file name of flow-based policy script where all incoming flows are analysed.

D. PO

For evaluating Bro default packet-based detection or PO, the following command is used:

```
bro –i eth0 default-packet-based.bro
```

It is required to specify the corresponding Ethernet interface using –i parameter. For network interface, eth0 is set since it is connected directly to the port-mirror on the switch is used to receive packets from tcpreplay machine. The local host's subnet to be monitored in packet-based detection are defined in the script itself instead of defining them in the command. The last parameter is the *default-packet-based.bro* policy script file name where all incoming packets are analysed. Figure 3.7 summarizes and show the procedure commands run in order. In all experiments in this research, as shown in Figure 3.7, Tcpreplay instance begins after all Bro instances have been launched. This is to ensure that Bro detection does not miss any packets.

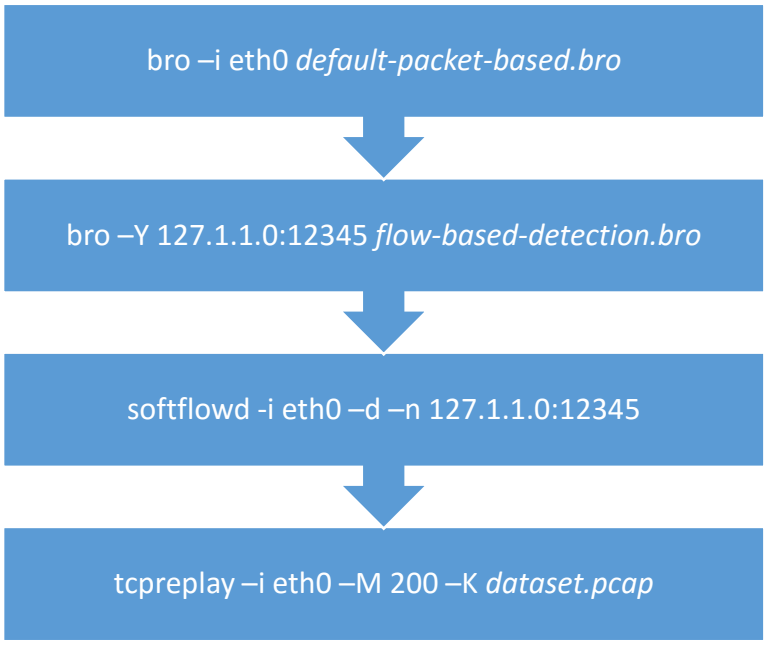

Figure 3.7. Experimental Commands

3.4.4 Dataset

The dataset is playing an important role when evaluating NIDS. For the dataset captured from traffic, it should contain potential attacks that can be tested by the proposed approach. To achieve this, traffic is captured in controlled testbed environments. From this environment, a dataset that carries packet traces can be obtained that represent the real traffic and contain potential attacks. Most of the researchers used public benchmark dataset based on the testbed for evaluating their NIDS approaches. In the literature, DARPA 1999 dataset [152] is the most widely used for validation NIDS. This dataset is contributed significantly to the NIDS researchers for the purpose of testing their proposed IDSs [153]. It was developed by MIT Lincoln Laboratory. Although the researcher still using the dataset, it faces challenges since it seems to be old and it does not contain the changes of Internet traffic and new attacks. With the absence of better benchmarks, a huge amount of

NIDS research experiments were relying on DARPA dataset [154, 155]. On the other hand, this dataset was criticized due to its age and inability to reflect real-world traffic by [156] and therefore discarded in this research.

The DEFCON data [157] is newer public benchmark dataset based on the testbed, but they did not explain what type of attacks this data carries. The reasons behind the lacking of the public benchmark dataset are a due to privacy concern especially when the traffic contains IP addresses that identify users or payload that identify sensitive information. This data leak information that might be valuable to the attackers or the competitors.

3.4.4.1 Malicious Datasets

To avoid the criticism against works that rely only on DARPA dataset, another dataset is collected that contains recent application traffic. Mainly, two public labelled datasets are used in this research. The first dataset used is the Information Security and Object Technology (ISOT) dataset published on 2011 [90]. The other datasets are generated from Czech Technical University (CTU) in different scenarios and published on 2013 [141]. Labelling these datasets is useful to validate the accuracy of the detection methods [92]. These datasets are presented in Table 3.5 with more statistical details in each dataset. More explanations about ISOT datasets and CTU scenarios by their authors are found in [90] and [141], respectively. The datasets mentioned in this table were generated based on lab environment and traffic communicating of physical hosts and servers with various operating systems and applications. In their testbed, many types and scenarios of attacks were carried out that

represent current threats especially the repetitive attacks such as DoS, Worm, and Botnets.

Table 3.5

Datasets and Statistics

Datasets	**Total Extracted Packets**	**Total Exported Flows**	**Average bytes per flow**	**Average packets per flow**	**Type of Activity**
ISOT	59.9 million	5.2 million	25,196	44	P2P-bot
CTU-50	2.1 million	159,704	10,438	21	Spam-bot
CTU-51	66.3 million	31.7 million	2,161	2	IRC-bot
CTU-52	3.9 million	1.7 million	2,311	2	IRC-bot
CTU-53	351,537	11,117	46,063	57	P2P-bot
PSCJ-1	10 million	4.3 million	3,432	23	Non-malicious
PSCJ-2	22 million	9.5 million	5,233	25	Non-malicious

The ISOT dataset was created purposely for the research community. It has a combination of several existing publicly available P2P-botnet malicious and non-malicious datasets (3.3% is malicious P2P botnet flows). The malicious dataset was obtained from the French chapter of the Honeynet project [158] captured on 2011 and involving P2P botnet activities such as Storm, Weledac, and Zeus. For the non-malicious datasets, it was collected from two incorporated different datasets, one from

the Traffic Lab at Ericsson Research in Hungary [159] and the other from the Lawrence Berkeley National Lab (LBNL) [160].

The Ericsson Lab dataset contains a large number of general traffic from popular applications such as HTTP web browsing behaviour, World of Warcraft gaming packets, and packets from popular BitTorrent[161] clients such as Azureus. The LBNL Institute is a research organization with a medium-sized enterprise network. The recording of the network trace of LBNL occurred over three month's period, from October 2004 to January 2005. This traffic contains a variety of network activities that involve everyday activity usages such as HTTP web behaviour, popular sharing file packets, emails, and streaming media.

CTU datasets were generated in different scenarios and published in 2013. These datasets are created under Malware Capture Facility Project [141] and obtained for botnet behaviour analysis. The datasets have several real captures that are labelled and generated by a number of computers infected with the IRC-bot, P2P-bot, HTTP-bot. Where as shown in Table 3.5, four datasets from CTU are considered in this research. CTU-50 is referred to as CTU-Malware-Capture-Botnet-50 in the literature [141]. Similarly, CTU-51 is referred to as CTU-Malware-Capture-Botnet-51; CTU- 52 is referred to as CTU-Malware-Capture-Botnet-52; and CTU-53 is referred to as CTU-Malware-Capture-Botnet-53. However, for privacy reasons, completed PCAP is made public for only infected machines. For background and normal traffic in PCAP format are not available. The malicious labelled data is made by the expert group in security

fields. ISOT and CTU datasets were also used by variety of studies such as [46], [162] with different research objectives.

3.4.4.2 Background Traces

For the non-malicious traffic, unfortunately, none of the datasets mentioned in Table 3.5 (except for the ISOT dataset) contains full-payload background traffic for privacy reasons. Background traffic is required for non-malicious training and false positive testing. In other words, if the NIDS generates intrusion alarm from this background traffic, it is considered as a false positive alert. To create this traffic, two separate background traces were captured, in a one-day complete payload trace. They were captured at Alfaisal University, Prince Sultan College Jeddah (PSCJ), Information Technology Centre, at the main gateway link that connects hundreds of hosts with an educational network to the Internet.

The first trace is named as "PSCJ-1" and captured on Monday, 5th May 2014 from 8:00 am for 24 hours. The size of this trace is 8 GB and contains 32 million packets, corresponding to approximately 1.9 million flows. For the second trace, it is named as "PSCJ-2" with 17 GB of size and contains 22 Millions of packets. It was captured on Monday, 12th May 2014 from 8:00 am for 48 hours. The PSCJ-1 and PSCJ-2 datasets involve everyday activity usages such as HTTP web behaviour, popular sharing file packets, IRC traffic, emails, and streaming media. All implementations in this research will be injected by these datasets using traffic replay tools, Tcpreplay.

3.4.5 Evaluation Metrics

Based on the literature, the metrics mostly used for NIDS evaluations are detection accuracy and resource consumption [23]. Notions presented in Table 3.6 are used in the evaluation to measure the common metrics: true positive and true negative rate and precision. Many researches used these metrics for NIDS evaluations [11, 46, 141, 155].

Table 3.6

Notion Matrix [163]

	Actual Non-Malicious	**Actual Malicious**
Detected as malicious	False Positive (FP)	True Positive (TP)
Detected as non-malicious	True Negative (TN)	False Negative (FN)

True Positive Rate (TPR) as calculated in:

$$TPR = \frac{TP}{TP + FN} \tag{3.1}$$

For False Positive Rate (FPR) and precision, the following formulas are used:

$$FPR = \frac{FP}{FP + TN} \tag{3.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

For the resource consumption, the following metrics are used for all implementations:

1. CPU and memory usage of the flow-based detection: this metric measures how flow-based detection processes such as threshold calculation, updating tables, and running scripts, consume the resources. This metric monitors the resource consumption of the flow-based NIDS to make valid assumptions on how much traffic volume can be processed until the resources are exhausted.

2. CPU and memory usage of packet-based detection: this metric evaluates how the proposed packet-based and the default Bro packet-based detection handle the packet processing. This metric monitors the resource consumption of the packet-based NIDS to make valid assumptions on how much traffic volume can be processed until the resources are exhausted.

3. Packet drop rate of packet-based detection: this is important metric to measure the packet drop rate of the proposed packet-based and the default Bro packet-based detection. To calculate the packet drop rate, the following metrics are required:

    - A number of received packets in all packet-based detections.

    - A number of processed packets in all packet-based detections.

The following formula is used to compute the packet drop rate:

$$\text{Packet drop rate} = \frac{\text{Number of received packets}}{\text{Number of processed packets}} \quad (3.4)$$

From these metrics, the efficiency of the proposed NIDS is evaluated. All of the tests are run three times to avoid any an

omalies or noise in results. Then, the CPU and memory parameters are measured in all experiments as an average value. This average value is calculated from the mean values of the results of these tests. These results are plotted in graphs over time and with different traffic speed rates. Bro provides statistical information that might be useful to get the important evaluation metrics. These resource consumption metrics are generated using *stats.bro* that is provided by Bro application. For the proposed method, CPU and memory of flow-based and packet-based are summed to enable a direct comparison with default packet-based detection.

3.5 Chapter Summary

This chapter presented the four phases in the research methodology. These phases included the design of the proposed mechanisms, components requirements, implementation of the proposed mechanisms, and evaluation phase. For the first phase, the design of the proposed mechanism named CHID was briefly explained. The components of the mechanism were then explained in details. These components included: traffic capture, flow aggregation, packet-based detection, and flow-based detection. Implementation of the proposed mechanism was also presented briefly. In this phase, verification and validation of the proposed mechanisms were also discussed. Finally, evaluation of the mechanisms in term of detection accuracy and resource consumption were presented. In this phase, the experiment testbed, datasets, and performance metrics used in this research were explained.

CHAPTER FOUR
TWO STAGES FLOW-BASED DETECTION

4.1 Introduction

Due to the severity of network threats, detection systems have attracted intensive research efforts. For botnet threats case, some detection systems have been proposed as presented in Section 2.3.3. Most of these systems focus on detecting botnets in the command and control stage. Although these systems demonstrated promising detection results, they suffer from poor scalability when they analyse a large volume of network traffic in high-speed networks. Their poor scalability originated mainly from their deep packet inspection (DPI) analysis on the payload of network packets as studied by [10]. BotHunter [67], for example, perform payload anomaly inspection and signature-based detection engine. TAMD [66] uses packet payload to compute the scores of content similarity. Parsing the content of IRC communication was required for BotSniffer [64] and Rishi [65]. BotMiner [63] also perform DPI for detecting binary downloading and remote exploitation. Smallwood and Vance [164] and Bremler-Barr, et al. [165] also develop DPI for intrusion detection but with reasonable resource consumption rate.

Because of packet-based detection is computationally expensive, the above systems cannot be directly deployed in their software platform design in high volume networks. In other words, with an increase in network volume and speed, existing network NIDSs face challenges when capturing full payload traffic for malicious inspection which in turn affect the performance and accuracy of NIDS [27]. To overcome this

issue, the main aim of this chapter is to design and implement flow-based detection approach. Flow-based detection approach reduces the amount of data to be analysed by looking at aggregated information of related packets in the form of flow. Also, many network operators have flow monitoring services at their disposal [106], thus deploying these flow aggregation comes at almost no cost. Golling, et al. [9] consider flow-based intrusion detection as a viable approach for operators of high-speed networks.

Several works showed the ability to detect malicious traffic using only flow-based attributes with several machine learning classifiers as presented in Section 2.4.5. Their methods extract malicious flow features from datasets. However, these methods hold the following limitations:

- Most of the approaches proposed by these researchers reported a significant number of false positive alerts [20, 21]. The reasons behind generating these false positives include: 1) limited information available for intrusion analysis, 2) complexity of characteristics of normal and abnormal traffic, 3) generic of signatures (when the signatures are developed to be more general, any similar pattern can be identified as intrusion), 4) network and system environment are dynamic.

- Wijesinghe, et al. [104] pointed out that their approaches observed high overhead in the aspect of CPU utilization that caused from flow capturing process.

- With focusing only on flow data analysis, they ignore the benefits of packet data analysis that provide more information about the traffic.

- Various researches focus on detecting botnet attacks using flow-based detection but in host-based behaviour (or HIDS) [166] with neglecting network-based behaviour (or NIDS). Host-based detection alone seems to be ineffective in current botnet attacks [27].

- Some of these researches lack real traffic data collection since they used artificial traffic which does not reflect real traffic [98].

- Other researchers [71, 72] evaluate their proposed botnet detection methods in a simulated environment. The basic problems with these approaches are the lack of the implementation model in real-world scenarios.

As discussed in Chapter 2, this research focuses on network-based behaviour analysis (or NIDS) which is mostly used today and promised high detection rate for botnet attacks [5]. Instead of using only flow-based data, a combination of flow-based and packet-based analysis is used in this chapter to collect more information about the malicious and non-malicious traffic in the datasets. Moreover, the data collected for traffic analysis is captured in an environment that represent real traffic. For more effective evaluation of the detection method, testbed environment is performed instead of simulating environments.

In summary, the main contribution of this chapter is to propose flow-based detection mechanism in two stages: precondition and threshold states. The aim of precondition stage is to identify all candidate hosts are potentially related to malicious botnet communications. To identify these candidates, they must pass certain rules that contain botnet activities behaviour. The aim of the threshold stage is to analyse the behaviours of these candidates classifies these them into either normal or bots. To achieve the proposed flow-based detection mechanism, the following contributions of this chapter are performed:

1. Design and implement a workflow process to build flow-based detection scripts (for precondition stage). These scripts obtain characteristics of malicious activities by extracting malicious flow features from a variety of labelled datasets. Packet and flow analysis (using Bro) in addition to attribute classification (using WEKA [167] machine learning) are involved in this process.

2. A flow-based detection threshold-based (the second stage) approach that analyses the botnet flows for suspicious identification.

This chapter is organized as follows: Section 4.2 presents the design of flow-based detection mechanism. In Section 4.3, the behaviour of the selected attacks to be implemented are discussed. A workflow for building flow-based detection scripts is implemented in Section 4.4. More details on how detection mechanisms are implemented are presented in Section 4.5. Then Section 4.6 explains how to validate

and evaluate the flow-based detection implementation on the testbed with the new datasets.

4.2 Design

To study the detection accuracy impacts on flow-based detection approach, in this section, design and implementation flow flow-based detection scripts are required (see Figure 4.1 (b)). The inbound and outbound traffic of the internal network is analysed to detect possible intrusive activities that the local machines are performing. The output of this processing is the list of machines IP addresses which perform malicious activities. Detection methods monitor only local hosts.

In packet-based detection approach, as shown in Figure 4.1 (a), all incoming packets are passed to packet-based detection. While packet-based detection listens directly from the network interface, the flow-based detection listens from the flow aggregator in order receive the flow records as input. Bro detecting signatures for packet-based inspection are available, but no directly implementable flow-based detection exist. In this chapter, flow detection scripts are build using statistical analysis of the traffic. Before the implementations of detection scripts are presented, discussion on attack selection and the datasets used are presented in next sections.

4.3 Attack Selection

Based on literature [25, 168, 169], botnet attacks perform repetitive traffic patterns. Repetitive patterns imply that the attacks generate similar traffic patterns in future connections. With this point in mind, and rather than review general botnet detection

in this work, the following bot-related malicious types are selected to be considered in this chapter: IRC-bot and P2P-bot. These botnets are the most popular active botnets [20]. Also, these cases of botnet attacks are believed to be the most strong–threat to the security of Internet-connected users and systems [24]. For more details on the characteristics of these malicious activities, refer to [25, 26]. In the following subsections, the behaviours of IRC-bot and P2P-bot are presented. These behaviours are useful for feature selection used for botnet detection implementation in this chapter.

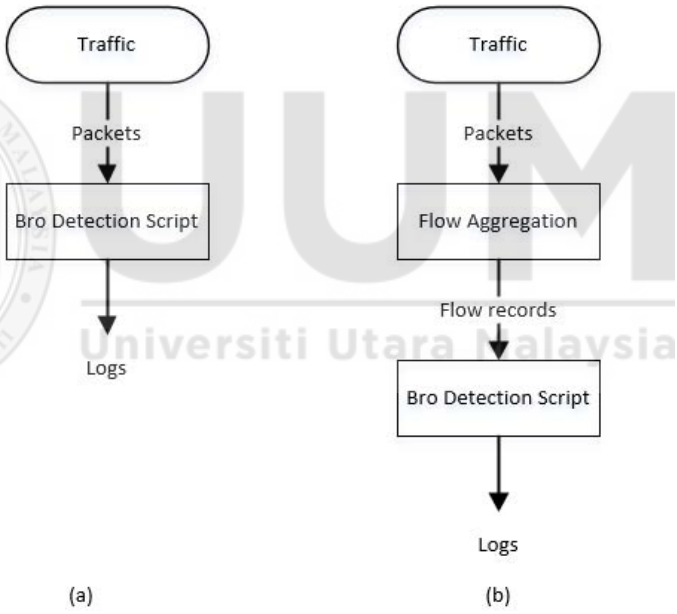

Figure 4.1. Illustration of (a) Packet-based Detection System and (b) Flow-based Detection System

4.3.1 IRC-bot Behaviour

Concerning IRC-bot, flow-based includes significant information about IRC and IRC-bot connections. Both connections use TCP communications. IRC connection such as

a ping-pong message exchange is easily identified in flow-level information. A ping-pong message or so-called keep-alive function is used by an IRC server to determine whether the user computer is alive and to prevent the channel from being idle. It works as follows: IRC server sends a ping message to the IRC client. This client replies with a pong message. This ping-pong message holds a fixed amount of packets and bytes per flow [170]. Typically, no messages are exchanged when a bot is communicating with the controller other than ping-pong messages because IRC-bot is mostly in idle mode; since it usually waits for commands from the controller. The absence of any messages other than ping-pong messages could be a significant indication of the presence of an IRC-bot which is utilized in this chapter.

Also, currently, a large fraction of spam comes from botnets (e.g. IRC or P2P botnet). Based on the literature, the most efficient and common activity that a master of botnet command the bots is spamming [171, 172]. Some reports indicate that around 80% of all email traffic is spam and that most of them are sent via botnets [5]. Thus, these observations imply that e-mail spam detection can be an effective strategy for subsequent botnet detection. Stringhini, et al. [172] pointed out that botnet tends to send a large number of spam in a relatively short period. Also, bot spammers send many SMTP connections and receive a few or no SMPT connections [173]. With taking this assumption into account, the botnet could be detected by identifying the associated spams.

For packet-based detection, the payload information for IRC-bot also contains potential information for detecting such attacks. With this information, it is possible

to identify whether an IRC connection is used for benign communication such as chatting or for malicious communication such as connecting to an IRC-bot command and control server.

4.3.2 P2P-bot Behaviour

In P2P bots, peers do not receive commands from a central server, as they do in IRC-bot; instead, they receive commands from peers. In other words, in P2P network, every bot acts as a client and server at the same time. Unlike client-server botnet (such as IRC botnet), P2P botnet is getting much attention by an attacker; this is because even if a single node in the P2P bots is detected, the entire botnet can still continue [169]. Unlike IRC-bot, P2P communication uses UDP protocol. It is reported in the literature [107] that P2P-bot involves higher numbers of connections between peers compared with the normal communication in P2P such as e-Donkey and Bit-torrent applications. A higher number of connections means higher flows occur in P2P-bot, which can be easily marked by the flow-based process.

For packet-based detection, payloads include significant information for P2P-bot intrusion detection. For example, in command and control (C&C) communication, a payload could contain the instructions (e.g., what task to perform) sent to peers. Even some part of this communication could be encrypted; it still contains static values that are useful for signature detection. Also, spam emails sent by a bot (in the attack phase) could be a sign of a P2P botnet. This occurs when compromised machines controlled by a botnet is commanded to send spam by searching email contact on that machine. For example, most Storm botnets serve as spam senders [18]. Thus the number of

SMTP packets may also be an indicator for such attacks. This implies that spam detection can be an effective strategy for subsequent botnet detection.

4.4 Detection Scripts Derivation

Since the main goal of this chapter is to study the false positive impact on flow-based detection approach, it is required to implement the flow-based detection mechanism by developing policy scripts for specific malicious detection. For developing detection scripts, existing labelled datasets are utilized. Packet and flow analysis along with machine learning are also used to collect and extract malicious flow and packet features from these datasets. Based on the findings, the flow-based detection scripts are implemented to detect P2P and IRC bots. Then different datasets are used to validate this detection method.

Figure 4.2 illustrates the different steps for deriving detection scripts for each botnet malicious type. These steps include: packet and flow analysis of datasets to generate log files, observation of the output log files, writing detection script policy and rules, and validating and evaluating the detection script. The following subsections details each step. In this chapter, several labelled datasets will be applied to Figure 4.2 to build detection scripts on each malicious activity.

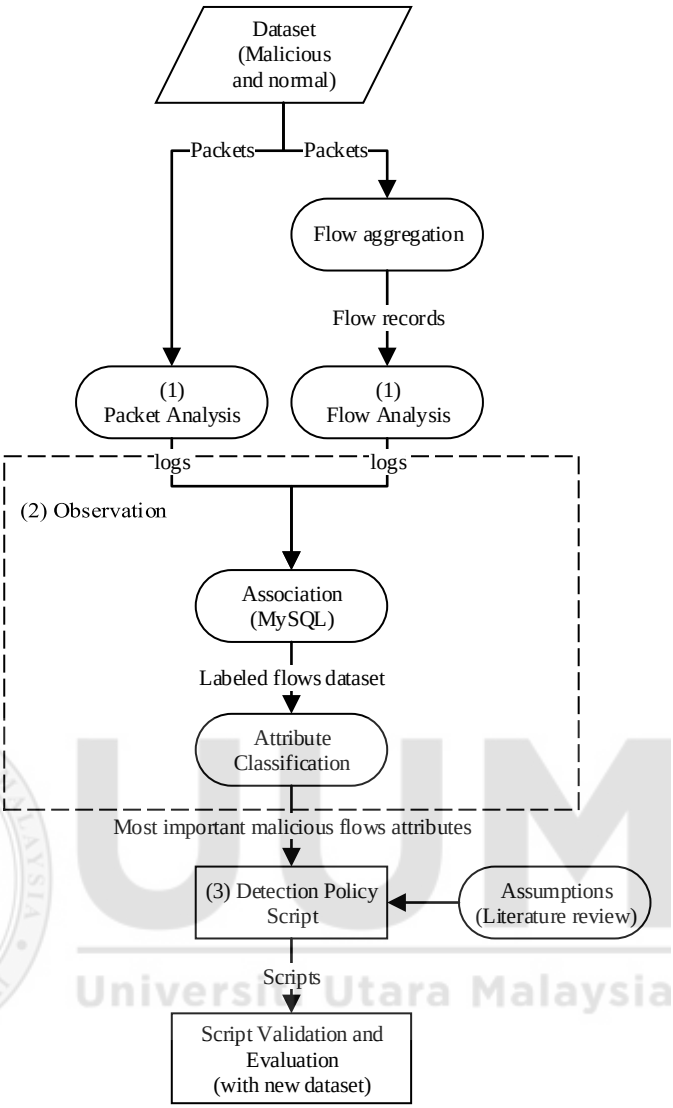

Figure 4.2. Workflow for Deriving Flow-based Detection Policy Scripts

4.4.1 Packet and Flow Analysis

In this step, packet and flow analysis against the given combined datasets are performed. The input is PCAP format datasets, and the output is log files generated from Bro analysis. The aim of packet analysis is to label flows as discussed in the next subsection. Non-malicious and malicious dataset are combined. Malicious dataset selection depends on the type of malicious activity type. In this case, CTU-52 and

CTU-53 datasets are used for IRC-bot and P2P-bot respectively. Also CTU-50 dataset is used to extract spamming flow behaviour. For non-malicious traffic, PSCJ-1 trace is used. For each malicious type, since the combined datasets are in PCAP format, it is converted into network flows (by using Softflowd) and then, the flows are used as input for the flow analysis for further analysis in this step.

However, before Packet and flow analysis begin, the datasets were cleaned by removing out non-IP, non-TCP/UDP, and irrelevant traffic. Also, to limit the amount of packet and flows processing of these datasets, eliminating non-relevant packets and flows within the datasets are performed using filtering scripts. These filtering scripts are based on known infected and normal machines IP addresses, port numbers, and protocols, depending on the malicious activities selection (e.g., in spam, packets and flows initiated from internal host IP addresses to external host on TCP protocol with port 25 only were extracted). Packet analysis also relies on Dynamic Protocol Detection (DPD) to determine the application protocol (such as BitTorent [161], eDonkey [174]) of the connections.

Also, the logs provide Unique Identifier (UID) of a connection which is used to correlated information across other Bro logs as shown in Table 4.1. This table also shows the fields and a brief description of *weird* log file. Other log files including, *notice.log, connection.log, and signatures.log* are presented in Appendix D. These files contain details regarding unusual activities that shed some insight on the behaviour of the malicious traffic. Other logs also added manually, that are not written by default, to get more deep analysis. The output of this step is the log files generated

by both flow and packet analysis. The features that are extracted from logs files are presented in Table 4.2. These logs are stored in MySQL database for the next step.

Table 4.1

Fields Description of Wired.log file

Field	Type	Description
Ts	Time	Timestamp of message
Uid	String	Connection unique id
Id	record	ID record
name	String	The name of the wired occurred
Addl	String	Additional information along with the weird
Notice	Bool	Indicate whether this weird is also turned into a notice
Peer	String	The peer generated this weird

Table 4.2

Features of Flow and Packet Generated from Logs

Type	Features	Description
Shared features	src_ip	IP address of source host
	dest_ip	IP address of destination host
	src_p	Source port number
	dest_p	Destination port number
	Proto	Protocol
Flow-based features	Octs	Total octets or bytes per flow
	pkts	Total packets per flow
	t_s	Flow time start (for the first packet in a flow)
	t_f	Flow time finish (for the last packet in a flow)
Derived flow features	*duration*	Flow duration (*duration* = $t_f - t_s$)
	bpp	Average bytes per packet in flow (*octs/pkts*)
	Bps	Average bytes per second (*octs/duration*)
	Pps	Average packets per second (*pkts/duration*)

4.4.1.1 Flows Labelling

The outcome of this step is flow datasets with labelling. In this step, flow labelling from packet analysis logs to flow analysis logs are made. Labelling the flows as malicious is based on IP address. First, matching the 5-tuple (source and destination IP addresses, source and destination port numbers, and protocol) and timestamps of both packet and flow analysis logs using MySQL are performed. Second, when a match is found, the flow is labelled, based on the packet-based analysis decision. The reason for selecting packet analysis for flow labelling is detailed below.

Unlike flow analysis, since packet logs are generated from full packet inspection, hence is rich of information; it is assumed that packet-based logs are the benchmark for alert decision and labelling. With this in mind, and to reduce the number of flows for further analysis, a decision, that flow is malicious or non-malicious, is based on the decision of its corresponding packet generated by packet analysis. In other words, flows that are alerted by packet analysis as malicious, are extracted and labelled herewith. Finally, flow labelled datasets mixed with malicious and non-malicious labels, for each malicious type, are generated. To obtain feature vectors that represent normal traffic, it is assumed that the traffic in PSCJ-1 trace is clean. Thus, they are then labelled as non-malicious.

4.4.1.2 Attribute Classifications

Attribute classifications on the labelled flow datasets are then taken place. The outcome of this step is the most significant attributes for malicious detection. For this purpose, the advantage of using an open-source toolkit, WEKA [167] data mining

package is utilized. WEKA has a collection of popular machine learning algorithms. These algorithms are used for learning the flow characteristics from the labelled datasets (in CSV format) based on the hidden features trained by both malicious and non-malicious traffic. The main goal of using these algorithms is for features classifications and then to find out the most important malicious flow attributes that provide maximum detection accuracy. Then their corresponding rules (from the classification process) of these significant attributes are considered in the next step to improve the accuracy of the detection scripts.

Table 4.3 shows the features selected as the input of WEKA for the purpose of attribute evaluator to generate the most important attributes in each malicious type. These features are widely used in recognition of botnet traffic [90, 99, 175-177]. Where as shown in the table, spam-bot is added since IRC and P2P bots behaviour involves in spamming activities as mentioned in Section 4.3. Thus, extracting spamming features from the datasets is useful for IRC-bot and P2P-bot detection. Where as shown in Table 4.3, source port number of spam and IRC-bot scenarios are not selected. This is because the source port number of these scenarios does not carry potential signs for detection [62]. Also, some features from Table 4.2 are neglected due to derivation based. For example, t_f and t_s are not considered since the *duration* is derived from these features.

These feature selections should be relevant to the behaviour of the malicious types as presented in Section 4.3. For example, in IRC-bot, a number of packets per flow are much related to Ping-Pong (keep-alive) communication used in regular IRC channel.

Ignoring unrelated features will avoid noisy attributes that affect negatively on the classification accuracy.

Table 4.3

Attributes Used for Classification

Attribute	Spam-bot	IRC-bot	P2P-bot
Source port (src_p)	X	X	√
Destination port (dest_p)	√	√	√
Flow duration (duration)	√	√	√
# Packets per flow (pkts)	√	√	√
# bytes per flow (octs)	√	√	√
# Bytes per packet (bpp)	√	√	√
# Bytes per second (bps)	√	√	√

(√) means selected while (x) means not selected

For generating the most significant attributes, Wrapper subset evaluation [93] is used to create all possible subsets from the feature vectors, with the best first search method. Wrapper approach finds appreciate features with a different set of features through a repetitive process of the classification algorithm. Then the best set is determined [176]. With every subset is classified, full training is set, by each classification machine learning algorithms. Wrapper strategy is select since the focus of this chapter is to analyse the relevant importance of the existing flow-level features and to determine the most effective subset of selected features.

Table 4.4 present the classification algorithms used in this research. They are chosen since they are widely used learning approach with better accuracy in botnet detection studies [46, 178, 179]. Narang, et al. [180] found out that J48 [95] and RepTree were

useful classification algorithms for botnet detection. Gomes, et al. [181] also stated that within the botnet traffic classification domain, the most common algorithms used are the tree structure. Based on the accuracy results generated from these algorithms, the most effective features are generated.

Table 4.4

Attribute Selection Setting and Classification Selection

Type		**Description**
Attribute evaluator		Wrapper
Search method		Best first
Attribute selection mode		Full training
Number of folds		5
Classification algorithm	Tree	J48 (C4.5)
		Random Tree
		RepTree
		BFTree
	Rule	JRip
		PART
		DTNB

4.4.2 Detection Policy Scripts

Writing the detection policy script presents the most important step in this chapter. To implement flow-based detection policy script, rules of the most important features derived from the previous step, and rules inspired from the literature are used in the first stage, precondition stage, as explain in the next section. For flow-based detection scripts, in addition to the rules derived from the previous step, P2P and IRC botnet detection are based on the fact that these botnets involve in repetitive patterns during

their communications. Bots require communicating to bot master to receive commands and send requested data.

Even in P2P, bots often exchange information with each other. As stated by Soltani, et al. [25], these communications usually occurs at specific time intervals and make repetitive patterns which can be useful for detecting these botnets. Also, Giroire, et al. [182] observed that a bot must repeatedly obtain new instructions from a C&C server. Thus, a host often contacting a remote host would have higher connection persistence.

4.5 Detection Implementations

4.5.1 Flow-based Detection

For implementing flow-based detection policy script, rules of the most important features derived from the previous step, and rules inspired from literature [183-185] are combined to a form called "rules". These rules are then converted into Bro script syntax. The aim of flow-based detection is to differentiate between normal and abnormal behaviour using statistical information and threshold value. In this work, flow-based detection mechanism primarily has two stages: precondition stage and threshold stage as shown in Figure 4.3. Algorithm 4.1 explains the principle of the flow-based detection mechanism.

Stage I: The aim of precondition stage is to identify all candidate hosts within monitored network that appear to be potentially related to malicious P2P or IRC communications. To achieve this, when flow-based detection receives flow records, these flows must pass the rules mentioned above, before further analysis. In other

words, this stage reduces the number of flows to be further processed at the threshold stage. These rules depend upon malicious activity types. For example, in P2P bots type, the output flows of this stage are the flows that carry P2P traffic characteristics along with malicious patterns.

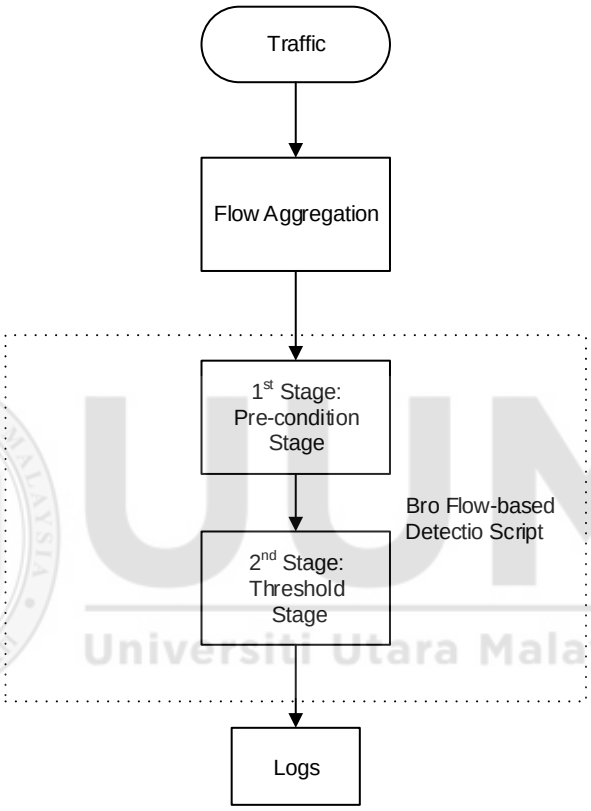

Figure 4.3. Two Stages Flow-based Detection Mechanism

Based on the experiments in this chapter, it was found that P2P bots engage with source and destination port numbers ranged from 1024 to 65535. In addition to port numbers, these rules contain flow statistical information of malicious flows such as number of bytes, packets per flow and flow duration (see the results in Section6.1.2). For example, the ping-pong message in IRC botnet generally holds a fixed amount of packets and bytes per flow [170]. More details on the threshold stage are discussed

further later in this subsection. However, to monitor only local network hosts, scripts were written that scope the monitored host's ranges and set IP address of the only local network hosts. Thus, external IP address is not considered to be detected.

Algorithm 4.1: Flow-based Detection Mechanism

Inputs: flows

Outputs: suspicious hosts

1: Receive flow
2: **if** flow is new and passes the precondition stage **then**
3: increment the number of flow of the host by 1
4: **if** the number of flows of the host >= threshold && the host is not previously marked as suspicious **then**
5: mark the host as suspicious and add it to output log
6: **end if**
7: **end if**

Stage II: The aim of the threshold stage is to analyse the behaviours of the hosts generated by the first stage and classifies these hosts into either normal or bots (IRC-bot or P2P-bot). This stage is to minimize the false-positive alerts generated from the previous stage. The threshold stage is implemented using statistical analysis and based on the following observation. Since bots such as P2P-bots or IRC-bots involve in repetitive communications (in addition to spamming), the number of flows to a distinct host is greater than that found in standard, benign communication (this fact was verified in the experiments).

With this fact in mind, a botnet may engage in malicious activities when the number of incoming or outgoing flows of a bot exceeds the normal behaviour. To achieve this, a script is added in the threshold stage to record and count the number of flows of every host IP address received, within a specified time interval. For this purpose, a counter value is given to each local host. Also, it is also required to create the so-called *Periodic table* for incrementing and updating the number of flows for each host. If the number of flows of a particular host exceeds a threshold, the host is considered malicious bots. To ensure that these repetitive-pattern malicious activities are repeatedly occurring at this stage, a script is added to track the malicious IP addresses every period. Such repetitive patterns occur when ping-pong messages are exchanged between client and C&C server at regular intervals to check whether the client host is alive. To avoid false negative alerts, the detection scope should not be narrow, but this will be on the account on false positive alert.

4.5.1.1 Threshold-based Mechanism

In threshold stage, fixed threshold is not the optimal choice in existing networks. In this work, however, the threshold is adaptively set according to the traffic volume changes. The principle of this dynamic threshold calculation is explained in Algorithm 4.2. However, the threshold is updated and calculated periodically (e.g., at a scheduled interval of 50 seconds), depending upon the type of attack. In this work, the threshold value is changed periodically based on the mean number of a total number of flows. Besides the mean value is a common statistic dimension, it is selected based on the findings from BotMiner [63], where stated that the number of connections a bot engages is more than the average number of connections of all participating hosts. In

another word, it assumes that the bots belonging to the same botnet would share similar communication patterns compared to the non-bot hosts.

Algorithm 4.2: Threshold Calculation in Flow-based Detection Mechanism

Inputs: *Periodic Table*

Outputs: threshold value

```
1: Set initial scale = 10
2: for every host in Periodic Table do
3:      total number of flows = total number of flows + Periodic Table [host]
4: end for
5: total number of hosts = length of Periodic Table
6: if number of host = 0 then
7:      Set average = 50 to avoid dividing number by zero
8: end if
9: if number of host != 0 then
10:     average = (total number of flows divided by total number of hosts)
11: end if
12: new threshold = average * scale
13: return new threshold
```


To get the mean threshold value, it is calculated based on the total number of potential concurrent flows associated with a host in relation to the total number of all hosts participating (in the periodic table) within a predefined period. To calculate the actual threshold value, this mean value is then multiplied by a fixed scale number. This scale number is set to save the operator effort of manual determination of an appreciate threshold. This scale number need a careful adjustment since it affects the threshold calculation. The high threshold value may miss potential malicious activities. On the

other hand, the low threshold value may trigger many suspicious hosts which could increase false positive rate. In the flow-based detection, to avoid any programming bugs when launching flow-based detection, and the threshold is not calculated yet, an initial threshold value is set. To adjust the initial threshold and scale values, an experiment was run to study the behaviour of botnet activities from the datasets. Based on this experiment, the best results were obtained when the average of the scale value is set to 10. Basically, for each botnet scenarios, a script that calculates the average total number of flows that bots involve was written at the threshold stage.

However, when the calculation considered all hosts exist in the periodic table even the old entries (hosts that entered into the table more than 15 minutes), the threshold value would increase quickly. This might be caused when these old entries have high flows numbers. In this case, the threshold becomes very high which might miss potential malicious flows to be marked. This problem was solved, to make sure that threshold value is reasonable, by adding expiry time for all entries in the periodic table. However, this mechanism records the time of each host entered the table, once the host reaches the expiry time, the host leaves the table. Also, the threshold calculation is calculated in a predefined period.

4.5.1.2 Proof of Concept

For proof of concept and to study the feasibility of the flow-based detection mechanism setup, an experiment with default Bro analysis and flow aggregator is implemented. The aim of this experiment is to verify and validate overall processes of flow-based detection. In the whole thesis, this experiment is referred as “live

experiment“. In this experiment, the following simple scenario is chosen. Bro scripting code reads live traffic from the Internet as shown in Figure 4.4. In this scenario, P2P normal activities connections are selected. To analyse P2P activities, BitTorrent [161], eDonkey [174] and Kazaa [186] P2P legitimate applications are executed on the machine and then flow-based detection processes the corresponding flows. These legitimate applications are selected since they are desirable in this verification in which they involved in repetitive UDP communications [187]. In this case, it is required to set a simple condition (for the first stage) in the flow-based detection script that considers UDP protocol with port ranges; depends on the type of application. Two command instances were run in this experiment:

- Softflowd was run to read live traffic and convert it into flows
- Bro flow-based was also run to analyse the flows received from Softflowd

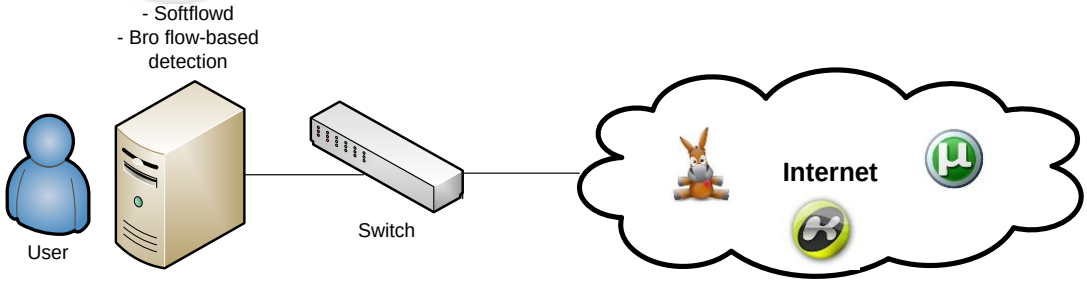

Figure 4.4. Live Experiment for Proof-of-Concept

From the proof of concept perspective, the scripting code of flow-based detection mechanism was verified and validated. For Netflow functionality, *netflow.bro* policy

file provided by Bro are extended for flow record analysis. Although no errors bugs are obtained, the following issues were observed.

Because flow aggregator can produce several flow records with the same connection (e.g., when downloading a large file) [16], repeatedly analysing these records in flow-based detection might waste resources [19]. It was observed in this experiment that flow-based detection processed a huge number of existing flows rather than new flows. Therefore, better resource consumption might be obtained if flow-based detection processes only new (unique) flows. This issue was solved by adding a script to determine whether the received flow is existing in the flow table by matching the flow key 5-tuple. When no match occurs, this flow is considered new flow and will further be processed (see the seond line in Algorithm 4.1).

4.5.2 Packet-based Detection

Having shown how the flow-based detection mechanism is implemented, this subsection implement the packet-based detection mechanism. This subsection presents how packet-based detection mechanism is implemented. Signatures obtained from literature [68, 188] are combined to form "signatures" for packet-based detection. Wurzinger, et al. [68] generated a network-level botnet signatures based on the botnet command-response pattern from real botnet traffic collections. They extract these signatures by observing bot behaviours captured in a controlled environment and recording its network traces. Then they identify points in a network trace that likely involved in IRC and P2P botnet activities. For example, one of the common payload signature for P2P-bots is such information form as `*.mpg;size=*`, where *

represent decimal numbers [68, 189]. More details on how they extracted the signatures are discussed below.

First, malware is collected and executed for a period of a few days in a controlled environment. The malware is then allowed to send its C&C traffic into the Internet. All incoming and outgoing traffic are recorded for the automatic signature extraction process. Change point detection is used to determine “most relevant” traffic patterns in which they will be extracted and collected. The idea is that these relevant patterns will be seen often and can therefore be automatically identified. Such patterns include when a bot master issues a command to its botnet, which results in scanning behaviour. The command and response signatures can be inferred for arbitrary C&C protocols, e.g. IRC or P2P. The generated signatures can be built into the Bro and can be then be used for botnet detection.

In this work, payloads of incoming packets are compared with these signatures. If these payloads match with these signatures, then it is considered as an intrusion. The signature engine of Bro provides high-performance pattern matching that is separately from the normal script processing. Bro relies mainly on its scripting language for defining and processing detection policies. For defining scripting language, the signature below describes an example of such scripts

```
Signature example {

ip-proto == udp

dst-port==6666
```

```
payload /.*.mpg;size=*/

event "P2P bot is found" }
```

This signature requests Bro to match the regular expression *.mpg;size=* on all UDP packets going to port 6666. When the matching occurs, Bro raises an event signature_match with alert named "P2P bot is found". It also returns the payload content which triggers this event. The expression matches against either the raw payload of a connection (for TCP connections as in IRC-bot) or for each packet (for UDP connections as in P2P-bot).

This work also utilizes default and built-in signature-based detection scripts, such as *ircbot.bro*, provided by Bro [190]. Also, Summary Statistics, or so-called "SumStats" mechanism is developed by Bro to observe and analyse application layer data by efficiently summarizing network activities [191]. Figure 4.5 shows sample of SumStats script approach where the rest of the script can be found in Appendix F.1. Signatures expression from Snort can be used and then converted to Bro script format. These signatures can be found at Bleeding Edge Threats [192, 193], an organization that delivers signatures for serious attacks including botnets. Signature conversion processes had been automated in the past, but for the time being, they are converted manually. In addition, Porras, et al. [59] provides Snort botnet payload rules that can be easily converted into Bro syntax in this work (see Appendix F.3).

```
event connection_established(c: connection)
{
# Make an observation!
# Each established connection counts as one so the observation is always 1.
        if(c$id$resp_p == 25/tcp) {
                SumStats::observe("SMTP conn",
                           SumStats::Key($host=c$id$orig_h),
                        SumStats::Observation($num=1));
        }
        if(c$id$orig_p == 25/tcp) {
                SumStats::observe("SMTP conn",
                           SumStats::Key($host=c$id$resp_h),
                        SumStats::Observation($num=1));
        }
}
```

Figure 4.5. Sample of SumStats Scripts

4.6 Evaluation Environment

Once detection scripts of flow-based are implemented, it is important to validate and evaluate this implementation to determine possible false negatives and positive alerts, in addition, to study the resource consumptions, compared with packet-based detection. To do so, the detection policy scripts that were implemented from Section 4.5, must be applied to packet-based and flow-based detection shown in Figure 4.1 (a) and Figure 4.1 (b). For effective validation, new datasets that include same malicious activities in addition to background traffic are used as input data. P2P-bot detection is validated with ISOT dataset while in IRC-bot, CTU-51 dataset is used (see Section 3.4.4). Each of this malicious dataset is combined with PSCJ-2 trace as additional background dataset. In the evaluating environment, datasets are replayed into the detection machine.

Testbed illustrated in Figure 3.6 is used in the experiments. All commands mentioned in the measurement procedures (see Section 3.4.3) are used in this chapter. However, in all experiments conducted in this chapter, two detection instances are running on

the second machine. The first instance is to launch flow based detection to collect flow records from flow aggregator (Softflowd) to find malicious flows based on the policy script. The second instance is for packet detection to collect packets from the live interface. All these detection instances run in an identical environment in term of input source (datasets), and traffic speed. FL and PO are experimented in sequence but with the same datasets and platform. The output logs are then manually observed to determine whether the known IP addresses of the infected machines (that produce malicious flows) are detected or undetected.

4.7 Chapter Summary

In this chapter, since there is no flow-based detection built-in in Bro; flow-based detection mechanism was implemented from scratch. To implement flow-based detection mechanism, characteristics of malicious activity were required. For this purpose, a workflow process was designed and implemented to extracts malicious flow features against several labelled datasets. Packet and flow traffic analysis are applied on these datasets to generate malicious features. From these malicious features, machine learning algorithms were used to generate the most important attributes and rules that will be useful for implementing the first stage of flow detection mechanism. Then flow-based detection policy scripts were implemented in two stages: precondition and threshold stages. These scripts analyse the behaviour of flows based on certain rules (precondition stage) and threshold-based (threshold stage) strategy. For validating flow detection methods, different recent labelled datasets were used. The evaluation results are presented in Chapter 6.

CHAPTER FIVE
CONDITIONAL HYBRID INTRUSION DETECTION

5.1 Introduction

In the previous chapter, flow-based detection mechanism was designed and implemented. The idea was to derive detection scripts from malicious labelled datasets and using machine learning to extract malicious features from those datasets. Evaluation results (see Section 6.1) showed that flow-based detection showed a significant amount of false positive alerts compared with packet-based detection. Also, several works implemented flow-based detection methods for botnet attack and their methods still suffer from generating false positive [18, 54]. Although a lot of efforts have been made to reduce the number of false alarms generated by NIDS (as discussed in Section 2.6), having an NIDS with no false alarm is almost impossible [194]. The impacts of producing false positive have negative consequences on the network performance. Such impacts include:

- The network operator has to be overwhelmed with a lot of annoying alarms. However, the attackers can exploit this situation by overloading the system monitored by an IT staff. Hence network infrastructure defence becomes weaker.

- With generating huge unjustified alerts, the value and urgency of true alert are diminished. When a real attack occurs, its alert is handled as it was within false positive alerts. Hence false negative occurs [111].

- System resources become exhausted when receiving and processing a large number of false positives.

As discussed in Section 2.6.1.6, a few works combine flow-based and packet-based detection for enhancing NIDS scalability and reducing false positive rates [9, 22, 23]. In [9], no results were reported since the implementation is under deployment. The resource consumption of the work [22] was challenged since all packets, regardless they are suspicious or non-suspicious, are processed by packet-base detection. In their performance results, no performance on detection accuracy analysis in NIDS was presented. In [23], the author highlighted the significant overhead consumption of Broccoli [131] method that enables communicating process between flow-based and packet-based detections. Also, in their work, high-speed volume measurements was not considered, and a full payload of the packets was captured and inspected. Instead of using labelled datasets, these approaches used live traffic for their evaluation measurements. Live traffic has limitations, regarding the repeatability of experiments, for any systematic performance evaluation study.

This chapter proposes a mechanism named Conditional Hybrid Intrusion Detection (CHID). The aim of this mechanism is to reduce the false positive rate of flow-based detection by combining flow-based with packet-based detection to compensate for their mutual drawbacks. To be specific, Flow-based detection identifies a reasonable number of suspicious hosts that are likely bots. The traffic of these hosts can be forwarded to packet-based detection for further analysis. Also, this combination approach can significantly reduce resource consumption of packet-based detection by

reducing the amount of traffic to be applied to this detection. Thus, the scalability of packet-based detection can be improved while preserving detection accuracy. Also, high volume traffic environment and partial payload inspection in packet-based detection are considered in this Chapter. However, this research proposed a mechanism that enables communicating process between flow-based and packet-based detections in order to exchange the flow-based suspicious IP addresses. The main contribution of this chapter are as follows:

- Design CHID mechanism to reduce the false positive alerts caused by flow-based detection and to reduce the resource consumption of packet-based detection.
- Design a mechanism that combines the method of IF and BPF filter to communicate between flow-based and packet-based detection modules in CHID approach.

This chapter is organized as follows: Section 5.2 presents the design and theory of the proposed detection mechanism. In Section 5.3, the practical requirements for designing and operating the proposed mechanisms are implemented. In this section, several strategies are presented for this CHID mechanism. To test the feasibility of these strategies, proof of concept is experimentally implemented until the final prototypical implementation is revised. This section also proposes Input Framework (IF) method to exchange data between packet-based detection and flow-based detection. Finally, Section 5.4 present the evaluation of the proposed detection

mechanism. This section discusses the attack selection and experimental environment where the proposed mechanism is evaluated.

5.2 Proposed Mechanism

This section discusses how the combination of two approaches, flow detection, and packet detection, can reduce false positives from flow-based detection and reduce the resource consumption of packet-based detection while maintaining the level of detection accuracy.

5.2.1 Design and Theory

The idea of the proposed mechanism design is to obtain the advantages of a flow-based NIDS approach by having a small amount of data to be processed and the advantages of a packet-based NIDS approach by having a low false-positive rate. With these two approaches, the proposed mechanism is based on the following strategy: one approach is used for the first inspection (to mark traffic as suspicious), and the second one is used for further inspection to confirm the decision made by the first inspection.

To identify the first and second inspector, Figure 5.1 and Table 5.1 are presented based on the literature and the experimental results from the previous chapter. As presented in Figure 5.1 (a), flow-based analysis has better scalability (see the y axis) due to light resources consumption compared with packet-based analysis. On the other hand, considering the high resource usage of packet-based detection, it degrades the system scalability. Where as shown in Figure 5.1 (b), however, flow-based detection generates false alerts that need to be verified by the other inspector. This degrades the level of

alert verification. On the other hand, because packet-based detection has high detection accuracy, it is more suitable to be used to verify an alert made from the other inspector.

In the combined approach theory, two approaches are possible. The first approach is to set flow-based detection as a first inspector and packet-based as a second inspector. The second approach is the opposite. To identify the candidate approach among them, Table 5.1 are derived based on Figure 5.1. It is observed that the first approach is anticipated to have better scalability and alert verification level. On the other side, when placing packet-based detection in the second inspector, low scalability level might occur with low alert verification from flow-based detection. Thus, the first approach has been chosen as a candidate for CHID for the rest of the thesis. A scenario describing the chosen approach is presented in the next subsection.

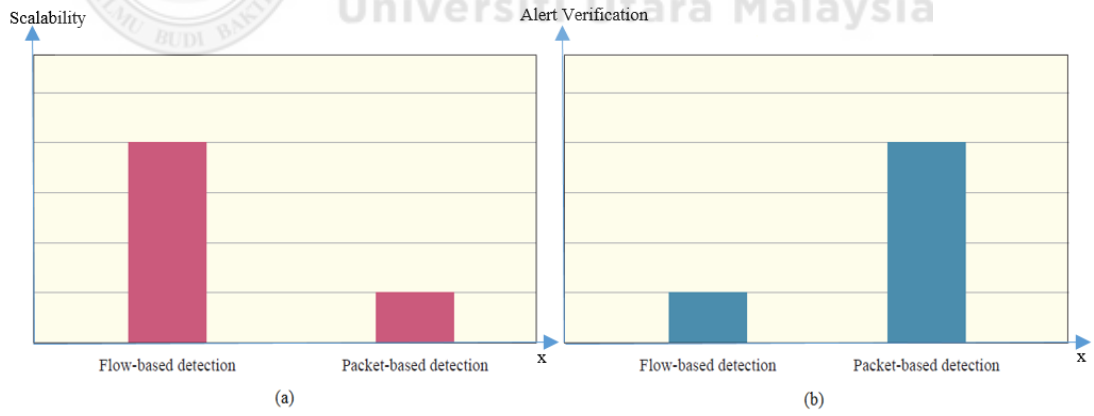

Figure 5.1. Flow-based and Packet-based detection with a) Scalability Level and b) Alert Verification Level (x-axis indicates the detection type)

Table 5.1

Two Combination Approaches

Combination Approach	Anticipated Performance
Flow-based → Packet-based	Good Scalability with High Alert Verification
Packet-based → Flow-based	Bad Scalability with Low Alert Verification

5.2.2 Combination Approach Scenario

The following scenario describes the importance of the chosen approach that combines flow-based and packet-based detection. This scenario shows the need of packet-based NIDS as the second layer to verify the false positive generated from flow-based NIDS. In the scenario, as shown in Figure 5.2, Host A communicates with Host B with the existence of flow-based NIDS. The NIDS processes the flows generated from the communication between the two hosts. It is assumed that the abnormal (intrusive) behaviour occurs when the number of flows in a connection exceeds the predefined threshold in a short time. This intrusion might be classified as port scanning that prepares for an attack.

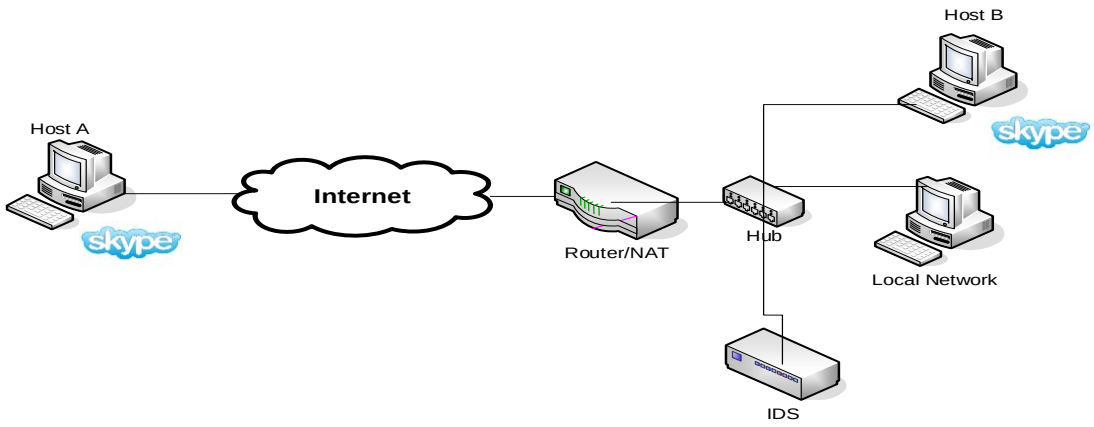

Figure 5.2. Two Hosts with P2P Communications

It was observed that host A communicates host B with matching the case above. Hence it was alerted as an intrusion. After investigation the cause of this activity, it was found that host A was trying to call host B via Voice Over IP (VoIP) application (Skype), at the back of NAT, many times in short period. Thus, the sender (host A) did not involve in malicious activity to attack the host B. Eventually, false positive was generated by the flow-based NIDS.

From this scenario, it can be seen that distinguishing between normal (benign) and intrusive activities are the main challenge of flow-based NIDS. Thus, NIDS decision relying on flow-based only cannot be certain and accurate. In this case, another detection layer is required which is packet-based NIDS to make a final statement for the decision. This is because packet-based data provides useful information to NIDS about the traffic such as the signatures and malicious code within the data [9].

Similar scenarios can be seen when a benign traffic is marked as an intrusion. For example, when large flows corresponding to HTTP connections attempt to a web server, this activity is not necessary to be always an intrusion. Even a sweep through the entire web addresses when looking for a web server, this should not be considered as an intrusion. Such activities might happen when some search engines perform port scanning to search for the web server to crawl and index the corresponding websites. Thus, flow-based NIDS does not proof the intention behind those activities whether they carry the malicious attack. Hence packet-based NIDS is required as second detection layer. This problem arises when proxies, NAT (as seen in the first scenario),

and DHCP are used since many IP addresses may correspond to a single address or vice versa.

5.3 Implementation

This section presents the practical requirements for designing and operating the proposed hybrid CHID mechanism. Two strategies of the hybrid mechanism are presented in this section in the context of the proof of concept: traffic recording and subsequent-packet. By implementing a first proof of concept, it is explained why the recording traffic strategy as hybrid mechanism had to be discarded. On the other hand, the subsequent-packet strategy will be considered in this chapter as CHID mechanism, as discussed in the following subsections.

5.3.1 Traffic Recording Strategy

Again, the goals of CHID mechanism are to verify alerts that are produced by flow-based detection and to reduce the resource consumption of packet-based detection. To achieve these goals, packets corresponding to (or triggered) these alerts are retrieved for further inspection by packet-based detection. However, the issue is that these packets might already have exited the link by when suspicious flows are generated; therefore, these packets become not retrievable. In this case, when suspicious flows occur, a packet-based NIDS would not be able to request the relevant packets for payload analysis. To overcome the mentioned issue and to ensure inspection of past traffic, the captured traffic is recorded into a PCAP file using Tcpdump. This strategy is named as Traffic Recording. Figure 5.3 shows how this strategy works. When

recording into the file, packet-based detection later reads the file in offline mode to retrieve and then inspects only suspicious traffic.

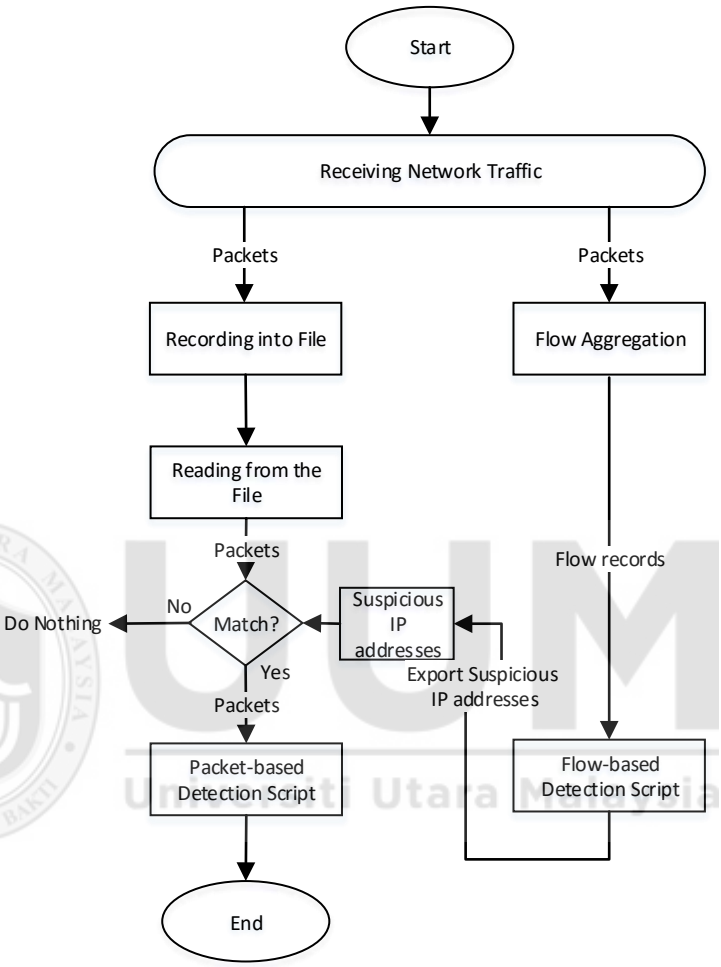

Figure 5.2. Proposed Flow Chart with Traffic Recording Strategy

To test the feasibility of this strategy method, proof of concept is implemented experimentally. The same scenario conducted in the live experiment (presented in Section 4.5.1.2) is used. In this experiment, flow-based detection read live traffic that carries P2P communication flows. After executing P2P applications on the machine

to inspect the transmitted traffic, the following instances are launched on the Bro machine:

- Softflowd command was run to read live packets and converted into flows.
- Bro flow-based detection was run to read and analyse the flows received from Softflowd.
- Tcpdump command was run to record the live traffic and save it into recording file.
- Finally, Bro packet-based was run in offline mode to read the recording file generated from tcpdump.

When to initiate the experiment, Tcpdump starts recording the live traffic into a recording file in PCAP format. Packet-based detection starts reading offline from the PCAP file and inspects only the traffic that matches with the log file based on host IP addresses. Flow-based detection performs well without problems and errors. However, after one minute of launching, packet-based detection stop reading the PCAP file, although tcpdump still running and recording. The cause of this issue was identified by monitoring the experiment. The cause is that packet-based detection was faster than tcpdump recording which leads to the internal issue that it was not possible to be solved. Since the recording approach could not be validated, it is discarded in this work.

Another method of reading the past traffic is to use Bro Timemachine method to go back in time and retrieve only certain parts of the captured traffic [195]. However, Timemachine method does not meet the requirements and is discarded for the following reasons. Timemachine method was developed to record traffic in days rather than hours. Thus, its query functionality requires much time to retrieve sufficient data from the captured traffic. This delay can negatively affect NIDS performance, particularly when the number of queries increases. Another challenge facing the Timemachine approach is interruptions that might occur when performing queries while capturing packet operation. In regular basis, Timemachine is more suitable for recording traffic, not for querying and retrieving traffic. To overcome the mentioned issues, the subsequent-packets strategy is proposed, which is explained in the next subsection.

5.3.2 Subsequent-Packet Strategy

In botnet activities, repetitive attacks frequently send similar traffic and patterns in future connections [25, 168]. AsSadhan, et al. [196] assumes that a bot involves in a repeated pattern behaviour. The authors then explore this behaviour by analysing and looking for periodic components in botnet traffic traces. As a result, botnet traffic exhibits periodic behaviour. The communications between bots and the C&C servers are based on either a pull or push mechanism [64]. Depending on the mechanism used, each bot often either contacts or is contacted by other bots to updating requests for command and control process, receive commands, discovering peers in P2P structure, and send keep-alive messages.

This pattern exists in bots regardless of the structure of the botnet (e.g., centralized or P2P) and the communication protocol being used between bots and the C&C server (e.g., IRC, HTTP) [196]. Moreover, this pattern happens in a periodic manner for a given bot, and it occurs because a bot's communication with the C&C server repeats itself with a certain pre-programmed period. This pre-programmed behaviour to perform similar routine communication with the C&C server and for the same botmaster [5, 196].

As a conclusion, these observations implies that these malicious activities can be detected and verified by inspecting their future packets using packet-based detection. With this fact in mind, these activity characteristics can be utilized to verify the flow-based alerts by inspecting related packets in the corresponding future traffic instead of past traffic. This strategy is named as "Subsequent-packet". However, the advantage of this strategy is the ability to read the traffic in live mode instead of offline mode.

Figure 5.4 illustrates how the hybrid mechanism using subsequent-packet strategy works. It works as follows: initially, Bro in packet-based is adjusted with the Berkeley Packet Filter (BPF) to exclude all traffic. Later, when flow-based detection generates suspicious IP addresses, the packet-based detection adds these IP addresses into the capture filter so that from now on, only incoming traffic that matches these suspicious IP addresses is subject to inspection by packet-based detection for further analysis. If packet detection finds an intrusion, it updates the intrusion log file. Otherwise, it is considered a flow-based false positive. From now and onwards, in this thesis, this strategy is used as the basis for CHID mechanism.

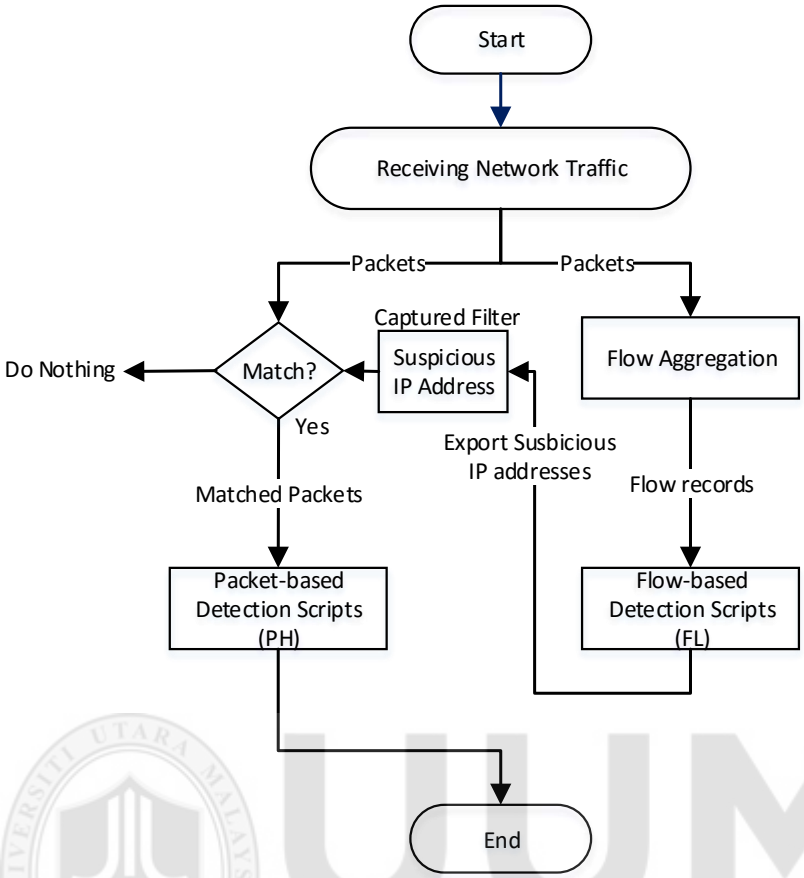

Figure 5.3. Proposed Flow Chart with CHID Mechanism

5.3.3 PH and FL Communicating Process Implementation

The proposed mechanism, CHID, mainly has two components: PH and FL. Each detection runs in separate Bro instance. Since PH has to import the suspicious IP addresses from FL, communicating process should be implemented. This section presents how the communicating process between these two detection approaches is implemented. Two methods of communicating process are presented in the context of the proof of concept: BPF-only and IF methods. By implementing first proof of concept, it is explained why the BPF-only method had to be discarded. On the other hand, IF method will be considered in this chapter as communicating process method

in CHID mechanism, as discussed in the following subsections. This section also explains the integration between IF method and Bro detection engine.

5.3.3.1 BPF-only Method

To ensure that packet-based detection in CHID approach, or PH, is only capturing the suspicious IP addresses, PH should communicate with FL to read the suspicious log file. This subsection presents a method for importing the suspicious IP address using BPF (by adding `-f` in the Bro instance command). It also presents the proof concept of this method with the same scenario from live experiment (presented in Section 4.5.1.2) is performed. The following instances were run in this experiment:

- Softflowd was run to read live traffic to export flows
- Bro FL was run to read and analysed these flows and generated a file that stored suspicious IP addresses.
- Bro PH was run to read live traffic and capturing based on the IP address filter BPF. It uses "`-f`" as parameter followed by IP addresses generated from FL.

FL mechanism is similar with one conducted on Section 4.5.1, but with a little customization to add IP addresses (which conduct P2P activities) to the file. For PH script, print function (to print all IP addresses that PH processes) was only added to make sure that PH was triggered based on the captured filter. When running the experiment, BPF managed to exclude all traffic when the suspicious file was empty. This means that the captured filter, as shown in the Figure 5.4, is working properly.

Then FL started reading flows from Softflowd and started generating IP addresses into the suspicious file as shown in the Figure 5.4. Unfortunately, after a while, PH did not capture any packets while FL instance was running. However, PH captured the packets only when PH instance was restarted; then it captured (using BPF) the future traffic matching the IP addresses successfully. PH printed all IP addresses that captured by BPF, and it worked without any fault and was error-free. However, it was observed that to update BPF with new IP addresses, it was required to restart (re-launch) PH. However, to automate BPF update, a script was added to schedule PH to be launched every 30 seconds.

This method (BPF only) worked faultlessly. Filtering must be updated as fast as possible for better performance. As experienced in this experiment, this method was not practical since scheduling this update resulted in a significant delay, thus should not be the optimal choice. Also, resource consumption was badly affected since restarting PH many times required additional CPU overhead. With these mentioned issues, this BPF-only method was discarded.

5.3.3.2 Input Framework (IF) Method

As an alternative to the BPF-only method, Broccoli method [131] can be used for this purpose, because it subscribes to events of other Bro instances. However, based on literature, it suffers from high overhead consumption as reported by [23]. The overhead consumption was caused due to more frequent updates of the BPF filter. Thus, the Broccoli method is discarded in this research.

The alternative method to the Broccoli method is to combine the BPF with the most recent novel framework developed by [132], which is called “Input Framework (IF)”. The IF architecture can be implemented on top of the Bro NIDS. Unlike Broccoli method, IF method integrates external information in real-time into an NIDS source without negatively affecting the NIDS’s main task, even in high-volume environments. In CHID mechanism, it allows PH instance to import the suspicious log file from FL instance and stores it in a table.

The following sections explain how to implement IF method and integrate it into detection scripts. Then a script-level Application Programming Interface (API) to configure the file input sources is discussed. Also, IF method is customized to meet the CHID requirements.

5.3.4 IF Method Integration

Figure 5.5 shows how IF implementation integrates into packet-based detection mechanism, PH. The input manager acts as the interface between the PH analysis engine core and file sources. It receives a request from the PH engine to open a stream, issue a new file reader thread, and then instructs it to connect to the corresponding file source. The reader passes it on to the manager when reading data. The manager then (after converting the data into a Bro format) feeds the data into PH engine, either the event stream or directly into the user scripts as shown in Figure 5.5.

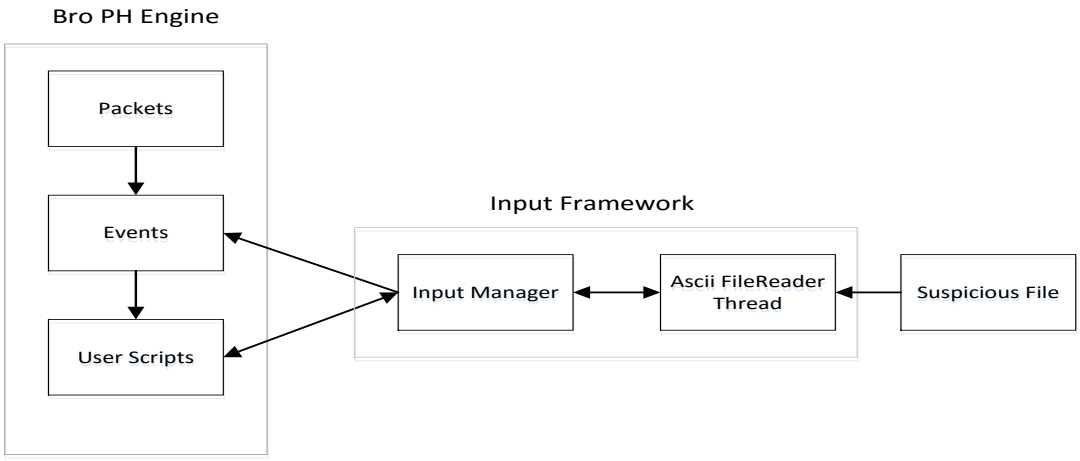

Figure 5.4. IF Method Integration into PH

The IF implementation integrates fully into PH's domain-specific scripting language. In the following, the main parts of the script-level interface that the IF method exposes to the user are discussed. As a simple running scenario, importing IP addresses of suspicious hosts from a file named *suspicious_log* (generated from FL), formatted as a 2-tuple (IP address, reason) where IP address is the host's address and the reason a textual description of the host's offense. Stored in a tab-separated file (log file), the list looks like this:

```
Ip              Reason
147.16.2.195    P2P-bot suspicious
147.16.2.192    P2P-bot suspicious
147.16.2.196    P2P-bot suspicious
```

5.3.4.1 Reading Files

The IF method can directly import files such as the above into tables. To do so, it is required to declare the columns to extract from the file by defining two corresponding record types (records are similar to *structs* in C programming language): one for the

table index and one for its values. In the scenario above, assuming the *IP address* as the table index and the *reason* as its value, the following types can be defined:

```
type Index: record { ip: addr; };

type Value: record { reason: string; };
```

When reading the *suspicious_log* file, the IF method will use the records' field names (*ip* and *reason*) to locate the corresponding columns, and it will interpret their content according to the fields' types (addr is Bro's built-in script types for IP addresses). Next, the table that will receive the content of the file is defined as follows:

```
global suspicious_log: table[addr] of Value;
```

Note that the types for table *index* and *values* must correspond to the *Index* and *Value* records, respectively. Now that the types and the table are defined, the IF API function is used to read the suspicious IP address from the *suspicious_log* file as shown below:

```
Input::add_table(source="/home/suspicious_log.log",
idx=Index, val=Value, destination=suspicious_log);
```

When executing the `add_table` function, the IF's manager issues a new reader thread (as shown in Figure 5.5). While the new thread is analysing the *suspicious_log* in the background, it continuously forwards entries to the manager, which in turn puts them on the suspicious table accordingly.

5.3.4.2 Updating Table

The *suspicious_log* sources will see frequent updates at regular intervals. This is since FL keep adding new suspicious hosts. IF implementation mainly provides two mechanisms to accommodate updates. The first mechanism is to call the API's function manually to refresh of an input stream. The IF method may also individually add, delete, or modify table entries once they get updated. This mechanism is preferable if one knows when to expect a change. The second mechanism is to put a reader into automatic update mode. In this mode, the reader thread continually re-read the source file for modifications and trigger the update operation automatically. The IF method will then add any new values to the corresponding table or remove ones that no longer exist. With the second mechanism, IF API function looks like this:

```
Input::add_table(source="/home/suspicious_log.log",
idx=Index,val=Value ,destination=suspicious_log,
$mode=Input::REREAD);
```


In this work, this mechanism (automatic update) is selected. This is because the changes in the suspicious file are unexpected and may occur at any time. The advantage of using IF method for re-reading processes is avoiding PH instances from extra overhead caused by being launched multiple times to update the table. It is required to customize IF method process to provide a file reader that can read a file once at start-up and then continuously monitor it for any changes and trigger the update operation automatically. In CHID mechanism, to minimize PH consumption,

this re-reading process is called only when there is a new entry added to the log file, as shown in the flow chart in Figure 5.6.

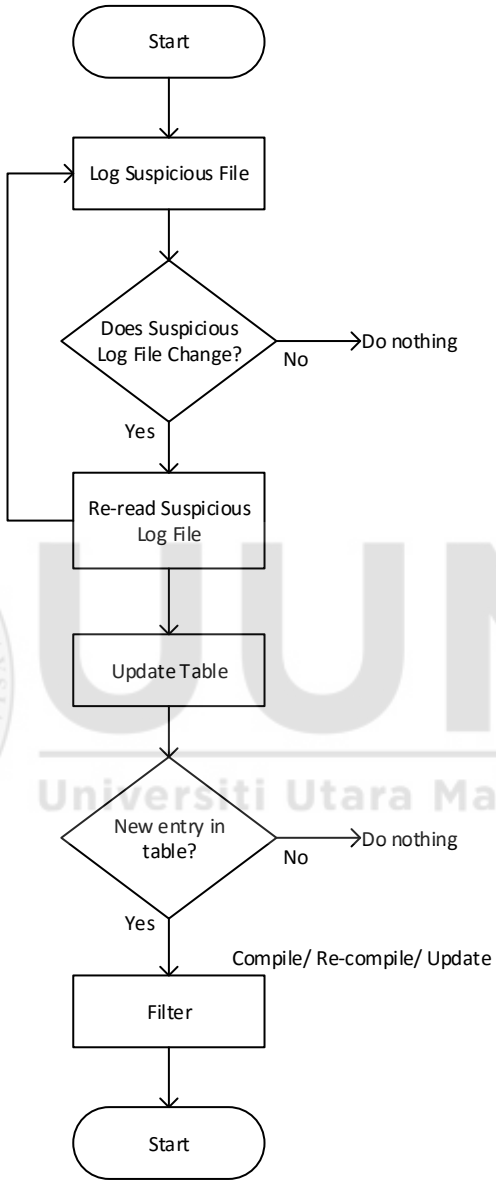

Figure 5.5. Combination of IF and BPF Filter Approaches

5.3.4.3 BPF Filtering

Now that the integration of IF method into PH detection is implemented, a combination mechanism of IF and BPF is implemented in this subsection so that BPF

filter can read the suspicious log file. To filter the incoming traffic in Bro based on IF entries, PacketFilter framework [197] is used. This framework supports how Bro sets its BPF capture filter. By default Bro sets a capture filter that allows all incoming traffic. If a filter is set, then filter takes precedence over the default open filter using capture filters variable. This variable is a BPF filter that is used by default to define what traffic should be captured.

Figure 5.6 illustrates how the BPF filter reads the suspicious log file through IF communication. After IF method updates its table, PH captures the packets that correspond to the IP addresses found in that table. When the table is empty, BPF excludes all traffic. The BPF filter frequently reads the entries from the table and updates the filter to add them to the capturing process. To perform this function, the compilation is needed. To avoid unnecessary filter compiling overhead occurring when updating the filter, this update (or filter re-compilation) should only occur when the table is changed.

5.3.4.4 Proof of Concept

The proof-of-concept is performed to test the feasibility of CHID mechanism (with subsequent-packet strategy) after integrating IF approach. Again, the live experiment (performed in Section 4.5.1.2) is used. In the experiments, three instances were running. One for Softflowd and the other two are for FL and PH. Softflowd and PH read live traffic. FL script is similar with one conducted on Section 4.5.1 and is customized to add IP addresses, which conduct P2P activities, to the file. For PH

script, it is suggested to add print function to print IP addresses of processed traffic and to make sure that PH was triggered based on the captured filter.

Initially, when started the experiment, BPF excluded all traffic when the suspicious file was empty. FL started reading flows from Softflowd and generated P2P IP addresses into the log file. PH added these IPs into the captured filter via IF method and kept updating this filter whenever needed. PH starts capturing incoming traffic that only matches with these IPs (it was verified by printing all IP addresses processed by PH). When added a new IP address in the file, the table and the filter were automatically updated with that address successfully. In this experiment, no errors or bugs are found. Appendix F.2 shows PH code for IRC-bot detection based on the flow chart shown in Figure 5.6.

However, in the log file, every entry should have an expiry time. Once this expiry time is reached, the entry (suspicious IP address) is deleted. This process ensures that entries are updated with only recent suspicious traffic. Also, in this experiment, it was proofed that filtering (or indexing) by only matching on IP address instead of on 4-tuple (IP address and port number of source and destination) was more practical. First, if filtering by 4-tuple is performed, many unique records will be entered into the flow log file, and hence, many requests will be made by PH to collect the relevant packets.

To reduce the number of records, it is suggested to filter the traffic by IP address only so that PH detection will inspect all connections to and from this IP address. Second, the compilation time of the filter, when IP addresses only were applied, requires less time compared with when port numbers were considered. With this simple

experiment, it was concluded that the CHID mechanism worked correctly and will be considered in the evaluation process.

5.3.5 Partial Payload Inspection Approach

Although PH detection mechanism inspects selected packets from traffic, the whole payload of the packet is analysed for detecting intrusions which might be time-consuming especially in of high volume network. Researchers such as [22, 198, 199] found methods to select portion of the payload to be captured and analysed without compromising the security. This portion should be very relevant to security properties. When the client and server intend to transfer bulk data, handshake process and data request and response must be performed first which it happened at the beginning of every connection. In the data request and response exchange, sensitive information for authentication is transferred, such as in HTTP GET or POST request. When the attacker uses web exploitation against the victim, the signature can be found in this request.

Münz, et al. [199] found that the most security relevant portion is focused at the beginning of a connection. The authors analysed rule sets of the Snort to get knowledge about the relationship between the rules and the amount of payload needed. It can also find the bytes of payload that do not contain interested information and ignore them without impairing the detection accuracy. They concluded that 90% of all Snort rules could be found in the first 145 bytes. With this fact in mind, the portion size of payload to be inspected by PH detection can be adjusted. Bro provides such scenario by resizing the packet payload using the *snaplen* function. This scenario is

considered in the evaluation to determine what the optimal payload size is that provides better resource consumption with accurate detection.

5.3.6 Switching Approach based on Traffic Rate

In current network environments, the amount of traffic is not at constant volume; it is rather in between low and high volume. In low volume network, under 80 Mbps for example, Bro packet-based mechanism can process all packets with enough resources and without dropped packet [200, 201]. In this case, one could say that flow-based detection or CHID approach is not needed to replace default packet-based detection. On the other hand, in high volume network (more than 100 Mbps), CHID is preferred and needed to reduce analysed packets. In other words, in low volume network environment, CHID approach is not preferable, and its advantages may be destroyed. Thus, switching between inspecting all packets (PO) and inspecting only suspicious packets (CHID) is proposed, as shown in Figure 5.7

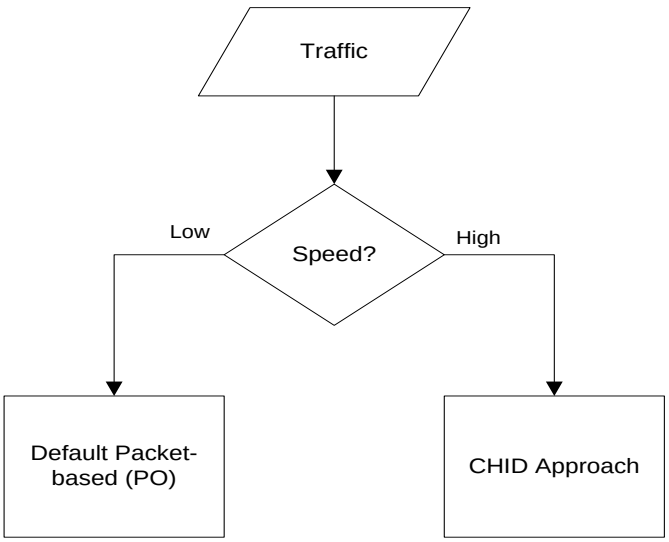

Figure 5.6. Switching between CHID and PO Approaches based on Traffic Rate

Algorithm 5.1 describes how switching between processing all packets approach and CHID approach. In this section, this switching approach uses the total number of received packets within a certain interval as an indication of the volume speed. For example, when Bro receives more than 1 Million packets (threshold) in 15 seconds (time interval) (that corresponding to 66,666 packets per second), it can be considered high volume network. This value (total number of Received Packets or RP) will be used to be compared with traffic volume threshold (that are set by a network operator), and it is required to set the time interval as well. On the other words, RP is the total number of packets received since the last stats interval [202].

Algorithm 5.1: Approach Selection based on Traffic Rate

Inputs: total current packet received all time, total number of last packets received

Outputs: change status of filter and flow aggregation if needed

```
1: Set traffic volume threshold value = 1000000
2: Set time interval = 10 seconds
3: If (reading live traffic) then
4:          current received packets since the last stats interval (RP) = total current packets
5: received all time – last packets received
6: if (RP > volume threshold) then
7:              if current traffic volume is already high
8:                      do nothing
9:             End if
10:              if current traffic volume is already low then
11:                          resume flow aggregation
12:                          Set filter “ON”
13:               End If
14: End If
15: If (RP < volume threshold) then
16:        If current traffic volume is already low then
17:                  Do nothing
18:           End if
```

```
19:         If current traffic volume is already high then
20:                 Pause flow aggregation
21:                 Set filter "OFF"
22:         End If
23: End If
```

To calculate the RP value, two values are required. The first value is the total current packet received at the end of the interval and the second value is the total packets received at the beginning of the corresponding interval. The RP value is calculated as following formula:

$$RP = \text{Total Number of Packets Received All Time} - \text{Last Number of Packets Received} \quad (5.1)$$

RP value is recalculated regularly based on the time interval set. In any time, if RP value is more than the volume threshold value, this means the traffic is under high volume and vice versa. When switching from low volume to high volume, flow aggregation (Softflowd) should be executed to feed FL with flows. In this case, PH activates IF and filter process. On the other hand, if the switching occurs from high to low volume, then Softflowd is suspended to relieve FL from extra unnecessary overhead. In this case, packet-based detection should inspect all packets with default configuration by inactivating filter process.

To study the feasibility of this method, proof of concept mechanism is experimented. The scenario in the live experiment (presented in Section 4.5.1.2) could not be used in this section because, in live traffic, it is not possible to adjust the traffic volume speed

manually as needed in this experiment. Thus, it is suggested to use the controlled testbed environment presented in Section 3.4.2 with taking CTU-52 dataset (see Section 3.4.4) as a scenario. This dataset is replayed using tcpreplay to adjust between high and low speeds.

This experiment adds *stats.bro* load as a module in PH script to calculate RP value. The volume threshold and time interval are set to be 1000,000 packets and 10 seconds respectively. This threshold value is chosen because based on the experiments at high traffic rate, it was found this value to a correspondent high megabit per second. For suspending and activating flow aggregation and filtering process, Linux script shell was added and linked to Bro system. In this simple experiment, the following instances are required:

- Tcpreplay was run on the first machine to replay the dataset to Bro in the second machine.

- Softflowd was run to read traffic from the network interface and convert the packets into flows.

- FL was also run to read and process the flows and then generate the suspicious file.

- PH was run to be able to read filtered traffic from the network interface.

During this experiment, traffic volume rate was purposely adjusted between low and high volume to verify the switching traffic approach. Traffic speed more than 100

Mbps was considered as high volume [47]. RP value was updated every 10 seconds and compared with a threshold value, and the Linux script was performing accordingly. The switching traffic approach was verified and validated without error or bugs reported from this experiment.

5.4 Evaluation

Having shown the implementation of the CHID mechanism, this section presents how this implementation is evaluated. This is to ascertain the possibility of CHID mechanism to reduce resource consumption while maintaining detection accuracy when compared with the default packet-based detection or PO.

The following subsections initially present and justify the attack scenarios selections used in this evaluation. This section also presents the comparative and experimental environments for the evaluation of the proposed implementations. Finally, it is required to explain measurement procedures used in this evaluation before discussing the datasets used in this measurement.

5.4.1 Attack Scenarios

To evaluate the CHID mechanism, attack scenarios that fit the mechanism should be prepared. Based on literature [25, 168, 169], flow-based detection yields promising results when detecting botnet activities that perform repetitive traffic patterns. Repetitive attacks mean that the attacks generate similar traffic patterns in future communications. Such repetitive patterns may include 1) when bot-infected machines frequently connect to C&C servers to receive commands, 2) when a bot keeps sending

spam emails to distribute a botnet or 3) when “keep-alive” messages are sent from time to time for an IRC-botnet. With these points in mind, IRC-bot and P2P-bot as malicious types are selected to be considered. Also, these cases of botnet attacks are believed to be the most strong–threat to the security of Internet-connected users and systems [24].

The other reason behind selecting these attacks is that they match the hybrid mechanism strategy (see Section 5.2). Thus, an attack can be marked as suspicious in the first inspector (FL) and can be detected and verified in the second inspector (packet-based detection mechanism or PH), which can access data (payloads) not available in FL. More details on this point concerning IRC and P2P botnets, refer to Section 4.3. In this chapter, FL and PO detection scripts, for IRC-bot and P2P-bot, are adopted from the previous chapter in Section 4.5.

Similar to IRC-bot and P2P-bot attacks, there are other attacks that can be detected in both flow and packet level. Such attacks include HTTP-bot and brute-force attacks. However, the proposed mechanism is not designed such that a combination of flow and packet detection is a suitable solution for enhancing performance in all cases. For example, DoS attacks, scanning, and probing are not suitable cases for the proposed approach because they do not include significant payloads for detection. Therefore, they do not meet this requirement.

5.4.2 Experimental Environments

In this chapter, comparative analysis is performed between CHID mechanism (FL+PH) that has customized Bro packet-based scripts and PO mechanism that has

default Bro packet-based detection configuration. Also, the performance comparison occurs in two situations: 1) packet-based detection is directly applied (PO), 2) packet-based is applied with CHID approach (PH). In other words, the detection accuracy and the total resource consumption in the CHID approach are calculated to perform a direct comparison with Bro PO. PH and PO share the same signature detection. The difference between them is that PH is involved in IF and BPF techniques. For accuracy detection evaluation, it is required to compare the log files generated from both PH and PO. The CHID and PO mechanisms are experimented in sequence with the same datasets and platform.

In this evaluation, both the CHID and PO mechanisms receive the traffic from datasets by replaying these datasets. CHID and PO mechanisms are run on testbed depicted in Figure 3.5 but with adding PH mechanism in the second machine. Bro is installed in the second machine with default configuration for PO while customized for PH to accommodate the IF and BPF methods.

In all experiments, three detection instances are running on the second machine. The CHID mechanism uses two detection instances, one for FL to collect the flow records from flow aggregator (Softflowd) to find suspicious flows based on the policy script. The second instance is for packet detection PH to collect packets from the live interface based on the filter process. The third Bro instance is used for independent packet detection PO, which inspects all incoming traffic. All of these detection instances run in identical environments regarding input sources (datasets), and data volume rate.

5.4.3 Measurement Procedures

To evaluate each of detection script (FL, PH, and PO) and to keep comparable results between them, the standard evaluation procedures defined in Section 3.4.3 are used. However, two instances are added to these evaluation procedures: PH instance and CPU affinity. These instances are explained below.

a. PH

PH is executed by launching this instance:

```
bro –i eth0 hybrid-packet-based.bro
```

The –i parameter specify the corresponding Ethernet interface where PH reads from. As seen in the command, eth0 is used since it is connected directly to the port-mirror on the switch where traffic forwarded from tcpreplay machine. For defining subnet monitored hosts, it is treated similarly to FL command. Finally, this command loads *hybrid-packet-based.bro* policy script that establishes BPF, IF, and corresponding signature engines. Appendix F.2 shows the content of this policy scripts. In this script, it is required to add the sub-directory of the suspicious file inside IF script. However, the file must be empty before the next experiment start. This is to make sure that PH exclude all incoming packets at the beginning of the experiment.

b. CPU Affinity

Bro is a single-threaded application, meaning that the design has been constructed in which it does not break its threads and send them to multiple cores. Thus, it does not

get the advantage of the multi-core architecture. Since the machines used in the evaluation support multi-core processing, each detection instance to be run assigned on separate core using CPU affinity tool [203]. This is to minimize resource consumption effects by the system. This can be achieved by using *taskset*, command that is Linux-based tool as follows:

```
taskset core-number processes-ID
```

5.4.4 Traffic Data for CHID Mechanism

This subsection presents datasets used for each resource consumption and detection accuracy for CHID mechanism measurements. To study the resource consumption of CHID and PO implementations, CTU-52 and ISOT datasets are selected for IRC-bot and P2P-bot scenarios, respectively. In addition, PSCJ-2 is used as background trace instead of PSCJ-1 due to the larger size of the PSCJ-2. However, each of malicious datasets is combined with PSCJ-2 background traffic and injected to detection machine.

For detection accuracy measurements, more datasets are included to get better testing results. Thus, IRC-bot scenario is evaluating using CTU-51 and CTU-52 datasets while P2P bot activities are evaluated on both ISOT and CTU-53 datasets as shown in Table 5.2. Since these datasets are labelled, it will be useful to validate the accuracy of CHID and PO methods. Both PSCJ-1 and PSCJ-2 are combined to these malicious datasets to support the evaluation of false positive measurements. However, CTU-50 is discarded in this evaluation since it is not considered as an attack scenario in this chapter.

Table 5.2

Datasets for Detection Accuracy Measurements

Scenario	Datasets for Detection Accuracy Measurements
P2P-bot	ISOT, CTU-53,PSCJ-1, PSCJ-2
IRC-bot	CTU-52, CTU-51, PSCJ-1, PSCJ-2

5.5 Chapter Summary

In this chapter, a mechanism named CHID for improving NIDS scalability was presented. CHID mechanism is based on combining flow-based and packet-based detections to build on their advantages and overcome their drawbacks to reduce the resource consumption of NIDS. CHID mechanism is proposed to verify alerts that are produced by flow-based detection and to reduce the resource consumption of packet-based detection. To achieve these goals, subsequent-packets corresponding to these alerts are retrieved for further inspection by only packet-based detection. CHID mechanism is implemented in Bro NIDS and utilize the IF method to communicate between flow-based and packet-based detection mechanisms. CHID and PO implementations are evaluated by replaying labelled datasets, ranging the traffic rate from 100 to 1000 Mbps. Evaluation results of the proposed approach are discussed in Section 6.2.

CHAPTER SIX
RESULT AND DISCUSSION

In this chapter, evaluation results and discussion of the proposed mechanisms are explained. First, Section 6.1 study the performance of the flow-based detection mechanism, which was implemented in Chapter 4. Then, the performance of CHID mechanism is presented in Section 6.2.

6.1 Flow-based Detection Scripts

After implementing the workflow processes that extracted malicious flow features from datasets as presented in Section 4.4, most significant attributes were generated to build flow-based detection policy scripts. This section presents these attributes and evaluation results in term of detection accuracy and resource consumption of these flow-based detection scripts.

6.1.1 Dataset Correctness

To check the correctness of labelling decisions in the datasets obtained from the public and presented in Chapter 4, these datasets were analysed and verified manually using Bro analysis logs. For example, after extracting and analysing the flows initiated from the internal hosts who were labelled as bots in the ISOT dataset, it was observed that these machines exhibit unusual network usage comparing to other machines. These machines engaged in a large number of SMTP connections (7,699 flows) within a short time of period, by sending packets (with randomly source email addresses and advertising words in the email's subject) to many external servers on port 25.

However, only one machine (172.16.0.2) in ISOT dataset seems to have either incomplete traffic or been wrongly labelled as bots. All other labelled machines as infected machines on the other datasets were found to be correctly labelled.

6.1.2 Most Significant Attributes

For the most significant attributes used in flow-based detection scripts, Table 6.1 lists the best three important attributes with generated by different classification algorithms for each malicious type. These attributes were observed to be very useful in the flow-based detection script. From these attributes, the following rules were observed and merged into flow-based detection:

- Spam-bot: *dest_p == 25/tcp && pkts < 83 && duration <= 5 sec*
- P2P-bot: *dest_p from 1024/upd to 65535/udp && pkts <= 17 && duration <= 12 sec*
- IRC-bot : *pkts < 8 && octs < 570 && dest_p ≠ 6667/tcp, 6668/tcp*

Destination port number was often employed as an absolute feature in identifying botnet traffic [176]. For example, for bots involves spam activity detection, it is obvious that destination port is among these attributes since port 25 on the SMPT destination server is a good sign of this malicious type. It is also observed that packets and bytes per flow in IRC-bot are the most significant attributes. These features are mostly intended to represent similar communication patterns and have been used for the purpose of botnet traffic identification [175, 176]. For example, the values of these

attributes are constant and really small as appear at regular intervals (PING/PONG communication, e.g. for when each time keep-alive is exchanged), where normal IRC traffic has more and larger packets as also appeared in [184].

Table 6.1

Best Three Important Attributes

Classification algorithm	Spam-bot	IRC-bot	P2P-bot
J48	dest_p, duration, pkts	dest_p, pkts, octs	src_p, dest_p, duration
Random Tree	dest_p, duration, pkts	dest_p, pkts, duration	src_p, dest_p, pkts
Rep Tree	dest_p, duration, pkts	dest_p, pkts, octs	src_p, dest_p, duration
BF Tree	dest_p, duration, pkts	dest_p, pkts, duration	src_p, dest_p, duration
PART	dest_p, duration, pkts	dest_p, pkts, duration	orig_p, dest_p, duration
Jrib	dest_p, duration, pkts	dest_p, octs, duration	orig_p, dest_p, pkts
DTNB	dest_p, duration, pkts	pkts, octs, duration	orig_p, dest_p, pkts

It was also observed that the traffic generated by bots is more uniform than traffic generated by non-bot hosts. Thus, these attributes are relevant to identify bot behaviour as promised by [90, 99]. Also, flow duration was showed to be a significant feature for malicious botnet detection. It is one of the most useful parameters used in the detection of botnets [176]. This is because the majority of botnets maintain long communication sessions. This characteristic has been widely applied in many botnet detections approaches [175-177].

6.1.3 Detection Accuracy

For detection script validation, when using newly labelled datasets in the experiments, flow-based detection mechanism marked all known infected machines IP addresses in these datasets as malicious bots (FN = 0). This mean that flow-based script can detect all the infected machines that generate malicious flows with 100% detection accuracy. Table 6.2 and 6.3 show the FPR and precision for each malicious type respectively. It shows that flow-based detection mechanism suffers from generating false positive alerts.

Table 6.2

False Positive Rate (FPR)

	IRC-bot	**P2P-bot**
FPR[#]	0.25	0.20

[#]the value is 0 if there is no FP

Table 6.3

Precision Results

	IRC-bot	**P2P-bot**
Precision[$]	0.66	0.42

[$]value is 1 if there is no FP

In IRC-bot detection results, for example, five benign IP addresses detected as malicious were reported from flow-based detection analysis. Also, unlike packet-based detection, incomplete data (no payload data) of the flow-based analysis also plays an important role in producing these false alarms. In other words, payloads provide a significant role for identifying non-malicious traffic. To test false positive

in packet-based detection mechanism, IRC-bot dataset was run and no false positive alerts generated which indicates the accuracy level when the payload is inspected.

6.1.4 False Positive Test

To make a statement on the false positive alert generated from flow-based detection mechanism, further investigation on the traffic features of these IP addresses are performed in two approaches. First, Bro analysis is launched and filtered based on these IP address for manual inspection. After investigating the output log files, the alerts from these IP addresses are caused by the similarity between malicious and non-malicious recorded data. Such data include flow duration, port number, and number of packets per flow.

The second approach involves in public blacklist investigations. However, these false positive alerts generated from hosts that reside at PSCJ campus monitored network where non-malicious traffic are assumed to be generated. To verify whether these hosts involve in any botnet activates, the following steps are performed; 1) identify all IP address of these hosts. 2) list all non-campus network IP addresses that were contacted by an identified IP address; and 3) for each non-campus network IP addresses, look it up in search engines and check whether the address can be found in any blacklisted server that involve any malicious activities. If an identified IP address has contacted a blacklisted peer, it could be most likely a bot. Also, publicly available blacklists such as [204] and [205] are also used. Based on the findings, none of the hosts residing on PSCJ campus are confirmed to be bots. As a conclusion, the

statement can be stated that the false positives generated by flow-based detection are indeed false positive.

6.1.5 Resource Consumption

In this subsection, evaluation results of the resource consumptions of both FL and PO are presented. The importance of this evaluation is to compare between the two scenarios: IRC and P2P botnets, in term of memory and CPU usages, for each FL and PO detection. In addition, this evaluation study how FL can handle the flow records in high speed link compared with PO. This subsection shows the memory and CPU usages while replaying the P2P-bot trace with speed ranging from 100 to 1000 Mbps, in Figure 6.1 and 6.2 respectively.

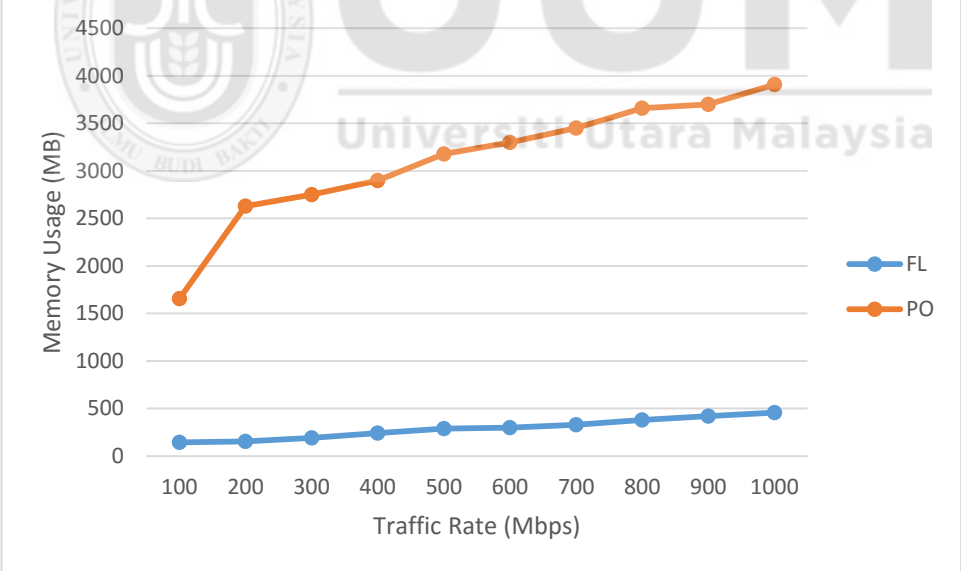

Figure 6.1. Memory Usage with Different Traffic Rates – P2P-bot (FL: flow-based detection; PO: default packet-based only)

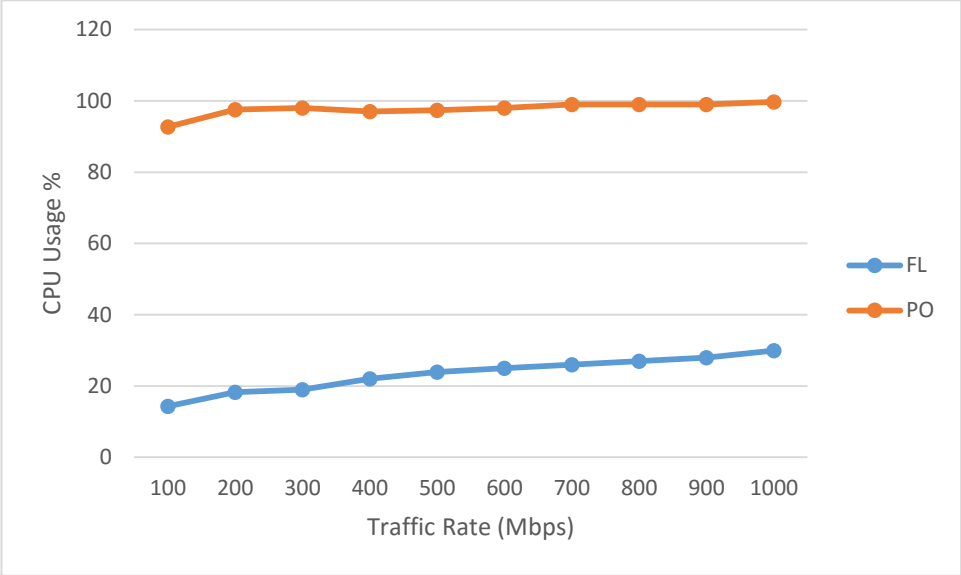

Figure 6.2. CPU Usage with Different Traffic Rates – P2P-bot (FL: flow-based detection; PO: default packet-based only)

It is observed in these figures that PO consumes higher memory and CPU usage compared with FL. When sending packets at a constant rate of 500 Mbps, for example, FL mechanism drops CPU usage from 97% with PO to 24%. This observation is expected since FL processes only flow records rather than inspecting whole packet data. Also, this observation was stated by Golling, et al. [9] that flow-based detection can handle the flow records of the 10 Gbps link. Similar to P2P-bot detection, this observation was true when IRC-bot scenario is measured as shown in Figure 6.3 and 6.4.

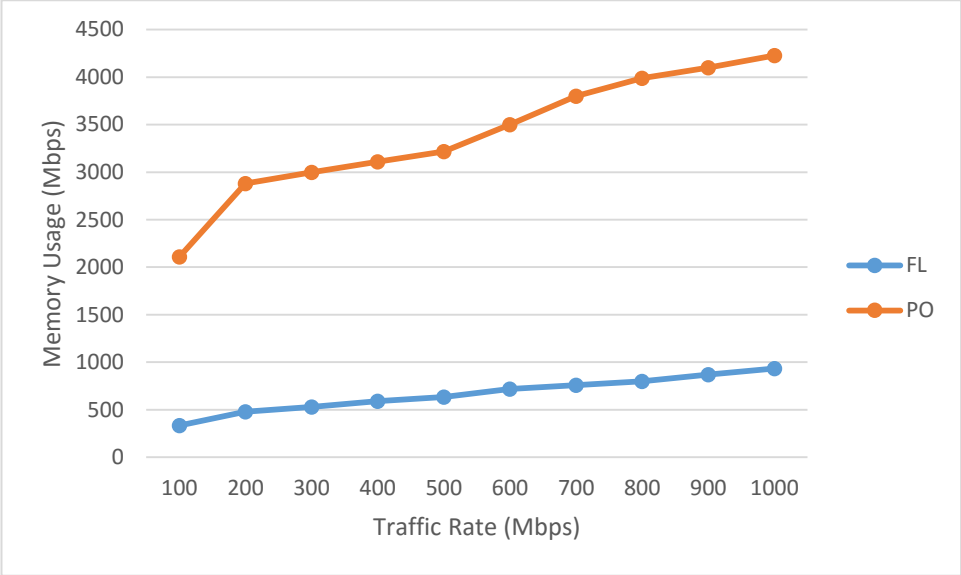

Figure 6.3. Memory Usage with Different Traffic Rates – IRC-bot (FL: flow-based detection; PO: default packet-based only)

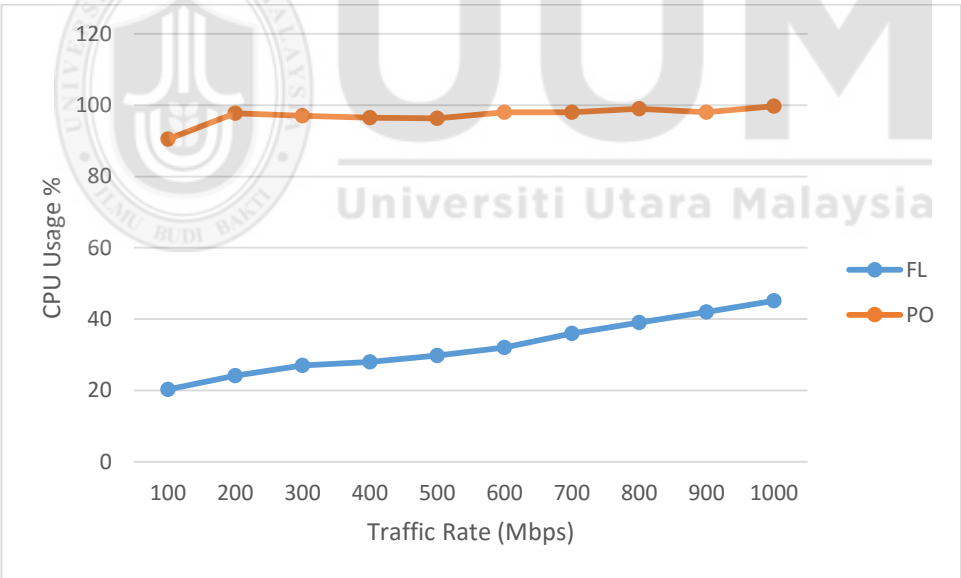

Figure 6.4. CPU Usage with Different Traffic Rates – IRC-bot (FL: flow-based detection; PO: default packet-based only)

FL plays an important impact on resources consumption. However, at all data rates, the CPU usage of FL is slightly higher in IRC compared with a P2P-bot scenario (see

Figures 6.2 and 6.4). When investigating the causes, FL received larger flows when IRC was used. Also, because the second stage in FL detection depends on upon the first stage (precondition), FL processes a huge number of hosts (101 hosts were processed in the second stage as shown later in Table 6.5) that carry IRC characteristics from the traffic compared with P2P (25 hosts were processed as shown later in Table 6.4). When more hosts passed the first stage, more hosts were processed in the second stage, which led to more resources being needed. Such resources include 1) updating the FL table for all hosts used in the second stage of flow-based detection and 2) dynamic threshold calculations for all hosts participating in the FL table.

To study how FL and PO behave over running time, Figures 6.5 and 6.6 are generated to show the memory and CPU usage over time at 200 Mbps rate. These figures show that both memory and CPU usages of FL and PO rise to a certain level and remains at the same level almost all the time. This observation was true with other traffic rates even when IRC-bot case was tested. For FL, for example, this observation could be a result of setting the timeouts, flushing of old states, and expiration values of all entries in the tables to be consistent.

It is also observed in all experiments that when similar flows within the same connection were discarded (as suggested in Section 4.5.1.2), FL process only 22% of total received flows; when P2P scenario at 200 Mbps was measured. This indicates that a huge number of flows, belong to the same connections, were exported in separate flows. Therefore, better resource consumption was obtained when flow-based detection processes only new (unique) flows.

As a conclusion from this section, although FL mechanism showed a significant improvement in resource consumptions, this improvement was on the cost of detection accuracy in term of false positive rate. In other words, the accuracy of FL is sacrificed for the sake of scalability. Since packet-based detection deserves higher score in accuracy and flow-based deserves higher score in scalability, this gives implication in placing this packet-based detection as the second layer to the flow-based detection mechanism to reduce the false positive rate as discussed in the next section.

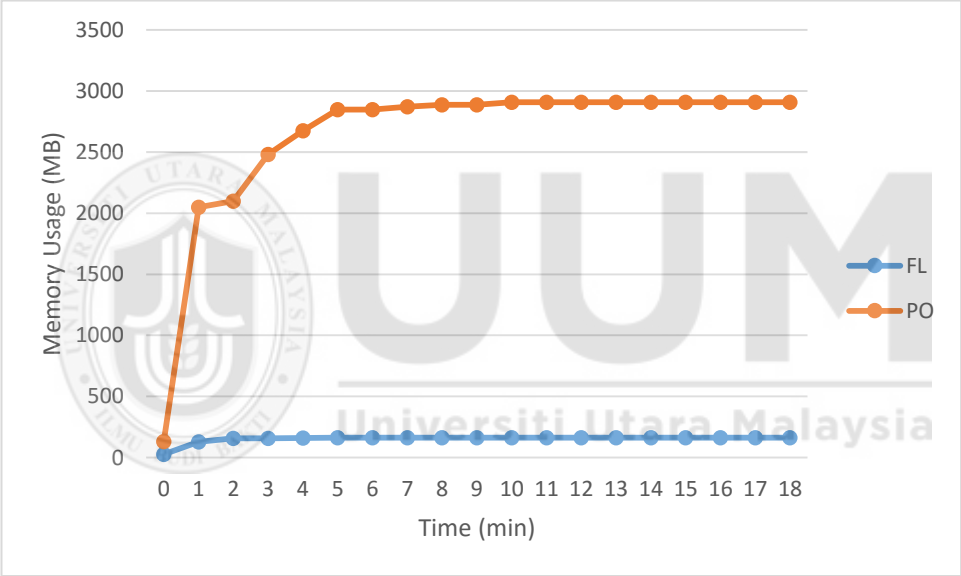

Figure 6.5. Memory Usage over Time at 200 Mbps` – P2P-bot (FL: flow-based detection; PO: default packet-based only)

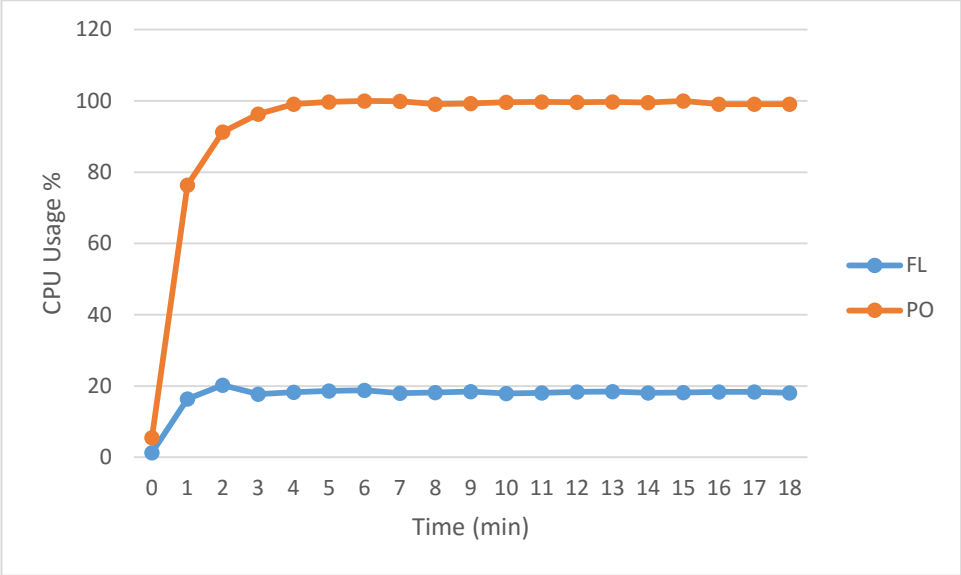

Figure 6.6. CPU Usage over Time at 200 Mbps – P2P-bot (FL: flow-based detection; PO: the default packet-based only)

6.2 CHID Mechanism

In this section, evaluation results of the measurement performance of the proposed CHID mechanism are presented.

6.2.1 Detection Accuracy

According to the findings from Chapter 4 where flow-based detection only was implemented, the results reported a significant number of false positive alerts. However, when the same datasets are input into CHID mechanism, no false positive alerts were identified. This is due to the extra layer, PH, added after flow-based detection in CHID. In all measurements, it is important to mention that the false positive reported from flow-based-only in Section 6.1.3 were mitigated when CHID mechanism is implemented. Based on the findings, PO and PH can detect all IRC-bot and P2P-bot infected IP addresses that are reported and labelled in the traces, see Table

6.4. As shown in the table, in the P2P-bot scenario, six hosts are reported as infected hosts in PH and PO. These six hosts are labelled in their respective traces as P2P-bot hosts.

Table 6.4

Detection Results with P2P-bot Scenario

Number of hosts in threshold-stage	25
Number of hosts marked as suspicious from FL	20
Number of hosts filtered in PH	20
Number of hosts detected as P2P-bot in PH	6
Number of hosts detected as P2P-bot in PO	6
Number of hosts labelled as P2P-bot in the traces	6

With this result in mind, PH yields a high accuracy rate with a zero false negative rate. Since CHID was able to detect all infected hosts reported in the traces, PH is not only able to detect intrusion activities but also means that FL detection plays an important role in detection accuracy as declared by [19]. This is because PH filtered incoming packets depend primarily upon the suspicious list generated from FL detection. Table 6.4 and 6.5 also shows the number of hosts involved in threshold stage in FL detection. Then among these hosts, suspicious hosts are marked based on the threshold mechanism. These tables also show that the number of the suspicious hosts is identical with the number of hosts in the filter. This validates IFs implementations.

Table 6.5

Detection Results with IRC-bot Scenario

Number of hosts in threshold-stage	101
Number of hosts marked as suspicious from FL	17
Number of hosts filtered in PH	17
Number of hosts detected as IRC-bot in PH	13
Number of hosts detected as IRC-bot in PO	13
Number of hosts labelled as IRC-bot in the traces	13

6.2.2 Resource Consumption

Concerning resource consumption, Figures 6.7 and 6.8 show how these resources behave at 200 Mbps over time when IRC-bot case was measured. The results show that CHID (FL+PH) mechanism saves much memory and CPU usage compared with the PO mechanism. This observation is expected because PH processes only potential (filtered) packets rather than inspecting all incoming traffic. To calculate the reduction rate of packet processed by PH, in the case of IRC-bot 200 Mbps, PH processed only 45.5% of total packets processed by PO.

It is clearly observed that it is possible to reduce the resource consumption of packet-based in the CHID approach compared with default packet-based that inspects all packets. These figures show that both memory and CPU usages rises to a certain value and remains at the same value almost all the time. Other results with P2P-bot scenario are shown in Appendix E. These results show similar observations with other traffic rates even when P2P-bot case was tested. These results also show that CHID

mechanism saves much memory and CPU usage compared with the PO mechanism. It was also observed the ability to reduce the resource consumption of packet-based in the CHID approach compared with PO.

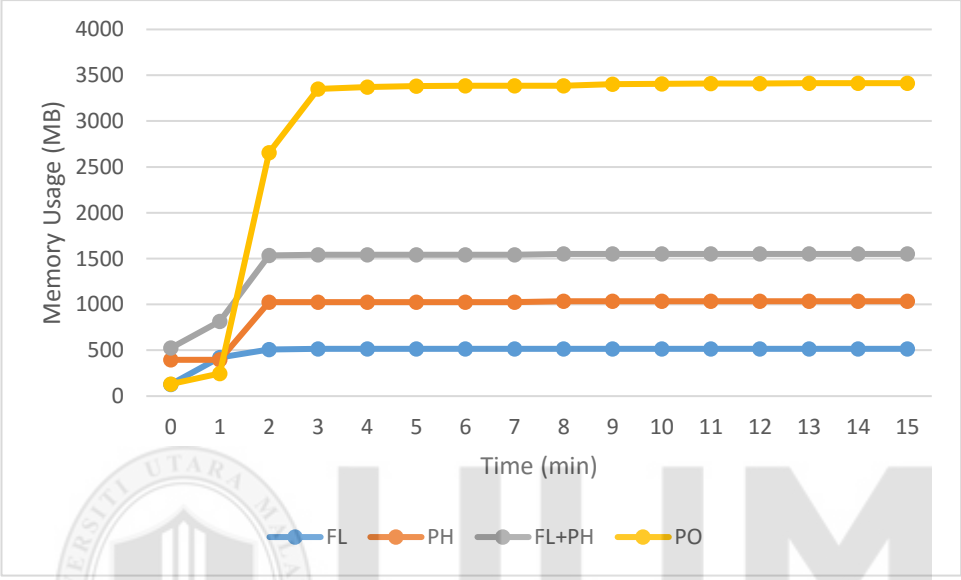

Figure 6.7. Memory Usage over Time at 200 Mbps – IRC-bot

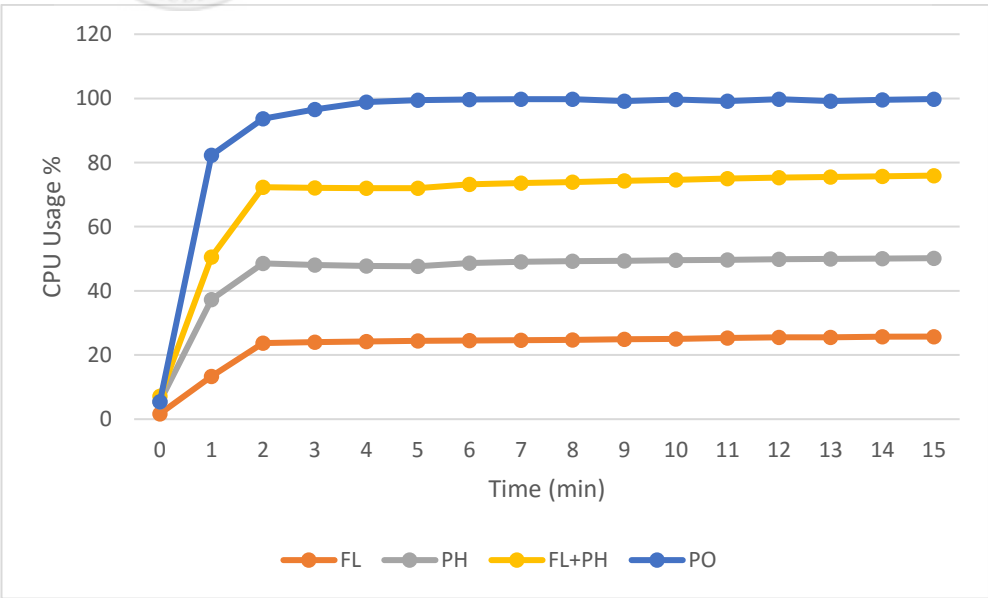

Figure 6.8. CPU Usage over Time at 200 Mbps – IRC-bot

To study the memory and CPU resources behaviour when increase traffic rate, Figures 6.9 and 6.10 show the memory and CPU usages while replaying the P2P-bot trace with speeds ranging from 100 to 1000 Mbps. The results show that CHID (FL+PH) generates less memory and CPU usage compared with the PO method. For CHID, this holds true until traffic rate reaches 500 Mbps. The reason for this is the increased number of filtered hosts as will be discussed in the next subsection. The more hosts that are in the filter, the more traffic is received and consumed. For PH, CPU usage reaches maximum utilization at 1000 Mbps. When sending packets at a constant rate of 200 Mbps, CHID approach drops CPU usage from 97.7% with PO to 72.4% with CHID approach.

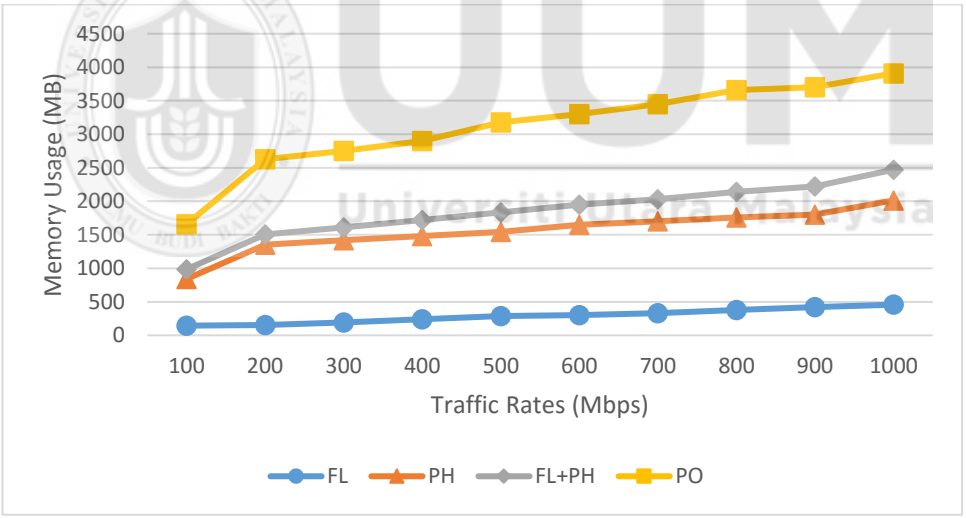

Figure 6.9. Memory Usage with Different Traffic Rates – P2P-bot

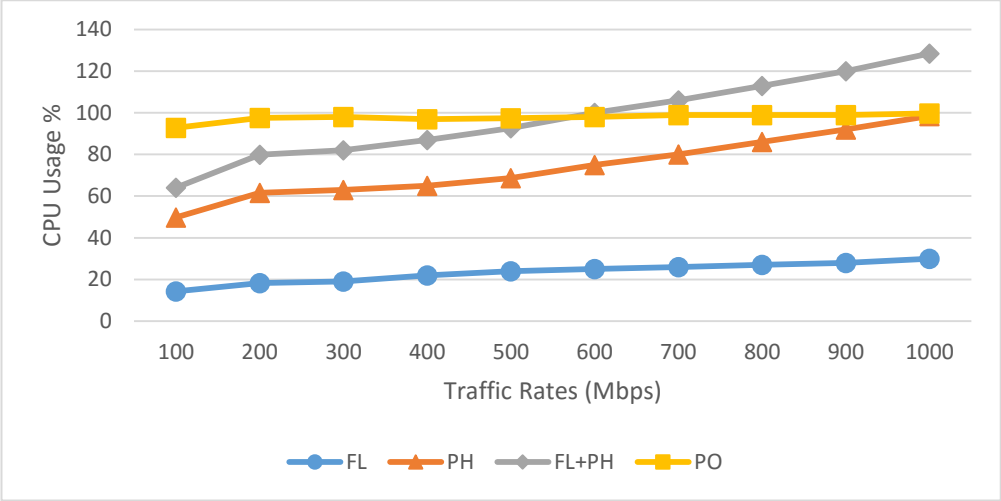

Figure 6.10. CPU Usage with Different Traffic Rates – P2P

Similar to the P2P-bot scenario, the above observation is also true when the IRC-bot scenario is measured. At a 200 Mbps rate, as shown in Figures 6.11 and 6.12, CHID can save 50.6% of memory and 18.1% of CPU usage compared with PO approach. In both scenarios, as shown in Figures 6.10 and 6.12, PH has less resource consumption compared with PO until 1000 Mbps rate. However, the total CPU consumption of CHID approach (FL+PO) reaches at almost the full CPU capacity when the traffic rate is about 500 Mbps. This means that CHID approach can save resource consumption until 500 Mbps rate takes place. Both CHID approach and PO show CPU increase when traffic rate increase. This artifact most likely results from packet capture library (libpcap), which is discussed later in Section 6.2.4.

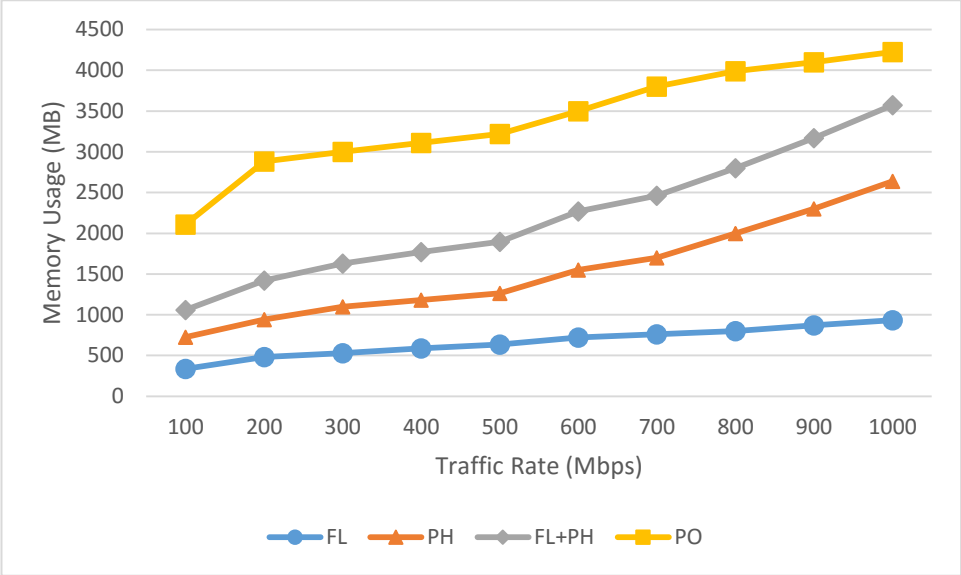

Figure 6.11. Memory Usage with Different Traffic Rates –IRC-bot

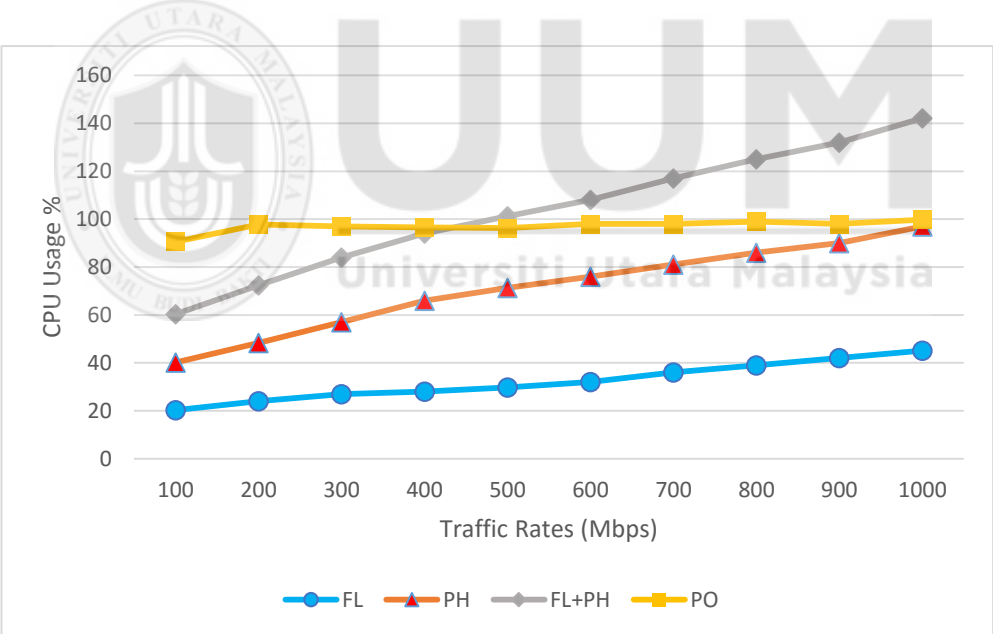

Figure 6.12. CPU Usage with Different Traffic Rates – IRC-bot

6.2.3 Filtered Hosts and IF Method

The number of filtered hosts in PH also affects the resource consumptions. When comparing between P2P-bot and IRC-bot cases, it is observed that the memory and

CPU usage of P2P-bot scenario are higher compared with the IRC-bot scenario. This holds until the traffic rate reaches about 500 Mbps. The reason for this is the increased number of filtered hosts reported from the P2P scenario. With the same reason, when receiving packets at 500 Mbps onwards, IRC-bot scenario consume higher resources compared with the P2P-bot case. The more hosts that are in the filter, the more traffic is received and consumed, and vice versa. However, these filtered hosts require more filter compilation overhead. This compilation is needed to apply filters in the packet capturing operation. In other words, the filter compilation or BPF update resource usages increases when the number of hosts added to filter increases, which might destroy the advantage of the CHID approach.

As mentioned earlier, the resource consumption of packet-based in CHID can be reduced compared with default packet-based that inspects all packets. This implies that the resource consumption of IF method, which is used for multi-instances communication, does not affect negatively on intrusion detection. This finding is also stated by Amann, et al. [132]. However, this implication shows that the approach of combination BPF and IF methods performs better performance compared with Broccoli [131] method implemented by Hensel [23] where the author declared its cause of the significant consumption overhead due to more frequent updates of the BPF filter.

However, IF method may contribute a negative effect on resource consumption if the filtered host number is high. More filtered hosts require more updates to the corresponding table. Moreover, the re-read function must be called every time a new

host is added to the log file generated by FL. Unfortunately, an optimum value of filtered hosts cannot be easily identified. This is because such identification largely depends upon the data in the network traffic characteristics in the traces themselves, i.e., session length, protocols, and the ratio of malicious traffic to benign traffic.

6.2.4 Packet Drop Rate

With concerning packets drop issues, Table 6.6 and 6.7 compare the packet drop with traffic rates from 100 to 1000 Mbps for P2P and IRC bot. Drop packet rate is calculated in relation to the total packet received.

Table 6.6

Packet Drop Rate in P2P-bot Scenario

Traffic Rates	100 Mbps	200 Mbps	500 Mbps	1000 Mbps
PH#	0.00	0.16	5.40	21.12
PO**	5.04	9.32	29.79	48.45

#PH represents packet-based detection in the approach
**PO represents the default packet-based only detection that inspects all packets

Table 6.7

Packet Drop Rate in IRC-bot Scenario

Traffic Rates	100 Mbps	200 Mbps	500 Mbps	1000 Mbps
PH#	0.00	0.00	7.98	27.47
PO**	2.50	5.12	26.02	48.55

#PH represents packet-based detection in the approach
**PO represents the default packet-based only detection that inspects all packets

As observed in the Figure 6.13, it is expected that PO has higher drop rate compared with PH even at 1 Gbps rate. At 500 Mbps in IRC-bot scenario, for example, PH reports 7.98% of drop packet while 26% in PO. This is because of light CPU usage in PH as stated early in this section. Experiments demonstrated that when P2P scenario is measured, PH solution can handle bandwidth up to 200 Mbps with a little drop of 0.16%. With the same traffic rate, no drop is reported when IRC-bot is tested. Based on findings, PO detection can handle up to 80 Mbps rate. For validating the PO implementation, the same observation was also stated by Alparslan, et al. [201] and had been proofed experimentally by Pihelgas [47]. Also, Bro official documentation also stated that a single Bro instance can handle approximately 80 Mbps with a normal load of traffic [200]. Comparing with these works, on the other hand, PH in CHID approach can handle higher rate, up to 200 Mbps. However, in the experiments, the default configuration PO start dropping packets at 100 Mbps.

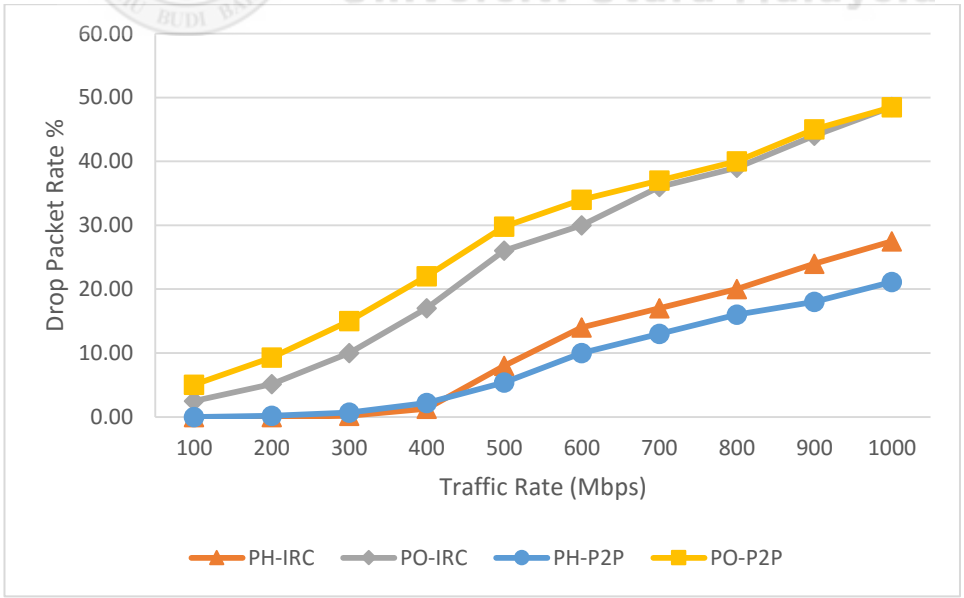

Figure 6.13. Drop packet Rate with Different Traffic Rates – P2P-bot and IRC-bot

It is also observed that PH in P2P starts to drop packets faster than IRC scenario. The reason behind this may refer to 1) the higher CPU consumption at the early traffic rates when P2P scenario is measured as mentioned earlier in this section, and 2) Per-packet processing time which might occur when packet-based detection spends much time on a single packet, either in libpcap or in Bro itself, it may miss the subsequent packets. Also, the processing time of each packet depends on the type of the packets received (e.g. protocol, packet length). For example, processing HTTP packets require a longer time than TCP packets.

However, both Bro PO and PH have shown to dropped packets increase when the speed rate of the packets increase. This is because Bro is a single-threaded technique which means Bro only fully utilizes one processor core and is not taking advantage of multi-core CPU. This will lead to overloading the Bro with a big amount of traffic. Thus Bro is not keeping up with the traffic rate since transmission rates are faster than what Bro can handle which lead the buffer to eventually fill up, and the packets had to be dropped. Another reason for packets drop is the packet capture library libpcap design as discussed later in the next chapter.

6.2.5 Partial Payload in PH

In this section, the experiments of CHID mechanism are run again but with partial payload capturing approach. As discussed in Section 5.4.1, the first N-byte portion of a packet could carry significant and related data to be useful for malicious detection. In the P2P-bot experiments, it was discovered that the first 105 bytes of the malicious traffic were a candidate for intrusion detection. However, for IRC-bot case, PH must

capture the first 447 bytes to detection IRC-bot infected machines. Based on the experiments with these partial payload values, Table 6.8 and 6.9 present comparison between capturing full payload and partial payload in term of CPU usage and packet drop rate, for P2P-bot and IRC-bot cases respectively. In this experiments, the data rate of 100 Mbps was discarded since no dropped packets were reported in this rate as mentioned in the previous subsection.

Table 6.8

Comparison between Full-Payload and Partial-Payload Inspection in PH for P2P-bot Scenario

Approach	Traffic Rate (Mbps)					
	200		500		1000	
	CPU Usage rate	Packet Drop Rate	CPU Usage Rate	Packet Drop Rate	CPU Usage Rate	Packet Drop Rate
Full Payload	61.658	0.162	68.760	5.400	98.562	21.116
Partial Payload	49.511	0.000	55.480	0.001	83.209	0.295

Table 6.9

Comparison between Full-Payload and Partial-Payload Inspection in PH for IRC-bot Scenario

Approach	Traffic Rate (Mbps)					
	200		500		1000	
	CPU Usage rate	Packet Drop Rate	CPU Usage Rate	Packet Drop Rate	CPU Usage Rate	Packet Drop Rate
Full Payload	48.334	0.000	71.378	7.984	96.950	27.473
Partial Payload	40.644	0.000	61.043	0.000	94.693	6.349

As shown in these tables, it is observed that the CPU usage and packet drop rates were decreased when PH capturing partial payload compared with full payload inspection.

Thus, partial payload inspection approach played an important role in improving resource consumption of PH. In this case, PH can handle the traffic transmission until 500 Mbps instead of 200 Mbps when full payload capturing occurred. However, although partial payload inspection approach proof a significant improvement compared with full payload processing, this should not be in all cases. Since N-byte values were determined from datasets that were captured in certain circumstances, it is not possible to make a universal statement of N-byte value for such scenarios since the malicious traffic is dynamic.

Also, selecting the correct N-byte length for a packet capture is critical. If the chosen N-byte value is too long, it may cause performance degradation as observed in the experiments. Moreover, this may cause packets to be dropped. On the other hand, if the chosen N-bye is too short, it is anticipated to miss potential attack which its signature exist in packets that can never be recovered. Thus, it is challenged to limit N-byte to the smallest number that will capture the most relevant malicious data.

6.3 Chapter Summary

In this chapter, it was clear that Bro is a powerful and useful tool for data analysis and feature extraction on the labelled dataset. The workflow process that extracts malicious flow features from datasets as input was implemented. This workflow is found to be useful to generate malicious features that can be used in detection policy script. However, this workflow was designed in a way that it can be applied to any datasets whether is labelled or non-labelled.

When flow-based detection mechanism was run against several labelled datasets to detect malicious activities, based on the experiments, no false negatives are reported. This indicates that flow-based detection implementation along with attributes selection (obtained from machine learning) and the threshold-based mechanism promises high detection rate for flow-based detection. However, false positive alerts were generated which degrades the accuracy of flow-based detection mechanism. It is also concluded that since only flow data is available in flow-based detection mechanism, it is hard to make complete potential behaviour about the malicious activities found in the datasets.

Since packet-based detection mechanism does not report any false positive alerts, the result shows that its resource consumption is far higher than when flow-based detection mechanism is tested. It was also expected that the resource consumption, CPU and memory usage, of flow-based detection was much less than packet-based detection. To reduce the false positive rate in flow-based detection and to reduce the resource consumption of packet-based detection, CHID mechanism was proposed.

The experimental evaluation shows that the CHID mechanism, in both P2P and IRC bot scenarios, could gain a significant performance improvement when using IF and BPF methods, compared with default Bro packet-based detection implementation. When IRC-bot scenario is implemented at 200 Mbps rate, CHID can save 50.6% of memory and 18.1% of CPU usage. With this in mind, it is possible to enhance the resource usage of packet-based in CHID approach. With scalability improving, CHID implementation maintained accuracy detection level and managed to eliminate false

positive alerts generated from flow-based detection and also detect all reported malicious hosts.

The consumption resources of IF method and BPF compilation depend on the number of suspicious hosts generated from flow-based detection. The more hosts produce from flow-based detection; the more resources are needed in packet-based detection CHID approach. A huge number of filtered host results in compilation process overhead which might destroy the advantage of CHID solution. For flow-based detection resources, the number of concurrent hosts processed in the FL table can affect the overall of the approach performance. This is because the hosts require table updating and dynamic threshold calculations.

The experiments also demonstrated that proposed packet-based detection mechanism or PH can handle about 200 Mbps network traffic rate without packet drops compared with 100 Mbps when PO was executed. However, drop packet rate increase slightly beyond this limit. This is because Bro is single-threaded in which Bro does not take advantage of multi-core CPU and only fully utilize one processor core. Moreover, the processing time of each packet may influence packet drop which depends on the type of the packet received. Also, the experiments showed that it is possible to reduce the packet drop rate of PH by capturing the first N-byte of each incoming packets instead of capturing full payload.

CHAPTER SEVEN
CONCLUSION

This chapter summarizes the research presented in this thesis as discussed in Section 7.1. Then Section 7.2 presents the achievements of the three objectives mentioned in Chapter 1. Also, main contributions of this research are highlighted in Section 7.3. Finally, Section 7.4 present the limitation and directions for future works.

7.1 Summary of Research

In this research, the challenges that are existing in IDS were considered. Chapter 1 introduced the motivation for this research and brought up the problems of the inability of intrusion detection to handle the growth of traffic rates. The aim of this research was also presented in this chapter. Chapter 2 provided a background of packet-based and flow-based NIDS approaches in relation to scalability and detection accuracy. It showed that flow-based is the good choice in high volume network while the packet-based is more suitable for detection accuracy. With the serious consequences and impact of false positive generation, this chapter presented several types of research that proposed various approaches to reduce the rate of false positive.

In Chapter 3, research methodology of the study was explained in five phases. For the first phase, it briefly presented the design of the proposed mechanism, CHID. The second phase explained each component in the proposed mechanism to be implemented. These components included: PCAP library was used for traffic capture, open-source Softflowd was selected for flow aggregation. Also, with the open-source, Bro NIDS as a central component, the design of CHID is implemented. In this phase,

verification and validation of these mechanisms and the needs of proof-of-concepts were also discussed.

Evaluation of the mechanisms in term of detection accuracy and resource consumption were presented. In this phase testbed was implemented performance evaluation. This testbed consists of two Linux desktop machines connected through a switch. One machine was assigned to replay the datasets, in different traffic rates ranging from 100 to 1000 Mbps, to the second machines where intrusion detections were installed. Finally, datasets used in this research were presented in details. These datasets were captured, labelled and shared in public. Chapter 4 was established to validate the first objective as discussed in the next section. Second and third objectives were established in Chapter 5. Chapter 6 presented overall performance results of all implementations as detailed in the next section.

7.2 Objectives Achievements

As in this section, the achievements the three objectives mentioned in Chapter 1 are presented.

7.2.1 First Objective

The first sub-objective stated as follows:

To investigate the researches that can improve NIDS efficiency.

To achieve this sub-objective, this research presented many approaches that attempts to improve NIDS scalability and detection accuracy. The efficiency of NIDS depends

mainly on the type of data to be processed: individual packet (with payload) or flows. Packet-based provides full information to NIDS to detect attacks while flow-based provides limited and aggregated information to NIDS for intrusion detection. In term of scalability, since packet-based NIDS process large amount of data, it degrades its scalability, especially in high volume network. On the other side, flow-based is a good choice for this issue since it deals with the small amount of data.

In term of accuracy, information gathered from packet-based is enough for NIDS to detect almost all kinds of attack, hence improving NIDS accuracy. On the other side, since the information gathered from flow-based is aggregated, NIDS accuracy suffers from false alarms. Flow-based detection promises to be able to process data in high volume network with limited data with a trade-off of a higher false positive rate, while in packet-based, it promises to be able to detect intrusion in low false alarms with a trade-off of higher resource consumption with additional data processing.

7.2.2 Second Objective

Since the performance results from Section 6.1 showed that flow-based detection generates a significant number of false positive compared with packet-based detection, the second objective stated as follows:

To develop appreciate NIDS mechanism by reducing the rate of false positive.

To achieve this objective, the performance results of flow-based detection mechanism are compared with the proposed mechanism, CHID. Before implementing CHID, flow-based NIDS was required to be implemented. Since there were no flow-based

detection scripts built-in in Bro NIDS, it was required to implement flow-based detection mechanism from scratch. For this purpose, Chapter 4 proposed a mechanism to build flow-based detection scripts in two stages: precondition and threshold stages. For the first stage, the mechanism was designed and implemented to obtain characteristics of malicious activity by extracting malicious flow features from several labelled datasets. From these malicious features, classification algorithms were used to generate the most important attributes and rules that would be useful for implementing the first-stage flow detection scripts. These rules in addition to threshold-based (second stage) mechanism were used for flow-based detection scripts.

For CHID implementation, Chapter 5 proposed CHID mechanism to reducing the false positive alerts caused by flow-based detection. CHID mechanism was based on combining flow-based and packet-based detections to build on their advantages and overcome their drawbacks. The aim of CHID mechanism was to verify suspicious alerts generated from flow-based detection. To achieve this aim, subsequent packets corresponding to these alerts were retrieved for further inspection by packet-based detection. CHID combined the use of so-called Input Framework and BPF filter methods to communicate between flow-based and packet-based detection modules.

7.2.3 Third Objective

The third objective stated as follows:

To evaluate the developed mechanism by measuring its efficiency through experiments.

To evaluate the flow-based NIDS and based on the findings from Chapter 6, the workflow that was used to extract malicious flow features, was able to generate malicious features that could be used in detection policy script. When validating the only flow-based detection mechanism, this mechanism was able to detect all infected hosts reported in the datasets. This indicated that detection implementation along with attributes selection (obtained from machine learning) and threshold-based mechanism played an important role for high detection rate in flow-based detection. The results also showed that resource consumption, CPU and memory usage, of flow-based detection mechanism was much less than packet-based detection. This was due to the data reduction achieved in flow-based detection.

For CHID evaluations, results in Chapter 6 showed that CHID implementation maintained accuracy detection level and managed to eliminate false positive alerts generated from flow-based detection and detected all reported malicious hosts. It was also observed that the consumption resources of IF and BPF compilation depend upon the number of suspicious hosts generated from flow-based detection. The more hosts identified by flow-based detection, the more resources are needed in packet-based detection in CHID mechanism.

In order to evaluate the performance results of proposed packet-based detection or PH, these results are compared with default packet-based detection, PO. CHID mechanism was used and extended to reduce the resource consumption of packet-based detection. Chapter 6 presented the measurement results of CHID implementation. The experimental evaluation showed that CHID mechanism could gain a significant

performance improvement compared with a default Bro packet-based detection implementation. This improvement holds true until traffic rate reaches a certain value. With this saving in mind, the resource usage of packet-based detection in CHID approach was enhanced.

Also, Chapter 5 implemented CHID but with the inspecting portion size of payload instead of full payload to reduce the resource consumption. The results showed that it is possible to reduce the resource consumption and packet drop rate of packet-based in CHID mechanism by capturing the first N-byte of each incoming packets. Since network volume is in a dynamic mode which is between low and high traffic rate, switching approach between default Bro detection and the proposed mechanism (CHID) was also proposed in Chapter 5. This would avoid using CHID in low traffic where extra unnecessary overhead might occur from both flow-based and packet-based detection.

7.3 Main Contribution

The overall contribution of this research was to develop a new mechanism named CHID that enhances the NIDS scalability. Moreover, the mechanism integrates and combines the two NIDSs approaches: packet-based and flow-based. The specific contributions are:

- The reduction of false positive alerts generated from flow-based detection system was achieved by developing CHID mechanism. The aim of CHID mechanism was to verify suspicious alerts generated from flow-based detection by inspecting packet-based data.

- The enhancement of resource consumption of NIDS with CHID mechanism compared with the default packet-based NIDS mechanism (Bro). For example, when IRC-bot scenario is implemented at 200 Mbps rate, CHID can save 50.6% of memory and 18.1% of CPU usage without any dropped packets.

- The improvements in resource consumption of the communication processes between flow-based and packet-based detection system were achieved in CHID mechanism by combining BPF and IF methods.

- The significant improvement in scalability of packet-based detection mechanism (PH) compared with the default packet-based detection (PO) by only inspecting certain suspicious traffic. Also, PH detection can adjust the payload length (e.g. the first N-bytes) to be captured to reduce the resource consumption while preserving the detection accuracy.

- The improvement of NIDS scalability by tuning (switching) between default Bro detection and the proposed approach (CHID) based on traffic volume rate. This is to avoid using CHID approach in low traffic where extra unnecessary overhead might occur from both flow-based and packet-based detection. On the other hand, default packet-based detection can handle low traffic volume better than CHID approach.

7.4 Limitations and Future Works

a. Multi-thread Approach

The proposed mechanism, CHID, had shown to dropped packets when the speed rate of the packets increases. This is because Bro is a single-threaded technique which means Bro only fully utilize one processor core and is not taking advantage of multi-core CPU. This will lead to overloading the Bro with a big amount of traffic. To address this issue, the developers of Bro implemented a proof-of-concept multi-threaded version of Bro, but is not released yet. The only option is to spread the workload across many cores using cluster-mode [200] that is provided by Bro. In cluster-mode, a set of worker nodes examines independent traffic streams and share their results through a central manager node.

b. PF_RING Packet Capturing

One of the major challenges of NIDS is packet loss in the capturing system. Unfortunately, lost packets cannot be retrieved afterwards. To attempt to optimize the performance of the capturing system, PF_RING can be used. It offers an improvement to the standard PCAP library, or libpcap, mechanism especially with small packets. Since it directs the received packets to a ring buffer, it allows a higher rate of packets per second for monitoring. Also, it allows saving computational costs which make it more suitable for high volume networks environment [133]. However, to enable PF_RING in Bro, cluster-mode [200] must be run. In order to implement cluster-mode, several computer machines are required which need budget and much effort for user configurations. In this research, the maximum traffic rate used in the experiments was

1Gbps. The current network volume, especially in a large organization, is beyond this rate. Higher traffic rate can be considered in future with the hybrid approach.

c. Diverse Attacks Scenario

However, CHID mechanism is not designed as a suitable solution for enhancing the performance of NIDSs in all scenarios. CHID mechanism just demonstrates the possible approach of combining flow-based and packet-based detection in specific scenarios. Similar to IRC-bot and P2P-bot attacks, there are other attacks that can be detected in both flow and packet level with repetitive traffic patterns. Such attacks include HTTP-bot and brute-force attacks. These attacks with combined approach can be considered in the future.

d. Tuning Flow Keys and Timeouts

There are serval factors that affect the performance of flow-based detection. These factors include flow keys and timeout length of flow exporting. Generally, flow records are exported to the collector when the active or inactive timeout of the flow expires. More flow keys and large timeouts can provide the NIDS more and specific information about the traffic. The longer timeouts are applied, the more specific characteristics are analysed, and hence, the accurate decision is made with the lower false positive rate. On the other hand, a short timeout may lead to detecting in near real-time. Tuning these values until the desired performance is obtained can be considered in the future work.

7.5 Chapter Summary

The research presented in this thesis was summarized in this chapter. This chapter also discussed in details the achievements of each of the three objectives mentioned in this research. Having explained the research objectives, the main contributions of this research were presented. Finally, this chapter highlighted the limitations and directions for future works.

REFERENCES

[1] Internet World Stats, "Internet Growth Statistics," 2016, [Online; accessed 8-Dec-2015]. [Online]. Available: http://www.internetworldstats.com/emarketing.htm

[2] J. Nazario and J. Kristoff, "Internet Infrastructure Security," *IEEE Security & Privacy*, vol. 10, pp. 24-25, 2012.

[3] E. Amoroso, *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, and Response*: New Jersey, 1999.

[4] G. Khalil, "Open Source IDS High Performance Shootout," *SANS Institute InfoSec Reading Room*, 2015.

[5] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: A survey," *Computer Networks*, vol. 57, pp. 378-403, 2013.

[6] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion Detection Systems," *Computer Networks*, vol. 31, pp. 805-822, 1999.

[7] N. Weng, L. Vespa, and B. Soewito, "Deep Packet Pre-filtering and Finite State Encoding for Adaptive Intrusion Detection System," *Computer Networks*, vol. 55, pp. 1648-1661, 2011.

[8] R. Koch, "Towards Next-generation Intrusion Detection," in *2011 3rd International Conference on Cyber Conflict*, 2011, pp. 1-18.

[9] M. Golling, R. Hofstede, and R. Koch, "Towards Multi-layered Intrusion Detection in High Speed Networks," in *6th International Conference On Cyber Conflict (CyCon 2014)*, 2014, pp. 191-206.

[10] J. Svoboda, "Network Traffic Analysis with Deep Packet Inspection Method," Master thesis, Faculty of Informatics, Masaryk University, Brno, 2014.

[11] M. Nor, "Malware Detection Using IP Flow Level Attributes," *Journal of Theoretical and Applied Information Technology*, vol. 57, 2013.

[12] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational Experiences with High-volume Network Intrusion Detection," in *11th ACM conference on Computer and Communications Security*, 2004, pp. 2-11.

[13] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "Improving the Accuracy of Network Intrusion Detection Systems under Load using Selective Packet Discarding," in *Proceedings of the Third European Workshop on System Security*, 2010, pp. 15-21.

[14] I. Sourdis, D. N. Pnevmatikatos, and S. Vassiliadis, "Scalable Multigigabit Pattern Matching for Packet Inspection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 156-166, 2008.

[15] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion Detection System: A Comprehensive Review," *Journal of Network and Computer Applications*, vol. 36, pp. 16-24, 2013.

[16] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards Real-time Intrusion Detection for NetFlow and IPFIX," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 227-234.

[17] Y. Abuadlla, G. Kvascev, S. Gajin, and Z. Jovanovic, "Flow-based Anomaly Intrusion Detection System using Two Neural Network Stages," *Comput. Sci. Inf. Syst.*, vol. 11, pp. 601-622, 2014.
[18] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, "Building a Scalable System for Stealthy P2P-Botnet Detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, pp. 27-38, 2014.
[19] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, pp. 343-356, 2010.
[20] T. Hyslip and J. Pittman, "A Survey of Botnet Detection Techniques by Command and Control Infrastructure," *Journal of Digital Forensics, Security and Law*, vol. 10, pp. 7-26, 2015.
[21] L. Sheng, L. Zhiming, H. Jin, D. Gaoming, and H. Wen, "A Distributed Botnet Detecting Approach Based on Traffic Flow Analysis," in *Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC)*, 2012, pp. 124-128.
[22] T. Limmer and F. Dressler, "Flow-based Front Payload Aggregation," in *IEEE LCN*, 2009, pp. 1102-1109.
[23] F. Hensel, "Flow-based and Packet level-based Intrusion Detection as Complementary Concepts," High Diploma Thesis, Department of Informatics, University of Zurich, Zurich, Switzerland, 2008.
[24] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A Taxonomy of Botnet Behavior, Detection, and Defense," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 898-924, 2014.
[25] S. Soltani, S. A. H. Seno, M. Nezhadkamali, and R. Budiarto, "A Survey on Real World Botnets and Detection Mechanisms," *International Journal of Information and Network Security*, vol. 3, p. 116, 2014.
[26] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han, "Botnet Research Survey," in *32nd Annual IEEE International Computer Software and Applications Conference*, 2008, pp. 967-972.
[27] S. Abt and H. Baier, "Towards Efficient and Privacy-Preserving Network-Based Botnet Detection Using Netflow Data," in *Internation Network Conference*, 2012, pp. 37-50.
[28] V. M. Igure and R. D. Williams, "Taxonomies of Attacks and Vulnerabilities in Computer Systems," *IEEE Communications Surveys & Tutorials*, vol. 10, pp. 6-19, 2008.
[29] Symantec Corp, "Internet Security Threat Report," 2016, [Online; accessed 4-Feb-2016]. [Online]. Available: https://www.symantec.com/security-center/threat-report
[30] S. X. Wu and W. Banzhaf, "The Use of Computational Intelligence in Intrusion Detection Systems: A Review," *Applied Soft Computing*, vol. 10, pp. 1-35, 2010.
[31] Snort IDS, "Snort," 2012, [Online; accessed 8-May-2013]. [Online]. Available: www.snort.org
[32] J. GERBER,"Suricata: A Next Generation IDS/IPS Engine," 2010, [Online; accessed 4-May-2014]. [Online]. Available: https://suricata-ids.org/

[33] P. Mehra, "A Brief Study and Comparison of Snort and Bro Open Source Network Intrusion Detection Systems," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 1, pp. 383-386, 2012.
[34] J. Beale, A. R. Baker, and J. Esler, *Snort: IDS and IPS toolkit*: Syngress, 2007.
[35] Bro, "Bro IDS," 2012, [Online; accessed 5-June-2013]. [Online]. Available: www.bro.org
[36] B. Morin and L. Mé, "Intrusion Detection and Virology: an Analysis of Differences, Similarities and Complementariness," *Journal in Computer Virology,* vol. 3, pp. 39-49, 2007.
[37] R. R. Singh, N. Gupta, and S. Kumar, "To Reduce the False Alarm in Intrusion Detection System Using Self Organizing Map," *International Journal of Soft Computing and Engineering (IJSCE),* vol. 1, pp. 27-32, 2011.
[38] K. Wang, & Stolfo, S. J. , "Anomalous payload-based Network Intrusion Detection," *Recent Advances in Intrusion Detection,* p. 19, 2004.
[39] M. Mahoney and P. Chan, "Learning Non-stationary Models of Normal Network Traffic for Detecting Novel Attacks," in *8th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, 2002, pp. 376–385.
[40] M.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong, "A Flow-based Method for Abnormal Network Traffic Detection," in *Network Operations and Management Symposium*, 2004, pp. 599-612.
[41] S. M. Hussein, F. H. M. Ali, and Z. Kasiran, "Evaluation Effectiveness of Hybrid IDS using Snort with Naïve Bayes to Detect Attacks," in *Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, 2012, pp. 256-260.
[42] Z. M. Fadlullah, T. Taleb, A. V. Vasilakos, M. Guizani, and N. Kato, "DTRAB: Combating Against Attacks on Encrypted Protocols through Traffic-feature Analysis," *IEEE/ACM Transactions on Networking (TON),* vol. 18, pp. 1234-1247, 2010.
[43] K.-K. Tseng, J. Lo, Y. Liu, S.-H. Chang, M. Merabti, F. Ng, CK, *et al.*, "A Feasibility Study of Stateful Automaton Packet Inspection for Streaming Application Detection Systems," *Enterprise Information Systems*, pp. 1-20, 2016.
[44] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Predicting the Resource Consumption of Network Intrusion Detection Systems," in *International Workshop on Recent Advances in Intrusion Detection*, 2008, pp. 135-154.
[45] F. Fusco and L. Deri, "High Speed Network Traffic Analysis with Commodity Multi-core Systems," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet measurement*, 2010, pp. 218-224.
[46] J. Morgan, "Streaming Network Traffic Analysis Using Active Learning," Master thesis, Department of Computer Science, Dalhousie University, Halifax, Nova Scotia, 2015.
[47] M. Pihelgas, "A Comparative Analysis of Open-Source Intrusion Detection Systems," Master thesis, Departement of Computer Science, Tallinn University of Technology, Tallinn, 2012.

[48] J. Korenek and P. Kobiersky, "Intrusion Detection System Intended for Multigigabit Networks," in *2007 IEEE Design and Diagnostics of Electronic Circuits and Systems*, 2007, pp. 1-4.

[49] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle, "Comparing and Improving Current Packet Capturing Solutions based on Commodity Hardware," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 206-217.

[50] P. Lambruschini, M. Raggio, R. Bajpai, and A. Sharma, "Efficient Implementation of Packet Pre-filtering for Scalable Analysis of IP Traffic on High-speed Lines," in *20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2012, pp. 1-5.

[51] D. Ficara, G. Antichi, A. Di Pietro, S. Giordano, G. Procissi, and F. Vitucci, "Sampling Techniques to Accelerate Pattern Matching in Network Intrusion Detection Systems," in *IEEE International Conference on Communications (ICC)*, 2010, pp. 1-5.

[52] G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, "A Static Packer-agnostic Filter to Detect Similar Malware Samples," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2012, pp. 102-122.

[53] Q. Zhao, J. Xu, and A. Kumar, "Detection of Super Sources and Destinations in High-speed Networks: Algorithms, Analysis and Evaluation," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 1840-1852, 2006.

[54] F. Haddadi, J. Morgan, E. Gomes Filho, and A. N. Zincir-Heywood, "Botnet Behaviour Analysis using IP Flows: with HTTP Filters using Classifiers," in *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*, 2014, pp. 7-12.

[55] C.-H. Lin and S.-C. Chang, "Efficient Pattern Matching Algorithm for Memory Architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 33-41, 2011.

[56] N. Weaver, V. Paxson, and J. M. Gonzalez, "The Shunt: an FPGA-based Accelerator for Network Intrusion Prevention," in *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, 2007, pp. 199-206.

[57] G. Munz and G. Carle, "Real-time Analysis of Flow Data for Network Attack Detection," in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 100-108.

[58] A. Karim, R. B. Salleh, M. Shiraz, S. A. A. Shah, I. Awan, and N. B. Anuar, "Botnet Detection Techniques: Review, Future Trends, and Issues," *Journal of Zhejiang University SCIENCE*, vol. 15, pp. 943-983, 2014.

[59] P. Porras, H. Saidi, and V. Yegneswaran, "A Multi-perspective Analysis of the Storm (Peacomm) Worm," Computer Science Laboratory, Tech. Rep., 2007

[60] G. Sinclair, C. Nunnery, and B. B. Kang, "The Waledac Protocol: The How and Why," in *4th International Conference on Malicious and Unwanted Software (MALWARE)*, 2009, pp. 69-77.

[61] D. Andriesse and H. Bos, "An Analysis of the Zeus Peer-to-Peer Protocol," 2013. [Online]. Available: http://www.few.vu.nl/~dae400/papers/zeus-tech-report-2013.pdf

[62] W. Zilong, W. Jinsong, H. Wenyi, and X. Chengyi, "The Detection of IRC Botnet based on Abnormal Behavior," in *2010 Second International Conference on Multimedia and Information Technology*, 2010.
[63] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection," in *USENIX Security Symposium*, 2008, pp. 139-154.
[64] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic," 2008.
[65] J. Goebel and T. Holz, "Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation," *HotBots*, vol. 7, pp. 8-8, 2007.
[66] T.-F. Yen and M. K. Reiter, "Traffic Aggregation for Malware Detection," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2008, pp. 207-227.
[67] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "Bothunter: Detecting Malware Infection through IDS-Driven Dialog Correlation," in *Usenix Security*, 2007, pp. 1-16.
[68] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, "Automatically Generating Models for Botnet Detection," in *European Symposium on Research in Computer Security*, 2009, pp. 232-249.
[69] D. H. Kim, T. Lee, J. Kang, H. Jeong, and H. P. In, "Adaptive Pattern Mining Model for Early Detection of Botnet Propagation Scale," *Security and Communication Networks*, vol. 5, pp. 917-927, 2012.
[70] S. García, A. Zunino, and M. Campo, "Botnet Behavior Detection using Network Synchronism," *Privacy, Intrusion Detection and Response: Technologies for Protecting Networks*, pp. 122-144, 2011.
[71] G. Jian, K. Zheng, Y. Yang, and X. Niu, "An Evaluation Model of Botnet based on Peer to Peer," in *Fourth International Conference on Computational Intelligence and Communication Networks (CICN)*, 2012, pp. 925-929.
[72] L. Dan, L. Yichao, H. Yue, and L. Zongwen, "A P2P-Botnet Detection Model and Algorithms based on Network Streams Analysis," in *International Conference on Future Information Technology and Management Engineering (FITME)*, 2010, pp. 55-58.
[73] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, *et al.*, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 2037-2064, 2014.
[74] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101, 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5101.txt
[75] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a Better NetFlow," *ACM SIGCOMM Computer Communication Review*, vol. 34, p. 245, 2004.
[76] U. Banerjee, A. Vashishtha, and M. Saxena, "Evaluation of the Capabilities of WireShark as a Tool for Intrusion Detection," *International Journal of Computer Applications*, vol. 6, 2010.
[77] L. MartinGarcia,"TcpDump and Libpcap," 2012, [Online; accessed 9-July-2012]. [Online]. Available: http://www.tcpdump.org

[78] V. Kumaran, "Event Stream Database based Architecture to Detect Network Intrusion," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, 2013, pp. 241-248.
[79] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta, "Analysis of the Impact of Sampling on NetFlow Traffic Classification," *Computer Networks*, vol. 55, pp. 1083-1099, 2011.
[80] N. Duffield, "Sampling for Passive Internet Measurement: A Review," *Statistical Science*, pp. 472-498, 2004.
[81] T. Zseby, T. Hirsch, and B. Claise, "Packet Sampling for Flow Accounting: Challenges and Limitations," in *International Conference on Passive and Active Network Measurement*, 2008, pp. 61-71.
[82] D. Brauckhoff, M. May, and B. Plattner, "Flow-level Anomaly Detection-Blessing or Curse," in *IEEE INFOCOM Conference*, 2007.
[83] J. David and C. Thomas, "DDoS Attack Detection using Fast Entropy Approach on Flow-based Network Traffic," *Procedia Computer Science*, vol. 50, pp. 30-36, 2015.
[84] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, and F. Tang, "Discriminating DDoS Attacks from Flash Crowds using Flow Correlation Coefficient," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, pp. 1073-1080, 2012.
[85] S. A. Abdulla, S. Ramadass, A. Altaher, and A. A. Nassiri, "Setting a Worm Attack Warning by Using Machine Learning to Classify Netflow Data," *International Journal of Computer Applications*, vol. 36, pp. 49-56, 2011.
[86] F. Dressler, W. Jaegers, and R. German, "Flow-based Worm Detection using Correlated Honeypot Logs," in *Communication in Distributed Systems Conference*, 2007, pp. 1-6.
[87] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "SSHCure: a Flow-based SSH Intrusion Detection System," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*, 2012, pp. 86-97.
[88] M. Vizváry and J. Vykopal, "Flow-based Detection of RDP Brute-force Attacks," in *Proceedings of 7th International Conference on Security and Protection of Information (SPI 2013)*, 2013.
[89] P. Amini, R. Azmi, and M. Araghizadeh, "Botnet Detection using NetFlow and Clustering," *Advances in Computer Science: an International Journal*, vol. 3, pp. 139-149, 2014.
[90] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, *et al.*, "Botnet Detection based on Traffic Behavior Analysis and Flow Intervals," *Computers & Security*, vol. 39, pp. 2-16, 2013.
[91] J. François, S. Wang, and T. Engel, "BotTrack: Tracking Botnets using NetFlow and PageRank," in *International Conference on Research in Networking*, 2011, pp. 1-14.
[92] M. Stevanovic and J. M. Pedersen, "Machine Learning for Identifying Botnet Network Traffic," *Journal of Aalborg University*, 2013.
[93] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijayalakshmi, P. Yogesh, and A. Kannan, "Intelligent Feature Selection and Classification

Techniques for Intrusion Detection in Networks: A Survey," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, p. 271, 2013.
[94] M. Stevanovic and J. M. Pedersen, "An Efficient Flow-based Botnet Detection using Supervised Machine Learning," in *International Conference on Computing, Networking and Communications (ICNC)*, 2014, pp. 797-801.
[95] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, "Decision Tree Analysis on J48 Algorithm for Data Mining," *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, 2013.
[96] M. N. Anyanwu and S. G. Shiva, "Comparative Analysis of Serial Decision Tree Classification Algorithms," *International Journal of Computer Science and Security*, vol. 3, pp. 230-240, 2009.
[97] A. Liaw and M. Wiener, "Classification and Regression by Random Forest," *R news*, vol. 2, pp. 18-22, 2002.
[98] A. Nogueira, P. Salvador, and F. Blessa, "A Botnet Detection System based on Neural Networks," in *Fifth International Conference on Digital Telecommunications (ICDT)*, 2010, pp. 57-62.
[99] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, *et al.*, "Detecting P2P Botnets through Network Behavior Analysis and Machine Learning," in *Ninth Annual International Conference on Privacy, Security and Trust (PST)*, 2011, pp. 174-180.
[100] S. Ting, W. Ip, and A. H. Tsang, "Is Naive Bayes a Good Classifier for Document Classification," *International Journal of Software Engineering and Its Applications*, vol. 5, pp. 37-46, 2011.
[101] D. Miller,"Softflowd: A Software Netflow Probe," 2012, [Online; accessed 7-June-2013]. [Online]. Available: http://www.mindrot.org/projects/softflowd/
[102] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding Bots in Network Traffic without Deep Packet Inspection," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012, pp. 349-360.
[103] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting Botnet Command and Control Servers through Large-scale Netflow Analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 129-138.
[104] U. Wijesinghe, U. Tupakula, and V. Varadharajan, "An Enhanced Model for Network Flow Based Botnet Detection," in *Proceedings of the 38th Australasian Computer Science Conference (ACSC 2015)*, 2015, p. 30.
[105] G. Schaffrath, & B. Stiller, , "Conceptual Integration of Flow-based and Packet-based Network Intrusion Detection," *Resilient Networks and Services*, pp. 190-194, 2008.
[106] J. Steinberger, L. Schehlmann, S. Abt, and H. Baier, "Anomaly Detection and Mitigation at Internet Scale: A survey," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*, 2013, pp. 49-60.
[107] M. A. Mehmood, A. Feldmann, S. Uhlig, and W. Willinger, "We Are All Treated Equal, Aren't We?—Flow-level Performance as a Function of Flow Size," in *Networking Conference, 2014 IFIP*, 2014, pp. 1-9.

[108] G. F. Guo, "The Study of the Ontology and Context Verification Based Intrusion Detection Model," in *Applied Mechanics and Materials*, 2014, pp. 3338-3341.
[109] U. Shankar and V. Paxson, "Active Mapping: Resisting NIDS Evasion without Altering Traffic," in *Proceedings Symposium on Security and Privacy*, 2003, pp. 44-61.
[110] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive Approach to Intrusion Detection Alert Correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 146-169, 2004.
[111] M. Sourour, B. Adel, and A. Tarek, "Environmental Awareness Intrusion Detection and Prevention System Toward Reducing False Positives and False Negatives," in *IEEE Symposium on Computational Intelligence in Cyber Security*, 2009, pp. 107-114.
[112] G. S. Kumar and C. Sirisha, "Robust Preprocessing and Random Forests Technique for Network Probe Anomaly Detection," *International Journal of Soft Computing and Engineering (IJSCE) ISSN*, pp. 2231-2307, 2012.
[113] D. G. Bhatti and P. Virparia, "Data Preprocessing for Reducing False Positive Rate in Intrusion Detection," *International Journal of Computer Applications*, vol. 57, 2012.
[114] D. G. Bhatti, P. Virparia, and B. Patel, "Conceptual Framework for Soft Computing based Intrusion Detection to Reduce False Positive Rate," *International Journal of Computer Applications*, vol. 44, pp. 1-3, 2012.
[115] G. P. Spathoulas and S. K. Katsikas, "Using a Fuzzy Inference System to Reduce False Positives in Intrusion Detection," in *2009 16th International Conference on Systems, Signals and Image Processing*, 2009, pp. 1-4.
[116] T. Pietraszek and A. Tanner, "Data Mining and Machine Learning—Towards Reducing False Positives in Intrusion Detection," *Information Security Technical Report*, vol. 10, pp. 169-183, 2005.
[117] D. Bolzoni, B. Crispo, and S. Etalle, "ATLANTIDES: An Architecture for Alert Verification in Network Intrusion Detection Systems," in *LISA*, 2007, pp. 1-12.
[118] T. Kaur, "A Hybrid approach using Signature and Anomaly Detection to Detect Network Intrusions," Ph.D. thesis, Thapar Univeristy Patiala, 2013.
[119] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley, "Worm Detection, Early Warning and Response based on Local Victim Information," in *20th Annual Computer Security Applications Conference*, 2004, pp. 136-145.
[120] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous Payload-based Worm Detection and Signature Generation," in *International Workshop on Recent Advances in Intrusion Detection*, 2005, pp. 227-246.
[121] A. D. Todd, R. A. Raines, R. O. Baldwin, B. E. Mullins, and S. K. Rogers, "Alert Verification Evasion through Server Response Forging," in *International Workshop on Recent Advances in Intrusion Detection*, 2007, pp. 256-275.
[122] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, "A Hybrid Intrusion Detection System Design for Computer Network Security," *Computers & Electrical Engineering*, vol. 35, pp. 517-526, 2009.

[123] E. Tombini, H. Debar, L. Mé, and M. Ducassé, "A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffic," in *20th Computer Security Applications Conference* 2004, pp. 428-437.
[124] Y.-X. Ding, M. Xiao, and A.-W. Liu, "Research and Implementation on Snort-based Hybrid Intrusion Detection System," in *2009 International Conference on Machine Learning and Cybernetics*, 2009, pp. 1414-1418.
[125] K. Hwang, M. Cai, Y. Chen, and M. Qin, "Hybrid Intrusion Detection with Weighted Signature Generation over Anomalous Internet Episodes," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, pp. 41-55, 2007.
[126] J. Yang, X. Chen, X. Xiang, and J. Wan, "HIDS-DT: An Effective Hybrid Intrusion Detection System Based on Decision Tree," in *International Conference on Communications and Mobile Computing (CMC)*, 2010, pp. 70-75.
[127] J. Zhang and M. Zulkernine, "A Hybrid Network Intrusion Detection Technique using Random Forests," in *First International Conference on Availability, Reliability and Security (ARES'06)*, 2006, p. 8 pp.
[128] S. M. Hussein, F. H. M. Ali, and Z. Kasiran, "Evaluation effectiveness of Hybrid IDS using Snort with Naïve Bayes to Detect Attacks," in *Second International Conference on Digital Information and Communication Technology and it's Applications*, 2012, pp. 256-260.
[129] D. J. Day, D. A. Flores, and H. S. Lallie, "CONDOR: A Hybrid IDS to Offer Improved Intrusion Detection," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012, pp. 931-936.
[130] V. Jacobson and S. McCanne, "libpcap: Packet Capture Library," *Lawrence Berkeley Laboratory, Berkeley, CA*, 2009.
[131] C. Kreibich and R. Sommer, "Policy-controlled Event Management for Distributed Intrusion Detection," in *25th IEEE International Conference on Distributed Computing Systems Workshops*, 2005, pp. 385-391.
[132] B. Amann, R. Sommer, A. Sharma, and S. Hall, "A Lone Wolf No More: Supporting Network Intrusion Detection with Real-time Intelligence," in *International Workshop on Recent Advances in Intrusion Detection*, 2012, pp. 314-333.
[133] L. Deri,"PF_Ring Packet Capture," 2011, [Online; accessed 4-May-2013]. [Online]. Available: http://www.ntop.org
[134] J. Stebelton,"Berkeley Packet Filters – The Basics," 2014, [Online; accessed 5-May-2013]. [Online]. Available: http://www.infosecwriters.com/text_resources/pdf/JStebelton_BPF.pdf
[135] L. Deri and N. Spa, "nProbe: An Open Source Netflow Probe for Gigabit Networks," in *TERENA Networking Conference*, 2003.
[136] S. Astashonok,"fprobe: a NetFlow Probe," 2007, [Online; accessed 25-October-2013]. [Online]. Available: http://fprobe.sourceforge.net/
[137] P. B. Ruthven, "Contextual Profiling of Homogeneous User Groups for Masquerade Detection," Master Thesis, Department of Computer Science and Media Technology, Gjøvik University, Norway, 2014.

[138] Logging Framework, "Bro 2.4.1 documentation Framework," [Online; accessed 19-Dec-2013]. [Online]. Available: https://www.bro.org/sphinx/frameworks/logging.html#streams
[139] R. G. Sargent, "Verification and Validation of Simulation Models," *Journal of Simulation,* vol. 7, pp. 12-24, 2013.
[140] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, pp. 45-77, 2007.
[141] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An Empirical Comparison of Botnet Detection Methods," *Computers & Security,* vol. 45, pp. 100-123, 2014.
[142] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, *et al.*, "Prudent Practices for Designing Malware Experiments: Status Quo and outlook," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 65-79.
[143] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward Credible Evaluation of Anomaly-based Intrusion Detection Methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, pp. 516-524, 2010.
[144] A. Papadogiannakis, D. Antoniades, M. Polychronakis, and E. P. Markatos, "Improving the performance of passive network monitoring applications using locality buffering," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*, 2007, pp. 151-157.
[145] F. Schneider and J. Wallerich, "Performance evaluation of packet capturing systems for high-speed networks," in *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, 2005, pp. 284-285.
[146] J. Corsini, "Analysis and Evaluation of Network Intrusion Detection Methods to Uncover Data Theft," Napier University, 2009.
[147] A. Turner and M. Bing,"TcpReplay," 2011, [Online; accessed 9-Dec-2012]. [Online]. Available: https://sourceforge.net/projects/tcpreplay/
[148] A. Folkerts, G. Portokalidis, and H. Bos, "Multi-tier Intrusion Detection by Means of Replayable Virtual Machines," Technical Report IR-CS-47, VU University2008
[149] A. Yeow,"Bit-Twist: Libpcap-based Ethernet Packet Generator," 2016, [Online; accessed 19-Jan-2016]. [Online]. Available: http://bittwist.sourceforge.net/
[150] S. Forge,"TOMAHAWK," [Online; accessed 10-December-2016]. [Online]. Available: http://tomahawk.sourceforge.net
[151] S. C. Smith, K. W. Wong, I. Hammell, J. Robert, and C. J. Mateo, "An Experimental Exploration of the Impact of Network-level Packet Loss on Network Intrusion Detection," DTIC Document, 2015
[152] J. W. Haines, R. P. Lippmann, D. J. Fried, M. Zissman, and E. Tran, "1999 DARPA Intrusion Detection Evaluation: Design and Procedures," 2001.
[153] N. Nwanze, S.-i. Kim, and D. H. Summerville, "Payload Modeling for Network Intrusion Detection Systems," in *MILCOM 2009-2009 IEEE Military Communications Conference*, 2009, pp. 1-7.

[154] C. Thomas, V. Sharma, and N. Balakrishnan, "Usefulness of DARPA Dataset for Intrusion Detection System Evaluation," in *SPIE Defense and Security Symposium*, 2008, pp. 69730G-69730G-8.
[155] H. Om and A. Kundu, "A Hybrid System for Reducing the False Alarm Rate of Anomaly Intrusion Detection System," in *1st International Conference on Recent Advances in Information Technology (RAIT)*, 2012, pp. 131-136.
[156] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection," *Computers & Security*, vol. 31, pp. 357-374, 2012.
[157] J. O. Nehinbe, "A Simple Method for Improving Intrusion Detections in Corporate Networks," in *International Conference on Information Security and Digital Forensics*, 2009, pp. 111-122.
[158] S. Tricaud,"French Honeynet Chapter Status Report," 2011, [Online; accessed 20-May-2013]. [Online]. Available: http://www.honeynet.org/chapters/france
[159] G. Szabó, D. Orincsay, S. Malomsoky, and I. Szabó, "On the Validation of Traffic Classification Algorithms," in *International Conference on Passive and Active Network Measurement*, 2008, pp. 72-81.
[160] Lawrence Berkeley National Laboratory, "Enterprise Tracing Project," 2005, [Online; accessed 8-July-2014]. [Online]. Available: http://www.icir.org/enterprise-tracing/
[161] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The Bittorrent P2P File-Sharing System: Measurements and Analysis," in *International Workshop on Peer-to-Peer Systems*, 2005, pp. 205-216.
[162] O. E. Elejla, A. B. Jantan, and A. A. Ahmed, "Three Layers Approach For Network Scanning Detection," *Journal of Theoretical & Applied Information Technology*, vol. 70, 2014.
[163] G. Kumar, "Evaluation metrics for intrusion detection systems-a study," *International Journal of Computer Science and Mobile Applications, II*, vol. 11, 2014.
[164] D. Smallwood and A. Vance, "Intrusion Analysis with Deep Packet Inspection: Increasing Efficiency of Packet Based Investigations," in *Cloud and Service Computing (CSC), 2011 International Conference on*, 2011, pp. 342-347.
[165] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep Packet Inspection As a Service," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 271-282.
[166] M. M. Masud, T. Al-khateeb, L. Khan, B. Thuraisingham, and K. W. Hamlen, "Flow-based Identification of Botnet Traffic by Mining Multiple Log Files," in *First International Conference on Distributed Framework and Applications*, 2008, pp. 200-206.
[167] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD Explorations Newsletter*, vol. 11, pp. 10-18, 2009.
[168] R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and Taxonomy of Botnet Research Through Life-cycle," *ACM Computing Surveys (CSUR)*, vol. 45, p. 45, 2013.

[169] X. Ma, X. Guan, J. Tao, Q. Zheng, Y. Guo, L. Liu, *et al.*, "A Novel IRC Botnet Detection Method Based on Packet Size Sequence," in *IEEE International Conference on Communications (ICC)*, 2010, pp. 1-5.
[170] S. Garg, A. K. Sarje, and S. K. Peddoju, "Improved Detection of P2P Botnets Through Network Behavior Analysis," in *International Conference on Security in Computer Networks and Distributed Systems*, 2014, pp. 334-345.
[171] H. R. Zeidanloo and A. B. A. Manaf, "Botnet Detection by Monitoring Similar Communication Patterns," 2010. [Online]. Available: http://arxiv.org/abs/1004.1232
[172] G. Stringhini, T. Holz, B. Stone-Gross, C. Kruegel, and G. Vigna, "BOTMAGNIFIER: Locating Spambots on the Internet," in *USENIX Security Symposium*, 2011, pp. 1-32.
[173] G. Vliek, "Detecting Spam Machines, A Netflow-data Based Approach," Master thesis, Faculty of Electrical Engineering, University of Twente, 2009.
[174] Y. Li, D. Gruenbacher, and C. Scoglio, "Reward Only Is Not Enough: Evaluating and Improving the Fairness Policy of the P2P File Sharing Network eMule/eDonkey," *Peer-to-Peer Networking and Applications*, vol. 5, pp. 40-57, 2012.
[175] D. Garant and W. Lu, "Mining Botnet Behaviors on the Large-Scale Web Application Community," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, 2013, pp. 185-190.
[176] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards Effective Feature Selection in Machine Learning-based Botnet Detection Approaches," in *IEEE Conference on Communications and Network Security (CNS)*, 2014, pp. 247-255.
[177] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet Detection based on Network Behavior," *Botnet Detection*, pp. 1-24, 2008.
[178] A. I. Madbouly, A. M. Gody, and T. M. Barakat, "Relevant Feature Selection Model Using Data Mining for Intrusion Detection System," *International Journal of Engineering Trends and Technology (IJETT)*, 2014.
[179] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical Real-time Intrusion Detection Using Machine Learning Approaches," *Computer Communications*, vol. 34, pp. 2227-2235, 2011.
[180] P. Narang, J. M. Reddy, and C. Hota, "Feature Selection for Detection of Peer-to-Peer Botnet Traffic," in *Proceedings of the 6th ACM India Computing Convention*, 2013, p. 16.
[181] J. V. Gomes, P. R. Inácio, M. Pereira, M. M. Freire, and P. P. Monteiro, "Detection and Classification of Peer-to-peer Traffic: A survey," *ACM Computing Surveys*, vol. 45, p. 30, 2013.
[182] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki, "Exploiting Temporal Persistence to Detect Covert Botnet Channels," in *International Workshop on Recent Advances in Intrusion Detection*, 2009, pp. 326-345.
[183] A. Sperotto, G. Vliek, R. Sadre, and A. Pras, "Detecting Spam at the Network Level," in *Meeting of the European Network of Universities and Companies in Information and Communication Engineering*, 2009, pp. 208-216.

[184] H. Weststrate, "Botnet Detection using Netflow Information," in *10th Twente Student Conference on IT, 23rd January*, 2009.
[185] Y. Liu, "Data Streaming Algorithms for Rapid Cyber Attack Detection," Ph.D. thesis, Department of Computer Engineering, Iowa State University, Ames, Iowa, 2013.
[186] H. Ma, S. Tan, and Z. He, "The Research of P2P Recognition Technology," in *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*, 2014, pp. 601-604.
[187] R. Keralapura, A. Nucci, and C.-N. Chuah, "A Novel Self-learning Architecture for P2P Traffic Classification in High Speed Ntworks," *Computer Networks*, vol. 54, pp. 1055-1068, 2010.
[188] M. Agnihotri,"DeepEnd Research: Library of Malware Traffic Patterns," 2013, [Online; accessed 9-May-2014]. [Online]. Available: http://www.deependresearch.org/2013/04/library-of-malware-traffic-patterns.html
[189] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the Storm and Nugache Trojans: P2P is here," *USENIX Login*, vol. 32, pp. 18-27, 2007.
[190] Bro IDS, "Signature framework — Bro 2.4.1 documentation," 2012, [Online; accessed 6-Nov-2013]. [Online]. Available: https://www.bro.org/sphinx/frameworks/signatures.html
[191] J. Amann, S. Hall, and R. Sommer, "Count Me In: Viable Distributed Summary Statistics for Securing High-Speed Networks," in *International Workshop on Recent Advances in Intrusion Detection*, 2014, pp. 320-340.
[192] M. Jonkman,"Emerging Bro Threats," 2008, [Online; accessed 30-June-2012]. [Online]. Available: http://doc.emergingthreats.net/bin/view/Main/EmergingBro
[193] M. Jonkman,"Storm Worm Emerging Threats," 2007, [Online; accessed 4-April-2013]. [Online]. Available: http://doc.emergingthreats.net/bin/view/Main/StormWorm
[194] M. Tavallaee, "An Adaptive Hybrid Intrusion Detection System," Ph.D. thesis, University of New Brunswick, 2011.
[195] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, "Enriching Network Security Analysis with Time Travel," in *ACM SIGCOMM Computer Communication Review*, 2008, pp. 183-194.
[196] B. AsSadhan, J. M. Moura, D. Lapsley, C. Jones, and W. T. Strayer, "Detecting Botnets using Command and Control Traffic," in *Eighth IEEE International Symposium on Network Computing and Applications* 2009, pp. 156-162.
[197] PacketFilter, "Packet Filter in Bro," 2013, [Online; accessed 20-June-2013]. [Online]. Available: https://www.bro.org/sphinx/scripts/base/frameworks/packet-filter/main.bro.html
[198] G. Carle, F. Dressler, R. A. Kemmerer, H. Koenig, C. Kruegel, and P. Laskov, "Network attack detection and defense–Manifesto of the Dagstuhl Perspective Workshop, March 2nd–6th, 2008," *Computer Science-Research and Development*, vol. 23, pp. 15-25, 2009.

[199] G. Münz, N. Weber, and G. Carle, "Signature Detection in Sampled Packets," in *Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2007), Toulouse, France*, 2007.
[200] R. Sommer,"Bro Cluster Architecture — Bro 2.4.1 Documentation," 2013, [Online; accessed 24-Jan-2015]. [Online]. Available: https://www.bro.org/sphinx/cluster/index.html
[201] E. Alparslan, A. Karahoca, and D. Karahoca, "BotNet Detection: Enhancing Analysis by Using Data Mining Techniques," *INTECH Open Access Publisher*, 2012.
[202] Bro IDS, "Policy Stats," 2008, [Online; accessed 7-Dec-2013]. [Online]. Available: https://www.bro.org/sphinx/scripts/policy/misc/stats.bro.html
[203] R. Love, "Kernel Korner: CPU Affinity," *Linux Journal*, vol. 2003, p. 8, 2003.
[204] Open BL, "Abuse Reporting and Blacklisting," 2014, [Online; accessed 4-July-2014]. [Online]. Available: https://www.openbl.org
[205] Black List, "URL Blacklist," 2013, [Online; accessed 2-May-2015]. [Online]. Available: http://urlblacklist.com/
[206] S. Hansman and R. Hunt, "A Taxonomy of Network and Computer Attacks," *Computers & Security*, vol. 24, pp. 31-43, 2005.
[207] K. Labib, "Computer Security and Intrusion Detection," *Crossroads*, vol. 11, pp. 2-2, 2004.
[208] Y. Gao, Z. Li, and Y. Chen, "A DoS Resilient Flow-level Intrusion Detection Approach for High-speed Networks," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006, pp. 39-39.
[209] T. Diibendorfer and B. Plattner, "Host Behaviour based Early Detection of Worm Outbreaks in Internet Backbones," in *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, 2005, pp. 166-171.
[210] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras, "Hidden Markov Model modeling of SSH Brute-force Attacks," in *International Workshop on Distributed Systems: Operations and Management*, 2009, pp. 164-176.

Appendix A
Attack Classification

In the literature, many attack classifications and taxonomies have been presented and surveyed. However, not all the taxonomies that outlined in the literature provide the same classification. Some studies classify the attack based on their goals, results, and tools [28] and others classify the attacks based on the network type [206]. The highest priority attacks are those who have a critical impact on the computer system. In this appendix, the following major types of attacks are described.

A.1 Denial of Service (DoS)

The main objective of DoS attacks is to deny a legitimate user from using or accessing his/her system in a normal mode. It often disturbs the service of a computer, a server, or a network. Thus it is impossible to use its resources. This kind of attack is frequent on the Internet. There are three types of Dos attacks: host based, network based, and distributed based.

Host-based DoS attacks: This attack targets a vulnerability in the operating system, application software, CPU, and memory. The main aim of this attack is to crash the host. It also works by exploiting the implementation of network protocols

Network based DoS attacks: Network resources are targeted in this attack by flooding the network with packets to disrupt legitimate use. In this case, the bandwidth is overwhelmed with packets so that there is no left bandwidth for the legitimate users.

TCP floods, ICMP floods, and UDP floods are the most network based DoS attack identified that stream their packets to the target.

Distributed based DoS or DDoS attacks: This attack use a large number of attacked computers to direct coordinated DoS attack against target or targets.

A.2 Information Gathering and Scanning

These attacks try to gather information about the system for further attacks. No actual attack is launched on the computer and the network; they are, however, sniffed, scanned, and probed. A packet sniffer is a simple tool to gather information about computer and network by listening to every packet at a particular point in a network. In conventional packet sniffer, the attacker set the Ethernet card into promiscuous mode so that the card accepts and read all traffic packets in the network, even when a packet is not addressed to this network card. MAC address, IP addresses, and running services for a particular host can be obtained using sniffer tools.

A.3 Malicious Software

Malware includes Worms, Virus and Trojan horse, are malicious programs that are inserted into a host to corrupt a system, deny access to a service. The worm runs random code on the victim's host and installs copies of itself in the memory, which infects other hosts on the network. It leads to network congestion, delay, and loss packets. A virus is a program that is attached to another program to run a particular harmful function on the victim's computer. The virus needs the user interaction to run it and propagate to other files or hosts. However, the spread of worms is extremely

faster than the virus. A Trajan horse is a program looks like a useful application, but in fact, performs unwanted actions such as controlling the victim's host remotely using backdoor installation.

A.4 IP Spoofing

This kind of attack is functioning on networks and TPC/IP protocols. Network spoofing is used when the attacker pretends himself as a legitimate user by spoofing who they are. Session Hijacking is the most popular attack in this kind of attack. The attacker usually takes over a session between two hosts and then cuts one of these hosts to be replaced by him. Session Hijacking usually operates at TCP layer and is used to take over sessions of services such as FTP and Telnet. TCP session hijacking also takes advantage of using IP spoofing and TCP sequence number. To make this attack easy to the attacker, the attacker has to guess the TCP sequence number of the session that is attempted to be hijacked by capturing and analysing the packets travelling between the two victims. After the attackers manage to get the sequence number, they spoof their IP address to be matched with one of the victim hosts and then send a TCP packet to the other host with the hijacked sequence number. When the other host accepts the packet and verifies the sequence number which is correct, this host starts to reply to the attacker and continue the hijacked session.

Other types of attacks may include:

- **Physical attacks:** The aim of this attack is to damage the computer hardware and network devices**.**

- **Buffer overflows:** This attack overflow the process's buffer of the victim's system to damage the process.

- **Password attacks:** This attack involves when the attacker is attempting to guess a password of a protected host. Password dictionary and brute force are the main example of this attack.

- **Botnet attack:** This attack was discussed in Chapter 4.

The following steps explain the nature and the methodology of the computer attacks [207]:

1. Reconnaissance: This step involves the process when the attacker collects information about its victim, including the network infrastructure, before launching its attack.

2. Scanning: In this stage, the attacker starts to look for vulnerabilities and holes by scanning the victim's system. Towards the end, the attacker can obtain precious information such as network topology, IP addresses of live hosts, open port numbers, and security devices rules.

3. Getting Access: This step takes place when the attacker attempt to gain access either using the operating system and application attacks if the attacker is a legitimate user, or using the network if the attacker is an outsider.

4. Retaining Access: After the attacker gained access to the compromised host, he/she has to maintain this access. Trajan horse and Backdoors are the famous techniques to perform this step.

5. Hiding Imprint: When the attackers have achieved what they want, they should not leave any track on the system. Backdoor and RootKit are among techniques that help the attacker to modify system logs and build hidden channel for data transmission.

Appendix B
NIDS Requirements

There are many requirements for efficient NIDS mentioned in the literature [6]. The main two requirements that attracted researchers currently are scalability and detection accuracy.

- **Scalability:** NIDS should operate in large volume networks without resource consumption. This happens when all potential packets and traffic are analysed without packet loss. Thus, detection analysis should be performed smoothly in a large data network as well as with increase traffic and network's size. Also, the data amount to be processed by detection methods should be as small as possible. Note that the term "potential packet" is used instead of "incoming packet", this is because potential packets are extracted after sampling processes as will be discussed later.

- **Detecting accuracy or detection rate**: beside all potential packets should be processed correctly; detection methods have to make the right decision, not to decide falsely. To achieve this requirement, the true-positive rate should be high while fewer false positive and negative rate.

Other requirements of NIDS may include:

- **Detecting unknown attacks:** novel intrusion should be detected

- **Detecting encrypted traffic:** encrypted payloads should be readable and analysed for intrusion detection.

- **Early detection:** intrusion should be detected as soon as possible

- **Large data storage:** all potential signatures, profiles, alerts, and reports should be stored for long-term and further usage.

- **NIDS security:** NIDS should be secured enough against attackers who direct attacks into the NIDS itself.

- **Events correlation:** For distributed attacks, NIDS should correlate single attack event with other resources such as firewall, routers or other NIDS for detection.

- **IPv6 compatibility:** NIDS should support IPv4 and IPv6

- **Success attacks identification:** NIDS should differentiate between successful and unsuccessful attack so that the operator should take a proper action against them.

- **Privacy:** NIDS should not violate privacy regulation of users by inspecting private information both in payload and header of the packets.

- **Attack classification:** After detection, NIDS should also identify and classify attacks. Each attack has to be labelled and be under a category for further analysis and measurements.

Appendix C
Attacks Detectable by Flow-based Approach

This appendix presents the attacks that are detectable by flow-based NIDS and how the current research community handles its limitation.

DoS Attack

Gao, et al. [208] proposed and implemented a DoS resilient High-speed Flow-level Intrusion Detection system, HiFIND. The authors developed a prototype that accepts flows exported from a Netflow router in real time. Their approach handles the problem of DoS using flow aggregation accounted in data stream called a sketch. A sketch is a hash table in one-dimension appropriated for quick storage of information. Sketch counts incidences of an event and studies how the traffic behaves over a period of time using statistics. It stores values that help an anomaly-based engine to trigger alarms based on a statistical forecast. So an abnormal deviation from this forecast values is detected as an intrusion. SYN flooding attack is one of DoS attacks that can be used by sketch to detect this type of attack with the following steps:

- The sketch stores and calculates the difference between the number of SYN packets and the number of SYN/ACK packets of each flow.

- If this difference is not within the normal range, a DoS SYN flooding attack is detected.

This approach can be implemented with relying on packet headers only instead of flows but, however; data reduction which is provided by flows cannot be achieved.

Zhao, et al. [53] proposed and designed data streaming algorithms that can detect super sources and super destinations attacks. Super source happens when a source or a host has a unusual number of outgoing connection (fan-out) within specified period. An example of the super source is port scanning that searches for vulnerable services among different hosts. Super destination is considered when a destination or a host receive abnormal number of incoming connection attempts within a small time interval (fan-in). Distributed Denial of Service (DDoS) attack is an example of super destination when a large number of hosts flood flows to a single destination. Data streaming algorithms used in their work is to identify flows that have an unusual number of connection after filtering part of the traffic. Unlike [208], the algorithms of used in [53] is based on two dimension hash tables. To reduce the amount of data to be processed, they perform flow sampling algorithm, hence improving the speed of the process. Since not all the flows are processed, data reduction may compromise the accuracy. The authors solve this problem by combining the power of data streaming and sampling.

Kim, et al. [40] presented a detecting method for detecting abnormal network traffic by analysing the traffic based on flows only. They use the term “traffic pattern” to express different types of DoS attacks. A traffic pattern is a signature that describes the number of flows, number of packets per flow, the size of flow, the size of packets, and the total bandwidth occupied during the session. The authors use these patterns to

differentiate between instances when detecting scanning or flooding attacks. For example, during scanning or SYN flooding attack, since the attacker makes many connection attempts, this pattern can be detected because of:

- a large number of flows generated since the attacker sends many packets to the victim,
- a small number of packets per flow,
- moreover, the small size of the packet as the attacker sends small SYN packets.

The authors also managed to detect ICMP and UDP flooding attack. These attacks have dynamic traffic patterns since it depends on the number of packets and hosts used in these attacks. However, these attacks can be detected since they create large bandwidth consumption and a high number of packets. Their approach can detect traffic of different attacks with a similar traffic pattern by identifying their metrics and then formalizing them into one detection function. However, certain attacks cannot be observed using their method since Kim, et al. focused on detecting DoS and DDoS attacks only. Since they used static threshold values of their parameters in the detection function, their method cannot be suitable for every network condition. So, the adaptive threshold for various network environments is required.

Munz and Carle [57] proposed a general system for DoS flow-based detection named "TOPAS" (Traffic flOw Packet Analysis System). This system operates as a flow collector from multiple sources. It receives data to be analysed in real-time. The

authors develop TOPAS so that it supports different kinds of DoS detection modules and it is publicly available. These modules are including SYN flood detection, Web Server overloading module using HTTP request, and traceback module that identifies the entry points of attack packet with spoofed source IP address. These modules can be adjusted by the network administrator to increase the detection opportunities and accuracy. An example of this is adjusting the number of SYN and SYN/ACK packets in case of SYN flooding detection module. Although the authors state that TOPAS can also analyse packet-base data, their approach does not support the combination of packet-based and flow-based to reduce the false alarms.

Worms

Worm mechanism such as Code Red usually has two stages: victim discovery and transfer code. In discovery stage, the worm surveys the network to find vulnerable holes in the systems while in transfer stage, the worm starts to spread the code to the systems. Unfortunately, the second stage cannot be detected using the flow-based system since the code is injected in the payload which is not analysed by the flow-based. Thus only the first stage of worm behaviour can be analysed and detected using flow-based approach. Some attributes on the hosts when worms infect them are used to detect worms attack. Such attributes include the number of connections, ratio of outgoing to incoming traffic, and response way. However, some researchers deal with worm detection the same way when dealing with scanning detection since they have some common characteristics. DoS detection methods achieved by [53, 208] can be used to detect the worm.

Diibendorfer and Plattner [209] proposed a near real-time method for outbreak worm detection in high-speed networks using flow-based approach. The method is based on examination the behaviour and the number of incomings and outgoing connection of the host. For detection method, the authors used the host behaviour and characteristics to classify hosts into three classes: traffic class, connector class, and responder class. Only suspicious hosts belong to these classes.

Hosts are classified as traffic class when the amount of traffic sent from the host is more than received. An example of this is the worms send out exploit code or when the worm spread in email attachments. Hosts that initiate an abnormal high number of outgoing connections are classified under connector class. Such class happens when hosts scan others. Responder class involves when a host holds bidirectional connections such as TCP connection. An example of this class is when the host responds to TCP handshake initiation or scan during a worm outbreak. In their approach, overlapping within these classes is possible, meaning that a host can be belonging to more than one class.

Figure C.1 illustrate this overlap. Worm outbreak attack can be detected by tracking the cardinality of each class of an entire network periodically. Thus, any unexpected or sudden changes in the cardinality of one or more classes are detected as worm outbreak. The authors validate their method by tracing archived flow-level of recent Internet emails and by tracing fast spreading worms such as Blaster.

Abdulla, et al. [85] proposed a worm warning system using IP flow and machine learning approach. The authors consider the case that when a host is infected by an

email worm or scanning, an unusual amount of traffic is initiated. This traffic is not relied on DNS. They classify flow-based records using Support Vector Machine (SVM) to extract features that belong to worm attacks. For training SVM, the features are gathered into a set of patterns. The authors propose a structure that consists of three modules: data collecting, data sampling, and classifier.

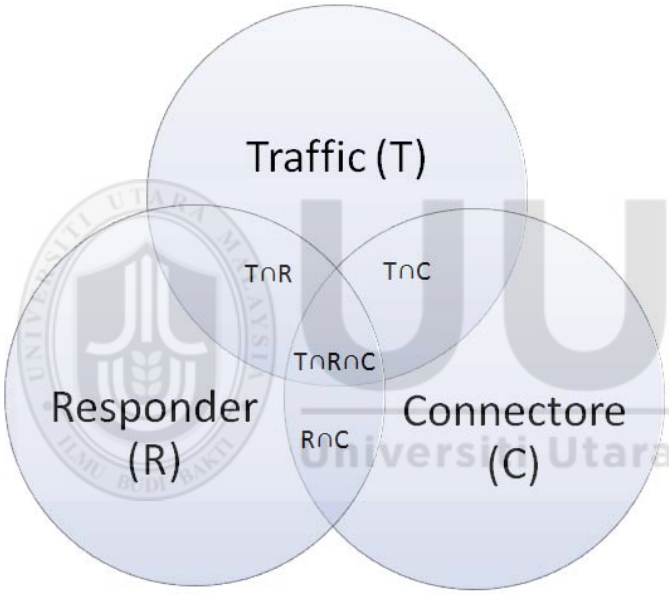

Figure C.1. Classes of Host Behaviour for Worm Detection

The first module collects the raw traffic and extracts the flow record information and stores them into a database. The authors address the problem of dealing with a large amount of flow data by creating the data sampling module. The classifier module classifies the sampled traffic into a worm and benign flow. The SVM was trained by the following scanning worms: CodeRed, Slammer, Doomjuice, and Witty. For email worms, it was trained by sobig, Netsky, Storm, MyDoom, and Conficker.

SSH

Secure SHell (SSH) is a communication protocol that allows a user to have full control over a host's resources remotely. Thus, hosts with SSH-enabled are unfortunately targeted by intrusions. Sperotto, et al. [210] have studied and analysed the flow traffic during SSH. They extract the flow data that is suspected to be malicious traffic. The authors then develop a model which presents the flow characteristics when SSH intrusion takes place. Although their model can detect these attacks, however, the possibility of this model to be in practice is still unknown. Based on their work, Hellemons (2012) develop an algorithm to test the practical applicability of the SSH intrusion model. The algorithm uses the processed flow data to construct attack metadata in the form of properties. Hellemons answered the question: "Can SSH intrusion attacks be detected and analysed in practice by using only flow data?" affirmatively. This method reduces the need for deep packet inspection system, allowing for more scalable NIDS solution.

Appendix D

Main Bro Log Files

D.1 *Connection.log*

Bro generates this log during run time. It consists of the complete connection log of incoming and outgoing traffic. Table D.1 shows the fields of the *connection.log* file.

Table D.1

Fields Description of Connection.log file

Field	Type	Description
ts	time	Timestamp
uid	string	Unique ID of Connection
id.orig_h	addr	Originating endpoint's IP address (AKA ORIG)
id.orig_p	port	Originating endpoint's TCP/UDP port (or ICMP code)
id.resp_h	addr	Responding endpoint's IP address (AKA RESP)
id.resp_p	port	Responding endpoint's TCP/UDP port (or ICMP code)
proto	transport_proto	Transport layer protocol of connection
service	string	Dynamically detected application protocol, if any
duration	interval	Time of last packet seen – time of first packet seen
orig_bytes	count	Originator payload bytes; from sequence numbers if TCP
resp_bytes	count	Responder payload bytes; from sequence numbers if TCP
conn_state	string	Connection state
local_orig	bool	If conn originated locally T; if remotely F. If Site::local_nets empty, always unset.
missed_bytes	count	Number of missing bytes in content gaps
history	string	Connection state history
orig_pkts	count	Number of ORIG packets
orig_ip_bytes	count	Number of ORIG IP bytes
resp_pkts	count	Number of RESP packets
resp_ip_bytes	count	Number of RESP IP bytes (via IP total_length header field)
tunnel_parents	set	If tunneled, connection UID of encapsulating parent (s)
orig_cc	string	ORIG GeoIP Country Code
resp_cc	string	RESP GeoIP Country Code

D.2 *Signatures.log*

This is log is generated when content matching occurs. Bro raises an event with the alert named. This log also contains the payload content which triggers this event. Table D.2 shows each field with its description for this log.

Table D.2

Fields Description of Signatures.log file

Field	Type	Description
ts	time	Timestamp of match
src_addr	addr	Host triggering the signature match event
src_port	port	Host port on which the match occurred
dst_addr	addr	Host which was sent the matching payload
dst_port	port	Port which was sent the matching payload
note	string	Notice associated with the signature event
sig_id	string	Name of the signature that matched
event_msg	string	More descriptive message of the event
sub_msg	string	Extracted payload data or extra message
sig_count	count	Number of sigs
host_count	count	Number of hosts

The following log text is a sample of *Signatuers.log* generated from PH when CTU-52 dataset is used. It shows three infected IRC-bot were detected: `147.32.84.165, 147.32.84.191, and 147.32.84.192`

```
#separator \x09
#set_separator ,
#empty_field   (empty)
#unset_field   -
#path   signatures
#open   2015-08-01-08-13-34
#fields ts      uid     src_addr        src_port        dst_addr        dst_port
        note    sig_id  event_msg       sub_msg sig_count       host_count
#types  time    string  addr    port    addr    port    enum    string  string  string
        count   count
1313675274.978894       CoX6Zn4wnPAUOfTuOk      147.32.84.165   1027    74.125.232.201 80
        Signatures::Sensitive_Signature         ircattack_client        147.32.84.165:
signature match         GET /service/check2?appid=%7B430FD4D0-B729-4F61-AA34-
```

```
91526481799D%7D&appversion=1.3.21.65&applang=&machine=0&version=1.3.21.65&osversion=5
.1...   -       -
1313675281.195719       CxIZuw1HkEATGTlkL6      147.32.84.191   1027    74.125.232.200 80
        Signatures::Sensitive_Signature         ircattack_client        147.32.84.191:
signature match         GET /service/check2?appid=%7B430FD4D0-B729-4F61-AA34-
91526481799D%7D&appversion=1.3.21.65&applang=&machine=0&version=1.3.21.65&osversion=5
.1...   -       -
1313675284.530430       CPfunv1ZCWV1ZnfWBj      147.32.84.192   1027    74.125.232.199 80
        Signatures::Sensitive_Signature         ircattack_client        147.32.84.192:
signature match         GET /service/check2?appid=%7B430FD4D0-B729-4F61-AA34-
91526481799D%7D&appversion=1.3.21.65&applang=&machine=0&version=1.3.21.65&osversion=5
.1...   -       -
#close  2015-08-01-08-13-48
```

D.3 *Notice.log*

Bro also generates this log at runtime. In this log, it contains activities that Bro recognizes as interesting or bad. Table D.3 shows the filed description of this log.

Table D.3

Fields Description of Notice.log file

Field	Type	Description
ts	time	Timestamp
uid	string	Connection unique id
id	record	ID record with orig/resp host/port. See conn.log
fuid	string	File unique identifier
file_mime_type	string	Libmagic sniffed file type
file_desc	string	Additional context for file, if available
proto	transport_proto	Transport protocol
note	string	The type of the notice
msg	string	Human readable message for the notice
sub	string	Sub-message for the notice
src	addr	Source address
dst	addr	Destination address
p	port	Associated port, if any
n	count	Associated count or status code
peer_descr	string	Description for peer that raised this notice
actions	set	Actions applied to this notice
suppress_for	interval	Length of time dupes should be suppressed
dropped	bool	If the src IP was blocked

Appendix E

Resource Consumptions Results

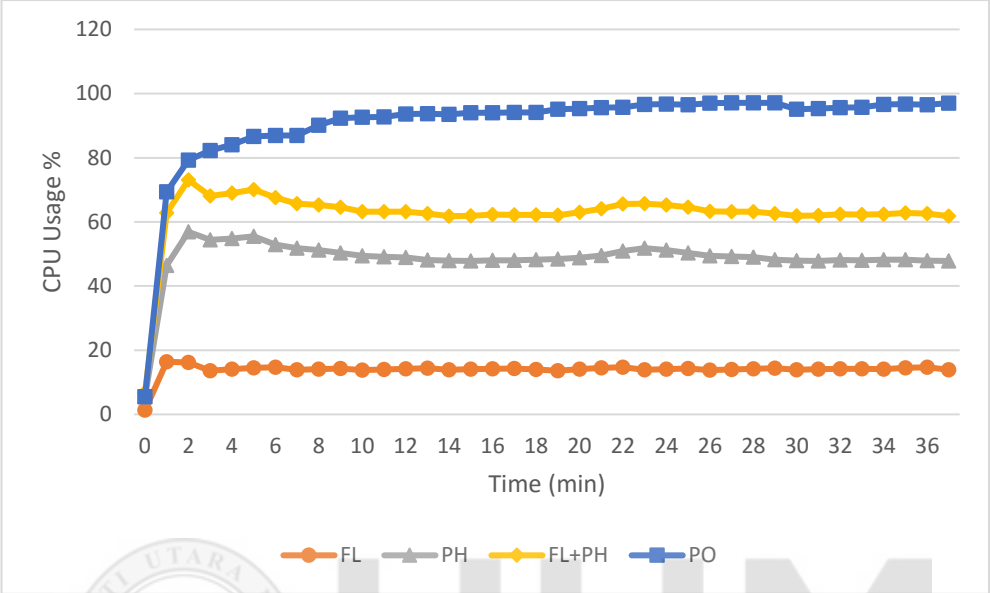

Figure E.1. CPU Usage over Time at 100 Mbps – P2P-bot

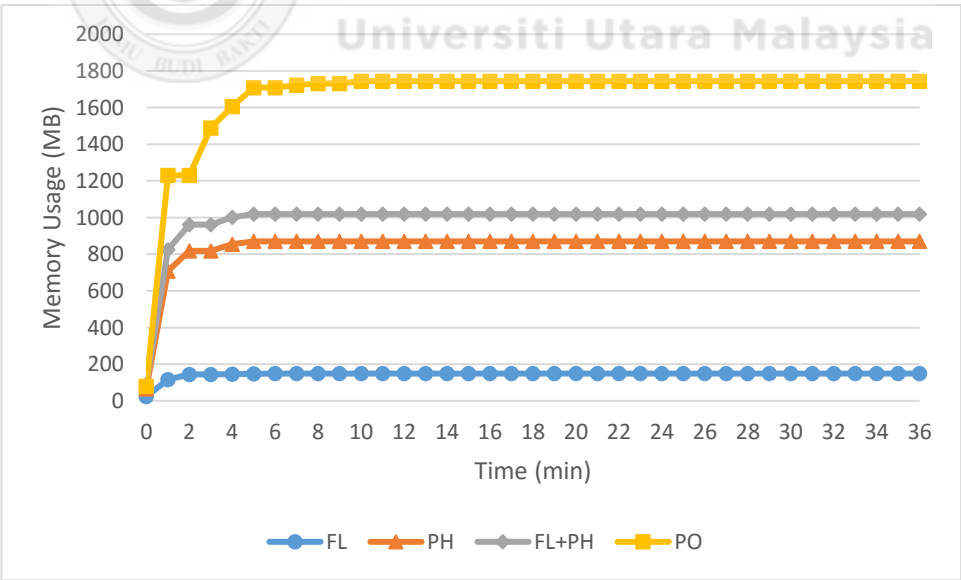

Figure E.2. Memory Usage over Time at 100 Mbps – P2P-bot

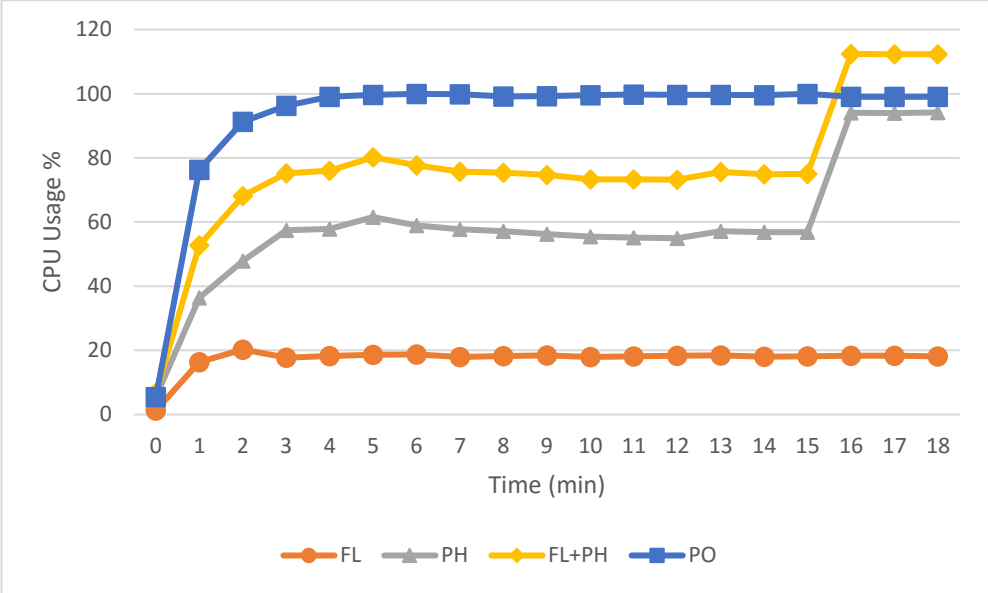

Figure E.3. CPU Usage over Time at 200 Mbps- P2P-bot

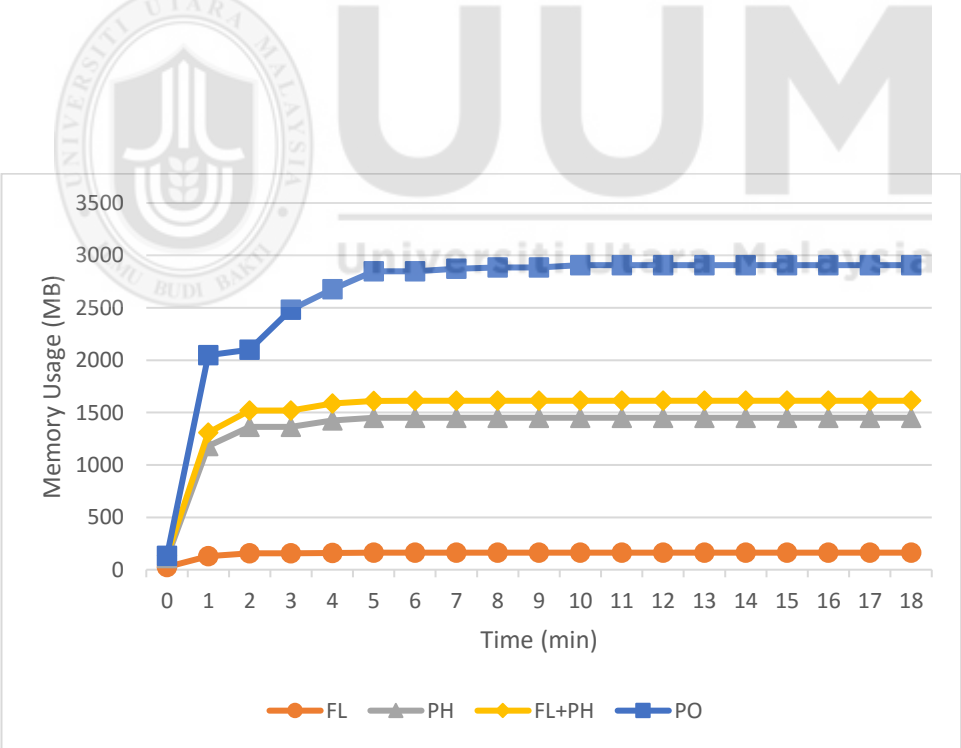

Figure E.4. Memory Usage over Time at 200 Mbps – P2P-bot

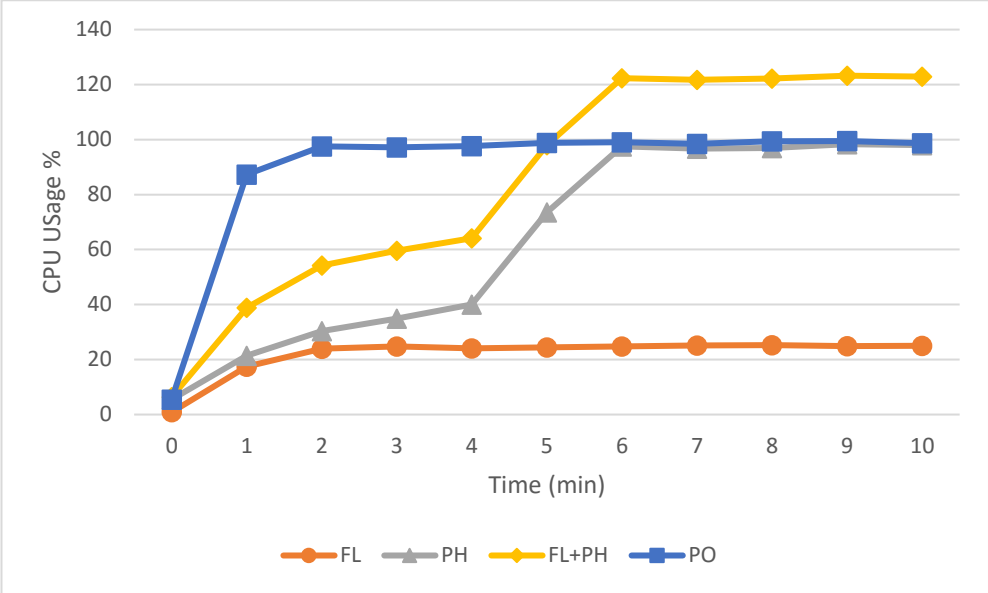

Figure E.5. CPU Usage over Time at 500 Mbps – P2P-bot

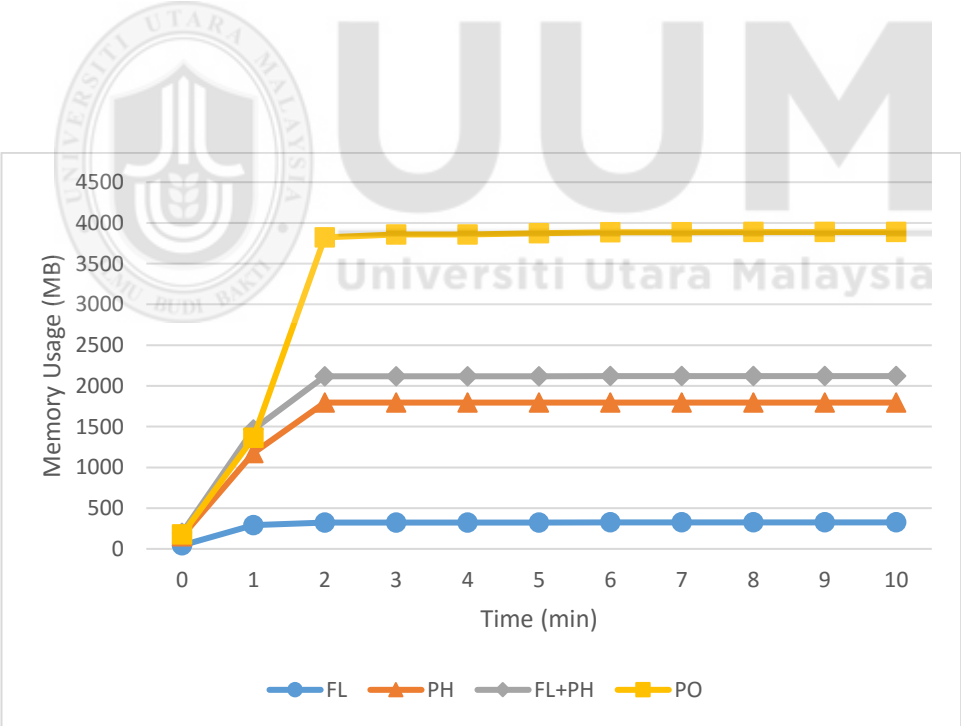

Figure E.6. Memory Usage over Time at 500 Mbps – P2P-bot

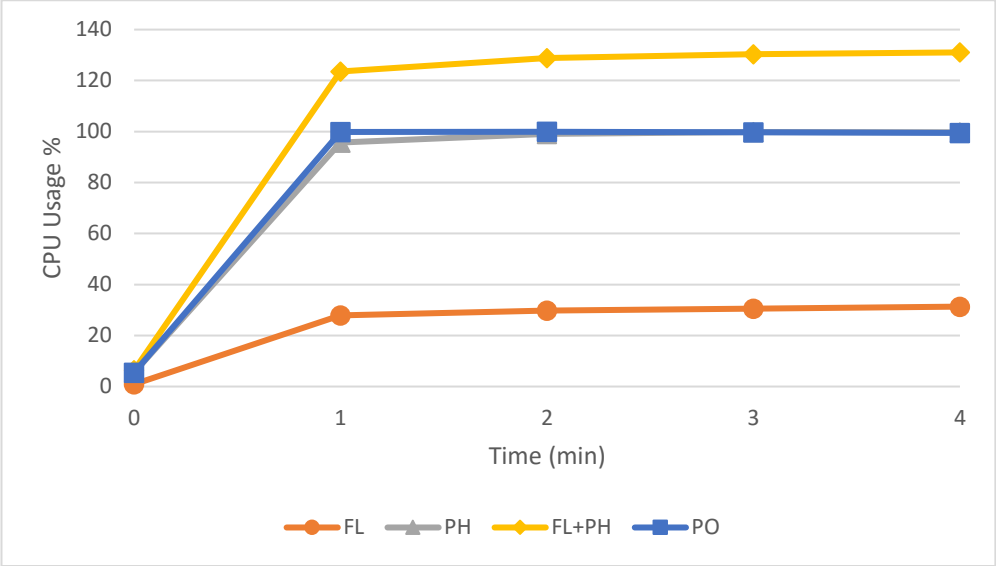

Figure E.7. CPU Usage over Time at 1000 Mbps – P2P-bot

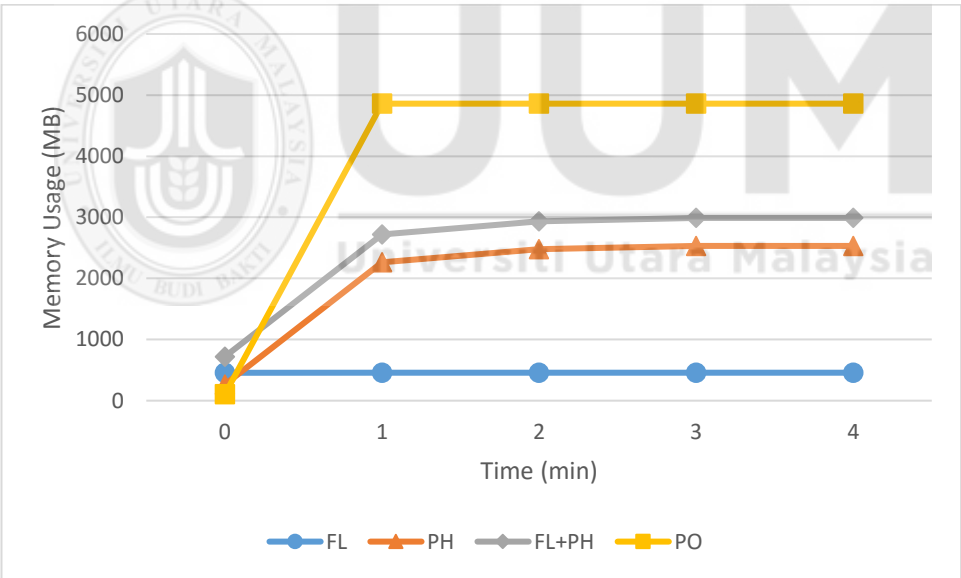

Figure E.8. Memory Usage over Time at 1000 Mbps – P2P-bot

Appendix F

Samples of Detection Code

F.1 Bro SumStats Mechanism Code for Packet-based Spam Identifications

```
@load base/frameworks/sumstats
## Networks that are considered "local":
 const private_address_space: set[subnet] = {
                10.0.0.0/8,
                192.168.0.0/16,
                172.16.0.0/12,
                147.32.84.0/8,
                100.64.0.0/10,
                127.0.0.0/8,
                [fe80::]/10,
                [::1]/128,
 } &redef;
const local_nets: set[subnet] &redef;
global spam_detect = open_log_file("spamhosts") &redef;

event connection_attempt(c: connection)
{
        # Make an observation!
        # This observation is about the host attempting the connection.
        if(c$id$resp_p == 25/tcp) {
                SumStats::observe("SMTP conn",
                        SumStats::Key($host=c$id$orig_h),
                    SumStats::Observation($num=1));
        }
        if(c$id$orig_p == 25/tcp) {
                SumStats::observe("SMTP conn",
                        SumStats::Key($host=c$id$resp_h),
                    SumStats::Observation($num=1));
        }
}

event connection_established(c: connection)
{
# Make an observation!
# Each established connection counts as one so the observation is always 1.
        if(c$id$resp_p == 25/tcp) {
                SumStats::observe("SMTP conn",
                        SumStats::Key($host=c$id$orig_h),
                    SumStats::Observation($num=1));
        }
        if(c$id$orig_p == 25/tcp) {
                SumStats::observe("SMTP conn",
                        SumStats::Key($host=c$id$resp_h),
                    SumStats::Observation($num=1));
        }
}

event bro_done()
{
}

event bro_init()
{
        Log::disable_stream(Conn::LOG);
        # The reducer attaches to the "SMTP conn" observation stream
        # and uses the summing calculation on the observations.
        local r1 = SumStats::Reducer($stream="SMTP conn",
                        $apply=set(SumStats::SUM));
        # Create the final sumstat.
        # $threshold_val.  The actual threshold itself is provided with
        # $threshold.
```

```
        # Another callback is provided for when a key crosses the
        # threshold.
        SumStats::create([$name = " Detecting spam activities",
                          $epoch = 10sec,
                          $reducers = set(r1),
                          # Provide a threshold.
                          $threshold = 10.0,
                          # Provide a callback to calculate a value from
                          # the result
                          # to check against the threshold field.
                          $threshold_val(key: SumStats::Key, result:
SumStats::Result) =
                              {
                              return result["SMTP conn"]$sum;
                              },
                          # Provide a callback for when a key crosses
                          # the threshold.
                         $threshold_crossed(key: SumStats::Key, result:
SumStats::Result) =
                          {
                        if (key$host in private_address_space) {
                              print fmt("%s attempted %.0f or more connections",
                                 key$host, result["SMTP conn"]$sum);
                              print spam_detect, fmt(
                                 "%s attempted %.0f or more connections",
                                 key$host, result["SMTP conn"]$sum);
                        }
        }]);
}
```

F.2 Bro PH Code for IRC-bot Detection

```
@load base/frameworks/notice
@load base/frameworks/signatures/main
@load base/protocols/irc
@load policy/misc/stats
@load-sigs ./ircattack.sig
@load base/frameworks/packet-filter

redef capture_filters = { ["filter_table"] = "" };
global print_logs = open_log_file ("print_log") &redef ;
global filter : string = "";

#To read a file into a Bro table, two record types have to be defined:
# This record contains the types and names of the columns that should constitute the
table keys.
#Our key record only contains the host IP
type Idx: record {
        ip: addr;
};

#This record contains the types and names of the columns that should constitute the
table values.
type Val: record {
        comment: string;
};
# Create an empty table that should contain the suspicious data
global suspicious: table[addr] of Val = table();

event update_filter ()
{
local ns = net_stats();
local filter_counter : count = 0;
local pre_filter : string = "host 100.101.102.103";

# 2) convert suspicious table into filter format string
```

```
for ( ip in suspicious )
{
pre_filter += fmt (" or host %s " , ip ) ;
++ filter_counter;
}
print "pre_filter is";
print pre_filter;

# 3) packet filter framework read the filters
if ( pre_filter != filter )
{
print " Filter has beed altered";
print " Perform Recompiling Filter";
captured_filter [filter_table] = pre_filter ;
}
else
{
print " Filter has not beed altered";
}
filter = copy ( pre_filter ) ;
print print_logs , " number of susp hosts marked ; hosts in filter";
print print_logs , fmt (" %s; %s", |suspicious| , filter_counter);
# to update the capture_filter from suspicious, but not to update the suspicious
itself (since Reread is there)
schedule 10 sec { update_filter () };
flush_all () ;
}

event bro_init() &priority = 5
       {
#1) transfer + update flow suspicious ips into suspicious table
Input::add_table([$source="/home/hashem-bro/b-irc/flowirc/suspicious_file.log",
       $name="suspicious", $idx=Idx, $val=Val, $destination=suspicious,
$mode=Input::REREAD]);
        Input::remove("suspicious");
              schedule 5 sec { update_filter () };
set_buf(detailed_log, F);
       set_buf(bot_log, F);
       }

global checkflag = 0;
global ircbotdetect = open_log_file("ircbot_packet_hosts") &redef;
global p_at_in : count = 0;
global p_es_in : count = 0;

module IrcBot;
export {
       global detailed_log = open_log_file("irc.detailed") &redef;
       global bot_log = open_log_file("irc-bots") &redef;
       global summary_interval = 1 min &redef;
       global detailed_logging = T &redef;
       global content_dir = "irc-bots" &redef;
       global bot_nicks =
              /^\[([^\]]+\|)+[0-9]{2,}]/           # [DEU|XP|L|00]
              | /^\[[^ ]+\]([^ ]+\|)+([0-9a-zA-Z-]+)/      # [0]CHN|3436036
[DEU][1]3G-QE
              | /^DCOM[0-9]+$/                     # DCOM7845
              | /^\{[A-Z]+\}-[0-9]+/               # {XP}-5021040
              | /^\[[0-9]+-[A-Z0-9]+\][a-z]+/              # [0058-X2]wpbnlgwf
              | /^\[[a-zA-Z0-9]\]-[a-zA-Z0-9]+$/   # [SD]-743056826
              | /^[a-z]+[A-Z]+-[0-9]{5,}$/
              | /^[A-Z]{3}-[0-9]{4}/               # ITD-1119
              ;
       global bot_cmds =
              /(^| *)[.?#!][^
]{0,5}(scan|ndcass|download|cvar\.|execute|update|dcom|asc|scanall) /
              | /(^| +\]\[ +)\* (ipscan|wormride)/
              | /(^| *)asn1/
              ;
       global skip_msgs =
              /.*AUTH .*/
              | /.*\*\*\* Your host is .*/
```

```
                | /.*\*\*\* If you are having problems connecting .*/
                ;
        redef enum Notice::Type += {
                IrcBotServerFound,
                IrcBotClientFound,
        };
        type channel: record {
name: string;
passwords: set[string];
topic: string &default="";
topic_history: vector of string;
        };
        type bot_client: record {
host: addr;
p: port;
nick: string &default="";
user: string &default="";
realname: string &default="";
channels: table[string] of channel;
servers: set[addr] &optional;
first_seen: time;
last_seen: time;
        };
        type bot_server: record {
host: addr;
p: set[port];
clients: table[addr] of bot_client;
global_users: string &default="";
passwords: set[string];
channels: table[string] of channel;
first_seen: time;
last_seen: time;
        };
        type bot_conn: record {
client: bot_client;
server: bot_server;
conn: connection;
fd: file;
ircx: bool &default=F;
        };
# We keep three sets of clients/servers:
#  (1) tables containing all IRC clients/servers
#  (2) sets containing potential bot hosts
#  (3) sets containing confirmend bot hosts
#
# Hosts are confirmed when a connection is established between
# potential bot hosts.
#
# FIXME: (1) should really be moved into the general IRC script.
        global expire_server:
                function(t: table[addr] of bot_server, idx: addr): interval;
        global expire_client:
                function(t: table[addr] of bot_client, idx: addr): interval;
        global servers: table[addr] of bot_server &write_expire=24 hrs
                &expire_func=expire_server &persistent;
        global clients: table[addr] of bot_client &write_expire=24 hrs
                &expire_func=expire_client &persistent;
        global potential_bot_clients: set[addr] &persistent;
        global potential_bot_servers: set[addr] &persistent;
        global confirmed_bot_clients: set[addr] &persistent;
        global confirmed_bot_servers: set[addr] &persistent;
# All IRC connections.
        global conns: table[conn_id] of bot_conn &persistent;
# Connections between confirmed hosts.
        global bot_conns: set[conn_id] &persistent;
# Helper functions for readable output.
        global strset_to_str: function(s: set[string]) : string;
        global portset_to_str: function(s: set[port]) : string;
        global addrset_to_str: function(s: set[addr]) : string;
}
function strset_to_str(s: set[string]) : string
{
```

```
        if ( |s| == 0 )
                return "<none>";
        local r = "";
        for ( i in s )
        {
                if ( r != "" )
                        r = cat(r, ",");
                r = cat(r, fmt("\"%s\"", i));
        }
        return r;
}
function portset_to_str(s: set[port]) : string
{
        if ( |s| == 0 )
                return "<none>";
        local r = "";
        for ( i in s )
        {
                if ( r != "" )
                        r = cat(r, ",");
                r = cat(r, fmt("%d", i));
        }
        return r;
}
function addrset_to_str(s: set[addr]) : string
{
        if ( |s| == 0 )
                return "<none>";
        local r = "";
        for ( i in s )
        {
                if ( r != "" )
                        r = cat(r, ",");
                r = cat(r, fmt("%s", i));
        }
        return r;
}
function fmt_time(t: time) : string
{
        return strftime("%y-%m-%d-%H-%M-%S", t);
}
event print_bot_state()
{
        local bot_summary_log = open_log_file("irc-bots.summary");
        disable_print_hook(bot_summary_log);
        print bot_summary_log, "----------------------------";
        print bot_summary_log, strftime("%y-%m-%d-%H-%M-%S", network_time());
        print bot_summary_log, "----------------------------";
        print bot_summary_log;
        print bot_summary_log, "Known servers";
        for ( h in confirmed_bot_servers )
        {
                local s = servers[h];
                print bot_summary_log,
                      fmt("    %s %s - clients: %d ports %s password(s) %s last-seen
%s first-seen %s global-users %s",
                                       "L",
                                      s$host, |s$clients|, portset_to_str(s$p),
                                      strset_to_str(s$passwords),
                                      fmt_time(s$last_seen), fmt_time(s$first_seen),
                                      s$global_users);
                for ( name in s$channels )
                {
                        local ch = s$channels[name];
                        print bot_summary_log,
                              fmt("        channel %s: topic \"%s\", password(s) %s",
                                ch$name, ch$topic,
                                strset_to_str(ch$passwords));
                }
        }
        print bot_summary_log, "\nKnown clients";
        for ( h in confirmed_bot_clients )
```

```
        {
                local c = clients[h];
                print bot_summary_log,
                      fmt("    %s %s - server(s) %s user %s nick %s realname %s last-
seen %s first-seen %s",
                                       "L", h,
                                      addrset_to_str(c$servers),
                                      c$user, c$nick, c$realname,
                                      fmt_time(c$last_seen), fmt_time(c$first_seen));
        }
        close(bot_summary_log);

        if ( summary_interval != 0 secs )
                schedule summary_interval { print_bot_state() };
}
function do_log_force(c: connection, msg: string)
{
        local id = c$id;
        print bot_log, fmt("%.6f %s:%d > %s:%d %s %s",
                       network_time(), id$orig_h, id$orig_p,
                       id$resp_h, id$resp_p, c$addl, msg);
}
function do_log(c: connection, msg: string)
{
        if ( c$id !in bot_conns )
                return;

        do_log_force(c, msg);
}
function log_msg(c: connection, cmd: string, prefix: string, msg: string)
{
        if ( skip_msgs in msg )
                return;
        do_log(c, fmt("MSG command=%s prefix=%s msg=\"%s\"", cmd, prefix, msg));
}
function update_timestamps(c: connection) : bot_conn
{
        local conn = conns[c$id];
        conn$client$last_seen = network_time();
        conn$server$last_seen = network_time();
# To prevent the set of entries from premature expiration,
# we need to make a write access (can't use read_expire as we
# iterate over the entries on a regular basis).
        clients[c$id$orig_h] = conn$client;
        servers[c$id$resp_h] = conn$server;
        return conn;
}
function add_server(c: connection) : bot_server
{
        local s_h = c$id$resp_h;
        if ( s_h in servers )
                return servers[s_h];
        local empty_table1: table[addr] of bot_client;
        local empty_table2: table[string] of channel;
        local empty_set: set[string];
        local empty_set2: set[port];
        local server = [$host=s_h, $p=empty_set2, $clients=empty_table1,
              $channels=empty_table2, $passwords=empty_set,
              $first_seen=network_time(), $last_seen=network_time()];
        servers[s_h] = server;
        return server;
}
function add_client(c: connection) : bot_client
{
        local c_h = c$id$orig_h;
        if ( c_h in clients )
                return clients[c_h];
        local empty_table: table[string] of channel;
        local empty_set: set[addr];
        local client = [$host=c_h, $p=c$id$resp_p, $servers=empty_set,
              $channels=empty_table, $first_seen=network_time(),
              $last_seen=network_time()];
```

```
        clients[c_h] = client;
        return client;
}
function check_bot_conn(c: connection)
{
        if ( c$id in bot_conns )
                return;
        local client = c$id$orig_h;
        local server = c$id$resp_h;
        if ( client !in potential_bot_clients || server !in potential_bot_servers )
                return;
# New confirmed bot_conn.
        add bot_conns[c$id];
        if ( server !in confirmed_bot_servers )
        {
                NOTICE([$note=IrcBotServerFound, $src=server, $p=c$id$resp_p, $conn=c,
                                $msg=fmt("ircbot server found: %s:%d", server,
$p=c$id$resp_p)]);
                add confirmed_bot_servers[server];
        }
        if ( client !in confirmed_bot_clients )
        {
                NOTICE([$note=IrcBotClientFound, $src=client, $p=c$id$orig_p, $conn=c,
                                $msg=fmt("ircbot client found: %s:%d", client,
$p=c$id$orig_p)]);
                add confirmed_bot_clients[client];
        }
}
function get_conn(c: connection) : bot_conn
{
        local conn: bot_conn;
        if ( c$id in conns )
        {
                check_bot_conn(c);
                return update_timestamps(c);
        }
        local c_h = c$id$orig_h;
        local s_h = c$id$resp_h;
        local client : bot_client;
        local server : bot_server;
        if ( c_h in clients )
                client = clients[c_h];
        else
                client = add_client(c);
        if ( s_h in servers )
                server = servers[s_h];
        else
                server = add_server(c);
        server$clients[c_h] = client;
        add server$p[c$id$resp_p];
        add client$servers[s_h];
        conn$server = server;
        conn$client = client;
        conn$conn = c;
        conns[c$id] = conn;
        update_timestamps(c);
        return conn;
}
function expire_server(t: table[addr] of bot_server, idx: addr): interval
{
        local server = t[idx];
        for ( c in server$clients )
        {
                local client = server$clients[c];
                delete client$servers[idx];
        }
        delete potential_bot_servers[idx];
        delete confirmed_bot_servers[idx];
        return 0secs;
}
function expire_client(t: table[addr] of bot_client, idx: addr): interval
{
```

```
        local client = t[idx];
        for ( s in client$servers )
                if ( s in servers )
                        delete servers[s]$clients[idx];
        delete potential_bot_clients[idx];
        delete confirmed_bot_clients[idx];
        return 0secs;
}
function remove_connection(c: connection)
{
        local conn = conns[c$id];
        delete conns[c$id];
        delete bot_conns[c$id];
}
event connection_state_remove(c: connection)
{
        if ( c$id !in conns )
                return;
        remove_connection(c);
}
event irc_client(c: connection, is_orig: bool, prefix: string, data: string)
{
        if ( detailed_logging )
                print detailed_log, fmt("%.6f %s > (%s) %s", network_time(),
id_string(c$id), prefix, data);
        local conn = get_conn(c);
        if ( data == /^ *[iI][rR][cC][xX] *$/ )
                conn$ircx = T;
}
event irc_server(c: connection, is_orig: bool, prefix: string, data: string)
{
        if ( detailed_logging )
                print detailed_log, fmt("%.6f %s < (%s) %s", network_time(),
id_string(c$id), prefix, data);
        local conn = get_conn(c);
}
event irc_user_message(c: connection, is_orig: bool, user: string, host: string,
server: string, real_name: string)
{
        local conn = get_conn(c);
        conn$client$user = user;
        conn$client$realname = real_name;
        do_log(c, fmt("USER user=%s host=%s server=%s real_name=%s", user, host,
server, real_name));
}
function get_channel(conn: bot_conn, channel: string) : channel
{
        if ( channel in conn$server$channels )
                return conn$server$channels[channel];
        else
        {
                local empty_set: set[string];
                local empty_vec: vector of string;
                local ch = [$name=channel, $passwords=empty_set,
$topic_history=empty_vec];
                conn$server$channels[ch$name] = ch;
                return ch;
        }
}
event irc_join_message(c: connection, is_orig: bool, info_list: irc_join_list)
{
        local conn = get_conn(c);
        for ( i in info_list )
        {
                local ch = get_channel(conn, i$channel);
                if ( i$password != "" )
                        add ch$passwords[i$password];
                conn$client$channels[ch$name] = ch;
                do_log(c, fmt("JOIN channel=%s password=%s", i$channel, i$password));
        }
}
global urls: set[string] &read_expire = 7 days &persistent;
```

```
event http_request(c: connection, method: string, original_URI: string,
                unescaped_URI: string, version: string)
{
        if ( original_URI in urls )
                do_log_force(c, fmt("Request for URL %s", original_URI));
}
event irc_channel_topic(c: connection, is_orig: bool, channel: string, topic: string)
{
        if ( bot_cmds in topic )
        {
                do_log_force(c, fmt("Matching TOPIC %s", topic));
                add potential_bot_servers[c$id$resp_h];
        }
        local conn = get_conn(c);
        local ch = get_channel(conn, channel);
        ch$topic_history[|ch$topic_history| + 1] = ch$topic;
        ch$topic = topic;
        if ( c$id in bot_conns )
        {
                do_log(c, fmt("TOPIC channel=%s topic=\"%s\"", channel, topic));
                local s = split(topic, / /);
                for ( i in s )
                {
                        local w = s[i];
                        if ( w == /[a-zA-Z]+:\/\/.*/ )
                        {
                                add urls[w];
                                do_log(c, fmt("URL channel=%s url=\"%s\"",
                                                channel, w));
                        }
                }
        }
}
event irc_nick_message(c: connection, is_orig: bool, who: string, newnick: string)
{
        if ( bot_nicks in newnick )
        {
                do_log_force(c, fmt("Matching NICK %s", newnick));
                add potential_bot_clients[c$id$orig_h];
        }
        local conn = get_conn(c);
        conn$client$nick = newnick;
        do_log(c, fmt("NICK who=%s nick=%s", who, newnick));
}
event irc_password_message(c: connection, is_orig: bool, password: string)
{
        local conn = get_conn(c);
        add conn$server$passwords[password];
        do_log(c, fmt("PASS password=%s", password));
}
event irc_privmsg_message(c: connection, is_orig: bool, source: string, target:
string,
                message: string)
{
        log_msg(c, "privmsg", source, fmt("->%s %s", target, message));
}
event irc_notice_message(c: connection, is_orig: bool, source: string,
                target: string, message: string)
{
        log_msg(c, "notice", source, fmt("->%s %s", target, message));
}
event irc_global_users(c: connection, is_orig: bool, prefix: string, msg: string)
{
        local conn = get_conn(c);
# Better would be to parse the message to extract the counts.
        conn$server$global_users = msg;
        log_msg(c, "globalusers", prefix, msg);
}

event Input::end_of_data(name: string, source: string) {
for(ip in suspicious) {
                #print ip;
```

```
        }
}

event bro_done()
{
}
event bro_init() &priority = -5
{
        if ( summary_interval != 0 secs )
                schedule summary_interval { print_bot_state() };
Log::disable_stream(Conn::LOG);
Log::disable_stream(HTTP::LOG);
Log::disable_stream(Files::LOG);
}
```

F.3 Sample of Snort Rules for Botnet Detection

```
alert udp $HOME_NET 1024:65535 -> $EXTERNAL_NET 1024:65535 (msg:"E7[rb] BOTHUNTER
Storm(Peacomm) Peer Coordination Event [SEARCH RESULT]"; content:"|E311|"; depth:5;
rawbytes; pcre:"/[0-9]+\.mpg\;size\=[0-9]+/x"; rawbytes; classtype:bad-unknown;
sid:9910013; rev:99;)

alert udp $HOME_NET 1024:65535 -> $EXTERNAL_NET 1024:65535 (msg:"E7[rb] BOTHUNTER
Storm Worm Peer Coordination Event [PUBLISH]"; content:"|E313|"; depth:5; rawbytes;
pcre:"/[0-9]+\.mpg\;size\=[0-9]+/x"; rawbytes; classtype:bad-unknown; sid:9910011;
rev:99;)
```