INVESTIGATING A BEHAVIOUR ANALYSIS-BASED EARLY WARNING SYSTEM TO IDENTIFY BOTNETS USING MACHINE LEARNING ALGORITHMS

by

Fariba Haddadi

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
September 2016

I dedicate this thesis to my beloved family

You have successfully made me the person I am becoming
You have always loved me unconditionally and taught me to work hard
for the things that I aspire to achieve.

Table of Contents

List of Tables

List of Figures

Abstract

Botnets represent one of the more aggressive threats against cyber security and botnet traffic analysis is one of the main approaches to study and investigate such threats. Botnets employ different techniques (*e.g.* fluxing and encryption), topologies (*e.g.* centralized and de-centralized) and communication protocols (*e.g.* HTTP and DNS) in different stages of their lifecycle. Therefore, identifying the botnets has become very challenging given that they can upgrade their methodology automatically at any time for one reason or another. To this end, different approaches are proposed for botnet traffic analysis and detection based on various botnet behaviours and structures. Hence, the main focus of this thesis is to investigate various botnet detection approaches based on the technique used and the available data.

Specifically, two main categories of solutions are explored: application data analysis-based solutions and network analysis-based solutions. In the application data analysis category, two different approaches are explored: one with *a priori* knowledge and the other one without any *a priori* knowledge. On the other hand, flow-based botnet detection approaches are explored in the network analysis-based category focused on using minimum a priori knowledge. In this case, various feature extraction methods, machine learning algorithms, protocol filtering, non-numeric feature representation, normal behaviour representation and time generalization issues are investigated. Finally, a flow-based early warning system is proposed.

The effectiveness of the solutions is shown on several botnet data sets from IRC botnets to peer-to-peer botnets. Results indicate that the proposed solutions can detect botnet behaviour with good performances. Moreover, two botnet detection systems from the literature and two publicly available malicious behaviour detection systems are employed for further evaluation of the proposed early warning system. The results indicate that the proposed system outperformed these four systems. Last but not least, the proposed system is evaluated as well on botnets in cellular networks on an exploratory basis. It is shown that the proposed system demonstrates promising performance under such circumstances as well.

List of Abbreviations Used

ACK	Acknowledgement
ANN	Artificial Neural Networks
BvL	Botnet versus Legitimate
C&C	Command and Control
CWR	Congestion Window Reduced
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DNS	Domain Name System
DoS	Denial of Service
DR	Detection Rate
ECE	ECN-Echo
FIN	Finish
FlowAF	Flow Aggregation/Fraction
FPR	False Positive Rate
FTP	File Transfer Protocol
IP	Internet Protocol
IRC	Internet Relay Chat
KNN	K Nearest Neighbour
MAC	Media Access Control

NIMS	Network Information Management and Security
P2P	Peer-to-Peer
PSH	Push
RST	Reset
SBB	Symbiotic Bid-Based
SSK	SVM String Kernel classifier
SSK-LP	SSK- Lambda Pruning
Stateful-SBB	Stateful Symbiotic Bid-Based Genetic Programming
SVM	Support Vector Machine
SYN	Synchronize
TCP	Transmission Control Protocol
TRR	True Positive Rate
TTL	Time To Live
UDP	User Datagram Protocol
URG	Urgent

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor, Professor Nur Zincir-Heywood for her continuous support during my Ph.D study, and for her patience, motivation, and enthusiasm. I want to thank you for all of the opportunities I was given to conduct my research and further my thesis.

My sincere thanks also goes to Professor Malcolm Heywood for his valuable guidance. You definitely provided me with the tools that I needed to choose the right direction and successfully complete my thesis.

My thanks and appreciation go to members of my dissertation committee and the Faculty of Computer Science, Dalhousie University who have given their time and expertise generously to better my work. I thank you all for your contribution and your good-natured support.

With great pleasure, I would like to thank my parents, Parivash Haddad and Ahmadreza Haddadi, for their wise counsel, unceasing encouragement and support throughout my life. You have always been there for me. Last but not least, I am grateful to my husband, Vahid Aghaei Foroushani, who supported me through this venture. I am truly thankful for having you in my life.

Thank you very much, everyone!

Chapter 1

Introduction

In the world of fast growing Internet and online activities where almost everyone has something to share and benefit from, having a secure infrastructure is the primary need for protecting users identity and information. A botnet is a network of compromised hosts which are controlled remotely by a master (a.k.a a botmaster). Different types of botnets have been created to perform various malicious tasks such as spreading spam, conducting Denial of Service attacks, performing identity thefts or simply taking advantage of victims' computational resources. In 2010, Damballa Inc. published a paper on the top ten active botnets indicating that the botnet infection rate is increasing rapidly by an average growth of 8% per week [101]. McAfee threat reports also confirm that this growth continued into 2013 [125]. These reports indicate as well that new powerful botnets enter the Internet realm every year. In January 2014, McAfee published a threat prediction report- an indication of how botnets will evolve and act in 2014 [124]. Hence, botnets are considered to be one of the main threats against cyber security [69].

Unlike the earlier botnets that had a list of exploits to launch on targets and all the commands were set at the time of infection, today a typical advanced bot uses multiple phases to create and maintain a botnet including: initial infection, secondary injection, connection, malicious Command and Control (C&C), update and finally, maintenance [69, 143], Fig. 1.1. In the first phase, the attacker infects the victim using several exploitation techniques to find its existing vulnerabilities. Once the target is infected, in the second phase, a script known as shell-code is executed on the victim machine. The shell-code fetches the image of the bot binary and the bot binary installs itself on the machine. At this time the host is converted completely to a zombie and malicious applications can run automatically on the host. In the connection phase, the bot binary establishes a C&C channel with the master. At this stage, the master uses the C&C channel to send the commands to its bot army

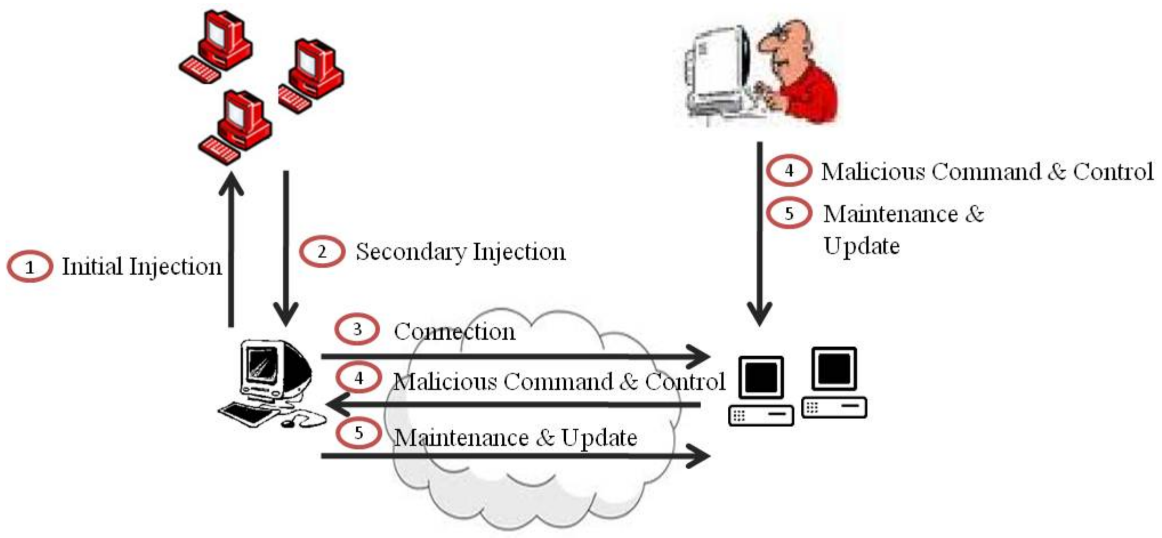


Figure 1.1: Botnet life cycle

(botnet) and the bot applications execute the commands as soon as they receive them. Finally, when the master needs to update the bots for various reasons such as avoiding an antivirus, changing the C&C server information, or adding a new functionality, the update and maintenance phase is entered.

In this lifecycle, the communication scheme is the main characteristic of the botnet architecture which evolved over time to enhance botnet functionality and avoid botnet classification (detection) systems. In the architecture, compromised bots interact with the C&C server to receive the instructions of the master. Until 2003, the Internet Relay Chat (IRC) protocol was the most common botnet communication protocol using centralized topology [102]. In the botnet arms race, security systems adapted to use solutions (such as firewalls) to block ports such as IRC ports or to perform content analysis/filtering which can reveal botnet communication information. Since 2003, not only have botnets started to use more ubiquitous protocols such as HyperText Transfer Protocol (HTTP) and Domain Name System (DNS) as well as a de-centralized topology such as Peer-to-Peer (P2P), but they have started to employ techniques such as fluxing and encryption as well to avoid detection. Hence, identifying the botnets and detecting them have become very challenging.

Internet Protocol (IP) Network traffic analysis (*e.g.* classification) has the capacity to assist in analyzing complicated network defence problems and therefore has been used widely by many researchers in the security field. The aim of this thesis is to design, develop and investigate an early warning system for identifying botnet behaviour via network traffic analysis. To this end, the approaches that are designed and the tools that are employed in such an early warning system very much depend on the botnet communication methods and the type of the network traffic available for analysis (*e.g.* having access to the packets payload and encrypted vs. unencrypted communication). Hence, several scenarios were considered and suitable approaches designed which match the goal and type of data that is accessible.

Botnets have been observed in wired and wireless networks with various topologies. In this thesis, the focus is on analyzing botnet behaviours specifically on wired networks. However, preliminary evaluations were conducted on data sets including botnets on cellular devices as the basis of exploring how far the proposed system can be pushed. Botnets found on wired networks have different topologies and take advantage of various protocols and techniques. Given that recent botnets tend to use de-centralized topologies (such as P2P), most of the data sets used in this research are placed in this category. Among the protocols utilized by these botnets in such topologies, HTTP is one of the more ubiquitous ones in Internet realm. Hence, the HTTP-based botnets with the de-centralized topology can be considered as the most dangerous botnets which can hide their malicious activity very well in the high volume of normal HTTP traffic. For this reason, special attention is paid to botnets with the HTTP protocol while older botnets, such as IRC botnets, are included in the evaluations as well.

Using this framework, two different approaches were designed and analyzed, each dealing with specific scenarios where both aim to detect botnet behaviour using the available information. In other words, the goal is to analyze the available information to recognize the behaviour of the botnet communication based on the footprints left by these attackers. Basically, various features of the data are extracted and analyzed to recognize such behaviour. Generally, there are two main types of network traffic: encrypted and unencrypted, Fig. 1.2. In each of these two main categories, the packet payload may or may not be accessible because of privacy and security issues.

Since 2004, DNS has been used in botnet traffic more and more to add mobility, remove the single point of failure [123] and thereby implementing a de-centralized

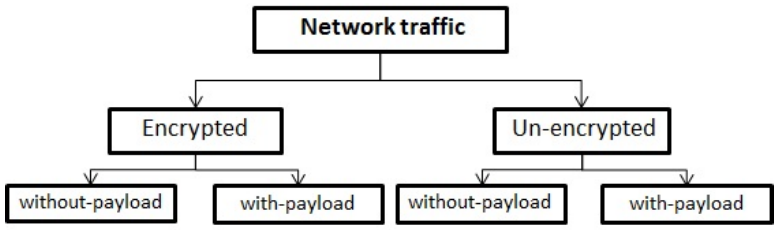


Figure 1.2: Types of network traffic

topology. To this end, fluxing techniques are designed to move the communication between the victims and the C&C server from one domain/IP to another domain/IP using the DNS protocol. Hence, the first scenario assumes that the botnet communication is using domain fluxing and the packet payload is accessible, specifically for DNS queries. Consequently, for this approach, an application data analysis-based system was designed.

In this case, the proposed approach works on the domain name strings extracted from the DNS packet payload. Two different methods are studied: feature extraction with *a priori* information and feature extraction without *a priori* information. The first method employs the features extracted from the domain name strings, based on *a priori* information, in order to detect the botnets. In the second method an evolutionary computation technique is designed, called Stateful-SBB (Stateful Symbiotic Bid-Based Genetic Programming), which employs the raw domain name strings (no *a priori* information) extracted from the DNS packet payload. These application analysis-based approaches address the important questions of this field:

(i) Can botnet DNS communication be detected based on domain name analysis without any further traffic analysis?

(ii) If yes, what domain name features should be extracted using *a priori* information?

(iii) Can a detection system be designed to recognize botnet behaviour at the application level without any *a priori* knowledge (a.k.a without a pre-defined feature set)?

(iv) Is the performance of a detection system with no *a priori* knowledge comparable to a system with *a priori* information?

(v) What would be gained/missed while using any of the aforementioned methods?

To evaluate this approach, publicly available malicious and legitimate domain name lists (from legitimate resources) are employed. Results show that the first method can detect the botnet's domain names with a detection rate (DR) of up to 96% and the second proposed method (Stateful-SBB) performed better with a detection rate of up to 100% without requiring a predefined feature set. Avoiding such a requirement is the most important contribution of this technique since this enables the approach to adapt to changes in the botnet upgrades of the domain generation algorithm. The performance of these approaches changed based on the type of botnet and the type of data set (balanced vs. unbalanced).

On the other hand, there are other possible scenarios in which the network traffic is encrypted (with/without payload), or it is un-encrypted but the packet payload is not accessible or the collected traffic does not contain any DNS packets. For such scenarios, a second approach was proposed and designed based on network data analysis. In this approach, the flow-based[1] network traffic features extracted using various flow exporters and machine learning (ML) classifiers are employed for differentiating legitimate traffic from the malicious. The goal is to explore the possibility of using publicly available tools and algorithms to design an early warning system with a minimum *a priori* knowledge. Specifically, several experiments are designed to address the following questions:

(i) Can all types of botnets be analyzed and detected with one feature set or should different feature sets be be introduced and employed depending on the botnet?

(ii) What role does the choice of machine learning play in the effectiveness and performance of a botnet detection system?

(iii) Given that botnets may use different protocols at each stage of their lifecycle, can data filtering (*i.e.* protocol filtering) make a difference in detection performance?

(iv) Given that most of the known machine learning techniques employ numeric features, should non-numeric features be represented in a numeric form? How effective can this approach be? What type of representation should be used?

(v) Having different types of botnets can a detection system distinguish them all from normal user behaviours? If so, can this system differentiate these types of

[1]Flow is defined as a logical equivalent for a call or a connection in association with a user-specified group of elements [126]. The most common way to identify a traffic flow is to use a combination of five properties (a.k.a 5-tuple) from the packet header, namely source/destination IP addresses and port numbers as well as the protocol.

botnets from each other?

(vi) Finally, how effectively can a machine learning-based early warning botnet detection system perform over a period of time, considering botnet evolution?

In this approach, six open source flow exporters (Maji, YAF, Softflowd, Tranalyzer, Argus and Netmate) along with several highly employed machine learning techniques [*e.g.* C4.5, SVM, Naive-Bayes, Bayesian Networks and Artificial Neural Networks (ANN)] are utilized. This approach was evaluated based on the effect of the factors listed below.

1. **Type of traffic.** For this purpose, three sets of experiments are conducted; the first set of experiments study the analysis and classification of botnet traffic using all of the traffic flows, the second set performs the same study using only the HTTP traffic flows, *i.e.* employing an HTTP protocol filter for HTTP-based botnets, and the third set uses only the DNS traffic flows (if available) utilizing a DNS protocol filter. The evaluation results show that the use of feature extraction tools, machine learning algorithms and protocol filtering can affect the performance of a detection system greatly. For the evaluations performed, the combination of the Tranalyzer tool with the C4.5 classifier using the HTTP filter gives the best performance in terms of the DR and false positive rate (FPR) on all the botnet data sets employed. However, it should be considered that HTTP filtering is not always applicable. Without using any filter, the combination of Tranalyzer and C4.5 still outperformed the other combinations.

2. **Feature representation.** Some of the features calculated and exported by flow features has non-numeric representation. Given that most of the known machine learning algorithms accept numeric type values as inputs, experiments were conducted on how these features should be presented. The main focus of the analysis is on the TCP-Flag feature which has been shown to be utilized by botnets. Overall, no notable performance increase was observed throughout the different representations.

3. **Botnet behaviour evolution.** Botnets can upgrade and evolve their topology and/or modify their algorithms on the fly based on the fifth phase of their lifecycle. The important question in this case is how the trained early warning system

would perform in this situation. Hence, experiments were conducted to test the performance of the proposed approach over a period of time in which different versions of the botnets were presented as inputs. These versions vary from minor changes in the communication structure to major changes in the topology (*e.g.* from a C&C-based de-centralized topology to a P2P-based de-centralized topology). The results show that the early warning system performance changes within a range of 80% to 100% with botnet version behaviour changes over four years while a sharp drop in the detection rate raise a red flag- a major change in topology.

4. **Normal behaviour representation.** In network traffic analysis systems in the literature, different traffic log files are employed to represent normal use behaviour. However, it appears no research has been done on how the choice of traffic files representing the normal behaviours might affect the evaluation of the proposed systems and the results. In this work three different data sets were used to represent normal behaviours, namely Alexa-D (NIMS), ETP (LBNL) and ISP (WiSNet), which will be introduced in Chapter 4. Evaluation results show that although these data sets have different characteristics, choosing one over the other does not have a notable affect on the performance of the early warning system proposed in this thesis.

Furthermore, apart from the importance of the proposed detection methods in the literature, in all cases, the first challenge is to obtain realistic network data that represents botnet traffic (behaviour). Due to the malicious nature of such data, it is difficult to find publicly available data sets reflecting real-life scenarios. Therefore, different approaches have been explored in the literature. These include (but are not limited to): (i) Running botnet binaries (that are publicly available or modifying such binaries) in sandbox environments and capturing the traffic [90, 147, 106, 139]; (ii) Obtaining captured Honeynet traffic [85, 156]; or (iii) Obtaining traffic from a network operator or a security company [71]. Using different versions of the Zeus and Citadel botnet toolkits, various scenarios were designed and these botnets were executed in a sandbox environment. The captured data in the sandbox is employed in this thesis which belongs to the first category. Moreover, a systematic approach is proposed as well to generate botnet traffic data representing the first stage of botnet communication (*i.e.* the connection phase of the botnet lifecycle) where the infected host intends to locate the C&C server and to make a connection with the server. This approach was evaluated under different conditions and shows that it can generate realistic botnet behaviour similar to the traffic captured in the wild [95, 98].

1.1 Objectives

The primary objective of this research is to explore an early warning botnet detection system based on a behaviour analysis approach with no or minimum *a priori* knowledge employing machine learning algorithms. To this end, the following issues were researched on both network and application level data that were publicly available to researchers.

1. Exploring the possibility of detecting the botnet using only the domain names appearing in the DNS packet payload without any traffic analysis.

2. Investigating the features of the malicious domain names and exploring the possibility of detecting the malicious automatically-generated domain names using raw text, string format domain names without any *a priori* knowledge;

3. Exploring the possibility of detecting botnet traffic when the traffic is encrypted, DNS analysis is not possible or the packet payload is not available, using flow-based features.

4. Exploring the possibility of creating a network flow-based early warning system that uses minimum *a priori* knowledge to detect botnet behaviours using open source tools and algorithms that are publicly available. This would make the system highly repeatable and usable for non-expert analysts in botnet detection.

5. Investigating the effect of various machine learning algorithms, feature extraction methods, various feature sets, traffic filtering, feature representation techniques on botnet behaviour detection.

6. Analyzing the features employed by the machine learning algorithms to distinguish different botnet behaviour, *e.g.* Zeus and Citadel.

7. Exploring the possibility of presenting a systematic method to generate botnet traffic which is similar to real botnet behaviours.

Indeed, most of the literature has analyzed botnet traffic in different ways to propose a detection method that can be applied to one or more types of botnets. In this regard, specific approaches could require specific types of network application/data to analyze which might not always be accessible. Hence, collecting/capturing complete real-life data to analyze is very challenging. For example, if a method uses packet payload analysis, it would not be able to detect a botnet that uses payload encryption. The objective in this research is to present a set of approaches and tools that can be employed to detect botnet behaviour in different situations and investigate what will be gained or missed when using a combination of the proposed approaches. This will then help the security analyst to choose a set of tools and approaches to analyze the available data and identify the botnet behaviour.

1.2 Contributions

As discussed earlier, the objective of this thesis is to investigate the possibility of designing and developing an early warning system to detect botnet behaviours with no or minimum *a priori* knowledge. To this end, a framework is proposed employing machine learning-based approaches and suitable feature sets depending on the available data for various botnet behaviour analysis purposes. This framework is evaluated on botnets with centralized, de-centralized and de-centralized P2P topologies utilizing IRC, DNS, HTTP and P2P protocols. In this research, one of the main challenges is the type of data that is analyzed and the features which should be extracted in order to recognize botnet communication. These features may differ from botnet to botnet based on the botnet's structure. Therefore, a specific feature set cannot be considered as a general feature set, *i.e.* one size does not fit all.

To achieve the above mentioned objectives, the contributions of this thesis can be summarized as follows.

1. **Data generation.** The first challenge in designing and developing a botnet detection system is to have access to realistic botnet data that represent botnet behaviour and can be analyzed and used for system design, benchmarking and

evaluation. In this thesis publicly available data sets were employed as well as lab-generated data sets to analyze botnet and normal behaviours for an early warning system. In the case of data generation, two approaches were used to generate the botnet data: (i) using botnet toolkits and publicly available botnet binaries, (ii) using publicly available botnet domain names to generate the botnet traffic which represents the first phase of the botnet communication. Although the first approach has also been used previously [85, 139, 147, 145], in this thesis different scenarios were designed and various settings were employed (e.g. enabling a web injection option) to generate botnet traffic. This way a collection of data sets were captured showing that the same botnet may have various functionalities and targets. A more detailed benchmarking and botnet behaviour analysis becomes possible with this approach. The second approach however is novel and based on the evaluations detailed in the following chapters, represents realistic botnet behaviours.

2. **Application data, specifically domain name, analysis.** This approach focuses on identifying the domain names generated by the botnets' Domain Generation Algorithm (DGA). In this case, all of the DNS-based analysis approaches in the literature, to the best of my knowledge, are *a priori* information-based detection systems where various DNS-based features are extracted from the network traffic and/or domain names [105, 53, 149]. Hence, in the application data analysis approach of this thesis, two different methods are investigated where both only take advantage of the DNS domain names without any further DNS traffic analysis. One of the proposed application data analysis methods employs a feature set defined based on *a priori* information for detection purposes. For the second method, a co-evolutionary algorithm has been designed and developed purely based on the string format domain names where the features are the ASCII code of the characters of the domain name. In other words, the proposed method (called Stateful-SBB) can detect the botnets' malicious domain names without any *a priori* knowledge. This is a great advantage when botnets can upgrade themselves at any time and the detection system need to cope with the upgrades as quickly as possible. With this method, there is no need for the human experts to analyze the malicious domain names in order to understand

the new algorithm behind the domain generation process to update the feature sets.

3. **Network Data Analysis.** The second approach is based on network data analysis, specifically traffic flows, where network traces are aggregated into flows only using the information embedded in the header of the packets and some statistics are then calculated. However, there is not only one standard flow feature set that can be used in this case and several different sets have been proposed and employed in the literature for various types of botnet communications. Hence, in this approach, the goal is to study the effect of various techniques and tools (leading to different feature sets) in detail. This form of detailed analysis on botnet behaviour analysis through flow feature sets has not been done by any work in the literature. This is one of the main contributions of this research which happens to be the most comprehensive analysis in this field. In this case, the effect of traffic filters, various feature sets, machine learning algorithms, feature representations and botnet evolution are investigated.

   Moreover, all of the approaches in the literature have chosen and employed a set of data to represent legitimate (normal) behaviour. It appears no research has been performed to understand how such a choice (representation of legitimate behaviour) will effect the performance of the designed system. Consequently, an investigation has been made of the importance of legitimate data selection (if any) which is part of the contributions of this thesis.

4. The two proposed approaches provides solutions which can be understood by network administrators easily. The analysis of the solutions aims to address an important question: can these solutions reveal any patterns or structures in botnet behaviours?

5. Although the focus of this thesis is on botnets in wired networks, the proposed botnet behaviour analysis framework has been applied and evaluated on the botnets seen on cellular networks. To the best of my knowledge, none of the work in the literature has been investigated and evaluated in this manner.

6. Twenty five botnet samples are employed in this thesis. Older botnets like IRC

decentralized to more recent botnets like de-centralized P2P are included in this data set collection. This is the most complete set of data sets that have been analyzed in the literature.

7. Given that different data sets are employed in the literature and some of these data sets are not publicly available, the repeatability of these experiments is impossible and the claimed performances are used for the comparison of the results. Thus, this thesis has taken advantage of the two publicly available tools (Snort and BotHunter) for evaluating these data sets, as well as implementing two other systems [118, 156] which have been proposed in the literature. The goal is to provide a baseline for the work done in the literature on these data sets.

   Although the easiest choices would be two flow-based detection systems with different feature sets, the goal was to choose two systems that would offer something new and different compared to the approaches employed. One of these systems is a flow aggregation/fraction-based detection system and the other one is a packet payload-based detection system [88]. All four of these tools/systems are based on *a priori* knowledge to define the feature set or the rule set. In summary, to the best of my knowledge, none of the work in the literature has been compared against other approaches at such comprehensive level as in this thesis.

Several outcomes of this thesis research have been published in different journals and conferences dealing with network operation, artificial intelligence, and machine learning [88, 89, 93, 90, 92, 94, 96, 97] which indicate that the proposed solutions are worth investigating. The application data analysis-based approaches have been published in [89, 93] in which the approaches are evaluated on different data sets and compared to the state-of-the-art techniques. On the other hand, the network data analysis-based early warning system and its evaluations have been published as well [90, 92, 96, 97]. The effect of the feature extraction/selection in addition to the effect of traffic filtering has been addressed specifically [94]. Moreover, the confirmation of data sets generated in this research has been published as well [95]. Finally, the evaluation and comparison of the network data-based early warning system with the other approaches in the literature has been published as well [88, 91].

1.3 Organization of the Thesis

This thesis is structured into eight main chapters as follows. Chapter 2 outlines the background information (*e.g.* a summary of the botnet lifecycle which will provide a better understanding of the concepts of the following chapters) in addition to a literature review. Chapter 3 introduces the machine learning algorithms and the tools employed in this research. This includes the flow exporter tools and the publicly available malicious behaviour detection systems. In Chapter 4 all the data sets that are used in the evaluation of early warning system proposed in this thesis are introduced. Chapters 5 and 6 present the application data analysis-based and network data analysis-based early warning systems, respectively. Each chapter discusses the methodology and presents the experimental results of the system according to the corresponding proposed data analysis method. In Chapter 7 the proposed early warning system is compared against four different malicious behaviour detection systems. Finally, conclusions are drawn and future work is discussed in Chapter 8.

Chapter 2

Background

A bot program is a self-propagating malware which infects vulnerable hosts known as bots (zombies) and is designed to perform a task after being triggered. Hosts can be infected with malware in different ways such as visiting an untrusted malicious website or opening a malicious email attachment. The infected bot network is referred to as a botnet, which is under the remote control of a master called a botmaster. Usually bots receive commands from the master through a C&C communication channel and carry out malicious tasks such as Distributed Denial of Service (DDoS), spamming, phishing and identity theft attacks [69, 143]. A DoS attack is an aggressive attempt with the purpose of making a network or a specific machine's resources inaccessible to its legitimate users. A DDoS is an attack in which multiple compromised systems attack a single target to interrupt or suspend the resources and services of the targeted system (victim) temporarily or indefinitely. Spam is any kind of a message, regardless of its content, which is sent to recipients who have not requested it. In spamming botnets, the botmaster commands its infected bots to send spam messages to the predefined list of victims selected by the master. Phishing is used to describe technical methods (such as spoof emails) which deceive targets into giving up their personal or professional confidential information.

2.1 Botnet Topology

Botnet topology is classified in two major categories based on the C&C communication method: centralized and decentralized [114]. In the centralized model, commands and data exchanges between the master and the bots are managed at the central C&C server. The C&C server uses different services/protocols to manage the botnet. Since all connections are going through a specific C&C server, its detection is relatively easier. As a result, attackers started to use decentralized communication methods such as Peer-to-Peer (P2P) architecture and decentralized C&C which is

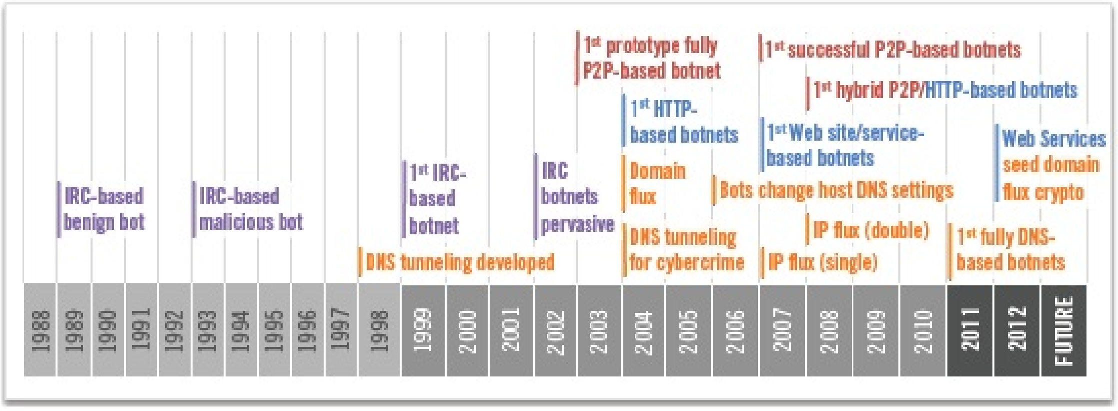


Figure 2.1: Botnet protocols and techniques timeline [102].

more complicated and harder to discover. In the P2P model bots can act as both clients and servers. They do not contact any specific server for commands directly but receive the commands from their peers.

It is believed that until 2003, most of the botnets were using a centralized topology, utilizing Internet Relay Chat (IRC) Protocol, Fig. 2.1, [102]. Since 2003, not only have botnets started to use several protocols such as P2P-based protocols, HyperText Transfer Protocol(HTTP) or DNS and a decentralized topology but also they have started to employ different mechanisms such as fluxing to avoid detection. Fluxing is a technique used to move the communication between the victims and the C&C server from one domain/IP to another domain/IP using the DNS protocol which forms a decentralized C&C-based architecture. Therefore, since 2004 DNS has been used in botnet traffic more and more to add mobility and to remove the single point of failure [123]. There are two categories of fluxing: (i) Domain fluxing, and (ii) Fast-fluxing (or IP fluxing). Fast-flux is a mechanism in which a set of IP addresses associated with a domain name change with high frequency. By contrast, domain fluxing is based on the idea of generating domain names using a Domain Generation Algorithm (DGA). A C&C server and its respective bots use the same algorithm and parameters to generate consistent domain names. Bots try to resolve the list of generated domain names in order to connect to the C&C server. Once one of the domain names is resolved, that domain name and its associated IP address will be used for server connections until the next round of the domain generating process.

2.1.1 Protocols Employed

Most recent botnets employ the HTTP protocol to hide their malicious activities within the normal web traffic given the wide range of HTTP usage on the Internet [43]. Moreover, DNS is an essential component for the Internet to function and is used in botnet architectures to provide robustness and mobility as well as to avoid the single point of failure [102]. Therefore, using these protocols, intruders (such as botnets) can hide their malicious activities easily within the normal traffic and therefore bypass firewalls and avoid detection mechanisms. As well, some of the data sets employed in this research have utilized the IRC protocol for communication. What follows is a brief description of these protocols.

2.1.1.1 HyperText Transfer Protocol (HTTP)

HTTP is an application layer protocol built on top of TCP, which is the foundation of data communication for the World Wide Web [104]. Although this protocol is designed mainly for the exchange/transfer of hypertext (*i.e.* structured text) based on a client-server model, it can be used for many other tasks such as distributed object management systems. In its initial client-server design, the client (*e.g.* a web browser) submits an HTTP request to the server (*e.g.* a web site server) and the server then responds properly based on the client's request. The three main HTTP message types are GET, POST and HEAD where the GET method indicates that the server is supposed to return an entity (*i.e.* information retrieval), the POST method submits data to be processed by the specified source/server (*i.e.* information submission) and the HEAD method is identical to GET except the server must not return a message-body in response (*i.e.* obtaining information about the entity without transferring the entity-body itself). Botnets with centralized or decentralized architecture (including P2P botnets) utilize the HTTP protocol to form the communication channel.

2.1.1.2 Domain Name System (DNS)

DNS is a hierarchical distributed system service which translate the domain names to their associated IP addresses needed to locate a system. Given that memorizing the domain names is much easier than memorizing the IP addresses, DNS is usually

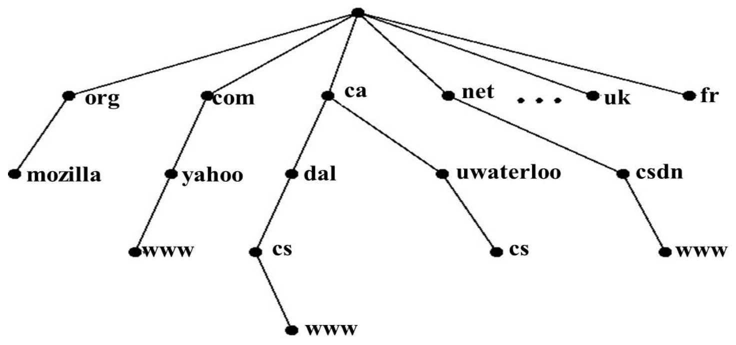


Figure 2.2: A simplistic view of the domain name space

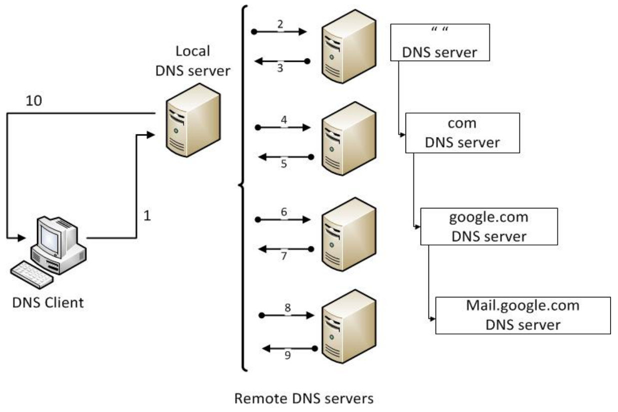


Figure 2.3: A simplistic view of the DNS look up

employed to locate the target by the given domain name before any type of communication can be established. Hence, this is one of the more commonly used protocols in the Internet. The domain name space is in the form of a tree of domain names in which each node or leaf holds information on zero or more resource records and the top level of the hierarchy is served by the root name servers that are responsible for the lower levels. Figure 2.2 is a very simplistic view of the domain name space.

When a DNS client needs to look up a name, it sends a DNS query to the DNS servers which contain specific information such as the domain name. This query can be resolved in different ways. It can be resolved locally by the client's cached information from previous queries or by the local DNS servers responsible. The DNS servers, however, can also ask to resolve the query from other servers recursively if no information is available on the authoritative zones. Figure 2.3 is a simplistic view of a DNS resolving process when no locally cached record is available.

2.1.1.3 Internet Relay Chat (IRC)

IRC is an application layer protocol that forms text-based communication. This protocol is designed for real time chat communication based on client-server architecture. The client consists of computer programs which can be installed on the users system. The clients communicate with chat servers to transfer text messages to other clients. In the main, the IRC protocol targets group communication through communication channels but also supports private messaging (one-to-one communication) and data transfer (such as file sharing).

When an IRC client connects to an IRC server, it must first register its connection. Once several users are connected and registered, they can send messages to each other. Each user sends the message that is intended for another registered user through the server. The server's duty is to receive the messages and forward them to the target user. On the other hand, users connected to an IRC server can also join existing channels on the server. Once a user has joined a channel, it can start sending/receiving messages. The difference is that the server will relay the message to all the users in the channel, instead of just a single user.

2.2 Botnet samples

What follows is a brief introduction to the botnets that are employed at some point in this thesis.

Zeus: Zeus is a well-known botnet that made a big comeback (after a takedown in 2012) with a new variant in 2013 [47]. This botnet has been collecting banking data by using man-in-the-browser keystroke logging and form grabbing but also can be configured for any type of identity theft attack. The HTTP protocol has been used as the carrier protocol for this botnet while the architecture has changed from C&C-based to P2P-based over the time.

Citadel: The Citadel botnet is the improved version of Zeus in which Zeus bugs are fixed and it is adapted to the newest security platforms [4]. Hence, this is also an HTTP-based botnet like Zeus. It is believed that Citadel stole more than $500 million and infected more than five million PCs in different countries. In June 2013, Microsoft and the FBI took down almost 90% of the Citadel botnet based on a court order in an operation. Yet, there is news of Citadel making a bold comeback [48].

Conficker: Conficker is an HTTP-based botnet which move from a C&C-based topology to P2P topology over a period of two years. Speculations about Conficker's purpose ranges from DDoS attacks or using distributed computing resources to steal banking credentials. In 2009 and 2010, the Conficker botnet was listed in the Damballa top ten botnets of the year. As of 2013, it is reported [72] that the Conficker botnet has infected many medical devices.

Cutwail: The Pushdo trojan was used originally to distribute other malware such as Zeus. It comes with its own spam module, known as Cutwail which is responsible for a large portion of the world's daily spam traffic. In 2013, this botnet evolved using domain generation algorithms (DGA) and became more resilient to takedown attempts [46]. Some variants of Pushdo utilize HTTP on top of the SSL/TLS protocol for C&C communication [44].

Kelihos: Kelihos is involved mainly with DDoS attacks and email Spam attacks. Capabilities of stealing Bitcoin wallets and spreading malicious links over social network sites such as Facebook were added to its latest versions. This botnet was first discovered in December 2010 and sent billions of spam messages in one day. Since then, three versions of Kelihos have been detected. The first two versions use HTTP while the latest version employs an HTTP-based P2P topology along with the Fast-Flux method [45].

Torpig: Torpig is an HTTP-based botnet designed specifically to harvest sensitive information (such as bank account and credit card data) from its victims [138]. In this botnet, victims are infected through drive-by-download attacks. In such attacks, first web pages on legitimate but vulnerable web sites are modifies with HTML tags which causes the victim's browser to request the Javascript code from a web site under the control of the attacker. This code then starts the vulnerability exploitation phase of the botnet lifecycle and then moves on to the other phases if possible.

Kraken: Kraken was the world's largest botnet in 2008 designed for spreading spams which was estimated to have sent almost nine billion spam messages per day. In contrast, Bobax, which is the newer version of this botnet, was listed in McAfee threat reports of 2011 and 2012 as one of the top active treats [125]. Hence, due to the usage of the domain fluxing technique and encryption and its high infection rate, this botnet was also considered as one of the top botnets which should be analyzed.

ZeroAccess: ZeroAccess is a P2P botnet discovered first in 2011 but which survived two take down attempts until 2015. Reports are indicating that this botnet is used primarily for click fraud and Bitcoin mining. Sophos Lab [31] estimated that this botnet has infected more that nine million systems and has cost advertisers millions of dollars. Exploit packs and social engineering have been the most common methods for spreading the ZeroAccess botnet.

Virut: This botnet came to existence in 2006 and has been used for various malicious activities such as DDoS attacks, spam, identity theft and pay-per-install activities. In 2013, Symantec estimated that the botnet had control of over 300,000 computers worldwide [34] whereas Kaspersky reported Virut as the fifth-most widespread threat in the third quarter of 2012 which was responsible for 6.8% of botnet infections [30]. As reported [74], Virut is an HTTP-based botnet given that it takes advantage of the HTTP protocol for communication.

Rbot: As previously discussed in this chapter, the IRC protocol is designed for real time chat communication based on client-server architecture. There are two basic levels of access to IRC channels: users and operators. The operator is the user who creates the channel and has more privileges compared to regular users. In an IRC botnet, the botmaster (attacker) controls the bots by sending commands to its own created malicious channel while switching channels to avoid being taken down. The advantage of an IRC botnet as opposed to HTTP botnets, for example, is that IRC allows for the interactive control of the bots. That means an attacker can send custom commands to a specific bot. In other words, IRC botnets have a much higher degree of control over the botnet network with a comparably low effort. However, this structure only works well for a relatively small bot network. This type of botnet has been used for for coordinating DDoS attacks and spam campaigns.

2.3 Related Work

Over time, botnets have employed different protocols, topologies and techniques to implement the aforementioned five phases of the lifecycle while avoiding detection. Hence, an arms race has started between the botnets and the detection systems. There are various techniques proposed for botnet detection due to the wide range of botnet methodologies. From the data perspective, while some techniques focus on malware source/binary analysis, others use host and/or network-based data. Given that this thesis is focussed on data analysis/behaviour extraction-based systems, this section focuses on the two main host-based and network-based detection systems proposed in the literature.

2.3.1 Host-based detection approaches

A host-based detection approach monitors and analyzes the internal behaviour of a computer system to detect anomalies and malicious botnet behaviour.

Masud *et al.* proposed a detection system which recognizes bot activities in the host machine using the temporal correlation of two host-based log files [117]. The approach uses data mining to learn temporal correlations between an incoming packet and outgoing packets, new outgoing connections and application startups. An incoming packet which correlates with one of these events would be flagged as a possible botnet packet. Connection features are then generated by the aggregation of the packet-based features. Finally, the system classifies the connections as botnet or normal based on the connection features. The system is trained on a combination of normal log files and botnet log files generated in a testbed and then tested on unbalanced new log files. The system showed an accuracy of 99% in the evaluations.

Stinson *et al.* proposed a host-based botnet detection system called BotSwat [137]. The system identifies potential botnet programs by focussing on the way these programs respond to data received on the network. It is claimed that a program shows external or remote control when it uses data received from an untrusted source on the network in a trusted system call argument. The system first scans the execution status of the Win32 library and monitors the runtime system calls created by a processor. It then tries to discover bots with generic properties despite the particular C&C architecture, communication protocols, or botnet structure. The system was evaluated on variants of the agobot, DSNXbot, evilbot, G-SySbot, sdbot, and Spybot botnets . The results suggest that the presence of network packet contents in selected system call arguments is an effective indicator for malicious Win32 bots.

Liu *et al.* [111] suggested that a typical botnet-infected host goes through three phases: startup, preparation and attack. In the startup phase, the bot program is launched without any user input. In the preparation phase, the bot makes a connection to the botmaster through C&C channels. Finally, the bot executes local or remote attacks. With such an assumption, BotTracer is proposed to track and detect these phases in order to determine whether a host is infected. To detect these phases, the system takes advantage of a virtual machine with the same image as the host and the known characteristics of the botnet communication channels and malicious behaviours. Evaluations on several bots such as Nugache, Graybird and variants of IRC bots like Agobot4 and Forbot showed that BotTracer is an effective detection system.

Szymczyk proposed a multi-agent botnet detection system (MABDS) which is made up of a user agent, an administrative agent, a central database of knowledge and collections of system analysis, network analysis, and honeypot agents [141]. These agents work collectively to analyze the events occurring in the operating system and network environment to detect bots. The database of knowledge holds the signatures derived from the analysis of the various types of malicious software that create bots. Szymczyk claimed that the evaluation confirmed the effectiveness of MABDS in detecting bots but no evaluation results were provided.

DeWare is a host-based malware detection system proposed by Xu *et al.* [148]. The system identifies dependencies between the system events and the corresponding user action that initiates them. As well, It uses policies to control how file systems are accessed in a host and how processes are created. In regards to botnet detection, DeWare can be used in detecting drive-by-download exploits used by botnets such as Torpig. Evaluating the system, it generated less than a 1% false positive rate separating legitimate user downloads from malicious downloads.

Moreover, there are several host-based intrusion detection systems proposed in the literature which are not specifically designed for botnets or evaluated on botnets such as [65, 120].

2.3.2 Network-based detection approaches

Network packets include two main parts: (i) the packet header, which includes the control information of the protocols used on the network, and (ii) the packet payload, which includes the application information used on the network. Some systems/approaches require both the payload and the header sections of the packets to extract the necessary analytic features while others only need the header of the packets (*e.g.* flow-based systems). On the other hand, some systems focus on per-packet analysis while others use aggregated packets of connections (*i.e.* flows).

Gu *et al.* proposed and developed two botnet detection frameworks called BotHunter and BotMiner[86, 85]. BotHunter is a botnet detection system based on Snort IDS alerts and bot activities correlation. Basically, a combination of Snort and a network data clustering approach forms the basis of this framework. The tool is designed based on the assumption that all botnet infection processes are similar and can be illustrated by a lifecycle model explained in the previous section. On the other hand, BotMiner is a botnet detection framework based on group behaviour analysis [85]. BotMiner uses a clustering approach to find similar C&C communication behaviours, which form clusters, and then employs Snort to find the type of activity in the detected clusters. Flow features such as the number of packets per flow, the average number of bytes per packet and the average number of bytes per second are extracted and employed for the detection. BotMiner was evaluated on several traffic data sets. These data sets included their campus network traffic, which represented the normal behaviour. Moreover, it included the Honeynet traffic and traffic captured by running bot binaries in a sandbox environment. Their results showed that BotMiner could detect botnets with detection rates between 75% and 100% on different types of botnets.

Strayer *et al.* developed an IRC botnet detection system that made use of machine learning techniques (classification and clustering) [139]. First, a classification technique is used to filter the chat type of traffic and then a clustering technique is applied to find the group activities in the filtered traffic. Finally, a topology analyzer is utilized on the clusters to detect the botnets. In this three layer approach, they employed flow-based features extracted from packet headers. Data employed in this work was gathered from a controlled testbed running bot binaries. They employed and evaluated Naive-Bayes, C4.5 and Bayesian Networks as the classifiers against a multi-Dimensional flow correlation technique that was designed and proposed.

Wurzinger *et al.* proposed an approach to detect botnets based on the correlation of commands and responses in the monitored network traces [147]. The two base assumptions in designing the approach were: bots receive commands from botmasters and then carry out actions in response to those commands. To identify traffic responses, the corresponding commands in the preceding traffic are located. Known content-based specifications (*i.e.* signatures) were utilized to identify the commands and responses. Then, using the command and response pairs, the detection model was built focussing on IRC, HTTP and P2P botnets. Data sets used in this work were collected by running bot binaries in a controlled environment. Traffic features such as the number of non-ASCII bytes in the payload were analyzed to characterize bot behaviour.

Zeidanloo *et al.* proposed a detection framework with a focus on P2P and IRC based botnets [151]. The framework is based on finding similar communication patterns and behaviours among the group of hosts that are performing at least one malicious activity. In the first phase, filters are applied to the traffic to filter out the irrelevant traffic flows in order to reduce the workload. By investigating the content of packets IRC, HTTP and P2P traffic is separated and forwarded to the malicious activity detector module in the second phase. Finally, a traffic monitoring module is used to detect the group of hosts that have similar behaviours. To analyze the traffic over the three phases, a flow-based approach was utilized while payload inspection was employed for traffic filtering.

Celik *et al.* proposed a flow-based botnet C&C activity detection system [61]. Specifically, they investigated the effect of the calibration of time-based flow features. In other words, they explored to what extent the presence of timing artifacts in botnet traces will affect the results of timing features-based classifiers. They employed machine learning algorithms such as C4.5, Naive-Bayes and logistic regression. For the evaluation, Lawrence Berkeley National Laboratory (LBNL) traces represents the normal traffic and IRC-based simulated botnet traffic represents the attack traffic. The results of four different time calibration scenarios showed some performance changes in the Naive-Bayes and Logistic Model Tree (LMT) overall accuracy but no notable change in the logistic regression and C4.5 classifiers. The best performing classifier in this work was C4.5 with an overall accuracy of 99%.

Francois *et al.* proposed a NetFlow monitoring framework to detect P2P botnets leveraging a simple host dependency model to track communication patterns. The system first creates a dependency graph between hosts by monitoring their interactions. Then, linkage and clustering algorithms are employed to identify similar botnet traffic patterns (*i.e.* identifying hosts that are highly linked together) [71]. The traffic data for evaluation was obtained from an Internet operator company in addition to synthetically-generated botnet flow records. In addition to the traffic flows, the Google web search engine was employed for linkage analysis. The results showed a TPR of up to 96%

Wang *et al.* proposed a fuzzy pattern recognition approach (called BBDP) to detect HTTP and IRC botnets behavioural patterns [145]. The traffic traces were first passed through a filtering process to narrow the search scope. The system analyzed the features of DNS queries (such as the number of failed DNS responses) and TCP flows of the filtered traffic to detect botnet malicious domain names and IP addresses. To evaluate the proposed approach, malicious bot binaries where collected using Honeytrap and were run in a controlled environment to generate bot traffic. The results showed up to a 95% detection rate.

Lu *et al.* proposed a two-phase botnet detection system [113]. In the first phase the system classifies the network traffic using traffic payload signatures and then a decision tree model is used in the second phase to classify the unknown traffic by the payload content of the first phase. Basically, the second phase takes advantage of clustering based on n-gram features selected and extracted from the content of the network flows. To evaluate the system, traffic traces had been collected on three networks including an indoor testbed network, a honeynet through a public Internet connection and a public network (a free wireless fidelity (WiFi) network).

Kirubavathi *et al.* designed an HTTP-based botnet detection system using a multilayer Feed-Forward Neural network [106]. Given that HTTP-based botnets do not maintain a connection with the C&C server but periodically make a request to the C&C server (over the HTTP) to download the instructions, the system extracts features related to TCP connections in specific time intervals based on the packet headers. The features are then normalized and used to train a multilayer feed-forward neural network. To collect data for evaluation purposes, botnets were simulated in the lab. The system showed a higher performance (99% accuracy) compared to other ML algorithms (such as decision tree and random forest) using the same feature set.

Guerid *et al.* proposed a collaborative and inter-domain botnet detection system which enables real-time analysis for large scale networks [87]. The system has several connected probes in which each probe correlates the information gathered from its network and the anonymized data received from other probes. The probe then uses two layers of analysis: (i) a community structure layer and (ii) a C&C server detection layer. The first layer detects the infected bots and groups them into communities with the same botnet behaviour. This detection is based on participation in the same ongoing attacks or in the similarity of their abnormal network traffic. The second layer receives the communities from the first layer and identifies the associated malicious servers. Finally, the system is evaluated on domain-flux botnets such as Conficker, Pushdo and Zeus. Given the known behaviour of such botnets, the first layer analyzes the behaviour of DNS requests and replies. The results demonstrated that collaboration architecture improves the detection rate.

Zhao *et al.* investigated a botnet detection system based on flow intervals [156]. Flow features of traffic packets were extracted based on several time intervals and utilized by several ML algorithms to find the best combination of flow features, time interval and ML algorithm. Accordingly, decision tree classifier was finally selected as the preferred classifier to detect botnets with a proposed feature set using a time interval of 180 seconds. The authors focussed on P2P botnets (such as Waledac) which employ the HTTP protocol and a fast-flux based DNS technique. Furthermore, based on the proposed approach, a web-based detection system was implemented to be used both in offline detection as well as live detection. To evaluate their proposed approach and web-based detection system, they employed a combination of normal and attack traffic some of which was generated in the lab, some was from Honeynet project traces and some was from the Lawrence Berkeley National Laboratory (normal traffic) data sets. Although their proposed detection approach resulted in up to 99% detection rates with a false positive rate of around 2%, they also tested their system with unseen botnet data and obtained detection rates of up to 100% while having a false positive rate of about 80% in some cases.

Dietrich *et al.* presented an approach (CoCoSpot) to detect botnet C&C channels based on traffic analysis features [64]. In this approach, flow features of TCP and UDP traffic are extracted from the packet header and payload. A hierarchical clustering technique is applied to the flows to form the clusters which are then labeled manually by the authors as malicious or normal. Using the clusters and cluster centroids, a classifier is trained to be used for classifying the unknown test traffic. The result indicated that CoCoSpot can detect more than 88% of the C&C flows at a false positive rate below 0.1%.

Beigi *et al.* investigated the effectiveness of flow-based feature sets employed in previous botnet detection studies and evaluated them using a proposed feature selection algorithm [54]. The results indicated that the byte-based group of features has less effect while the packet-based group has more impact. In the evaluation, IRC, HTTP and P2P botnet data sets were utilized. Beigi *et al.* obtained DRs between 75% and 99% on different combinations of data sets including ISOT (Uvic), Virut (CVUT), ZeroAccess (CVUT) and NSIS (CVUT) [54].

Yan *et al.* proposed PeerClean as a three layer peer-to-peer botnet detection system which investigates the traffic flow statistics and network connection patterns for this purpose [150]. The first layer clusters the hosts based on similar traffic patterns using the flow features. This layer was determined not robust enough to distinguish botnet behaviour from benign behaviour given the dynamic Internet traffic. Hence, the second layer applies a dynamic group behaviour analysis (DGBA) on the first layer clusters to extract the group-level aggregated connection features. An SVM classifier is then used to reveal the botnet clusters. Finally the third layer identifies the botnet types of the detected botnet clusters. Data collected on the edge router of a campus network that consists of traffic from known p2p applications, and botnet data collected from a controlled environment running botnet samples are employed in this work to evaluate the proposed system. The results showed TPRs between 95% and 100%.

Stevanovic *et al.* proposed three traffic analysis methods to detect botnet behaviours using the TCP, UPD and DNS protocol as the main carrier of botnet C&C communication [136]. After extracting features for each of these protocols, a random forest classifier is utilized to generate the detection models. For the TCP and UDP-based methods conversation features (flow features) are extracted from the packet headers without using the IP addresses while for the DNS-based method, features are extracted from the DNS queries and responses. The novel idea of extracting the conversation features is to calculate the features based on a sub-sample of the packets in each conversation. Forty botnet sample data sets were used for the evaluation of the proposed methods. The TCP classifier performed with up to 98% precision by using only ten packets per conversation while the UDP classifier was less sensitive to the number of packets analyzed and resulted in up to 99% precision. Finally, the DNS classifier showed an overall precision of 98.9%.

Wang *et al.* proposed a two-stage approach to detect botnets [144]. The first stage detects network anomalies which are linked to the botnet presence in the network while the second stage detects the infected machines (bots) based on these anomalies. Two anomaly detection methods were used for the first stage: (i) a flow-level anomaly detection method which quantifies the flow data and monitors the histograms of data and (ii) a graph-based packet-level anomaly detection method which aggregates the packet-level data into graphs and then monitors the distribution degree. In the second stage, highly interactive nodes and their relationships are identified by constructing a Social Correlation Graph. This type of analysis is based on the assumption that the botmaster and bots are highly interactive nodes and the interactions correspond to C&C communication. The proposed approach was evaluated on the CTU-13 and CAIDA 2007 DDoS data sets and compared against BotHunter and showed promising results.

To identify specifically DNS-based botnets which use DNS for either locating their C&C server or as the information carrier, many works employ DNS network traffic analysis and/or use domain name textual characteristics [100, 122, 116, 135].

Holz *et al.* implemented a heuristics-based malicious fast-flux service network (FFSN) detection system based on recursive DNS traffic analysis [100]. Fluxiness and Flux-Score measures are introduced into the system which is based on features such as Time-to-Live (TTL), multiple Address records (A records) and the number of unique Autonomous Systems. The system performed well in detecting Fast-Flux traffic. However, an enhanced version of this system was introduced [121] in which ten features (similar to the previous effort [100]) and their corresponding boundaries were introduced and analyzed to measure the Fluxiness of DNS traffic. Some of these features are a Time-to-Live (TTL) shorter than 900 seconds, more than five unique IP addresses in the Address record (A record) queries and more than two distinct Autonomous Systems (ASNs) highlighted by the A record. If more than four of these features are flagged, the traffic is marked as 'fluxy'. Moreover, a domain whitelist is then checked to ensure that legitimate operations using fast-flux techniques for load balancing are not falsely detected. The system identified over 900 fast-flux domain names from early- to mid-2008 and monitored their behaviour across the Internet.

Perdisci *et al.* developed a FFSN detection system by passively analyzing the recursive DNS traces [122]. The same features introduced previously [100] such as a short TTL and a set of resolved IP addresses returned at each query were also identified in this work. By collecting the recursive DNS traces passively from multiple large networks, filtering the traces to gather the suspicious traffic and clustering based on the related domain names, a system was built to identify the fast-flux domain names.

Manasrab *et al.* proposed a botnet detection framework (a.k.a BDM) based on the fact that bots query the DNS server as a group of hosts periodically [116]. Monitoring and capturing the DNS traffic at different time intervals and using the Jaccard index to measure the similarity, the system was developed to detect the malicious DNS behaviours of botnets. The results of the evaluation on the NAv6 network showed an average DR of 89% and average FPR and FNR of approximately 11%.

Ma *et al.* employed supervised learning techniques (*i.e.* Naive-Bayes, SVM) to detect malicious web sites from suspicious URLs [105]. To characterize the URLs, two categories of features are used: lexical features (the length of a domain name, the number of dots, *etc.*) and host-based features (IP address properties, WHOIS properties, *etc.*). For normal URLs, DMOZ Open Directory Project [16] and random URL selector for Yahoos directory[1]. On the other hand, PhishTank [18] and Spamscatter [52] are used to collect malicious URLs. The evaluation resulted in a performance accuracy of 95% to 99%.

Antonakakis *et al.* presented a dynamic reputation system, Notos [53]. Notos

[1]http://random.yahoo.com/bin/ryl

builds models of legitimate and malicious domain names using DNS query information as well as the zone-based (*e.g.* the occurrence frequency of different characters) and network-based (*e.g.* the total number of IPs associated historically with a domain) features of the domain names. Models were built in an off-line mode using a clustering approach and then reputation scores were assigned to label the online mode data as malicious or legitimate. Data from two ISP-based sensors were collected and used for evaluation in this work. The evaluation indicated a 96.8% TPR and 0.38% FPR.

Dietrich *et al.* presented a technique that distinguishes between DNS-based C&C and real-world normal DNS communication DNS traffic [62]. They discovered a malware family, Feederbot, which uses DNS sub-domain labels and DNS response packet payloads to exfiltrate and infiltrate malicious data. The detection technique extracts several features from the DNS response payload data such as the number of ASCII digits in a DNS message and utilizes k-Means clustering and a Euclidean Distance-based classifier to detect the C&C channels. The result showed the system was able to correctly classify more than 14 million DNS transactions of 42,143 malware samples.

Stalmans *et al.* developed a system to detect fast-flux domain names using DNS queries [135]. Analyzing the DNS query responses, two groups of features were extracted to identify legitimate and malicious queries: DNS features (*i.e.* unique Autonomous System Numbers (ASNs) and TTL) and Textual features (*i.e.* alpha numeric character frequency distributions). Given the extracted features, C5.0 and Bayesian classifiers were employed to identify fast-flux queries. The legitimate domain data was obtained from the Google Doubleclick ad planner top-1000 most visited sites list and malicious data was collected from multiple sources, including the fast-flux trackers for Zeus, SpyEye and other botnets. The system could detect the malicious DNS queries with 82% to 87% accuracy.

Yadav *et al.* proposed a methodology to detect malicious algorithmically-generated domain names, addressing the domain fluxing mechanism [149]. To this end they used several methods and features to group the DNS queries. Then for each group, metrics which characterized the distribution of alphanumeric characters– such as the Kullback-Leibler divergence and the Levenshtein edit distance– as well as bigrams in all domains which are mapped to the same set of IP-addresses were computed and used to identify domain names with structural anomalies. Data obtained from a Tier-1 ISP in Asia, a crawling IPv4 address space with the corresponding domains and campus DNS traces are utilized in this work. The performance of the system varies based on the distance metric and the characteristics of data such as the number of domains per TLD and the type of botnet. In this case, the system performed with an accuracy of up to 100%.

Futai *et al.* proposed an approach for detecting fast-flux domain names by processing the recursive DNS server traffic in real time in addition to a long-term monitoring phase [73]. Using the extracted features from the DNS response packets, a decision tree-based classifier is created to detect the fast-flux domain names. Next, a pre-defined heuristic rule set is used to label the domain names as malicious. The rule set reflects botnet behaviours on the extracted features of the previous phase (*e.g.* applying thresholds on the features) as well as defining additional rules like specifying commonly-used Top Level Domains (TLDs) by botnets. Finally, the long term monitoring phase monitors the domain names flagged as suspicious in the previous phase for 48 hours to make the final decision. The evaluation of 180 days of deployment on the university's DNS servers showed a higher detection rate compared to the flux-score-based algorithms proposed by Holz *et al.* [100].

Schiavoni *et al.* proposed Phoenix for classifying DGA- and non-DGA-generated domains apart using a combination of lexical (*e.g.* meaningful character ratio and n-gram normality score) and IP-based features [129]. In the first phase, lexical features are extracted and analyzed by statistical measures and thresholds. In the second phase, IP-based features and a DBSCAN clustering algorithm are used to group the domains that are resolved into similar sets of IP addresses. The cluster features are analyzed to characterize the DGAs and their respective botnets. The system could distinguish the automatically-generated domain names from the non-automatic ones with 94.8% accuracy on the Conficker A, B and C, Torpig and Bamital botnet domain names.

Bilge *et al.* proposed a system called Exposure to detect malicious botnet domain names in real time, by applying 15 complex features grouped in four categories: time-based features (*e.g.* Short life), dns-answer-based features (*e.g.* Number of domains sharing the IP with), TTL-based features (e.g Number of TTL changes), and domain-name-based features (*e.g.* percentage of numerical characters) [56]. These complex features are formed by 26 atomic features. To train the initial classifier model, the C4.5 classifier is employed. The benign data set consists of domain names from Alexa and a number of servers that provide detailed WHOIS data while reported malicious domain names in malwaredomains.com, Phishtank and Anubis are employed to represent malicious data samples. The initial model could perform with a 99% accuracy. Exposure was deployed in a real-time network for seventeen months with an initial model while the model was retrained every day. The results showed the system could perform well in a real-time network as well.

2.4 Summary

In short, botnets use different protocols such as HTTP and DNS for communication. Two main categories of botnet detection systems are proposed in the literature: host-based and network-based systems.

One advantage of host-based detection approaches is that they are very effective in drive-by-download attacks. However, these approaches are complex, costly, non-scalable and cannot detect the group-based behaviour of botnets. In this case, network-based detection approaches are more effective. This is why most recent botnet detection approaches proposed in the literature are based on network data analysis. For the same reason, the early warning system in this thesis is not a host-based system.

Network-based detection systems can be categorized in different ways: (i) some use group behaviour analysis [85, 139, 87] while others are focussed on single user behaviour and not on finding behaviours that are shared between a group of users. Using group behaviour analysis limits the applicability of the detection approach to the large networks which are being monitored. In other words, if the network under investigation/monitoring does not include multiple infected bots, group-based monitoring approaches might not be very useful. However, the approaches that are not focussed on group behaviour analysis can be effective regardless of the number of infected hosts on the network. This is why the early warning system in this thesis is not a group behaviour analysis-based system. (ii) From the data perspective, some of the systems use the packet payload and/or header information [145, 147], while others employ only the packet header information (*e.g.* flow-based systems) [154, 156, 64]. The importance of the approaches in the second group can be understood better knowing that the most recent aggressive botnets employ encryption to hide themselves and their information from the detection systems. Moreover, there are several studies on flow-based botnet detection systems in which each has proposed a different set of features [154, 156]. Some studies have analyzed the feature selection algorithms to extract the most effective feature sets [54]. Such feature selection processes can cause the models to be focussed on specific type(s) of botnet(s) which may not be very effective for other types. This is why this research has proposed two complimentary approaches to cover all forms of data even encrypted. (iii) While some of the approaches are focussed on botnets with specific communication structures or protocols [71, 106, 150], there are approaches which can be applied to various botnet structures and protocols [85, 153, 144]. (iv) Most of the DNS analysis-based systems employ some sort of DNS traffic analysis with/without domain name lexical analysis [100, 149, 73].

In summary, a pure domain name analysis-based system is proposed in this research. The idea is that DNS traffic analysis can be covered in a general network data analysis approach. Two methods are suggested and evaluated for this purpose. The first one is a lexical domain name analysis which extracts features from domain names and then uses ML algorithms to detect the malicious domain names. The second approach uses the domain name characters as the features and employs a ML evolutionary technique to classify the domain names. It appears that the lexical domain analysis approach has the most comprehensive feature set compared to similar works in the literature. Most importantly, no work in the literature has investigated only using the characters of the domain names (without any other network-based analysis) to classify legitimated domain names from malicious ones. By contrast, a flow-based network analysis system is proposed. The approach does not use group behaviour analysis, but packet header information which makes the approach effective for botnets which use encryption. Moreover, to detect botnets with various structures and communication protocols, a wide range of flow features are introduced. By using an ML algorithm which has the ability to perform attribute selection as an implicit property of constructing, it should be possible to develop a better way to approach feature selection while utilizing all the possible extracted flow features. Unlike most of the works in the literature, the proposed system in this thesis is evaluated on a wide range of botnets data sets, from centralized IRC botnets to decentralized P2P botnets.

Chapter 3

Learning Algorithms and Tools

In this thesis, eight well-known machine learning algorithms are employed: C4.5, SBB, SVM, ANN, KNN, Bayesian Networks, Adaboost and Naive Bayes. These algorithms have been selected because they have been widely used in the literature ([139, 61, 156, 153, 134]) for network traffic analysis. The use of machine learning algorithms requires that network traffic be represented in ways these algorithms can work on them. Hence, five flow exporters are employed in this research to extract different feature sets to represent traffic, namely Maji [10], YAF [24], Softflowd [21], Tranalyzer [22] and Netmate [13]. YAF was introduced by the known Network Situational Awareness (NetSA) group at CERT from Carnegie Mellon University which has also developed other known open source tools for monitoring large-scale networks, such as SiLK. Maji was introduced by the WAND Network Research Group from Waikato University which is known for its Weka machine learning framework. Tranalyzer and Netmate are widely used for network traffic analysis in the literature [70, 51, 84, 49, 132]. All of these flow exporters support extended and custom netflow features while Softflowd performs as a basic netflow exporter. Hence, using Softflowd as a base for evaluation would be very useful. Finally, two publicly available malicious behaviour detection systems, Snort and BotHunter, were used for evaluation purposes.

3.1 Machine learning algorithms

The machine learning algorithms employed throughout the different chapters of this thesis are described bellow.

3.1.1 C4.5

C4.5 is a decision tree algorithm which is an extension of the earlier ID3 algorithm developed by Quinlan [50]. This supervised learning method aims to find the small decision trees (using pruning) and then generate an if-then rule set based of the trained tree which can be used for classification.

Applying the Information Entropy concept, C4.5 constructs the decision trees based on a training data set. Each record of the set has the same structure consisting of a number of attributes in which one of them represents the class of the record. The algorithm employs a normalized information gain criterion to select attributes from a given set of attributes to determine the splitting point. In other words, the feature with the highest information gain value is chosen as the splitting point.

Let p_i be the probability that an arbitrary sample in data set D belongs to class C_i:

$$p_i = \frac{|C_{i,D}|}{|D|} \tag{3.1}$$

Then, the amount of information (entropy) required to classify an instance in D, where m is the number of unique instances of the data set:

$$\mathit{Entropy}(D) = -\sum_{i=1}^{m} p_i \log_2 p_i \tag{3.2}$$

Expected information needed to classify the objects of the data set D in all v sub-trees (after using attribute A to split D into v partitions) is:

$$\mathit{Entropy}_A D = -\sum_{j=1}^{v} \frac{|D_j|}{D} \times \mathit{Entropy}(D_j) \tag{3.3}$$

and finally, information gained by branching on attribute A is:

$$\mathit{Gain}(A) = \mathit{Entropy}(D) - \mathit{Entropy}_A(D) \tag{3.4}$$

A decision node is created based on the selected splitting node with the highest information gain. The same procedure applies recursively to the corresponding sub-lists obtained by the splitting process until all of the data samples associated with the leaf nodes are of the same class or the classifier runs out of training samples. More detailed information on the C4.5 learning algorithm can be found in [50].

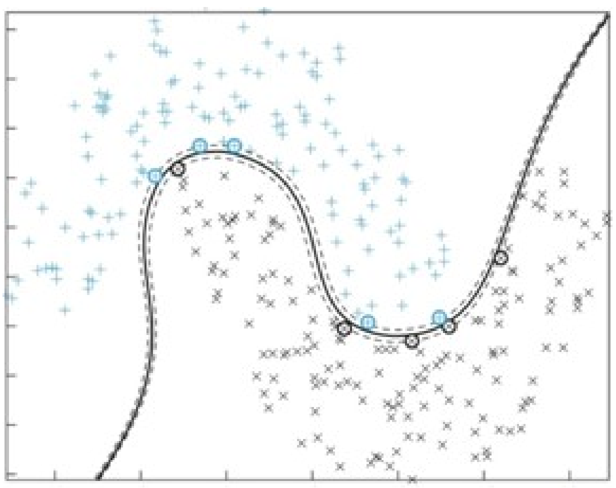


Figure 3.1: SVM hyperplane and support vectors [99]

3.1.2 Support Vector Machine (SVM)

SVM is a binary classification algorithm that can be used for classification and rule regression. The goal of this classification algorithm is to build an N-dimensional hyperplane that separates the samples of data optimally into two classes with maximal margin. The data points that form the hyperplane are called support vectors which are shown with cycles in Fig. 3.1. The classifier can be extended easily to a K-class classification by constructing k two-class (binary) classifiers.

In order to use an SVM to solve a classification problem on non-linearly separable data, a non-linear mapping of input data into a high dimensional feature space is required. Then, an optimal hyperplane for separating the high dimensional features of input data can be constructed which maximizes the separation margin. Finally, a linear mapping from the feature space to the output space is required. Generally, mapping the non-linearly separable data explicitly into a high dimension feature space has a very high computational cost. Therefore, to handle the problem, kernel functions are utilized to map the data points implicitly to the feature space. Mathematically, for any mapping $\phi : D \to F$ the function $K : K(x_1, x_2) = (\phi(x_1), \phi(x_2))$ is a kernel function where $(.,.)$ denotes the dot product. Several kernels have been introduced so far such as Polynomial, Gaussian and Hyperbolic tangent.

3.1.2.1 SVM with string kernel (SSK)

Lodhi *et al.* proposed a string kernel, called SSK (String Subsequence Kernel), for the purpose of text classification [112]. The idea is to define the dot product of two text data points by means of their sub-strings. The more sub-strings they have in common, the more similar they are. Note that the sub-strings do not need to be contiguous. However, the subsequences are weighted by an exponentially decaying factor of their length, emphasizing on the sub-strings that are close to contiguous. In other words, the contiguity degree of a sub-string determines its effectiveness (weight) in the comparison.

The main parameters of SSK are n (subsequence length), and λ (decay factor). The kernel maps the input string data into a feature space F implicitly for every subsequence u of n characters using the following formula:

$$\phi_u(s) = \sum_{i:\ u=s[i]} \lambda^{l(i)} \tag{3.5}$$

$\phi_u(i)$ is the overall result for all the occurrences of u in a string s. λ^l is the value of each occurrence of u where l denotes the length of that subsequence in s (length of u plus interior gaps of the occurrence). The Kernel function result for two strings s and t is the dot product of their features mapping:

$$K_n(s,t) = \sum_{u\in\sum^n} (\phi_u(s).\phi_u(t)) = \sum_{u\in\sum^n} \lambda^{l(i)} \sum_{i:\ u=s[i]} \sum_{j:\ u=s[j]} \lambda^{l(j)} = \sum_{u\in\sum^n} \sum_{i:\ u=s[i]} \sum_{j:\ u=s[j]} \lambda^{l(i)+l(j)} \tag{3.6}$$

Given that even for a small substring size (*i.e.* four) and an average-sized text, the direct computation of feature space is impractical. Thus, SSK utilizes an efficient recursive formulation using dynamic programming techniques with the complexity of $O(n|s||t|)$.

3.1.2.1.1 SSK- Lambda Pruning (SSK-LP)

Seewald *et al.* introduced SSK-LP to decrease SSK computational time and memory consumption along with a little loss in accuracy. In SSL-LP, the recursion is stopped as soon as it gets to an acceptable result in the current branch. This approach decreases the computational effort required for the SKK. A detailed explanation of the algorithm can be found in [130].

3.1.3 Artificial Neural Networks (ANN)

The ANN algorithm is inspired by the structure of the biological neural network. It is composed of neurons and the connections between them. Backpropagation network is the most frequently used version of this algorithm organized in layers which consists of an input layer, an output layer and at least one hidden layer. All the neurons of each layer (except the output layer) are connected to all the neurons of the next layer by an axon associated with a weight factor where they signal forward, and then the errors are propagated backwards. The supervised backpropagation algorithm uses the inputs and outputs to compute the error (the difference between the actual and the expected results). Starting with random weights, each training iteration adjusts the neurons' weights in order to minimize the error rate. The training process stops after reaching the maximum epoch (time step) or meeting a specific stopping criterion (*e.g.* error rate increases). Detailed information on ANN can be found in [50].

3.1.4 Naive Bayes

A Naive-Bayes classifier is a simple probabilistic classifier based on the Bayes theorem, which assumes that the presence of an attribute in a given class is independent of other attributes. The classifier uses the method of maximum likelihood (probability) for parameter estimation. Given a training set (X, Y) where for each sample (x, y), x is an n-dimensional vector and y is the class label out of k number of classes, $C_1, C_2 ... C_k$, the classifier predicts that the sample belongs to the class Ci having the highest posteriori, conditioned on x:

$$P(C_i \mid x) > P(C_j \mid x) for 1 \leq j \leq k, j \neq i$$

Where:

$$P(C_i) = \frac{P(C_i)P(x|C_i)}{P(x)}$$

Which equals to

$$Posterior = \frac{Priori*likelihood}{evidence}$$

in plain English. All classifier parameters (*i.e.* class priori) can be calculated using different assumptions (*i.e.* $prioris = \frac{1}{k}$ where k is the number of classes). A more detailed explanation of the algorithm can be found in [50].

3.1.5 Bayesian Networks

Given a set X of discrete attributes, Bayesian Networks are graphical representations for probabilistic relationships among the variables of the set. In other words, a Bayesian network is a directed acyclic graph which represents the probability distribution over X. The graph nodes that are associated with the attributes, are connected through the links that correspond to the direct influence from one attribute to the other. Given the Bayesian network structure (with nodes and direct influence links), the conditional probability distribution of the graph is then computed. The learning process aims to find a Bayesian Network structure that describes the training data in the best possible way. To this end, two categories of approaches are proposed: score and search-based and greedy-based approach. A more detailed explanation of the algorithm can be found in [50].

3.1.6 Adaboost

Machine learning techniques' goal is to generate a rule which can predict the new test samples with a high accuracy. Creating a highly accurate rule is a difficult task but on the other hand, generating a set of rough rules of thumb with moderate accuracy is not that hard. Based on this observation, the boosting method starts with finding the rules of thumb called weak learner. Given the training set, AdaBoost, an acronym for Adaptive Boosting, calls the weak learning algorithm repeatedly, each time feeding it with a different distribution over the training data to build a complex classifier.

Given the training examples (x_i, y_i) where x_i is the feature vector from domain X, and $y_i \in 1, +1$ is the label, in each round t, a given weak learning algorithm is used to find a weak hypothesis $h_t : X \rightarrow -1, +1$ using a distribution of the training data (D_i). The goal here is to find a weak hypothesis with a low weighted error over the

D_i. Finally, the combined/final hypothesis is created based on the sign of a weighted combination of weak hypotheses:

$$H(X) = Sign(\sum_{t^T}(\alpha_i * h_t(X))) \tag{3.7}$$

This is equivalent to saying that H is computed as a weighted majority vote of the weak hypotheses h_t where each is assigned weight α_t.

3.1.7 K Nearest Neighbour (KNN)

KNN is a non-parametric lazy learning algorithm. This means it does not make any assumption about data distribution and does not use the training data samples to do any generalization. Hence, this classifier has no explicit training phase and requires the keeping of all training data for classification. Defining a training data sample with a set of vectors and a class label, the classifier basically tries to find the K nearest neighbour and perform a majority voting for the testing phase. The Euclidean distance is the most commonly used distance measurement to find the nearest neighbours in this classifier. A more detailed explanation of the KNN algorithm can be found in [50].

3.1.8 SBB

The Symbiotic Bid-Based (SBB) algorithm is a form of genetic programming (GP). Although this is an open source algorithm, it is not one of the routine highly-employed machine learning algorithms in the literature. SBB will be explained in more details as this algorithm is used as basis for the proposed application data-based early warning system design.

This form of genetic programming algorithm builds teams of programs co-operatively while simultaneously identifying useful exemplars to learn from [67]. To do so, three distinct populations are utilized: a point population, a team population and a learner population. The learner population represents a set of symbionts (learners) which associate a GP-bidding behaviour with an action. The team population identifies subsets of learners to define team membership under a variable length representation; the implication of the latter is that team size as well as composition evolves. Finally, the point population denotes a subset of training data exemplars.

Host evaluation takes the following form. Each symbiont (s_j) consists of a program, $s_j.p$, and an action (class), $s_j.a \in \{1, ..., C\}$ where C denotes the maximum number of classes. All symbionts are a member of the target host h, and have their program evaluated on the *same* training exemplar, x_k, (from the point population). The symbiont with the maximum output is identified or $sym^* = \arg_{sym_j \in h} \max(s_j.p(x_k))$. It is this symbiont which has 'won' the right to present its corresponding action, $sym^*.a$ as the class label on exemplar x_k. Any form of GP could be assumed for symbiont programs. In this thesis, a linear GP representation is employed [59]. Each program consists of a sequence of binary instructions representing one- and two-operand operations with function set $\{cos, exp, log, +, *, -, \div, \%\}$. To standardize the bid values between zero and one, the sigmoid function $f(y) = (1 + exp^{-y})^{-1}$ is applied to the real valued program output y.

Fitness evaluation is only conducted against the current content of the point population, thus decoupling fitness evaluation from the cardinality of the entire training partition. The interaction between point and team population takes the form of Pareto archiving [66]. Thus, if an individual is not dominated by any other individual, it is set to be a part of Pareto-front. This relation is used by the SBB training algorithm to determine the points and the teams that survive to the next generation.

At each generation, $Pgap$ new points are generated by sampling the training data while enforcing a heuristic to ensure all classes see equal representation in the point population. Conversely, $Hgap$ new teams are generated through variation operators (add, delete, swap and mutate) as applied to the existing teams. Potentially, new symbiont programs appear through mutation alone, resulting in a variable size symbiont population. That is to say, there is no symbiont 'fitness' as such, however, should a symbiont not see any host index, it is deleted. After fitness evaluation the point and team population content is deterministically ranked with $Pgap$ points and $Hgap$ teams deleted before a new generation commences.

The above ranking process utilizes Pareto archiving, thus the Pareto non-dominated teams with the highest ranks are selected. Likewise, the non-dominated points are also preserved. Meanwhile, if a point/team ranking is required in these non-dominated subsets, a form of competitive fitness sharing is employed in order to introduce a bias

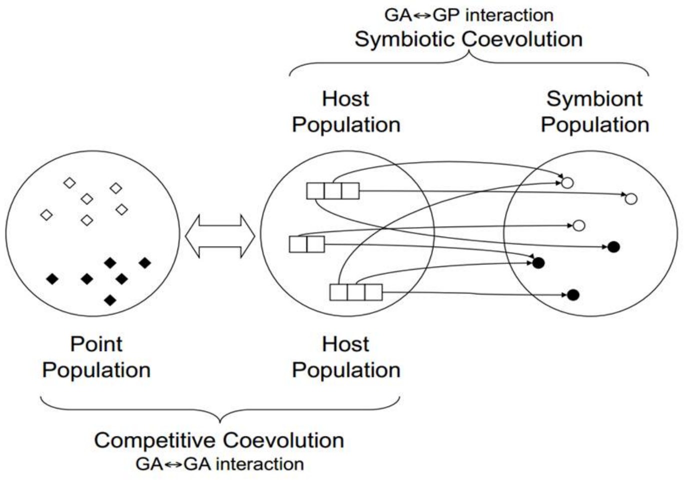


Figure 3.2: SBB team-based mechanism [110]

in favour of the points/teams that exhibit non-overlapping behaviour. Algorithm 1 describes the SBB training algorithm. A more detailed explanation of the algorithm can be found in [67].

3.1.9 Performance criteria

In data classification, two metrics are typically used in order to quantify the performance of the classifier: Detection Rate (DR) and False Positive Rate (FPR). In this thesis, DR reflects the number of correctly classified specific botnet exemplars in a given traffic file/domain name list and is calculated using $DR = \frac{TP}{TP+FN}$; whereas TP (True Positive) is the number of botnet exemplars that are classified correctly and FN (False Negative) is the number of botnet exemplars that are classified incorrectly (as normal). FPR, on the other hand, shows the number of normal exemplars that are classified incorrectly as botnets and is calculated using $TPR = \frac{FP}{FP+TN}$; whereas TN (True Negative) is the number of normal exemplars that are classified correctly.

Although typically, classifiers are evaluated using the DR, given an unbalanced data set or a multi-class data set, this metric can be misleading. In this regard, a classwise detection rate is defined as: $DET_c = \frac{TP_c}{FN_c+TP_c}$; where $DETc$ is the class c detection rate and TPc and FNc are the True-Positive and False-Negative counts for class c. Finally, to summarize the classwise detection rates of a classifier over all

Algorithm 1 Overview of SBB traning algorithm

```
1: procedure
2: $t = 0$
3: $P^t = INITPOINTS(P_{size})$
4: $(M^t, L^t) = INITTEAMS(M_{size})$
5: while $t \leq t_{max}$ do
6:     $P^t = GENPOINTS(P^t)$
7:     $(M^t) = GENTEAMS(M^t, L^t)$
8:     for all $m_i \subseteq M^t$ do
9:         for all $P_k \subseteq P^t$ do
10:            $EVALUATE(m_i, p_k)$
11:        end for
12:    end for
13:    $P^{t+1} = SELPOINTS(P^T)$
14:    $(M^{t+1}, L^{t+1}) = SELTEAMS(M^t, L^t)$
15:    $t = t + 1$
16: end while
17: return $BEST(M^t)$
18: end procedure
```

classes [110], the average DR criteria is defined by:

$$Score = \frac{1}{|C|} \sum_{c \in C} DET_c \tag{3.8}$$

Moreover, classifier complexity can be measured by different criteria such as memory consumption and the training time of the learned model by the learning algorithms. In this work, three complexity criteria are utilized.

1) Training (computation) time is employed where this is estimated on a common computing platform.

2) Solution complexity. A direct comparison between solutions from different representations is impractical since the underlying units of measurement are different. However, the solution complexity can be compared using a correct set of measurement units for each algorithm if these units have the same interpretation for complexity. The tree size for C4.5 and the program size of the solution team for SBB can be considered as the units of measurement. These units can be interpreted as a rule or an if-else statement. Hence, the complexity of C4.5 and SBB are comparable.

3) Feature complexity reflects the number of unique attributes employed per solution and potentially gives additional knowledge regarding botnet communication.

3.2 Publicly Available Tools Employed

As mentioned at the beginning of this chapter, network traffic should be presented to the machine learning algorithms in a meaningful way. Five publicly available flow exporters are used in this thesis. As well, Snort and BotHunter are utilized for the comparison and evaluation purposes of the proposed early warning system.

3.2.1 Flow exporters

Flow generation tools (flow exporters) summarize traffic utilizing the network packet headers. These tools collect packet information with common characteristics such as IP addresses and port numbers, aggregate them into flows and then calculate statistics such as the number of packets per flow, etc. In RFC 2722, a traffic flow is defined as a logical equivalent for a call or a connection in association with a user-specified group of elements [126]. The most common way to identify a traffic flow is to use a

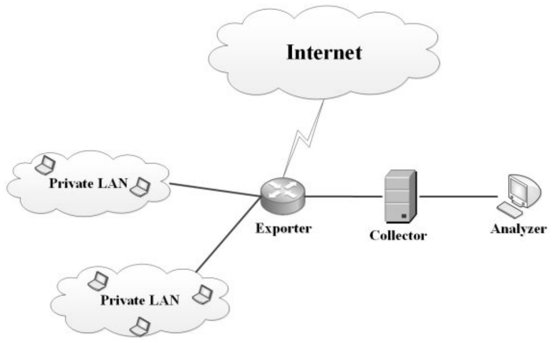


Figure 3.3: Flow exporting mechanism

combination of five properties from the packet header including the Network and the Transport layer headers of the TCP/IP network protocol stack. These are: Source IP address, Destination IP address, Source Port Number, Destination Port Number and Protocol which will be referred to as the 5-tuple information.

Companies producing and managing network equipment such as routers and switches provide different types of flow exporters to summarize network traffic in terms of flows using the 5-tuple information. As well, they provide flow analysis tools for analyzing the flow streams. Some examples of COTS (commercial off the shelf) flow exporter and analysis techniques are: Cisco systems' NetFlow [3], Juniper systems' J-Flow [8] and InMon systems' S-Flow [19]. As well, there are some open source exporters such as Maji [10] by the WAND research group from Waikato University or YAF [24] by the Cert NetSA research group.

However, over time different versions of NetFlow were developed and the very recent version is now the IETF standard for flow exporters as IPFIX (RFC 5101). To collect and analyze traffic flow data the three network elements should work together. Fig. 3.3 illustrates the three components: (i) the flow exporter which generates the flow data, (ii) the flow collector which collects (stores) the flow data from the exporter, and finally, (iii) the flow analyzer which analyzes the collected data. A variety of network devices (*e.g.* routers and switches) support different versions of flow data. Therefore, in this research different open source flow exporters and collectors are employed with the pre-captured data sets to extract the flow features with the objective of exploring the effect of the flow exporter and collector tools on the performance of the analyzers.

The exporters employed in this work are listed bellow.

(i) **Maji** [10] is an open source implementation of IPFIX supported by the WAND research group at Waikato University. This tool exports uni-directional flows from live PCAP interfaces as well as the more common trace file formats. Maji is designed to support custom template definitions and can relay the IPFIX information to TCP, UDP and SCTP collectors. Additionally, it has a simple exporting method that outputs the flow information in the standard output. Therefore, no IPFIX collector was employed for this tool.

(ii) **YAF** (Yet Another Flow Sensor) [24] is a bi-directional flow exporter designed by the Network Situational Awareness (NetSA) group at CERT. This tools collects and exports IPFIX-based flows. Similar to Maji, YAF can process packet data from pre-captured traffic files or live captures from an interface to export flows. Although the YAF package comes with a YAF-Ascii that outputs NetFlow packet information in an ASCII format (which is employed in this work); any other NetFlow collector can be used for this purpose, as well.

(iii) **Softflowd** [21] is a light-weight uni-directional flow exporter that supports different versions of the NetFlow. This tool exports NetFlow data using the traffic on a simple device interface or a pre-captured traffic file (pcap format). In this work, once the network packets are exported into flows using Softflowd, NfDump was employed as the flow collector. NfDump is an open source flow collector tool. NfDump supports different versions of NetFlow, too [17].

(iv) **Tranalyzer** [22] is a light weight uni-directional flow exporter and analyzer that employs an extended version of NetFlow feature set. This tool exports both the binary and ASCII formats and therefore does not require any collector (such as NfDump). Two versions of Tranalyzer have been used in this research, Tranalyzer v. 0.5.8 and Tranalyzer v. 0.5.9 since the later version became available after the first analysis was conducted in this research. These versions will be referred to as Tranalyzer-1 and Tranalyzer-2 hereafter. Tranalyzer-2 has five additional features compared to Tranalyzer-1[1].

(v) **Netmate** (Network Measurement and Accounting System) [13] is a bi-directional flow exporter and analyzer. This exporter can process live captures and pre-captured traffic and uses different rule sets to control the exported flow format. One of the

[1]Additional Features are: ICMP Status, TCP Ack Trip Jitter Average, TCP Ack Round trip average Jitter, Subnet number of source IPv4, Subnet number of destination IP.

Table 3.1: The properties of the flow exporters employed.

Exporter	**Standard**	**Flow**	**Input**	**Collector**	**# of features**
Maji	IPFIX	uni-direction	live/pre-captured	included	59
YAF	IPFIX	bi-direction	live/pre-captured	included	46
Softflowd	NetFlow (V. 5, 7, 9)	uni-direction	live/pre-captured	NfDump	41
Tranalyzer-1	-	uni-direction	live/pre-captured	embedded	93
Tranalyzer-2	-	uni-direction	live/pre-captured	embedded	98
Netmate	-	bi-direction	live/pre-captured	embedded	44
Argus	NetFlow	uni-directional bi-directional	live/pre-captured	included	126

former NIMS lab members has developed "Netmate-flowcalc" which is a bundle including Netmate v.0.9.5 packaged with NetAI modules from v0.1 [14]. Not only does this open source program ease the installation process but also the NetAI module has been extended for additional output features which have been employed in this research.

Table 3.1 summarizes the properties of the flow exporters employed in this research. The third column indicates whether the exporter requires a collector (if so, does the tool provide it in the package) or it exports the flow information directly in a human readable format. In this column of the table, "included" means a collector is included in the package which, however, can be replaced with another one if desired. In contrast, "embedded" means a collector is embedded in the exporter and no external collector can be used. Table 3.2 shows whether the tool supports specific categories of features. These categories are defined based on the features that are used frequently in the literature [85, 156, 153]. The "Time" category refers to those features indicating start-time, end-time and duration of the flow. The time-based category is important when flow aggregation or window-based analysis is employed. The "Inter-arrival" category refers to the statistics calculated over a flow such as the average inter-arrival time which represents the average number of milliseconds between consecutive packets of a flow. The "Packets&Bytes" category refers to the features computed based on the number of packets which form the flow and their size (in bytes) such as the average number of bytes per packet. The "Flags" category refers to any feature related to the packet header flags such as the TCP SYN flag.

Table 3.2: The feature categories of the flow exporters employed.

Exporter	Time	Inter-arrival	Packets&Bytes	Flags
Maji	✓	✗	✓	✓
YAF	✓	✓	✓	✓
Softflowd	✓	✗	✓	✓
Tranalyzer (1 and 2)	✓	✓	✓	✓
Netmate	duration only	✓	✓	✓
Argus	✓	✓	✓	✓

3.2.2 Malicious behaviour detection systems

In order to understand how well the proposed early warning system in this thesis would perform, two publicly available malicious behaviour detection system were chosen: Snort and BotHunter. These two systems were selected specifically because that BotHunter is the botnet detection system used in the literature [83, 152] for performance evaluation purposes and Snort is a highly employed malicious behaviour detection system in literature as well as in industry [152, 63].

3.2.2.1 Snort

Snort is an intrusion detection and prevention system that analyzes packet payloads as well as packet header data to detect any evidence of harmful actions which match predefined signatures (rule sets) [20]. Some of these pre-defined rules/signatures take advantage of payload information while others require only the header features to be analyzed. Snort has been supported by two rule sets: VRT (Vulnerability Research Team), which is the official rule set for Snort, and ET (Emerging Threat), which is published by emergingthreats.com. As discussed in more detail in Section 7, the VRT rule set was used. These two main rule sets have come with many rules which aim to cover all possible network conditions. Users should be careful to enable only those rules that fit their network conditions and alert priority settings and disable the others. Since 1998, Snort has been known and used in the network security area given that it is a cross-platform open source IDS/IPS which can be modified to fit the network security challenges and needs as shown by the BotHunter research.

3.2.2.2 BotHunter

Gu *et al.* introduced BotHunter as a botnet detection system and made it publicly available. This tool uses the combination of Snort and a clustering approach to detect botnet infections. BotHunter is based on the idea that all botnet infection processes are similar and can be illustrated by a lifecycle model explained in Section 2. Hence, it uses a modified version of Snort with its plugins to detect the specific bot actions of the lifecycle and then correlates the Snort alerts to detect the botnets' behaviour and infected machines. The developers have modified and selected the botnet-related Snort rules, developed many additional rules and inserted IP address checking into the Snort rules to make BotHunter's Snort sensors work more efficiently. The initial version of BotHunter used two plugins: *SLADE* and *SCADE*, which are designed for anomalous traffic pattern and payload detection. Recently, these plugins have been replaced by three new plugins: (i) *bhDNS* for malicious DNS analysis, (ii) *bhSD* for scanning detection; and (iii) *Con-P2P* for Conficker-C P2P detection and ethernet tracking. All the new plugins use existing information like DNS lists, IP lists, port lists and so on to detect malicious (botnet) behaviour. It should be noted here that to be able to detect new botnets, BotHunter relies on Snort signature updates of new malicious behaviours where the signatures use header and payload information.

3.2.2.3 Performance Criteria

Both BotHunter and Snort use Snort rule sets to generate alarms. BotHunter, however, goes a step further and aggregate those alarms to form a profile. In both of these tools alarms are raised on the IP addresses with/without supporting information (*e.g.* the protocol or port number). To evaluate the performance of these tools, the percentage of detected infected bots (with the corresponding IP addresses) is usually used. However, other approaches in this thesis use either the percentage of detected malicious flows or packets which is much more detailed compared to IP addresses. To have a better performance evaluation and comparison, the detected IP addresses by Snort and BotHunter are used to label the data samples (*e.g.* flows) as detected and then the detection rate is calculated. The detail of this relabelling and evaluation method are explained in Chapter 7.

3.3 Summary

The tools and algorithms which have been used throughout the different chapters of this research were introduced in this chapter. From the wide range of machine learning algorithms available, SVM, SBB, C4.5, ANN, KNN, Bayesian Networks, and Naive Bayes have been selected and employed. These are algorithms used widely in the literature for network traffic analysis. To apply these machine learning algorithms to the network traffic traces the data should be represented in a feature-based format. Using flow exporters is one of the methods used in this research for such representation. In this case, Maji, Netmate, YAF, Tranalyzer and Softflowd are the tools introduced here. Finally, the proposed botnet detection system is evaluated against four different detection systems, two of which are the publicly available systems called Snort and BotHunter. These two systems were introduced as well in this chapter.

Chapter 4

Data Sets

Botnet data can be analyzed in different forms. The type of data and specific part of the data for behaviour analysis should be defined based on the approach. In other words, different forms of analysis require different types and fractions of data. Hence, in this chapter, the data sets that are used in this thesis and their characteristics are described.

For analyzing botnet behaviours in application-based data such as domain name data analysis, in Chapter 5, lists of malicious and legitimate domain names are required. For this purpose, several domain name lists have been collected from legitimate resources. The ZeusTracker site [25] and the DNS-BH site [7] are monitoring the Zeus and the Citadel botnets actively. Hence, the C&C domain name lists representing the Zeus and Citadel botnets are downloaded from these two web sites. The Conficker C&C domain name lists published by the Bonn university [5] and DNS-BH [7] websites are collected as a representative of Conficker domain names. As well, the C&C domain name lists published by Twitter API and DNS-BH for the Torpig botnet are employed [142, 7]. Finally, for the Kraken (Bobax) botnet, domain name lists published by the Damballa [103], DNS-BH [7], DV lab [27] and ThreatExpert blog [26] web sites are collected and employed in this thesis. All of these domain name lists represent the botnet lists. In addition to these botnet domain name lists, some of the most frequently requested domain names from the Alexa lists are used to represent the legitimate domain names. Alexa Internet Inc. [2] ranks the websites based on their page views and unique site users. This ranking is then published as the list of the most popular websites. Given that even Alexa lists might have malicious domain names [127], 500 benign domain names from the Alexa lists were extracted manually to represent the legitimate domain names.

On the other hand, for analyzing botnet behaviours in network-based data (such as traffic flows or packets, in Chapter 6), network log file traces are required.

4.1 Data Sets Employed

What follows is an introduction to the different categories of data sets employed based on the capturing/generation methods and/or the public sources which provide the data sets.

4.1.1 Public log files

Although there were only a few small botnet data sets available publically before 2014, since then different research centers have started publishing their data sets for research evaluation purposes. What follows is a list of the public data sets employed in this research. These data sets are categorized by the source of data.

4.1.1.1 Snort Sourcefire VRT lab repository

The snort Sourcefire vulnerability research team lab has provided three sample traffic log files for the Zeus botnet in 2010 [1]. Information regarding the log files as well as the manner in which they should be analyzed and labeled is also provided by the research team. Two of them are very small (only 110 and 1105 packets) and therefore could not be used in these experiments. Hence, "Sample_1" is the only Zeus traffic file utilized from the Snort archive in this research. This data is referred to as Zeus (Snort) hereafter.

4.1.1.2 NETRESEC repository

The NETRESEC repository provides sample traffic log files for the Citadel, Cutwail, Kelihos and Zeus botnets. There are no detailed descriptions of these log files available in this repository and the sample log files have captured the botnet traffic (behaviour) for a very limited period of time.

4.1.1.3 CVUT malware captures facility repository

The Malware Capture Facility Project is a research project defined by the Czech Technical University Agent Technology Center (ATG) for capturing, analyzing and publishing malware traffic [82]. The goals of this project are to execute real malware for long periods of time, to monitor and capture the traffic during execution and finally, the analyzing and publishing of the captured traffic. Since 2013 more than one hundred botnet captures have been published by ATG . However, not all of the captures and their corresponding botnet executables are linked to known botnets, so only the captures that are named properly and linked to known botnets have been selected. In this way the analysis results and the extracted behaviours of the botnets can be analyzed as well by the published information. Zeus [80], Kelihos [78], Neris [79], NSIS [81], Rbot [75], ZeroAccess [77] and Virut [76] are the botnets selected from this repository.

4.1.1.4 University of Victoria repository

In 2013, the University of Victoria made a data set publicly available [128]. This data set has combined two separate data sets of malicious botnet traffic from the French chapter of Honeynet project on the Strom and Waledac botnets. This combination represents the malicious side of the data set. Unlike other data captures employed in this dissertation, this log file also has a legitimate side which consists of two traffic log files: one log file from the Traffic Lab at Ericsson Research in Hungary and another log file from the Lawrence Berkeley National Laboratory (LBNL) in the U.S. Hereafter, this data set will be referred to as ISOT (UVic). Information published on this data set by the University of Victoria was used for labelling and analysis.

4.1.1.5 Center of Applied Internet Data Analysis (CAIDA) repository

The CAIDA organization has captured and made publicly available a Conficker botnet data set [60]. This traffic log,captured over three days, was collected by the CAIDA UCSD network telescope when the Conficker botnet was active in 2008. The first and the second day covers the Conficker-A botnet infection while during the third day Conficker-A and B were active. This data set has been anonymized, the payload has been removed from the packets and the CAIDA network addresses have been masked (destination IP addresses) and Hereafter, will be referred to as the Conficker (CAIDA).

4.1.1.6 Lawrence Berkeley National Lab (LBNL) repository

The LBNL enterprise Tracing Project was focussed on collecting and characterizing internal enterprise traffic [9]. The data was captured in the medium-sized enterprise network of the research institute from October, 2004 to January, 2005. There is a variety of network activities included in this traffic capture such as web, email and media streaming. In the literature, this data is widely used as the non-malicious traffic trace which is a good representative of day-to-day enterprise network usage.

4.1.1.7 Wireless and Secure Networks Research Lab (WiSNet) repository

The WiSNet research lab at the National University of Sciences and Technology (NUST) has published several benign traffic log files [23]. Among those data sets, the ISP data set is the biggest data set with a wide range of communication types (*e.g.* HTTP communication). Hence, in this research the ISP data set is used to represent a legitimate log file. This data set was collected from the edge routers of an Internet Service Provider (ISP).

4.1.2 Dalhousie University NIMS lab repository

In addition to the publicly available log files, several botnet traffic traces were generated in the NIMS lab at Dalhousie University. These log files can be categorized into two major types: Sandbox log files and Domain-based HTTP log files.

4.1.2.1 Sandbox log files

Many researchers in the literature [147, 122, 145] have generated botnet traffic in a sandbox environment using the public botnet binaries and toolkits. Therefore, in addition to the public data sets, the Zeus toolkit (version 1.2.7.19 and 2.1.0.1) and Citadel toolkit (version 1.3.5.1) have been to generate botnet data samples. The Zeus toolkit version 1.2.3.19 is analyzed and employed by Binsalleeh *et al.* [57]. The newer version of the Zeus botnet and Citadel botnet are both improved versions of that toolkit with some enhanced capabilities. Twelve bots were set up in the testbed (machines infected with Zeus and Citadel botnets) as well as two C&C servers

Table 4.1: The sandbox configuration.

Data set	No. of bots	No. of servers	Server type	Description
Zeus-T1 (NIMS)	12 Windows machines	2	Linux and Windows	Zeus botnet version 1.2.7.19
Zeus-T1-W (NIMS)	12 Windows machines	2	Linux and Windows	Zeus botnet version 1.2.7.19 Web Injection enabled
Zeus-T2 (NIMS)	12 Windows machines	2	Linux and Windows	Zeus botnet version 2.1.0.1
Zeus-T2-W (NIMS)	12 Windows machines	2	Linux and Windows	Zeus botnet version 2.1.0.1 Web Injection enabled
Citadel-T1 (NIMS)	12 Windows machines	2	Linux and Windows	Zeus botnet version 1.3.5.1
Citadel-T1-W (NIMS)	12 Windows machines	2	Linux and Windows	Zeus botnet version 1.3.5.1 Web Injection enabled

(a Windows server and/or a Linux server). Table 4.1 shows the different testbed configurations used for generating botnet data sets in this category. As mentioned in the table, in some of the configurations, the Zeus/Citadel web injection capability is activated. Web injection refers to injecting rouge content in the web pages (usually bank websites) in order to steal sensitive information. For this purpose, the web injection plugin is installed on the bot which is then used to trick the browser for injecting information into specific web pages. Given that the sandbox environment is isolated from the outside world (not connected to the Internet), an HTTP server was set up and a fake bank website created. This website is then used as the target for the web injection attack of the Zeus and Citadel botnets. The generated network traces in this sandbox will be used as representative of the Citadel and Zeus botnet behaviour. Hereafter, these data sets will be referred to by the names used in the first column of Table 4.1.

4.1.2.2 Domain-based HTTP log files

In this category, the focus is on the botnets, which employ the HTTP protocol as their communication protocol. Rather than running botnet binaries in sandbox environments, publicly available lists (from legitimate resources) of C&C domain names that are used as the data sets for chapter 5, were employed for generating the representative botnet/legitimate traffic. Although using sandbox environments with botnet binaries and toolkits would create close to real world data samples, this approach would not result in up-to-date botnet data samples. This is because these toolkits are usually not accessible while they are recent/new. Basically, botmasters attempt to sell their toolkits after they are done using them for their purposes. Moreover, these toolkits can be found in underground blogs and black-hat websites usually one or two years after being used for the botnet attacks.

A new method for creating botnet data samples was developed using the newly employed domain names (specifically for the botnets). This approach ensures that the generated traffic is a good representative of up-to-date traffic and avoids the possibility of representing old botnet behaviour when old binaries are used. To this end, a program was developed to establish HTTP connections with the domain names (both botnet C&C servers and legitimate web servers) from the aforementioned lists to generate the traffic. First, the program sends DNS queries for the domain names in the lists. Then, if it receives a proper DNS response indicating that the domain name is registered and is associated with a valid IP address, it attempts to establish an HTTP connection with that IP address. The traffic generated in this approach represents the botnet communication of the communication phase of the botnet lifecycle. As discussed earlier, botnet domain name lists (such as Conficker and Zeus) were obtained from the Bonn University, ZeusTracker, Damballa and DNS-BH project blocklists [7, 5, 25, 103] which represent the botnet lists. All the NIMS-generated log files using the above domain name lists are purely malicious. However, the systems based on various data mining techniques (similar to the proposed network data analysis-based system) require legitimate traffic (to represent normal behaviour) for training purposes as well. The Alexa most common domain names list was used to represent the legitimate domain names. Using the Alexa domain name list, a legitimate log file was generated using the same method as the domain-based botnet data samples. All generated traffic was captured while the program was attempting to establish connections with the domain names and no sampling rate was used. Hereafter, all the generated log files in the NIMS lab will be referred to as the NIMS log files. Specifically, the domain-based generated data sets will be named with a "-D" extension: *e.g.* like "Alexa-D (NIMS)".

4.1.3 Data evaluation

All the data types employed in this research can be grouped into three main categories: (i) string format domain names, (ii) publicly available traffic traces (iii) traffic traces generated and/or captured at the NIMS lab. These are made publicly available as well at: http://web.cs.dal.ca/~haddadi/data-analysis.htm.

In the first group, domain name lists were collected from the ZeusTracker [25], Bonn University [5], DNS-BH [7] websites and Damballa Inc. [103]. Given that these are all known legitimate resources, no additional analysis has been done regarding the provided domain name lists and their labels. However, since even the Alexa lists may have malicious domain names [127], 500 benign domain names were extracted from the Alexa list for the data employed in this thesis.

For the second group data was obtained from the Snort VRT lab [1], the NETRESEC repository [15], the CVUT malware repository [82], the University of Victoria [128], the LBNL research lab [9], the WiSNet research lab [23] and the CAIDA organization [60]. Snort provides a report describing the sample traffic log files and therefore the files are not analyzed any further. NETRESEC, on the other hand, provides the Citadel, Cutwail, Kelihos and Zeus botnet traffic log files (one for each) but does not provide any information regarding these files. Although this is also a legitimate repository, investigations were done on the protocols employed, the domain names (if available) and the communication patterns in these traffic log files, matched them with the published characteristics of these botnets and verifiying that the data was correct. As discussed in Section 4, seven botnet traffic traces were employed in this research from the CVUT malware repository, namely ZeroAccess, NSIS, Rbot, Zeus, Neris, Virut and Kelihos. In addition to the log files, CVUT has published explanatory information as well for each botnet sample such as log file analysis, a protocol analysis and the infected hosts IP range. From a research group at the University of Victoria, Saad *et al.* have combined several botnet log files and legitimate log files which are published as the ISOT data set. Information regarding the scenarios which were used to combine the log files and the IP address mapping is provided by this research team. ISP (WisNet) and ETP (LBNL) are both employed in the literature as legitimate log files. Information such as the IP address ranges and the point of capture (at the enterprise level) is provided by these two organizations. Finally, since the UCSD telescope carries no legitimate traffic and given that there is other malicious background traffic than the Conficker infections in their captures, the information provided by CAIDA was used as the ground truth for this data set. This includes a two-day capture of the UCSD telescope network prior to the Conficker infection captures. In the end, all of these botnet traffic files were used in this research. All of the aforementioned information provided by the corresponding organizations for each of these log files were used for data set labelling and protocol analysis in this thesis.

In the third group, there are two subcategories of log files: (i) domain-based log files captured through HTTP communication of the NIMS machines and the botnet C&C domain names (mentioned in the first group); and (ii) toolkit-based log files captured while infected machines in the NIMS testbed were communicating with the configured C&C servers on the same testbed. For the first subcategory, the domain names, their corresponding IP addresses (if available) and the NIMS machine IP addresses were used for analyzing and labelling the data. Moreover, in the second subcategory, different scenarios and configurations were used to setup the botnet sandbox which resulted in six different log file captures. The IP address range used in the testbed was utilized for labelling the resulted data sets. It should noted that no legitimate application was running in the background of the infected machines.

4.1.3.1 Verifying the generated data

Section 4.1.2.2, a systematic approach was proposed for generating botnet traffic data representing the communication phase of the botnet lifecycle. The generated data explained in this section forms the first subcategory of the third group of data. Prior to using these log files, evaluations were conducted to confirm that the data generated by the proposed approach represents behaviour similar to the botnet traffic that is captured in the wild and the botnet traffic that is generated in a controlled environment based on publicly available bot binaries and toolkits. In order to do this, two steps were followed to evaluate and confirm the data after it is generated: (i) a visualization of the flow features of the generated traffic vs. the flow features of the traffic captured in the wild and the controlled environment; and (ii) the employing a decision tree based learning classifier (rule-based) for post-classification analysis of both the generated and the collected traffic. It should be noted that this data evaluation can be applied only to those botnets with the corresponding samples captured in the wild and controlled environments. Hence, the evaluation is confined to the Zeus and Citadel botnets.

In the first step, the data sets are compared based on the key features of the traffic. These features are highly employed in the literature for botnet behaviour analysis

[147, 139, 156, 90]. To extract these features, IP-flow technology is utilized, through which the network traffic is summarized into flow features. Specifically, Softflowd is employed [21], which is an open source tool based on the NetFlow standard [3] focussing on the main flow features (such as flow duration and number of bytes) and not detailed statistics. In the second step, the C4.5 learning technique is employed [50], which is a decision tree-based (rule-based) classifier, for post-classification analysis. This classifier has shown to perform very well in network traffic analysis and is also the best performing classifier in this thesis.

4.1.3.1.1 First Step for Confirming the Data Generated – Visualization of Flow Characteristics

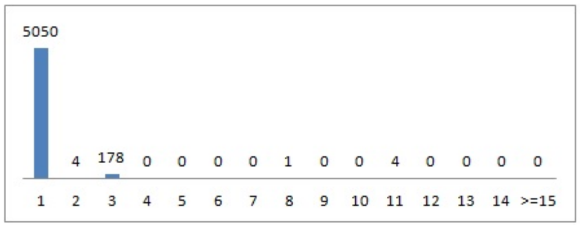
As a first step it will be beneficial to analyze the data sets on some of the more important features of a flow used by other researchers [147, 139, 156, 90] to understand the data sets better. As discussed earlier, given the wide range of HTTP usage on the Internet, many recent botnets employ the HTTP protocol to hide their malicious activities among the normal web traffic [32]. Citadel and Zeus fall under this category, too. They utilize the HTTP protocol to communicate with their bots. To analyze and compare the aforementioned data sets, the non-HTTP flows of the data sets are filtered out. This way all the background information is removed from the data and the data sets can be compared on their fundamental properties.

Figure 4.1 shows the frequency graph of the flow durations for the Zeus botnet data sets. As shown in the figure, the majority of flow durations (96%, 70%, 91% in Zeus-D (NIMS), Zeus (Snort) and Zeus (NETRESEC), respectively) are less than 50 seconds long (placed in the first bucket). However, 20% of Zeus (Snort) and 67% of Zeus-T1 (NIMS) flows last longer than the other two Zeus data sets. Figure 4.2 and Fig. 4.3 show that Zeus-T1 (NIMS), Zeus (Snort) and Zeus (NETRESEC) have more flows with a higher number of transmitted packets (30 packets or more) and more bytes, respectively. The main cause of this type of difference between the Zeus-D-generated data and the other Zeus data is that the connection between the system and the botnet C&C servers could not be kept open longer in the program. That is because botnet communication is based on a client-server command and response pattern but not all of the correct responses for the commands and requests sent by the C&C servers available.

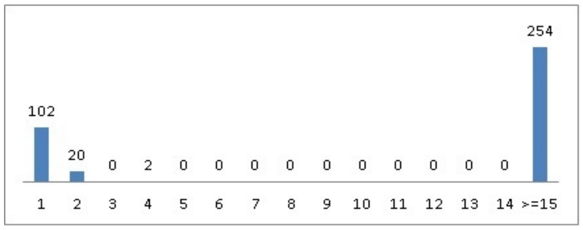
Figure 4.4 shows that the pattern for the duration of the flows for the Citadel-D (NIMS) botnet is different from the pattern of the duration of the flows for Citadel (NETRESEC) but it is similar to the pattern of Citadel-T1 (NIMS). Moreover, Fig. 4.6 shows that the data sets are similar in terms of the number of packets. However, no obvious similarity is shown in Fig. 4.5 regarding the number of packets per flow in these data sets. Although Citadel-D (NIMS) and Citadel-T1 (NIMS) tend to send and receive more packets during communication, the size of the data being transferred does not seem to be very different. With these observations, it can be concluded that the Citadel C&C connections in the NETRESEC data were kept open without any actual client-server command and response communication while in Citadel-D (NIMS) and Citadel-T1 (NIMS) the connections were terminated after the communication was finished. Since keeping a connection open for a long time is one of the signs of malicious activity, botnets tend to close the connection when the necessary information is sent/received and open a new connection for the next round when needed. This may indicate that the Citadel-D (NIMS) and Citadel-T1 (NIMS) data sets are closer to normal behaviour. However, a high number of connections also raises flags for malicious behaviour. Therefore, a trade-off between these two parameters is necessary for the botnets to hide their malicious activity. With regard the connection termination setup, it was decided to keep the program developed for communicating with the C&C servers in the Citadel-D (NIMS) capture the same, because it gives a good balance for the aforementioned trade-offs.

4.1.3.1.2 Second Step for Confirming the Data Generated– Post-Classification Analysis by Using a Decision Tree Classifier

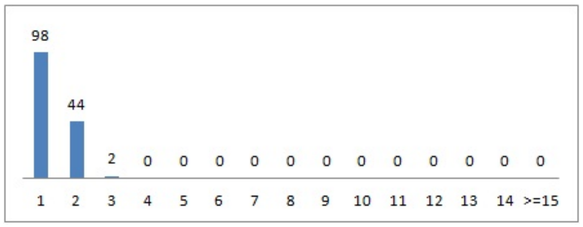
To investigate further how similar the data generated by the proposed approach is to the data captured in the wild and the sandbox data, a post-classification analysis was employed using a decision tree. To this end, a C4.5 classifier was trained on the domain-based NIMS data and the trained model tested on the botnet data from the NETRESEC and Snort web sites (captured in the field) and the sandbox Zeus-T1 (NIMS) and Citadel-T1 (NIMS) data. Table 4.2 presents the features that are utilized in this evaluation. These features are extracted by Softflowd and are the basic netflow



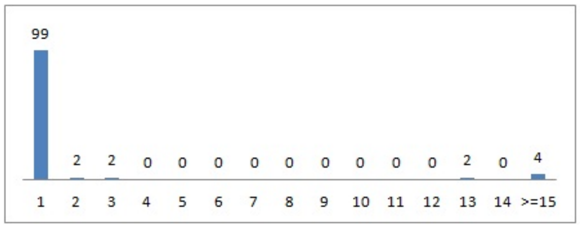
(a) Zeus-D (NIMS)



(b) Zeus-T1 (NIMS)

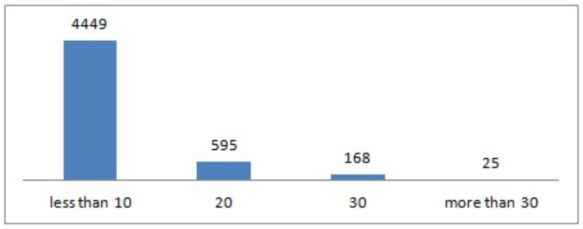


(c) Zeus (Snort)

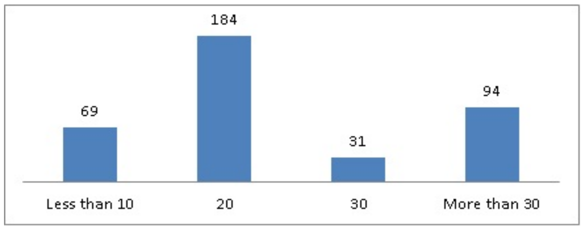


(d) Zeus (NETRESEC)

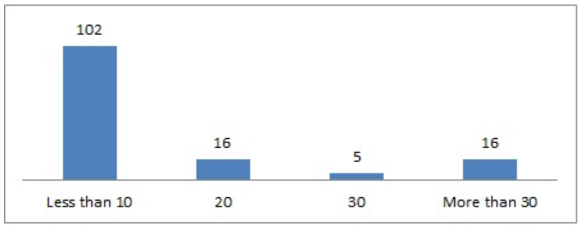
Figure 4.1: The frequencies of the flow durations in the Zeus data sets. The X-axis denotes the buckets of 10 seconds.



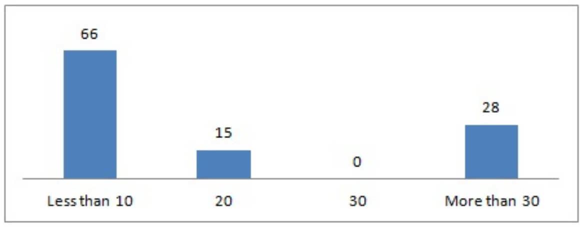
(a) Zeus-D (NIMS)



(b) Zeus-T1 (NIMS)

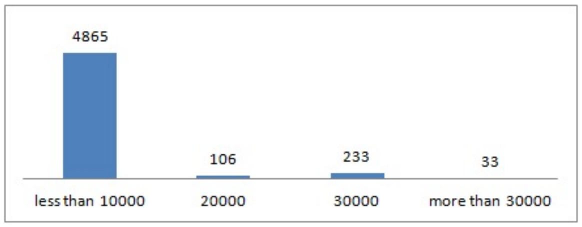


(c) Zeus (Snort)

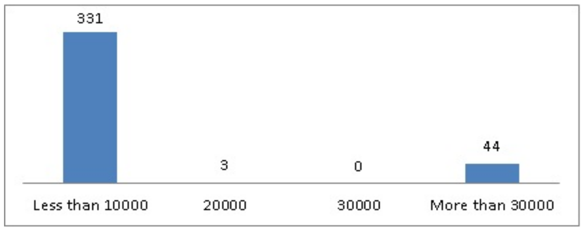


(d) Zeus (NETRESEC)

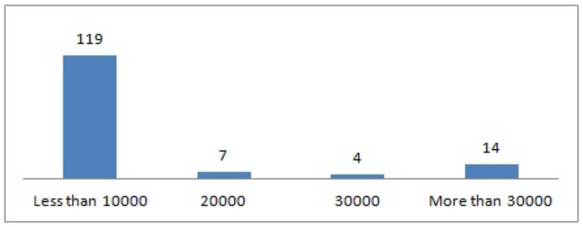
Figure 4.2: The frequencies of the number of packets per flow in the Zeus data sets. The X-axis denotes the number of packets per flow.



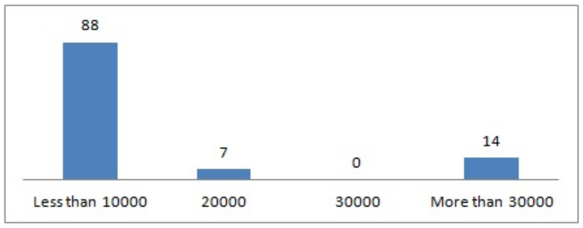
(a) Zeus-D (NIMS)



(b) Zeus-T1 (NIMS)

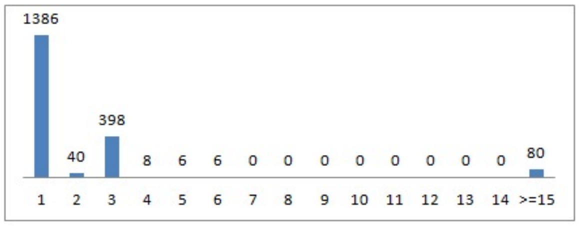


(c) Zeus (Snort)

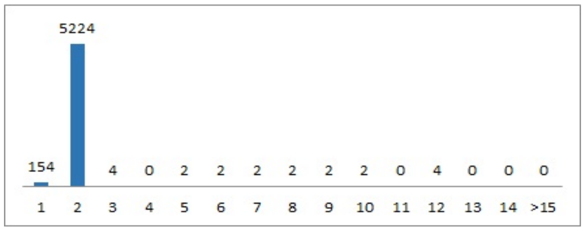


(d) Zeus (NETRESEC)

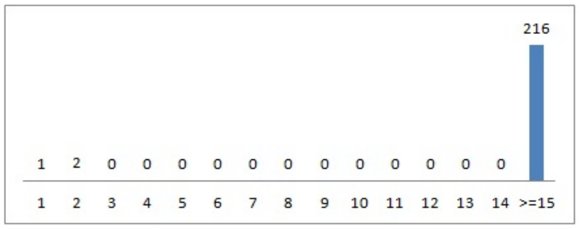
Figure 4.3: The frequencies of the number of bytes per flow in the Zeus data sets. The X-axis denotes the number of bytes per flow.



(a) Citadel-D (NIMS)

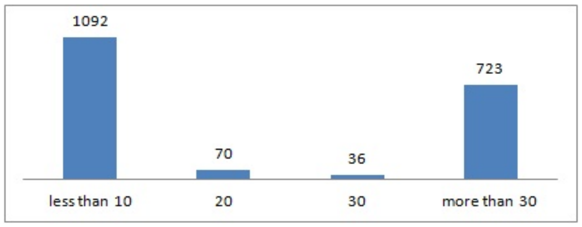


(b) Citadel-T1 (NIMS)

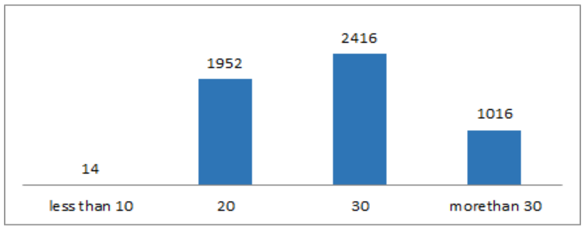


(c) Citadel (NETRESEC)

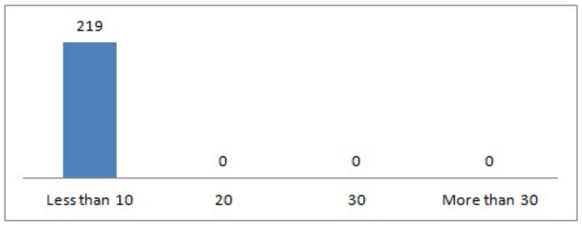
Figure 4.4: The frequencies of the flow durations in the Citadel data sets. The X-axis denotes the buckets of 10 seconds.



(a) Citadel-D (NIMS)

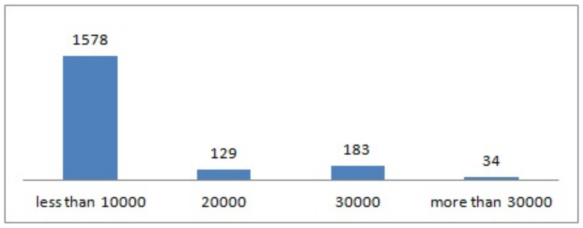


(b) Citadel-T1 (NIMS)

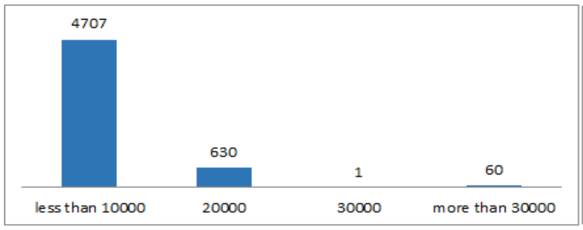


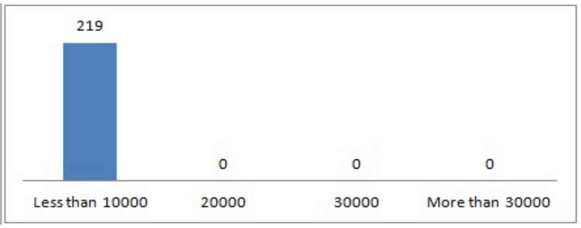
(c) Citadel (NETRESEC)

Figure 4.5: The frequencies of the number of packets per flow in the Citadel data sets. The X-axis denotes the number of packets per flow.



(a) Citadel-D (NIMS)





(b) Citadel-T1 (NIMS)

(c) Citadel (NETRESEC)

Figure 4.6: The frequencies of the number of bytes per flow in the Citadel data sets. The X-axis denotes the number of bytes per flow.

Table 4.2: Softflowd feature set definition

	Softflowd Features
Duration	Flow duration
Src-AS	Source AS number
Dst-AS	Destination AS number
In-If	Input interface
Out-If	Output interface
Total-Pkt	Total number of packets
F-Pkt	Forward number of bytes
B-Pkt	Backward number of bytes
Total-Byte	Total number of bytes
F-Byte	Forward number of bytes
B-Byte	Backward number of bytes
Flows	Number of aggregated flows (if any)
ToS	Type-of-Service
Src-ToS	Source Type-of-Service
Dst-ToS	Destination Type-of-Service
Src-Msk	Source mask
Dst-Msk	Destination mask
FWD	Forwarding Status
Src-Vlan	Source Vlan label
Dst-Vlan	Destination Vlan label
bps	Bits per second
pps	Packets per second
Bpp	Bytes per packet

features (highly employed in the literature). For the legitimate side of these data sets, Alexa-D (NIMS) is used in this experiment. Table 4.3 shows the results of this step. Detailed definitions of all the features can be found on the NfDump project web site [17].

As shown in Table 4.3, the model trained on the Citadel-D (NIMS) could detect all of the Citadel (NETRESEC) and Citadel-T1 (NIMS) exemplars of the test data (a 100% TP rate). This observation shows how similar these data sets are in terms of the basic netflow features exported by Softflowd that the C4.5 classifier employed to create the training model. Our post-classification analysis shows that the main features the C4.5 decision tree classifier employs for the Citadel-D (NIMS)-trained model are: Duration, Total-Pkt, Total-Byte, bps, pps and Bpp (from Table 4.2). In the first step of the analysis the duration feature of a flow presented the most discrepancy

Training data set	**Testing data set**	**DR**	**Botnet**		**Legitimate**	
			TPR	**FPR**	**TNR**	**FNR**
Citadel-D (NIMS)	Citadel (NETRESEC)	100%	100%	0	0	0
Citadel-D (NIMS)	Citadel-T1 (NIMS)	100%	100%	0	0	0
Zeus-D (NIMS)	Zeus (Snort)	81%	81%	0	0	19%
Zeus-D (NIMS)	Zeus (NETRESEC)	79%	79%	0	0	21%
Zeus-D (NIMS)	Zeus-T1 (NIMS)	88%	88%	0	0	12%
Zeus-D (NIMS)	Zeus-D (NIMS-unseen)	84%	84%	0	0	16%

Table 4.3: The C4.5 decision tree classification results.

between the NIMS botnet data set and that captured in the wild (the NETRESEC data set). However, the classification results indicate that this discrepancy does not seem to have a negative effect on the identification of Citadel botnet behaviour in the given traffic traces. This might be because of the importance of the other features employed or the relationships between the thresholds of these features (revealed by the C4.5-trained model). This implies that Citadel-D (NIMS) data set is similar to the real-life Citadel (NETRESEC) and Citadel-T1 (NIMS) data sets. Therefore, it can be confirmed that the proposed approach generates realistic Citadel botnet traffic.

Table 4.3 presents the classification results on the Zeus botnet trafficas well. Again, a C4.5 decision tree classifier was trained on the Zeus-1 (NIMS) data set and the trained model tested against the Zeus-T1 (NIMS) and Zeus data sets from Snort and NETRESEC [*i.e.* Zeus (NETRESEC) and Zeus (Snort)]. As the results indicate, the DRs of the Zeus-T1 (NIMS), Snort and NETRESEC data sets are from 79% to 88% which are less than the DR for Citadel, but it is still a promising performance. However, in previous work [90] where a C4.5 classifier trained and tested solely on the Zeus-D (NIMS) data set, the performance of the classifier could be pushed up to 86% DR. This indicates that Zeus botnet behaviour is more complicated than Citadel botnet behaviour and seems to hide within legitimate HTTP behaviour very well. This makes it more difficult to differentiate Zeus traffic from legitimate traffic, hence the lower DR (compared to Citadel's case).

To analyze whether this lower DR is caused by the discrepancies (indicated in 4.1.3.1.1) between the Zeus-D (NIMS) traffic captures vs. the other Zeus traffic captures, another experiment was done. Since the Zeus-D (NIMS)-generated traffic was bigger than the Alexa legitimate data (and therefore had more flows), only a fraction of the Zeus HTTP traffic was used to build a balanced Zeus-Alexa training data set for the second step of the analysis. Hence, there were some unseen Zeus flows that were not included in the training process. Thus for further analysis of Zeus-generated traffic, the trained Zeus model was tested on these unseen Zeus-generated flows. As the result shows, an almost identical DR was obtained (84%), Table 4.2. This seems to indicate the complicated behaviour of the Zeus botnet and confirms that the results are not a side effect of generated data, but rather that the classifier performs the same on both the generated data and the data captured in the wild for the Zeus botnet.

In short, the two-step analysis indicates that the domain name-based NIMS generated traffic captures are valid and comparable to botnet data simulated and/or captured in the wild. In other words, these generated data sets can be employed in botnet behaviour analysis as representing real data. It appears that there is no data generation and validation work in the literature similar to the methods discussed in this section. The importance of this data generation-validation method is two-fold: (i) The same data generation method can be used to create new data sets for benchmarking purposes which represent real data, (ii) The two-step validation method (principal feature-based analysis and machine learning-based analysis) can be applied to any generated data for confirmation and validation purposes. To this end, this is considered as one of the main contributions of this research.

4.1.3.2 Properties of the traffic log files employed

Table 4.4 shows the number of domain names for each of the botnets in addition to the legitimate Alexa. The aforementioned domain name lists employed in this thesis are available publically[1].

Table 4.5 shows the properties of the log files introduced in this chapter as well. It

[1] *http://web.cs.dal.ca/~haddadi/data-analysis.htm*

Table 4.4: The number of domain names in the data sets.

data set	No. of Domain names
Alexa-D (NIMS)	500
Citadel-D (NIMS)	42
Zeus-D (NIMS)	684
Conficker-D (NIMS)	1537546
Kraken-D (NIMS)	5739
Torpig-D (NIMS)	130

should be noted that different combinations of these log files were formed and features extracted from them in order to create the data sets in this thesis. The data sets were then employed for the evaluation of the proposed approaches.

4.2 Summary

This chapter reviewed the data sets and the data sources utilized in this research. Some of the data sets were collected from public repositories while some were generated/captured in sandbox-controlled environments in the Dalhousie University NIMS lab. Table 4.5 shows the main characteristics of these data sets. Some of the generated data sets were evaluated and compared to public data sets to show how the behaviours represented by these data sets are similar/different from the public data sets. Data generation and validation process is one of the contributions of this thesis which not only discusses two methods to generate more botnet traffic for benchmarking purposes but also provides a mechanism to analyze and validate the generated data.

Table 4.5: The data specification.

Data set	**Type–SubType**	**Size**	**No. of Packets**	**Year**	**Source**	**Description**
Kraken-D (NIMS)	Malicious	22.3MB	119865	2013	NIMS lab	Domain-based
Torpig-D (NIMS)	Malicious	1.47MB	16277	2013	NIMS lab	Domain-based
Citadel-D (NIMS)	Malicious	9.53MB	79516	2013	NIMS lab	Domain-based
Zeus-D (NIMS)	Malicious	19.7MB	108947	2013	NIMS lab	Domain-based
Conficker-D (NIMS)	Malicious	1.49GB	15713662	2013	NIMS lab	Domain-based
Alexa-D (NIMS)	Benign	2.11MB	21210	2013	NIMS lab	Domain-based
Zeus-T1 (NIMS)	Malicious	104MB	525354	2014	NIMS lab	Zeus toolkit (v. 1.2.7.19)
Zeus-T1-W (NIMS)	Malicious	6.39MB	50202	2014	NIMS lab	Zeus toolkit (v. 1.2.7.19)
Zeus-T2 (NIMS)	Malicious	18.7MB	91189	2014	NIMS lab	Zeus toolkit (v. 2.1.0.1)
Zeus-T2-W (NIMS)	Malicious	7.96MB	65551	2014	NIMS lab	Citadel toolkit (v. 2.1.0.1)
Citadel-T1 (NIMS)	Malicious	40.4MB	167303	2014	NIMS lab	Citadel toolkit (v. 1.3.5.1)
Citadel-T1-W (NIMS)	Malicious	20.5	112954	2014	NIMS lab	Citadel toolkit (v. 1.3.5.1)
Zeus (Snort)	Malicious	5.85	6995	2010	Snort VRT lab	–
Zeus (NETRESEC)	Malicious	5.34MB	7453	2012	NETRESEC repository	–
Citadel (NETRESEC)	Malicious	1.61MB	15239	2012	NETRESEC repository	–
Cutwail (NETRESEC)	Malicious	6.16MB	24328	2012	NETRESEC repository	–
Kelihos (NETRESEC)	Malicious	1.21MB	13961	2013	NETRESEC repository	–
Conficker (CAIDA)	Malicious	183GB	39384872	2008-2009	CAIDA	Conficker version A and B
ISOT (Uvic)	Malicious and Benign	10.6GB	55916710	2010	University of Victoria	–
ISP (WiSNet)	Benign	2.17GB	28152476	2011	WiSNet research lab	–
ETP (LBNL)	Benign	10.3GB	56076228	2004-2005	LBNL lab	–
Kelihos (CVUT)	Malicious	409MB	1930987	2013	CVUT malware repository	Botnet-3 capture
Zeus (CVUT)	Malicious	422MB	2661214	2013	CVUT malware repository	Botnet-5 capture
Neris (CVUT)	Malicious	1.13GB	2629167	2011	CVUT malware repository	Botnet-42, 43, and 50 captures
ZeroAccess (CVUT)	Malicious	59.2MB	206494	2013	CVUT malware repository	Botnet-28 capture
NSIS (CVUT)	Malicious	281MB	352266	2011	CVUT malware repository	Botnet-53 capture
Virut (CVUT)	Malicious	109MB	440625	2011	CVUT malware repository	Botnet-54
Rbot (CVUT)	Malicious	74.92GB	75861170	2011	CVUT malware repository	Botnet-44, 45, 51, and 52 captures

Chapter 5

Application Data Analysis: Feature extraction

Monitoring network traffic at the application level, specifically the DNS level, provides a suitable solution for detecting botnet attacks because, in addition to its many legitimate uses, DNS can be used by botnets as well for managing their infrastructure. In a typical botnet, for example, the infected computer may locate the C&C server by querying a list of domain names which are supplied at the time of the infection or after. The C&C server will instruct the infected host to engage in malicious activities, such as data ex-filtration, denial of service attacks or serving spam, without user's knowledge. The list of domain names provided to the victim host is large enough so that it cannot be blacklisted manually or at the firewall level. Thus, to create a long list of domain names, attackers usually generate a list of algorithmically built domain names which exhibit structural and syntactical anomalies compared to regular domain names. Therefore, it is possible to detect botnet C&C activity by monitoring high volume access to unusually-structured domain names.

Most of the existing works in this field employ DNS network analysis [100, 122, 116], and some works combine such an approach with domain name lexical analysis [135, 149, 53, 105]. The traffic analysis approaches require a considerable amount of traffic in order to identify specific behaviours, which is a resource consuming process. In contrast, it appears that all of the lexical analysis approaches employ the specific pre-defined features of the domain names for classification purposes. This thesis explores the possibility of detecting botnets domain names with two methods, one based on *a priori* information and the other one with no *a priori* information in order to determine what would be gained and/or missed with either of the methods. In doing so, the first method employs the features of the domain names extracted from the a priori information while the second method investigates the possibility of using the raw domain names as an option for detection purposes.

In the first method, specific features of the domain names are defined based the

1	Domain starts with 'www'
2	Total sub-domain length: number of characters in all sub-domains (minus the dots)
3	Number of sub-domains: number of sub-domain blocks.
4	Maximum sub-domain length: the largest sub-domain block length
5	10+ character sub-domain count: number of sub-domains longer than 10 chars
6	1 character sub-domain count: number of sub-domains with one char
7	Contains IP: A Boolean flag. Whether there exists four sub-domain blocks between 0-255, following each other.
8	Alphabetic ratio: Num. of alphabetic characters in all sub-domains divided by the character count
9	Hexadecimal ratio: Num. of hexadecimal digits (A-F, a-f, 0-9) divided by the character count
10	Standard deviation of sub-domain lengths
11	Non-alphanumeric ratio: Number of non-alphanumeric characters
12	Contains imbedded TLD, if the sub-domains contain any items in the Mozilla suffix list
13	Contains imbedded file extension
14	Number of alphabetic to non-alphabetic and vs. transitions
15	Core-domain length
16	Core-domain alphabetic character ratio
17	Core-domain alpha to non-alpha and vs. transition count

Table 5.1: The Domain name's feature set definition.

on *a priori* information. Heuristics-based feature extraction was employed on the components of a given domain name. Each domain name has three components: (i) a top level domain (TLD), (ii) a core domain, and (iii) a sub-domain. For example, in *mail.google.com*, *com* is the TLD, *google* is the core domain, and *mail* is the sub-domain. Given that the TLD names are distinctive and fixed, the other two components (core domain and sub-domain) are used only in feature extraction. Consequently, in this work, for each domain name, a set of 17 features is extracted (Table 5.1). The first 14 features are based on the sub-domain component and the last three are based on the core-domain component. Overall, the features aim to highlight the structural anomalies in the domain names (as seen in the literature)– in other words, those domain names that are not likely to have been typed by a person. To detach the top-level domain and to extract feature # 12, the Mozilla suffix list is employed [119]. Appendix A shows some samples of botnet domain names and legitimate domain names.

However, identifying the correct set of attributes which represent the domain name characteristics is challenging, specially given that the DGAs are moving targets. The second method proposed here is an evolutionary computation technique based on SBB [67]. As discussed in Section 3.1.8, SBB is a form of linear GP with a co-evolutionary architecture. Three populations are co-evolved in this algorithm: A point population, a team population and a learner population. The learner population represents a set of learners, which associate a GP-bidding behaviour with an action. The team population comprises a set of learners and finally the point population denotes a subset of training data exemplars. Evaluating a team on the points, all of the teams learner programs are executed while only the learner with the highest bid suggests its action as the team's action. The bidding procedure employs linear GP in addition to a sigmoid function to standardize the bid values between zero and one.

The second method employs a modified version of SBB, called Stateful-SBB (Stateful Symbiotic Bid-Based Genetic Programming), to classify malicious vs. non-malicious domain names by using only the string sequence (raw) of a domain name. In other words, it is exploration of how far classification performance can be pushed without any *a priori* knowledge about the characteristics of the domain names, *i.e.* without any lexical features or packet level features. Hence, in this case the input exemplars would be variable length character-based domain names.

In the original SBB classifier bids are based on all the attributes of the points as with the known classifiers such as C4.5, while the exemplars should all have the same number of attributes. However, given a dataset of variable length domain names, neither the original SBB nor most of the other known classifiers can be used. Therefore, for this approach, the SBB interface is changed to bid based on each character of a domain name. The new model keeps the state information for each exemplar, hence it is being called the Stateful-SBB. Figure 5.1 summarizes the team-learner interaction mechanism in the Stateful-SBB. Data set exemplars in the new layout are the variable length domain names. Features are the ASCII codes of the domain names' characters. A team receives a complete domain name but it passes the domain name to its team of learners character by character. Each learner then provides a bid per character as opposed to per exemplar. The learner's action that outbids the others is assigned as the team output for that specific character. Domain name characters are related to each other and are not independent. To achieve the correlation of characters reflected in the bidding process, learners reset their registers only at the beginning of each specific domain name, not for every bid process on every character in a domain. At the end of each domain name (when all the learners bid on the entire domain name characters), a team will have a sequence of the best learner actions as the team output sequence. Finally, the team will decide on its final action for that specific domain name.

Given that the domain names are a composition of related characters in a meaningful order, there are some important questions that need to be answered. Is it necessary to use all the domain name characters in the learning process to have a relatively good output label? Should the combination of all actions in the sequence be considered or just the last one? The answer to these questions can result in different policies for the final action selection of a team, which are discussed in a previous analysis [89]. Hence, seven different final action selection policies were evaluated. The methods are summarised briefly below.

- **The last-best action in the action sequence** which returns the learner's action on the last character of the input domain name that out bids other learners of the team learner set, called 'Last-best'. As the learners would not reset their registers during the biding process of a domain name, the last action of the sequence is affected by all the actions in the sequence in which all the domain characters are considered.

- **The most frequent action in the action sequence of a team on all characters of a domain name.** As the 'Last-best' heuristic team might not always reflect all the best actions of the bidding process, this selection method, called 'Most-freq', chooses the most frequent action of a team action sequence to be the team final decision.

- **Last output action of a team after 50% of the domain characters are used in training.** A meaningful sequence of characters forms a domain name. So, a team might be able to gather enough information about a domain name and select its action after being trained with some part of the domain name. In this experiment, the team's last action of the action sequence is selected after it has been trained with half of the domain name.

- **Last output action of the best learner after 60%, 70%, 80% and 90% of domain name characters are used in the learning process.** As with the previous approach, 60%, 70%, 80% and 90% of the domain characters were tested.

As expected, the method that bids only on the first half of the domain names (50% method) had the lowest training time (because it uses fewer characters for the training). The 60%, 70%, 80% and 90% had lower training times, and after those, the two other methods ('Last-best' and 'Most-freq') had the highest training time. Based on the results in [89], the 'Last-best' method has been employed for action selection in this thesis.

5.1 Pruned Stateful-SBB

Post evolution, all the generated teams in the learning procedure are evaluated on the training data set and the one with the best performance selected as the 'champion' solution. The performance metric assumed for this purpose takes the form of the class-wise average detection rate (Eqn. (3.8) Section 3.1.9). The solution team is a combination of a set of learners with their corresponding GP instructions. In these evaluations, the maximum program size is set to 48. Thus, each learner in the solution can have a maximum of 48 instructions including the non-effective code, called introns. Given that introns were found to count for between 60% to 90% of instructions in a linear GP [59], intron removal is employed to reduce the complexity of SBB [93]. A more detailed explanation of the algorithm can be found in [89, 93].

5.2 Domain name lists employed

In this approach in which the focus is on detecting botnets' malicious domain names, publicly available lists (from legitimate resources) of C&C domain names are collected, representing various botnets and legitimate domain names. The Conficker, Zeus and Citadel domain name lists are obtained from the Bonn University [5], ZeusTracker [25] and DNS-BH project [7] blocklists. Moreover, Kraken and Torpig domain name lists

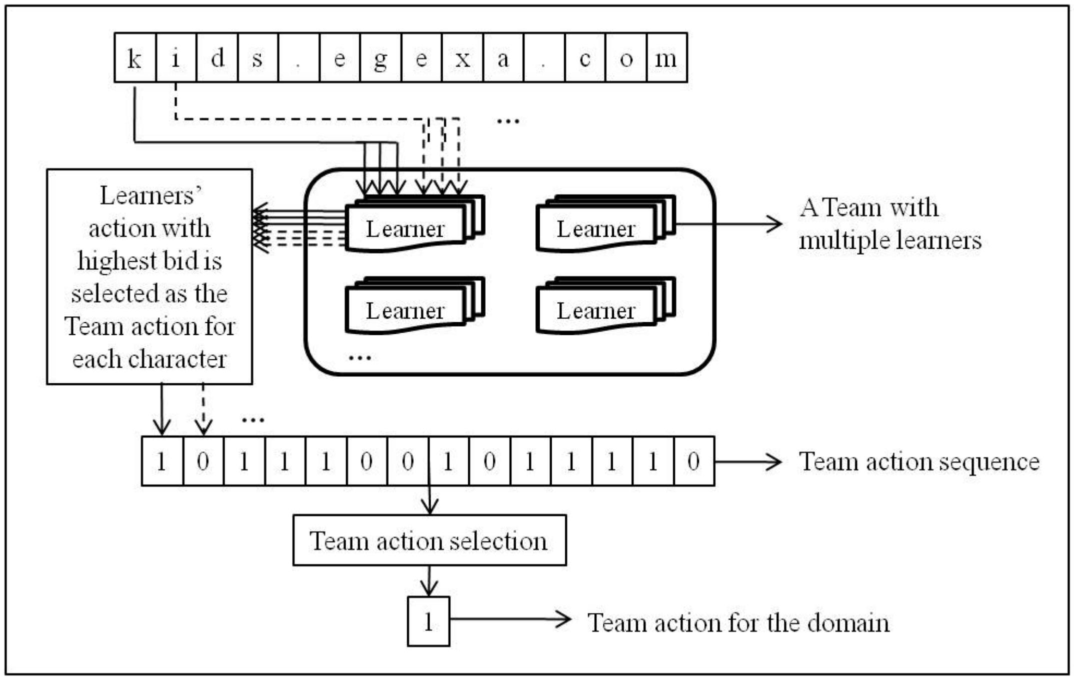

k i d s . e g e x a . c o m
Learners' action with highest bid is selected as the Team action for each character
Learner
Learner
Learner
Learner
A Team with multiple learners
1 0 1 1 1 0 0 1 0 1 1 1 1 0
Team action sequence
Team action selection
1
Team action for the domain

Figure 5.1: The Stateful-SBB mechanism.

are collected from DVLabs [27], DNS-BH [7], Damballa [103] and Twitter [142]. These domain name lists are representatives of botnets' malicious automatically-generated domain names. To represent legitimate domain names, 500 Alexa legitimate domain names were selected and used [2].

5.3 Empirical evaluation and Results

As demonstrated in this chapter, two methods are proposed for the application data analysis approach. The first method detects the botnets' malicious domain names based on a pre-defined feature set while the second method employs raw string format domain names (using the characters of the domain names as features). To evaluate the proposed approach, three classifiers were employed: string kernel SVM (SSK and SSK-LP), Stateful-SBB and C4.5. The reasons behind choosing the aforementioned classifiers are: (i) the C4.5 classifier was the best performer for detecting malicious automatically-generated domain names from legitimate ones using the first method in a previous analysis (compared to Naive Bayes, AdaBoost, SBB) [89] and (ii) string kernel SVM can be a good baseline given that it can be applied to the same string-based dataset as Stateful-SBB.

Classifier	Score	Kraken		Conficker		Alexa		Complexity	
		TPR	FPR	TPR	FPR	TPR	FPR	Time (sec)	Solution
Stateful-SBB	98.3%	98.4%	0.4%	99.3%	0.1%	97.3%	1.5%	2227.64	674
Pruned Stateful-SBB	98.3%	98.4%	0.4%	99.3%	0.1%	97.3%	1.5%	2227.64	116
SSK	99.6%	100%	0%	98.9%	0%	100%	0.1%	431.53	1094
SSK-LP	99.4%	100%	0%	98.9%	0.1%	99.3%	0.1%	166.2	805
C4.5	93.7%	100%	1.4%	96.4%	1.0%	84.7%	0.4%	0.06	19

Table 5.2: Domain name-based analysis results of the Conficker, Kraken and Alexa: unbalanced datasets.

For the C4.5 classifier, seventeen features are extracted from each domain name (defined in Table 5.1) and then a domain name label is added to the set as the last feature. By contrast, SSK, SSK-LP and Stateful-SBB can use raw domain names in string format without requiring any *a priori* known feature set. Hence, for SKK and SKK-LP each testing and training data exemplar has two attributes: domain name (string format) and a class label. Stateful-SBB, however, requires the characters of the domain names to be presented in their ASCII code in addition to their class labels.

For the first experiment, 5739, 934, and 500 domain names were chosen from the Kraken, Conficker and Alexa domain name lists, respectively. The Kraken and Conficker domain names were selected from the lists that are generated based on one version of the DGA algorithms of these botnets from [7, 27]. In other words, the domain names of each class are generated by one DGA algorithm, and therefore by a specific version of botnet. After creating the this unbalanced dataset, it is then divided into two parts (training and testing) based on: (i) an almost 30-70% breakdown for testing and training, respectively. This breakdown is specifically used here given that all classifiers should have been provided with the same data sets and SBB only accepts separate testing and training input files. It should be noted that in cases SBB and Stateful-SBB are not included in the experiments and evaluations, 10-fold cross validation is used instead; and (ii) keeping enough samples of each class in both of the datasets. C4.5, SSK, SSK-LP, Stateful-SBB and pruned Stateful-SBB were run several times, changing different parameters (C parameter and pruning option for C4.5, Lambda and subsequence length for SSK and SSK-LP and finally maximum team and point population sized for Stateful-SBB). Table 5.2 presents the best result for each classifier on the unbalanced data sets. Given that the goal is to have a low complexity with a high Score which, most of the time are conflicting, the decision as to the best run needs to be taken in consideration of trade-offs between these two criteria.

To prevent any changes to the individuals in the learner population during evolution, the non-effective instructions were removed after the training. Therefore, the training times of Stateful-SBB and Pruned Stateful-SBB are the same, as is indicated in Table 5.2. Given that only the introns were removed from the learners' instruction set, the TP and FP rates are also not changed, hence the Score. Since training is a one time process but the solution will be used several times, the goal of pruning Stateful-SBB was to reduce the solution complexity which will speed up the botnet malicious domain name detection process in real-time. The results indicate that using the pruned version of Stateful-SBB reduces the complexity by 83% which is a significant improvement. As well, the SSK-LP reduces the complexity of SSK by 27%. These results suggest that Pruned Stateful-SBB is a promising classifier as an automatically-generated malicious domain name detector, which can achieve high accuracy with no *a priori* feature set.

To analyze the classifiers further, another set of experiments was performed. For this round, 130, 642, 684, 16007, 17043 and 500 domain names were selected from the Torpig, Citadel, Zeus, Kraken, Conficker and Alexa domain name lists, respectively. In this case, the domain names of the Conficker, Citadel and Zeus botnets were from different version of the botnets' DGA but Torpig and Kraken were still generated with one version of DGA. Using these domain names, binary datasets were created in order to classify the botnet DGA-generated domain names from the Alexa legitimate domain names. The datasets were then divided into two parts (training and testing) based on the criteria mentioned earlier in this section. Given the different number of domain names for each botnet, these datasets are unbalanced. Table 5.3 shows the classification results of this experiment. Although, pruned Stateful-SBB still outperformed the other classifiers based on the Score and FPR, the average Score is around 80% which is not as good as the 98% in Table 5.2. Looking into the Kraken and Conficker botnets specifically, the results show that pruned Stateful-SBB, SSK and SSK-LP could still detect the Kraken domain names very well while the Conficker results dropped in the second experiment. This is caused likely by the fact that the Kraken domain names are only increased by the number but the DGA behind the scene is still the same. In contrast, Conficker has evolved from version A to versions B and C over time and therefore the domain names included in the new Conficker list

	Data Set	DR	Score	Botnet		Legitimate		Complexity	
				TPR	FPR	TNR	FNR	Time (sec)	Solution
C4.5	Citadel	79.86%	79.65%	81.5%	22.2%	77.8%	18.5%	0.11	9
	Conficker	97.75%	60.9%	100%	78.2%	21.8%	0%	0.48	11
	Kraken	68.675%	98.15%	100%	60.8%	39.2%	0%	0.38	7
	Zeus	77.45%	78.05%	74.1%	18%	82%	25.9%	0.13	17
	Torpig	84.13%	67.5%	39.2%	4.2%	95.8%	60.8%	0.08	9
Pruned Stateful-SBB	Citadel	88%	88.48%	84.3%	7.3%	92.7%	15.7%	6188.53	110
	Conficker	92.5%	85.95%	92.9%	21%	79%	7.1%	4989.38	122
	Kraken	100%	100%	100%	0%	100%	0%	4386.06	31
	Zeus	81.4%	81.7%	79.5%	16%	84%	20.5%	8286.9	111
	Torpig	85.7%	81.5%	74.4%	11.3%	88.7%	25.6%	6083.75	93
SSK	Citadel	81.88%	83%	73.8%	7.8%	92.2%	26.2%	48.27	827
	Conficker	98.24%	69.7%	100%	60.6%	39.4%	0%	1695.68	1196
	Kraken	100%	100%	100%	0%	100%	0%	3594.98	1089
	Zeus	77.45%	77.7%	76.2%	20.8%	79.2%	23.8%	49.6	859
	Torpig	83.97%	61.15%	22.3%	0%	100%	77.7%	2.56	385
SSK-LP	Citadel	81.09%	82.25%	72.5%	8%	92%	27.5%	6.6	802
	Conficker	97.83%	62.1%	100%	75.8%	24.2%	0%	138.12	1188
	Kraken	100%	100%	100%	0%	100%	0%	482.8	998
	Zeus	83.65%	60.4%	20.8%	0%	100%	79.2%	0.58	352
	Torpig	76.35%	76.65%	74.7%	21.4%	78.6%	25.3%	0.58	362

Table 5.3: Classification results of the botnet domain name-based datasets: unbalanced.

are generated by different DGAs. There are different possibilities for lower detection rate in the second experiment.

(i) Through botnet evolution, DGAs have evolved and therefore, newer botnets are better capable of generating more legitimate domain names. In other word, newer botnets like Citadel, Zeus and Conficker B/C are performing better in terms of the DGAs.

(ii) The datasets are unbalanced and there might not be enough samples in the legitimate and/or botnet classes of these botnets. Hence, the classifiers overlook the minority class in favour of the majority class.

(iii) It is possible that 130 domain names for a botnet like Torpig is just not enough to detect the classification patterns by the classifiers. Unfortunately, since the Torpig botnet discontinued and did not evolve and grow into newer botnets, more data samples are not available.

In this regard, a third set of experiments with balanced datasets was created to investigate the effect of unbalanced datasets. The Table 5.4 results for the balanced binary datasets indicate three conclusions.

(i) C4.5, SSK and SSK-LP showed increased performance when the datasets were changed to balanced. However, pruned Stateful-SBB still outperformed the other three classifiers.

(ii) In contrast, the pruned Stateful-SBB performed slightly better on unbalanced datasets (except for the Conficker botnet). This shows that Stateful-SBB can do

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
C4.5	Citadel	81.9%	78.6%	15.4%	84.6%	21.4%	0.1	7
	Conficker	64.9%	78.6%	48.8%	51.2%	21.4%	0.11	15
	Kraken	86.9%	78%	4.2%	95.8%	22%	0.1	7
	Zeus	77.9%	72.6%	16.8%	83.2%	27.4%	0.12	13
	Torpig	71.15%	71.5%	29.2%	70.8%	28.5%	0.07	7
Pruned Stateful-SBB	Citadel	87.3%	86%	11.3%	88.6%	14%	4592.2	96
	Conficker	87.7%	84%	8.6%	91.4%	16%	5423.9	133
	Kraken	100%	100%	0%	100%	0%	6982.17	24
	Zeus	81.7%	86.7%	23.3%	76.7%	13.3%	4521.53	69
	Torpig	74.4%	64.1%	15.4%	84.6%	35.9%	149.151	37
SSK	Citadel	80.4%	97.2%	6.4%	93.6%	32.8%	44.78	771
	Conficker	87.5%	97.6%	22.6%	77.4%	2.4%	17.48	738
	Kraken	100%	100%	0%	100%	0%	24.64	419
	Zeus	77.3%	73.6%	19%	81%	26.4%	38.6	764
	Torpig	73.46%	65.4%	18.5%	81.5%	34.6%	0.63	240
SSK-LP	Citadel	78.4%	62.8%	6%	94%	37.2%	8.39	757771
	Conficker	87%	97.8%	23.8%	76.2%	2.2%	3.92	707
	Kraken	100%	100%	0%	100%	0%	4.21	388
	Zeus	75.2%	70.8%	20.4%	79.6%	29.2%	6.59	755
	Torpig	73.08%	64.6%	18.5%	81.5%	34.5%	0.19	238

Table 5.4: Classification results of botnet the domain name-based datasets: balanced.

better when it gets to choose the point pareto-front from the larger pool of data samples in the unbalanced datasets.

(iii) In both the balanced and unbalanced experiments, Statefull-SBB showed an overall better performance in terms of FPR. FPR is specifically important in malicious domain name detection systems since mis-classifying legitimate domain names as malicious ones (which ends in a blocking the domain names) can interfere with a genuine website activity.

5.3.1 Summary

The domain name system (DNS) is an essential component of the Internet. On a daily basis, many Internet users utilize this protocol to be able to connect to the destination machine. As it is expected to be used by all legitimate users and applications, generally there are fewer inspections, restrictions and filters on it. Botnets have been taking advantage of this open component to accomplish their malicious operations since 2008. To overcome the single point of failure of static C&C servers and evade static blacklists and firewalls, botnets employ DNS-based methods frequently to generate new automatic domain names. As reviewed in this chapter, several approaches for the early detection of botnet Command and Control (C&C) activity were investigated by monitoring the DNS traffic. In other words, the investigated approaches can differentiate malicious automatically-generated domain names from legitimate domain names in a very early stage. Therefore, instead of monitoring user behaviour to flag any abnormal communication, these systems can predict whether a queried domain name is suspicious and not generated by human users. Hence, not only the infected machines which query such domain names can be detected, but also the communication can be blocked at the first attempted access.

In this chapter the possibility of detecting botnet domain names with two methods was explored, one based on a priori information and the other one with no *a priori* information. Stateful-SBB is an evolutionary computation-based solution that was designed to reveal the malicious automatically-generated domain names without *a priori* knowledge. As well, SVM String Kernel classifier (SSK and SSK-LP) was used for comparison. Stateful-SBB and the SSK classifier investigate the possibility of using the raw domain names as an option for detection purposes while the other approach focuses on extracting useful features from the domain names in order to highlight the abnormality.

The preliminary results show that the Stateful-SBB-based system performs as well as the other classification approaches when the dataset consists of the domain names generated by one DGA (with a Score of 99%), without *a priori* knowledge. However, botnet DGAs have evolved over time and as its result, the collected domain name list of a specific botnet may include different types of domain names. Although all of these domain names are malicious, some are more similar to legitimate domain names given a more complex and enhanced DGA. In a secondary evaluation based on more complex domain names, Stateful-SBB outperformed the other methods. However, the Score dropped to about 80%. Considering the fact that botnets can generate huge numbers of C&C server domain names every day (*e.g.* Conficker C generated 50,000 domain names each day), early detection rate of 80% can narrow the search scope of malicious behaviour significantly.

Given that the approach suggested in this chapter is applicable only to botnets which utilize DNS as a part of their structure, a generic network data analysis-based system would be a better choice for detecting botnet malicious behaviour. Hence, in the upcoming chapters other tools and approaches are investigated in order to design an early warning botnet detection system based on network data analysis.

Chapter 6

Network Data Analysis: Feature extraction

Although the proposed application data analysis early warning system in the previous chapter could limit the search scope of detecting botnet malicious behaviours by detecting the C&C server domain names, there are scenarios in which this system cannot be used: (i) the botnet does not use DNS for communication (*e.g.* IRC botnets), (ii) the botnet may use IP-fluxing or fixed IP addresses rather than domain-fluxing, (iii) the traffic being analyzed does not include DNS traffic or the payload information is not accessible (where the domain name is located) and (iv) the traffic is encrypted (for example when botnets use methods similar to DNScrypt to encrypt the DNS communication). Hence, for this approach an early warning network data analysis-based system is proposed. Among the traffic analysis-based systems in the literature, some focus on specific types of botnets while others attempt to build a general model for more than one botnet. In early 2000, most of the proposed systems were focussed specifically on botnets utilizing IRC (*e.g.* [139]) while recent research has focussed more on P2P- and HTTP-based botnets [106, 156, 90]. Hence, the aim in this thesis is to cover several types of botnet.

Given that botnets use automatic update mechanisms, botnet monitoring and detection approaches need to be active and continuous as well. Potentially, this could enable them to learn new patterns and adapt to changes in botnet evolution. Hence, machine learning techniques (*i.e.* classification and clustering) are among the highly employed techniques in this field. The clustering and classification techniques used for traffic analysis require the network traffic to be represented in a meaningful way to enable automatic pattern recognition. Thus, an important component for such systems is extracting those features (attributes) from the network traffic. However, feature extraction has always been a challenge. To this end, different botnet detection and analysis systems have come up with their own sets of features to represent the network traffic consisting of network packets. Network packets include two main parts: (i) a packet header, which includes the control information of the protocols used on the network and (ii) a packet payload, which includes the application information used on the network. In this case, some detection and analysis systems only use network packet headers as the basis for their features [61, 156, 90] while others take advantage of packet payloads [118, 147, 151]. Among the group using packet headers, flow-based feature extraction methods are highly employed according to the recent literature [153, 61, 71, 145]. In such methods, communication packets are aggregated into flows and then statistics are calculated. Systems that generate flows and extract such features are called flow exporters. Given that botnets employ encryption techniques to avoid the detection systems that analyze the communication information embedded in the packet payload, flow exporters can be very effective since they summarize the traffic utilizing only network packet headers. In this approach a flow-based botnet detection system was developed focussing on the critical phase of such approaches which is feature extraction and investigating the magnitude of the effect of flow exporters in botnet traffic detection and analysis systems. Thus, six open source flow exporters (Maji, YAF, Softflowd, Tranalyzer, Argus and Netmate) along with several highly employed machine learning techniques are utilized [C4.5, SVM, Naive-Bayes, Bayesian Networks and Artificial Neural Networks (ANN)].

Other important aspects which should be investigated and analyzed while designing and evaluating a traffic analysis-based early warning system include the following.

1. **Type of network traffic to be analyzed.** As discussed in Section 2, botnets employ different protocols and techniques which are part of their signatures. Hence, it is the effect of these protocol filters which are studied. For this purpose, the focus is on HTTP botnets given that the availability of a good number of data sets in this category. Three sets of experiments are conducted: the first set of experiments studies the analysis and classification of botnet traffic using all of the traffic flows; the second set performs the same study using only the HTTP traffic flows (employing an HTTP protocol filter); and the third set uses only the DNS traffic flows utilizing a DNS protocol filter.

2. **Feature representation.** Most of the known ML algorithms (e.g C4.5, KNN and SVM) accept only attribute sets with numeric types. Therefore, almost all of the classification works in the literature employ specifically numeric flow

features. Flow exporters, however, have non-numeric representation for some of the features. In this category, for instance, all of the eight TCP flags (which have binary values) are usually combined and presented as a hexadecimal TCP flag feature by the flow exporters. By contrast, implementations of the classifiers (like the implementations in Weka) usually interpret this feature (Hexadecimal type) as a string or nominal value. Therefore, it is usually removed from the feature set. Having said this, TCP flags were shown to be used by malware in ways that were not intended for legitimate use [108]. Consequently, the effect of three different TCP flag representations were investigated (numerical, nominal and binary). In the numerical representation the hexadecimal TCP flag value was converted into a numerical (integer) value whereas in the binary representation the hexadecimal value is broken down into eight separate flag values in binary format. These flags are: Congestion Window Reduced (CWR), ECN-Echo (ECE), Urgent (URG), Acknowledgement (ACK), Push (PSH), Reset (RST), Synchronize (SYN) and Finish (FIN) flags. Finally, in the nominal representation a set of hexadecimal flag values utilized in a data set is prepared and used as a nominal set of possible values for the TCP flag feature.

3. **Time generalization– the effect of botnet behaviour evolution.** Based on the fifth phase of the botnet lifecycle (*i.e.* maintenance and update phase) it is clear that botnets upgrade their methodology to defeat detection systems. Designing a botnet detection system that can cope with such changes is challenging. On one hand, using ML-based detection systems for this purpose has the advantage of being able to re-train on a new botnet setting with minor human expert involvement. On the other hand, it is important to know how effective an older trained classifier can perform facing the same botnet with the new setting or behaviour. Therefore, the performance of the ML-based botnet early warning systems was investigated over a period of time.

4. **Normal behaviour representation.** To build a classifier which can differentiate normal behaviour from botnet malicious behaviour, the ML algorithms require data samples from both classes. For this purpose, several public normal data sets have been used in the literature. The question is, how does the choice

of a normal data set and the way we represent normal behaviour to the classifier affect the performance evaluation (if at all)? It would appear that no research has been done on this issue. This chapter will investigate the effect of normal behaviour representation.

6.1 Empirical evaluation

As discussed earlier, the proposed approach develops an early warning network data analysis botnet detection system using flow-based features. Although, this approach has been explored by other researchers in the literature, none of them seems to have investigated the effect (if any) of flow exporters on the representation of network traffic in botnet detection while they have all introduced the preferred flow-based feature set for specific type(s) of botnets. Hence, the goal in this approach is not to introduce a network data analysis-based detection system but to investigate the effect of the flow features extraction process and the type of network traffic employed (*e.g.* HTTP flows only). To achieve this, five open source flow exporters were employed on eight different botnet traces using five different machine learning algorithms. In this section the effect of five flow exporters (Maji, YAF, Softflowd, Netmate and Tranalyzer) was explored using C4.5, SVM, ANN, Bayesian Networks, and Naive Bayes as the classifiers for the botnet traffic identification. Eight different traffic traces for the Citadel, Zeus, Conficker, Kelihos and Cutwail botnets were used for this analysis.

The aforementioned classification algorithms were selected in this analysis, because of their high performances as reported in the literature regarding network traffic classification, specifically in botnet detection [139, 61, 156, 153, 90, 134]. The Weka [146] implementation of these classification techniques was used since it is a well-known open source tool in this field. In general, a machine learning classifier requires a number of steps. First, a matrix of instances (in this chapter, instances are flows) versus features (attributes) is needed to describe the data set. A vector of features describes each instance (flow) in a given traffic file. The features are used as values to quantify different characteristics of a flow such as the average packet size or the minimum inter-arrival time. Second, a label (ground truth) is provided for each flow, which is the class description. Given that for the analysis in this section

Alexa-D (NIMS) is employed as a representative of legitimate data, the label for a flow in the Alexa traffic file is 'normal' whereas the botnet traffic files are labelled 'botnet'. Finally, a classifier needs to be trained using a data set. This is called the training phase. This phase produces a solution as the output. Then this solution can be verified on a test data set (unseen instances). To evaluate these classifiers on the traffic flows, first a balanced training data set is formed by selecting randomly (uniform random selection) from the non-malicious flow data set as well as from each of the malicious data sets. Then classifiers are run on each balanced training data set using 10-fold cross-validation to avoid any data set biases that might affect the results.

Although selecting a suitable feature set and a proper machine learning algorithm is the principal phase of designing a flow network analysis-based system, there are other factors to be considered. The next phases of the evaluation further analyze the performance of the designed system and its effects of feature representation, the time generalization issue and biases cased by the normal data set selection (if any).

6.1.1 Results

After analyzing the domain-based NIMS-generated traffic, it was observed that successful DNS responses were received from 95% of the Alexa, 75% of the Zeus-D (NIMS), 25% of the Citadel-D (NIMS) and 1% of the Conficker-D (NIMS) domain names. In a real-life botnet communication, a bot queries the C&C domain names provided by the DGA (or the botmaster) and only one of them is resolved as the live C&C server at the time of the communication. In Section 4.1.3.1 the generated data was confirmed using such an approach. Therefore, the combination of these resolved and unresolved domain names in the list employed by this analysis can be considered as representation of real botnet behaviour.

In the first step, the aforementioned five flow exporters are applied to the traffic captures. Figure 6.1 presents the number of extracted flows by each of the tools on all of the traffic log files. However, since the Conficker-D (NIMS) data (approximately 15 million packets) is much bigger than the other log files, only 0.5% of it was presented (about 10,000 flows for Softflowd) in order to have a readable Fig. 6.1. As shown in Fig. 6.1, the Tranalyzer, Maji and Softflowd tools export almost the same number of flows for all of the log files. Also, they provide the highest number of flows for any given log file. This is not surprising since all three of these tools are uni-directional flow exporters. On the other hand, YAF and Netmate did not show any consistent behaviour over the log files. This might be caused by the rules defined by the tools. For example, Netmate did not export any flow for Kelihos (NETRESEC) because in this log file, the number of out of order packets was higher than the acceptable threshold defined in Netmate. A comparison between the number of flows exported by uni-directional and bi-directional exporters shows that the number of exported flows by uni-directional exporters is almost twice as higher as the number of flows by bi-directional exporters for all of the log files except for Citadel-D (NIMS) and Zeus-D (NIMS). This shows that Citadel-D (NIMS) and Zeus-D (NIMS) have more one-way communications than the other files which can be caused by the number of unsuccessful connection requests or long (un-closed) connections that are split into multiple flows by the exporters according to the maximum lifetime threshold.

Table 3.1 shows the number of features supported by each of the exporters. All of features provided by the exporters/collectors were employed as inputs to the machine learning classifiers except the IP addresses, port numbers and any non-numeric features. The reason for this is that IP addresses can be anonymized whereas port numbers can be assigned dynamically. Employing such features may decrease the generalization abilities of the detection systems for unseen behaviours. Moreover, the presentation of non-numeric features may introduce other biases into the classifiers [115] which might effect the performance in a positive/negative way. More analysis is done on one of these features (the 'TCP-flag') in Section 6.1.1.2.

Figure 6.2 shows the classification results of the five classifiers on the traffic flows generated by the five flow exporters. These results demonstrate that the performance of a classifier does indeed change depending on which flow exporter is used. For example, on these data sets, it seems that each classifier works better if the flow exporter is Tranalyzer or Maji. In particular, the performance of C4.5 and ANN on Maji and Tranalyzer indicates that ANN did perform better (but not statistically significantly better) than C4.5. However, the C4.5 decision tree-based classifier performs not only competitively but also with considerable low time complexity, Table 6.1. Therefore, if the time criteria is an important factor in an environment, the C4.5

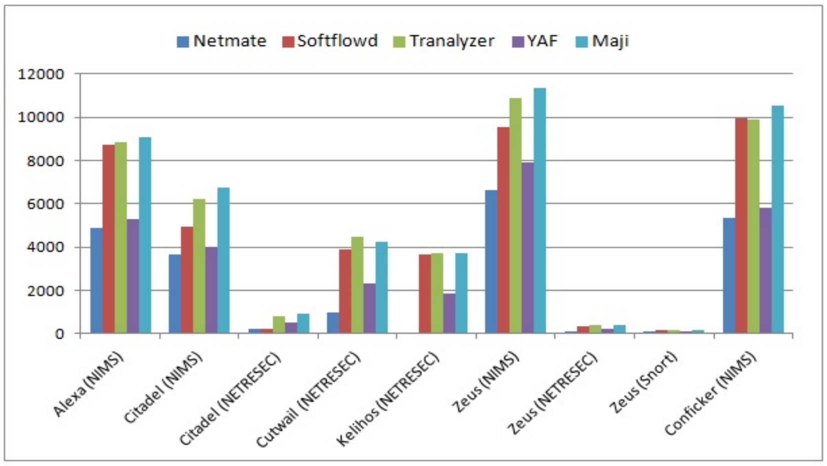
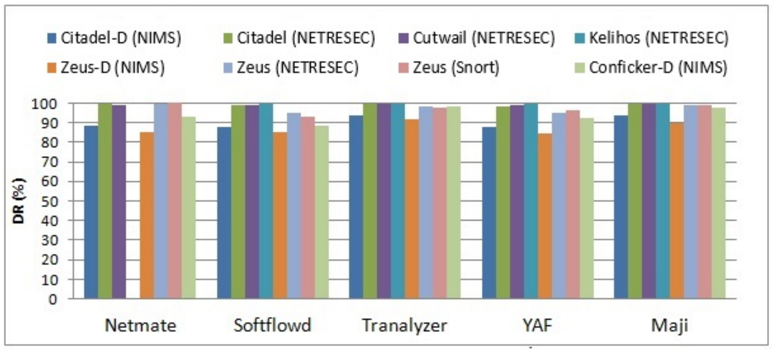


Figure 6.1: Number of Extracted Flows

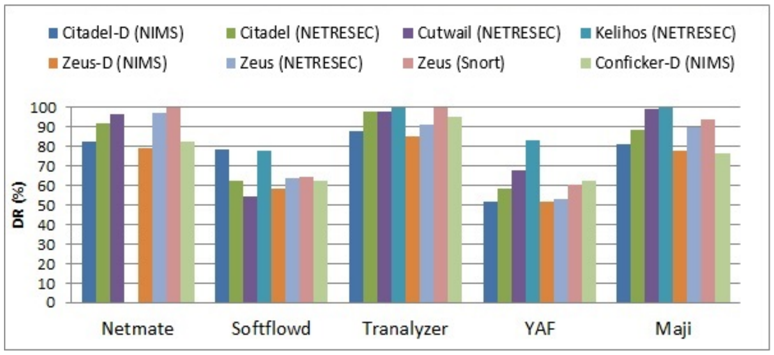
classifier has an advantage over ANN. Moreover, the C4.5 classifier has the ability of choosing the most appropriate features from all the features given to it. Such an ability enriches any analysis that can be done post classification. This enables the human expert (security analyst/system administrator) employing this system to have a better understanding of the solution and the botnet behaviour. These properties of the decision tree classifier along with the results imply that the C4.5 classifier seems to be is a better choice in terms of the performance metrics used and the data sets employed in this experiment.

While detection systems with high DR are important in understanding and identifying the behaviour of interest (in this case the specific botnet), the effect of FPR can be very important as well. In the case of botnet classification (detection), any normal behaviour that is mistakenly identified as botnet increases not only the false

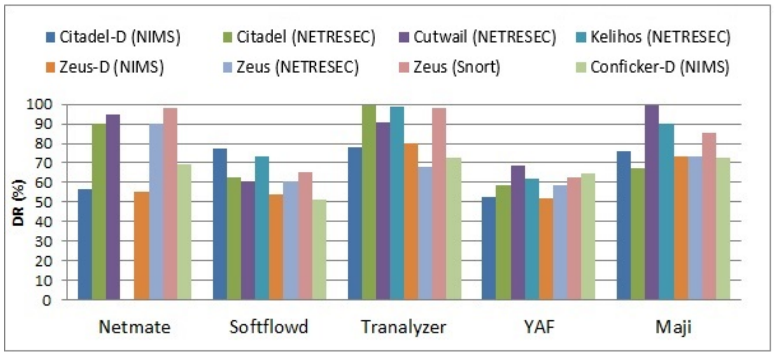


(a) C4.5

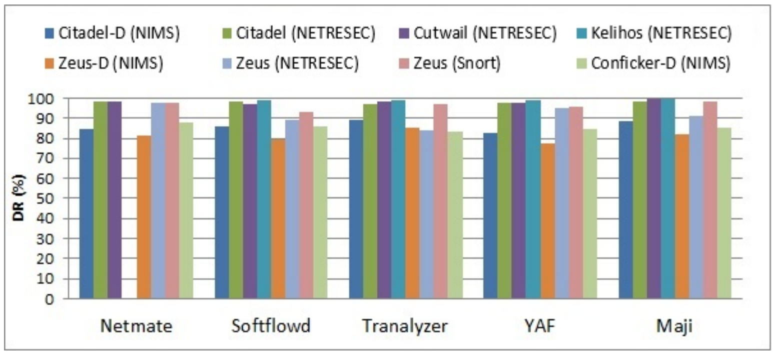
Figure 6.2: DR of all classifiers on the five flow exporters



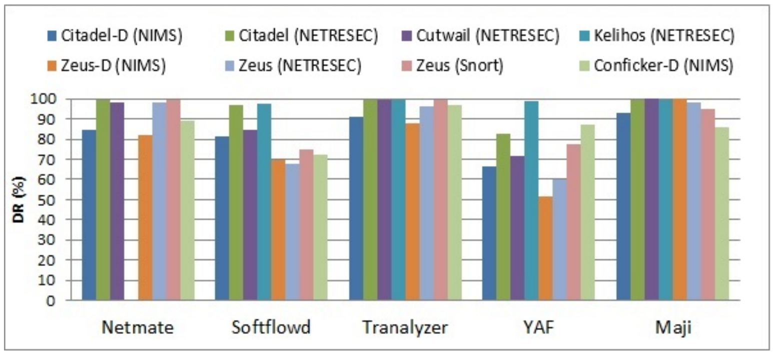
(b) SVM



(c) Naive-Bayes

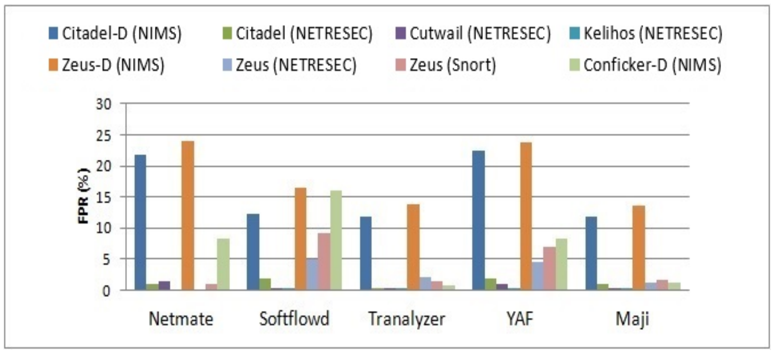


(d) Bayesian Networks

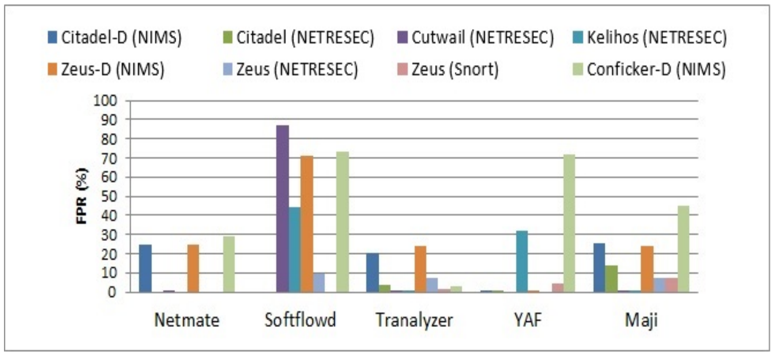


(e) ANN

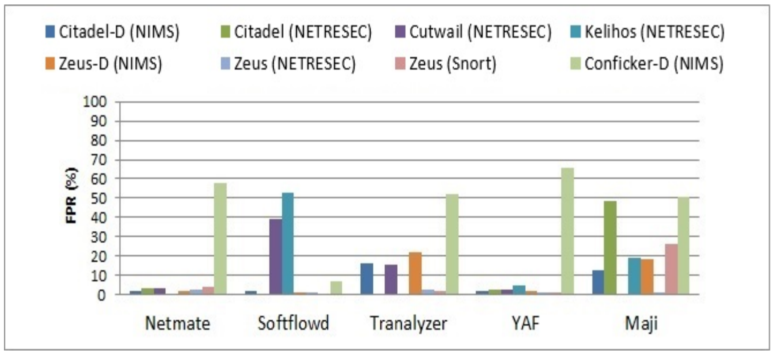
Figure 6.2: DR of all classifiers on the five flow exporters (Cont.)



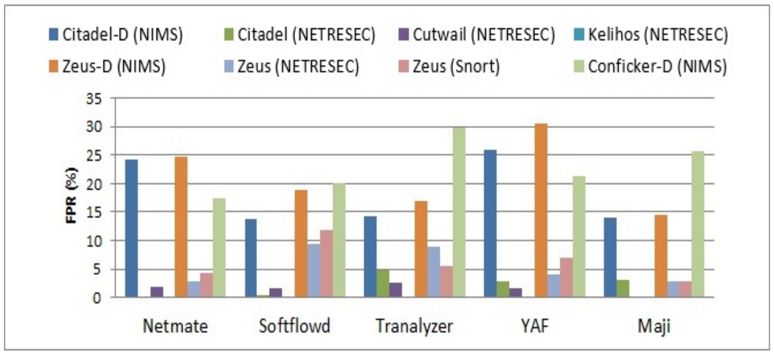
(a) C4.5



(b) SVM

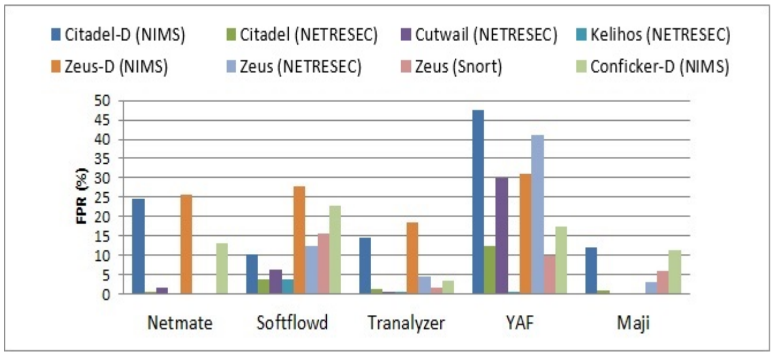


(c) Naive-Bayes



(d) Bayesian Networks

Figure 6.3: FPR of all classifiers on the five flow exporters



(e) ANN

Figure 6.3: FPR of all classifiers on the five flow exporters (Cont.)

Table 6.1: Average training time (in seconds)

Flow Exporters	**C4.5**	**ANN**
Netmate	1.42	0.82
Softflowd	0.30	21.60
Tranalyzer	1.42	156.68
YAF	0.06	4.1
Maji	0.82	75.88

positive (alarm) rate but also decreases the trust in the detection system. Therefore, the FPR of the five classifiers was analyzed as well as the effect (if any) of the flow exporters on this performance metric. Figure 6.3 shows the FPR of the five classifiers. As shown in the figure, the trend is the same for all classifiers in terms of FPR. The C4.5 and ANN classifiers were also the best performers in terms of minimizing the FPR among all the classifiers employed in this research on all the exporters and data sets. These results demonstrate that the flow exporter used also has an effect on the FPR rate obtained for all classifiers. On these data sets, the lowest FPRs were provided by ANN using Maji as the exporter (2% lower FPRs on average, compared to C4.5). However, even when using the Maji feature set, some of the FPR obtained is still too high to accept in practice. What follows is a further examination of this phenomenon.

6.1.1.1 Type of network traffic to be analyzed– On the effect of protocol filtering

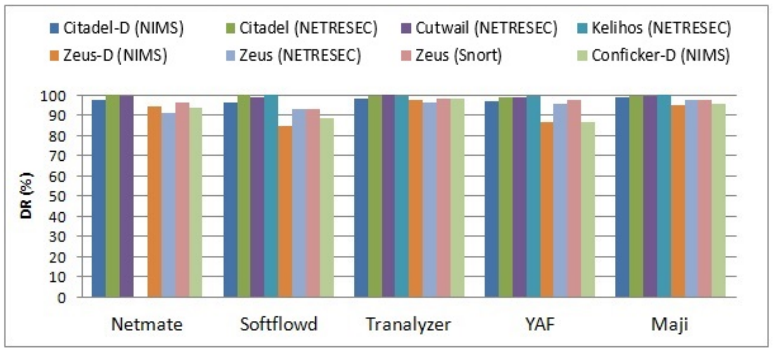
In the evaluations presented above all the network packets available in the networks traces were utilized to convert them to flows. Thee next step analyzes how the performance of the aforementioned systems would change (if at all) when traffic filters are in place on a given network. In practice, almost all network operation centers run packet filters to analyze and shape their traffic according to their organizational needs and policies. To this end, the data should first be analyzed in order to form a link between the type of data and the filter. A series of experiments was conducted applying HTTP and DNS filters to the selected data sets before classifying them for botnet detection. The logic behind these filters is explained in the corresponding sections.

6.1.1.1.1 HTTP filtering

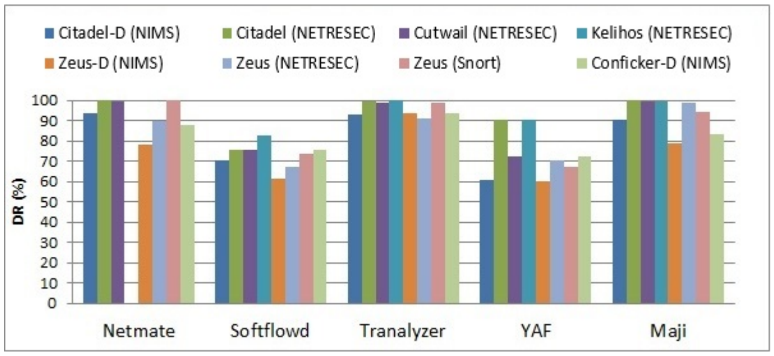
Given the wide range of HTTP usage on the Internet, most recent botnets employ the HTTP protocol to hide their malicious activities within the normal web traffic [43], easily bypassing firewalls and avoiding botnet detection mechanisms. Thus, the botnets employed in the above evaluation also utilize the HTTP protocol to communicate with their bots. In short, to investigate the effect of protocol filtering on botnet detection via machine learning approaches using different flow exporters, just the HTTP traffic flows were filtered and forwarded to the botnet classifiers. Then, The above approach was repeated train the classifiers and evaluate them again.

Figure 6.4 shows the botnet classification results versus normal traffic using the HTTP filter. As with the traffic classification without packet filtering, C4.5 and ANN outperformed the other classifiers. The results also support the observations that the classifiers could differentiate botnet behaviour with higher performances when a specific flow exporter is used as opposed to the others. Almost all of the five classifiers showed performance increases in terms of average DR and FPR when Tranalyzer and Maji were in use on all botnets except one. In this case, the classifiers showed a decrease in DR on the Zeus data sets when using Maji while an average 2% increase was observed with Tranalyzer on the Zeus data sets. Therefore, in these experiments in which the HTTP filter is used, there is a clear winner in terms of the flow exporters used, and the winner is Tranalyzer. Tranalyzer performs better than Maji on these data sets. This observation might imply that HTTP traffic could be presented more clearly by Taranalyzer flow features rather than the other flow exporter features. A close look at the features that the C4.5 classifier employed using Tranalyzer demonstrates that inter-arrival-based features are the ones that are highly employed in the decision tree solutions while such features are not available in Maji. It would appear that the features supported by Tranalyzer are very useful in terms of representing HTTP traffic.

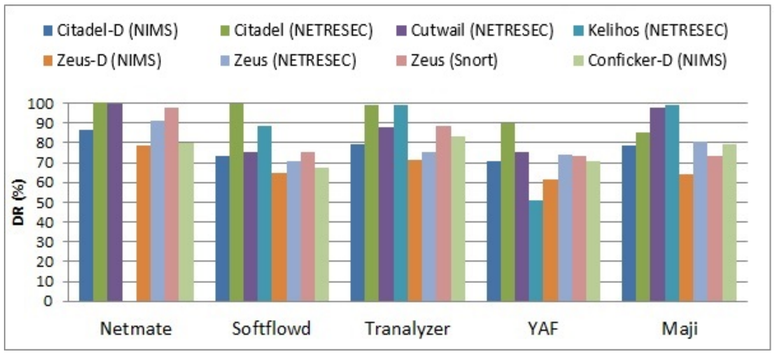
As discussed earlier, DR is not the only parameter which should be analyzed to evaluate the performance of a classifier. Figure 6.5 shows the FPR of the best two classifiers, C4.5 and ANN. Compared to Fig. 6.3, both classifiers showed at least a 1% reduction in the FPR on all data sets. Moreover, these results indicate that although the DR of the classifiers might not change significantly when traffic filtering is in place, the FPR (misclassification) can change significantly (lowered false alarm rates). This shows how useful traffic filtering can be when employed to refine the data on a very specific area which could be the center of focus specifically for challenging botnet detection research. To investigate the effect of protocol filtering on botnet detection further, a series of experiments on DNS filtering was run, given that DNS is another essential protocol in botnet communication.



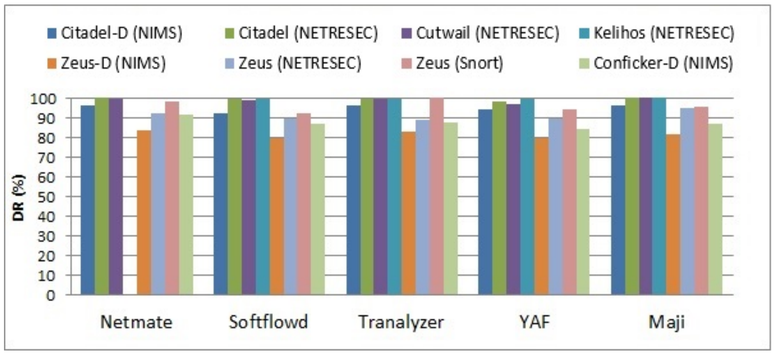
(a) C4.5



(b) SVM

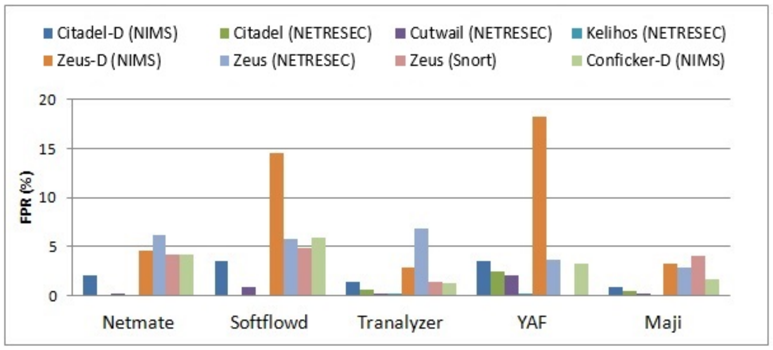


(c) Naive-Bayes

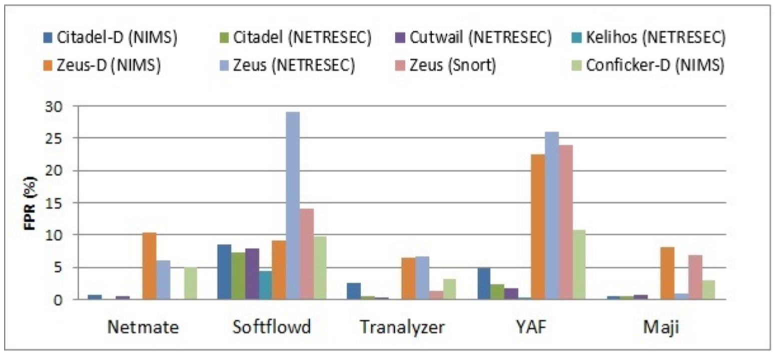


(d) Bayesian Networks

Figure 6.4: DR of all classifiers on the five flow exporters using HTTP traffic only



(a) C4.5



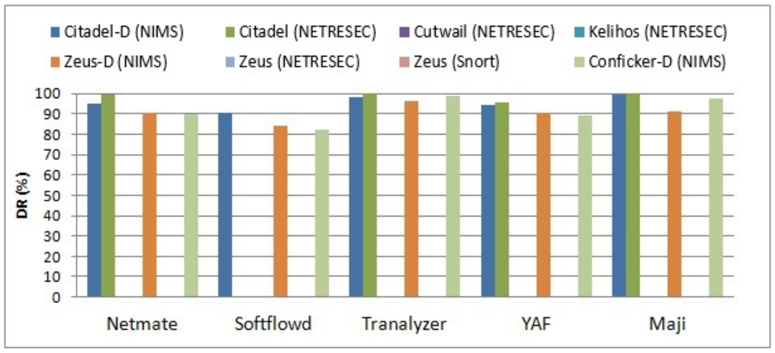
(b) ANN

Figure 6.5: FPR of C4.5 and ANN classifiers on the five flow exporters using HTTP traffic only

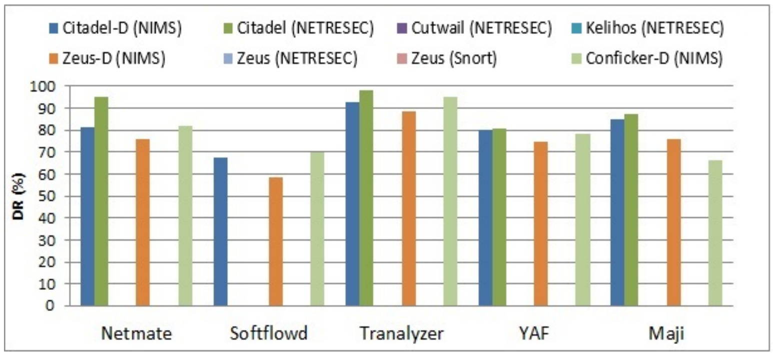
6.1.1.1.2 DNS filtering

In addition to its many legitimate uses, DNS is also used by botnets to manage their infrastructures. Botnets employ the DNS protocol along with fluxing techniques to solve the single point of failure problem on their C&C servers and to achieve mobility [102]. Many botnets (such as Zeus, Conficker and Pushdo) use the legitimate DNS infrastructure to locate their C&C servers to avoid static configuration of C&C servers IP addresses.

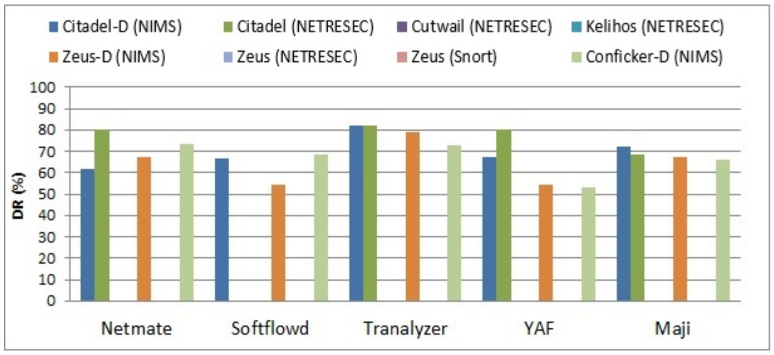
Consequently, the aforementioned systems are evaluated using only the DNS traffic flows via the DNS filters applied the captured traffic. In this case, some of the data sets are very short traces (Cutwail, Kelisos and Zeus from Snort and NETRESEC) and do not have enough DNS packets to generate enough DNS flows when the filters are in process. Thus, in Fig. 6.6 where the results of this evaluation are presented, there are no performance indicators for these data sets. On the rest of the data sets though, all of the classifiers showed some level of improvement in their performance when DNS filters were set compared with the classification performance of the traffic without filtering, Fig. 6.2. As shown in Figs. 6.6 and 6.7, C4.5 performance (as the best performing classifier in this experiment) is improved by DNS filtering by an average of 3% in terms of DR and 4% in terms of FPR using all five exporters. However, these results suggest again that Tranalyzer seems to be the winner when DNS filtering is in use with better DR and an acceptable FPR.



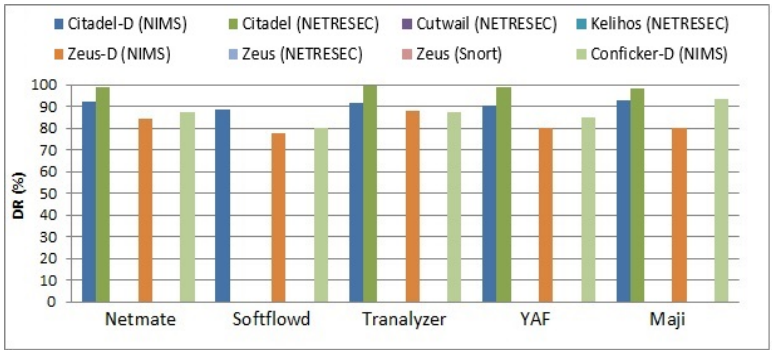
(a) C4.5



(b) SVM

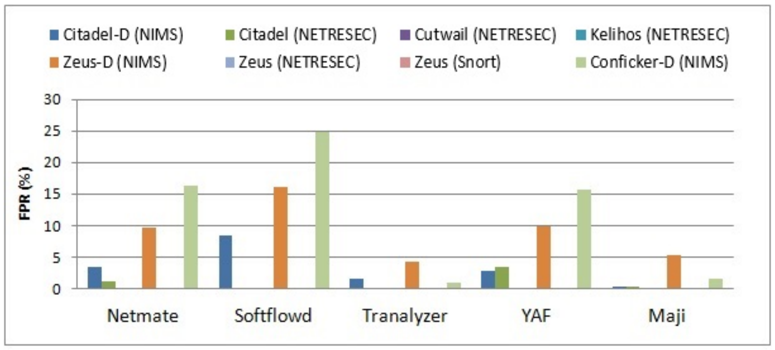


(c) Naive-Bayes

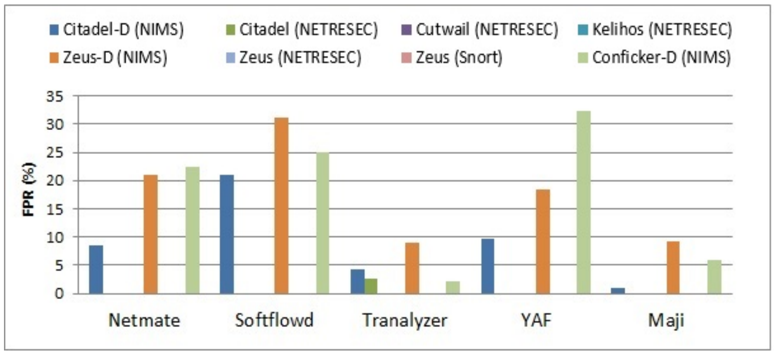


(d) Bayesian Networks

Figure 6.6: DR of all classifiers on the five flow exporters using DNS traffic only



(a) C4.5



(b) ANN

Figure 6.7: FPR of C4.5 and ANN classifiers on five flow exporters using DNS traffic only

6.1.1.1.3 Highlights:

The highlights of the evaluation results presented above are summarized bellow.

(i) Although some of the flow exporters in this work are based on the same standard (IPFIX), the number of features they support varies (e.g. Maji has 59 whereas YAF has 46 features). Some of these features are the ones employed in the machine learning solutions. For example, "MinimumIPTotalLength" is utilized by C4.5 on the Maji feature set while YAF does not support such a feature.

(ii) Since flow exporters have different mechanisms to generate flows (based on the standard they follow), these tools export a different number of flows for a given traffic file. In this case, not only is the number of exported flows different, but also sometimes the exporter may not generate any flows due to its pre-defined rules (*e.g.* Netmate on Kelihos traffic trace).

(iii) For the five flow exporters, all of the classifiers performed better when Tranalyzer and Maji were employed; third place belongs to Netmate. This confirms that the tool employed for flow exporting analysis does have an effect on traffic classification performance.

(iv) In contrast, the protocol filtering experiments show that such filtering improves botnet classification by focussing on a specific portion of the traffic in more detail (e.g. HTTP connections only). As indicated by the results in the two sets of filtering experiments, the Tranalyzer flow exporter and the C4.5 classifier are the winners with almost no exception. ANN follows as the second best performing classifier.

(v) Comparing HTTP filtering with DNS filtering shows that the FPRs and DRs did not change significantly in terms of focussing on one filtering versus the other. However, HTTP filtering seems to increase performance more than DNS filtering. This might be because all of the botnets included in the evaluations employed HTTP as their communication protocol whereas the DNS protocol was used only to find the C&C servers by the botnets employed in this research. Table 6.2 and Table 6.3 show the classification results and the features employed by C4.5 on all of the eight botnet data sets while using HTTP filtering. These 50 features are the automatically selected features by C4.5 based on the information gain criterion from the complete feature set given to it. As shown in the table, C4.5 could obtain a DR of up to 99.9% and an FPR of up to 0.1% by using only 50 features out of the 93 features of the

Table 6.2: The C4.5 classification results using the Tranalyzer feature set with the HTTP filter.

	Data Set	Score	Botnet		Legitimate	
			TPR	FPR	TPR	FPR
C4.5	Citadel (NIMS)	98%	98%	1%	99%	2%
	Citadel (NETRESEC)	99%	100%	1%	99%	0%
	Cutwail (NETRESEC)	100%	100%	0%	100%	0%
	Kelihos (NETRESEC)	100%	100%	0%	100%	0%
	Zeus-D (NIMS)	98%	99%	3%	97%	1%
	Zeus (NETRESEC)	97%	99%	6%	94%	1%
	Zeus (Snort)	99%	99%	1%	99%	1%
	Conficker-D (NIMS)	99%	98%	1%	99%	2%

Table 6.3: The Tranalyzer features employed by C4.5 for botnet classification– all botnets.

Flow Features					
AvePktSize	BytAsm	**Bytps**	**ConnDst**	TcpS-SA/SA-ATrip	connSrcDst
Duration	ExcIat	ExcPl	ipMaxdIPID	ipMaxTTL	**ipMinTTL**
ipTTLchg	IqdIat	LowQuartileIat	**MaxIat**	MaxPktSz	**MeanIat**
MedianIat	**MinIat**	minPktSz	ModeIat	**NumBytesRcvd**	NumBytesSnt
NumPktsRcvd	**NumPktsSnt**	PktAsm	**Pktps**	RobStdIat	**SkewIat**
SkewPl	StdIat	TcpAveWinSz	TcpInitWinSz	TcpMaxWinSz	TcpMinWinSz
TcpMSS	**TcpRTTAckTripMax**	TcpOptPktCnt	TcpPAckCnt	TcpPseqcnt	TcpRTTAckTripAve
TcpOptCnt	**TcpRTTAckTripMin**	TcpRTTSseqAA	ConnSrc	**TcpSeqSntBytes**	TcpWinSzDwnCnt
		TcpWS	UppQuartileIat		

complete Tranalyzer feature set on the HTTP-filtered traffic. The performance of the proposed combination is higher (e.g. [122, 147]) or similar (e.g. [106, 155, 85]) to the results reported on the HTTP-based botnets in the literature (not necessarily the botnets or even the data sets employed in this work), Table 6.4.

The bold features in Table 6.3 show the highly utilized features indicating the importance of the inter-arrival and Packets&Bytes feature categories which are also supported by other works in the literature [155, 61]. Tranalyzer has combined the most important features that are required for detecting various types of botnets reported in the literature [155, 61, 151, 85]. The conclusion is that the proposed feature set by Tranalyzer can be useful for other types of botnets as well and can be used for a real time detection system.

6.1.1.2 Feature representation– the effect of non-numeric features

As demonstrated in the previous sections, the choice of flow-based feature sets can have great impact on the performance of the machine learning-based botnet detection

Table 6.4: Performances reported in the literature.

Proposed systems	**DR**	**FPR**
Kirubavathi *et al.* [106]	up to 99%	1%
Zhao *et al.* [155]	99%	0.01%
Rerdisci *et al.* [122]	up to 79%	~ 0%
Wurzinger *et al.* [147]	88%	11%
Gu *et al.* [85]	up to 100%	~ 0%

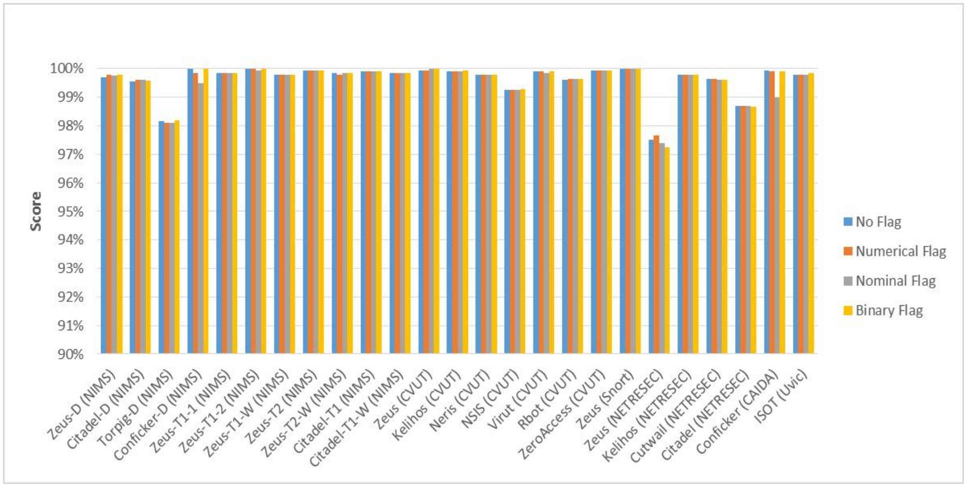
system. Although in the aforementioned analysis all usable flow features exported by the five various flow exporters were employed for detection purposes, non-numeric features were left out due to the format issues. Various methods have been investigated regarding the methods that can be used to introduce such features to the machine learning techniques and the biases they might cause [115]. This section explores the effects of non-numeric feature representation. The 'TCP-flag' was chosen for the study since these flags have been shown to be used by malware [108]. Two ML algorithms were employed: the C4.5 decision tree and the symbiotic bid-based (SBB) framework for evolving teams of programs to detect botnet behaviour. Both of these learning algorithms generate solutions (models) that are in human readable format and therefore enable the analysis of the learned models.

It has been shown that TCP flags are employed by the Torpig botnet for communication but not by the Conficker botnet [92]. The analysis was done using basic Netflow features exported by Softflowd. The evaluation and results are available in Appendix B. However, given that in the previous section Tranalyzer was the best performing flow exporter, this exporter is specifically used in this section to explore the effect of non-numeric feature representation. As discussed previously, Tranalyzer extracts 93 features for each flow. All of the features provided by Tranalyzer were employed as inputs to the machine learning classifiers except the IP addresses and port numbers because IP addresses can be spoofed or anonymized and port numbers can be assigned dynamically. Hence, such features may decrease the generalization abilities of the detection systems for unseen behaviours. In summary, without using the TCP flag as part of the feature set, the size of the feature set is 71. However, once the numerical, the nominal and the binary TCP flag representations are introduced, the feature set size changes to 72, 72 and 79, respectively. After extracting the relevant feature set for each of the experiments, a balanced data set is formed by selecting randomly (uniform random selection) from the non-malicious flow data set as well as from each of the malicious data sets. For this experiment, ETP (LBNL) is employed to represent the non-malicious side. This data set has been used to represent normal behaviour in the literature [153, 61].

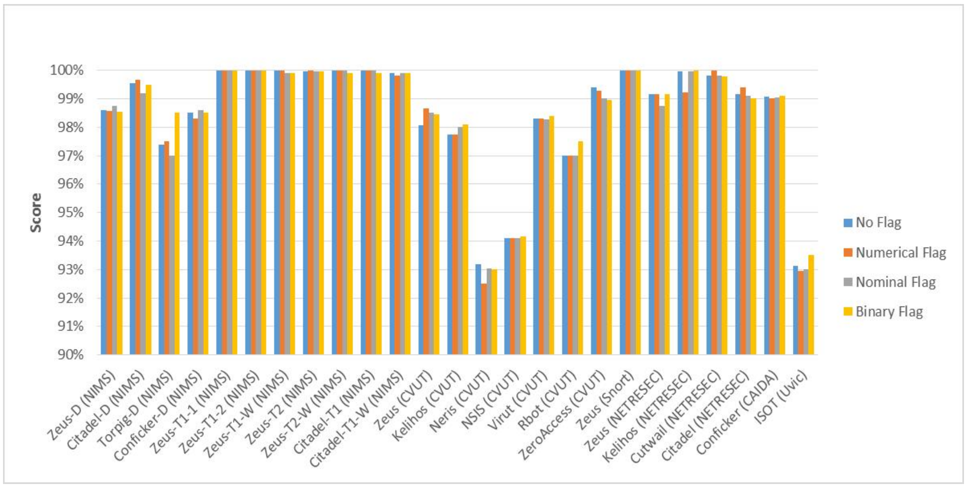
It should be noted that the Weka decision tree implementation accepts 'nominal' as a type for the features. Having a feature with the nominal type, the first string value is assigned index 0. This means that internally, this specific string value is stored as a 0 in the data set for the training/testing purposes. For consistency, the same index-based interpretation was used for the features of this type in SBB. The C4.5 classifier is run on each balanced data set using 10-fold cross-validation to further avoid any data set biases that might affect the results. However, SBB requires the training and testing data sets to be provided separately. Hence, the data set was divided into two parts (training and testing) based on an almost 30-70% breakdown for testing and training, respectively.

Figure 6.8 shows the Score of the C4.5 and SBB classifiers on all 25 data sets without utilizing the TCP flag data (indicated as No Flag in the figure) as well as using the flag information with the Nominal, Numerical and Binary representations. Since the Scores are all higher that 90% and very close to each other, the figure is zoomed into the [90%-100%] range. Although the results demonstrate changes (increase and decrease) in the performance, neither the C4.5 classifier nor the SBB classifier could display a winner among the four representations. Hence, adding the TCP flags in any form (representation) is not beneficial in designing early warning botnet detection systems but may result is some pre-processing overhead in some cases. Detailed results of this analysis are attached to Tables C.1, C.2, C.3 and C.4 of Appendix C.

Figures 6.9, 6.10, 6.11 and 6.12 show the results of the C4.5 and SBB classifiers on the data sets without the TCP flag. As the Figures 6.9 indicates C4.5 outperformed the SBB classifier based on Score. However, in general, SBB performed better in terms of solution complexity (obtaining smaller solutions), as shown in Figures 6.10 and 6.11. This difference is more noticeable for bigger data sets such as Conficker (CAIDA)



(a) C4.5



(b) SBB

Figure 6.8: Score analysis of the C4.5 and SBB classifiers based on TCP flags representations.

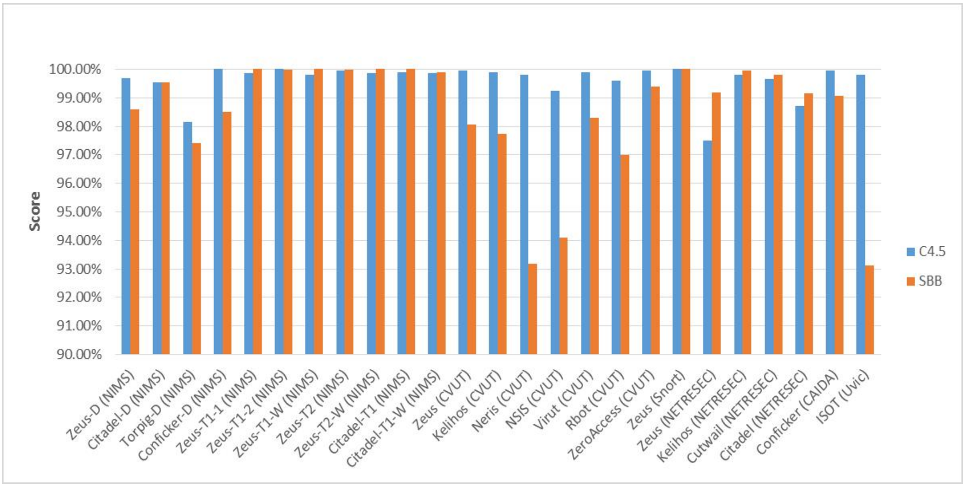


Figure 6.9: Score analysis of C4.5 vs. SBB– without TCP flags.

and Zeus (CVUT). The lower solution complexity enables SBB to implement solutions more efficiently. Given that such solutions need to operate at network flow speeds, simpler solutions are more advantageous because the early warning detection system can perform faster with fewer number of rules/signatures. By contrast, the C4.5 time complexity (for training) was usually lower than the SBB except for five data sets, Figure 6.12. Given that training is a one time off-line process, the indication is that SBB is the winner in terms of the complexity criterion between the two classifiers. However, C4.5 outperformed SBB by up to 8% of the Score in some cases. In this case, given that an early warning detection with better accuracy is more desirable (specially with the observed gap) compared to a less complex system, the C4.5 classifier with Tranalyzer feature set is the winner combination in this experiment.

6.1.1.3 Time generalization– On the effect of botnet behaviour evolution

Based on the botnet lifecycle/structure and the different versions of botnets being reported, it is evident that botnets evolve over time. Hence this section investigates how effective the C4.5, ANN, KNN and SBB-trained botnet detection models can perform facing newer/different versions of the same botnet behaviour. In other words, this section investigated the robustness of the relatively older trained detection model on a newer version of the corresponding botnet. For this purpose, three robustness scenarios are tested: (i) Time robustness when the test data set is from a different time window. (ii) Location robustness when test data sets are captured in a different

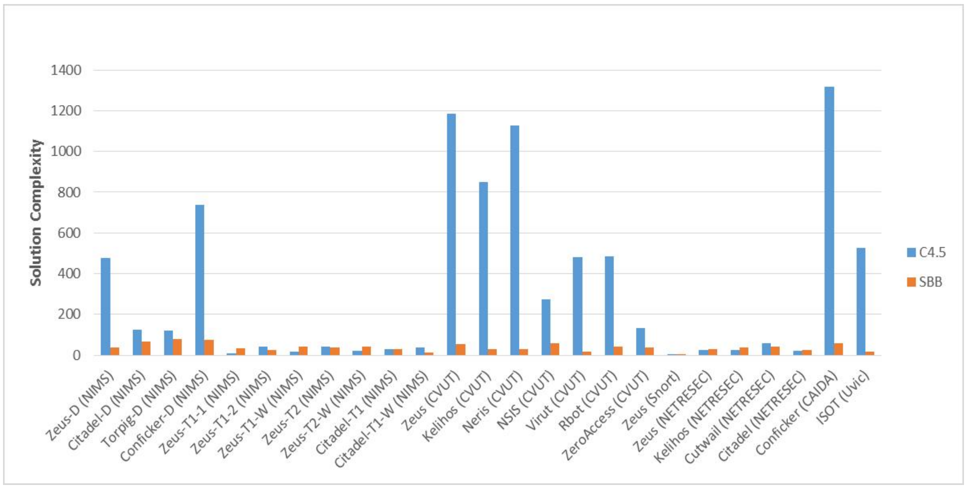


Figure 6.10: Solution complexity analysis of C4.5 vs. SBB– without TCP flags.

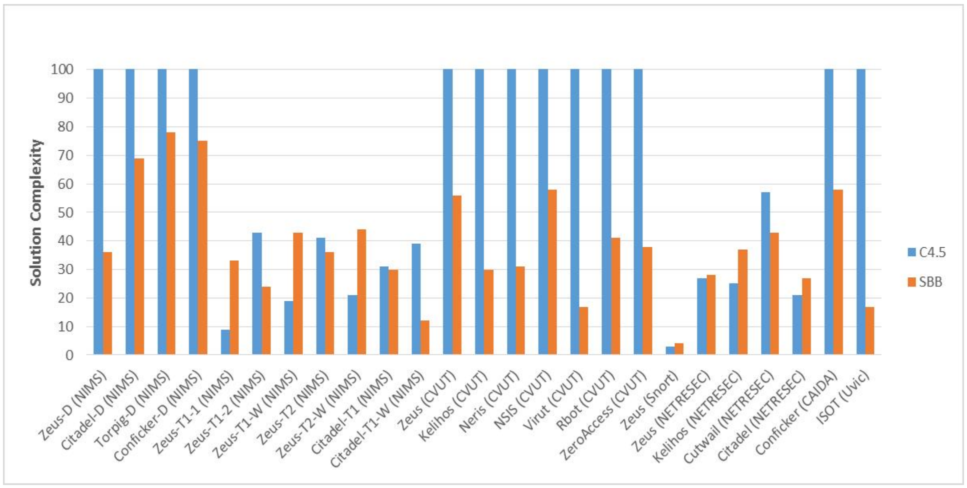


Figure 6.11: Zoomed solution complexity analysis of C4.5 vs. SBB– without TCP flags.

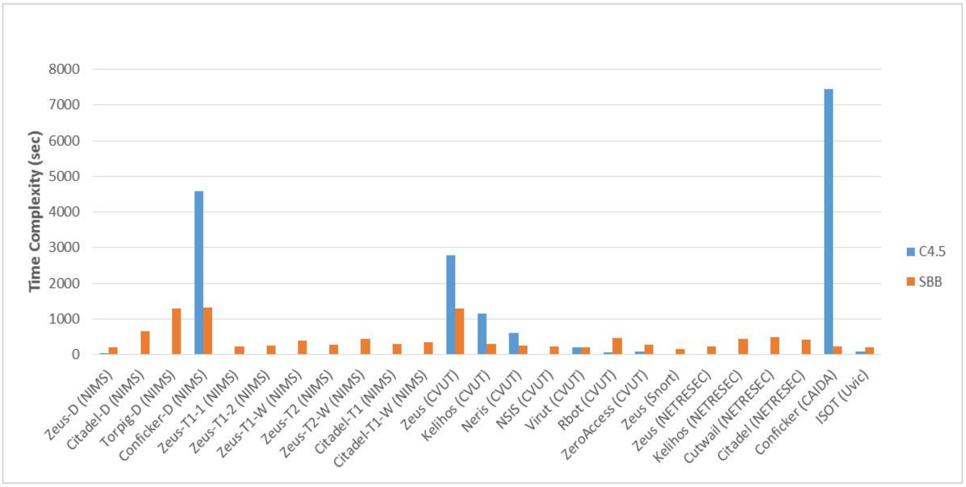


Figure 6.12: Time complexity analysis of C4.5 vs. SBB– without TCP flags.

location (*i.e.* a different network). (iii) Objective robustness when the objective of the test data set is different (*e.g.* Zeus botnet with web injection). The C4.5, ANN and SBB were chosen for this section given their good performances in the previous sections of this chapter. In addition to these classifiers, KNN is used as well in this section given its reported high performance in the literature on network traffic analysis [55, 133]. Regarding the data set for the analysis in this section, Zeus and Citadel were the only botnet available with several version of traffic traces over a period of time. Therefore, these were the only botnets that are studied. For this experiment, ETP (LBNL) represents normal behaviours in all of the data sets.

Regarding the Zeus botnet, the data sets were collected/generated over a period of four years. Among them, Zeus (NETRESEC) and Zeus (Snort) are small data sets. The Zeus-D (NIMS) data set was generated based on the C&C domain name list and hence, is a good representative of the communication phase of the botnet lifecycle. It was decided to use these as the testing data sets and use Zeus-T1-1 (NIMS) as the training data set. Zues-T1-1 is the oldest data set in the collection (excluding the Zeus-D (NIMS), Zeus (NETRESEC) and Zeus (Snort) data sets). Table 6.5 and Figure 6.13 show the results of these experiments. As expected, all of the classifiers could detect the legitimate side of the data sets with a high performance (TNRs of up to 100%) given that all of the data sets used the ETP (LBNL) traffic traces for this purpose. The classifiers are compared bellow.

(i) C4.5 and SBB were the best performing algorithms based on Score and TPR.

(ii) In both of these classifiers the trained model was very successful detecting the Zeus-T1-2 (NIMS), Zeus-T2 (NIMS), Zeus-T1-W (NIMS) and Zeus-T2-W (NIMS). This indicates that although there were toolkit, configuration and version changes between these data sets, the underlying behaviour of Zeus detected by the trained model is very similar to the older version of the data [Zeus-T1-1 (NIMS)].

(iii) None of the classifiers performed well when tested on the Zeus (CVUT) data set. In the Zeus (CVUT) read-me file, this data set's probable name is considered to be 'Zeus'. The read-me file also suspected that this data set may be a P2P version of the Zeus botnet. The experimental results confirm that Zeus (CVUT) data set behaviour is not similar to the C&C HTTP-based version of the Zeus botnet by any means. Hence, it more likely represents a P2P HTTP-based Zeus botnet behaviour, if it is a Zeus botnet at all.

(iv) The results indicate as well that the Zeus botnet behaviour presented by Zeus-T1-1 (NIMS) is different from the behaviour presented by the Zeus-D (NIMS) data set. This might be caused by the fact that Zeus-D (NIMS) is only a representative of one of the phases of the botnet lifecycle (C&C communication). Another set of tests was done with an HTTP filter for the experiments with low performance. Given that Zeus is an HTTP-based botnet, this filter only keeps the core botnet communication of the data set which was proven to increase performance [94]. Table 6.6 and Figure 6.14 show that the trained model on the Zeus-T1-1 (NIMS) performed much better in Zeus-D (NIMS) and Zeus (NETRESEC) detection but the performance did not change for the Zeus (CVUT) data set. These results indicate that even when different samples of the Zeus botnet do not seem to be similar [*e.g.* 32.73% TPR for detecting the Zeus (NETRESEC) sample with a model trained on Zeus-T1-1 (NIMS) using the SBB classifier], the core botnet communication behaviours are in fact similar [*e.g.* 75% TPR for the Zeus (NETRESEC)]. Overall, C4.5 performed better using the HTTP filter on Zeus-D (NIMS) and Zeus (CVUT) but not on Zeus (NETRESEC) in this experiment.

A similar experiment was conducted for the Citadel botnet. In this case, the classifiers trained on Citadel-T1 (SBB, ANN, KNN and C4.5) were tested against Citadel-D (NIMS), Citadel (NETRRESEC) and Citadel-T1-W (NIMS). The results of this experiment are shown in Table 6.7 and Figure 6.15. The resulting conclusions

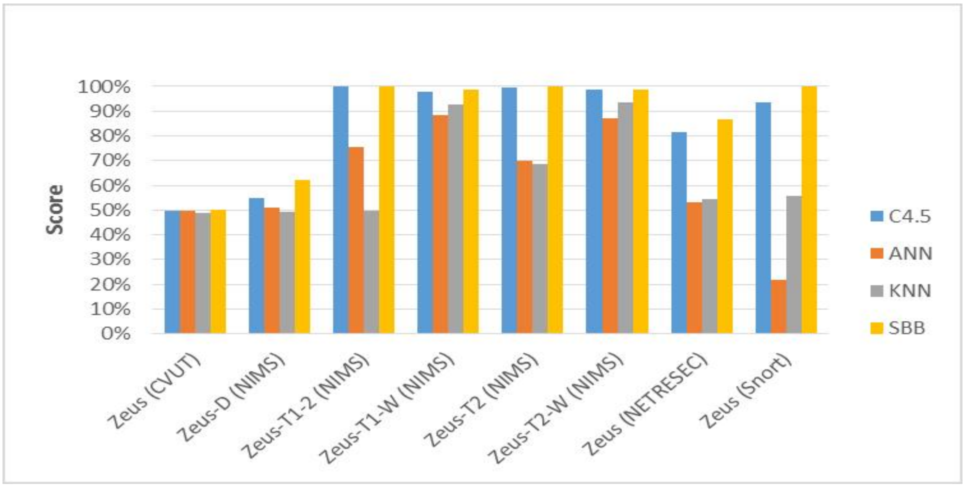


Figure 6.13: The Zeus botnet score analysis of C4.5, ANN, KNN and SBB using a model trained on Zeus-T1-1.

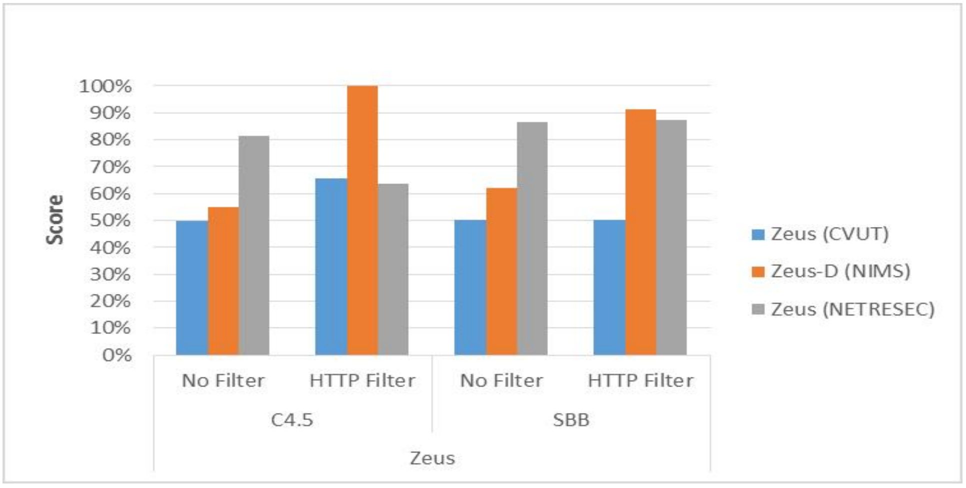


Figure 6.14: The Zeus botnet score analysis of C4.5 and SBB using a model trained on Zeus-T1-1– with and without HTTP filtering

Table 6.5: Classification Results.

	Data Set	Score	Botnet		Legitimate	
			TPR	**FPR**	**TNR**	**FNR**
C4.5	Zeus (CVUT)	49.79%	0%	0.4%	99.6%	100%
	Zeus-D (NIMS)	54.79%	10%	0.4%	99.6%	90.0%
	Zeus-T1-2 (NIMS)	99.79%	100%	0.4%	99.6%	0%
	Zeus-T1-W (NIMS)	97.75%	96%	0.5%	99.5%	4%
	Zeus-T2 (NIMS)	99.53%	99.5%	0.4%	99.6%	0.5%
	Zeus-T2-W (NIMS)	98.5%	97.4%	0.4%	99.6%	2.6%
	Zeus (NETRESEC)	81.53%	63.3%	0.2%	99.8%	36.7%
	Zeus (Snort)	93.75%	88.2%	0.7%	99.3%	11.8%
ANN	Zeus (CVUT)	49.54%	1%	1.9%	98.1%	99%
	Zeus-D (NIMS)	50.82%	3.5%	1.8%	98.2%	96.5%
	Zeus-T1-2 (NIMS)	75.55%	53%	1.9%	98.1%	47%
	Zeus-T1-W (NIMS)	88.26%	78.8%	2.3%	97.7%	21.2%
	Zeus-T2 (NIMS)	69.95%	42.1%	2.2%	97.8%	57.9%
	Zeus-T2-W (NIMS)	87.3%	76%	1.4%	98.6%	24%
	Zeus (NETRESEC)	53.12%	8%	1.7%	98.3%	92%
	Zeus (Snort)	21.73%	4.9%	1.4%	98.6%	95.1%
KNN	Zeus (CVUT)	48.87%	0.5%	2.7%	97.3%	9.5%
	Zeus-D (NIMS)	49.1%	1.1%	2.9%	97.1%	98.9%
	Zeus-T1-2 (NIMS)	49.59%	2.2%	3%	97%	97.8%
	Zeus-T1-W (NIMS)	92.54%	88%	3.3%	96.7%	11.7%
	Zeus-T2 (NIMS)	68.78%	40.5%	3%	97%	59.5%
	Zeus-T2-W (NIMS)	93.43%	90.1%	3.2%	96.8%	9.9%
	Zeus (NETRESEC)	54.36%	11.5%	2.7%	97.3%	88.5%
	Zeus (Snort)	55.90%	13.9%	2.1%	97.9%	86.1%
SBB	Zeus (CVUT)	50.26%	0.54%	0.02%	99.98%	99.46%
	Zeus-D (NIMS)	62.07%	24.14%	0%	100%	75.67%
	Zeus-T1-2 (NIMS)	99.98%	99.97%	0%	100%	0.03%
	Zeus-T1-W (NIMS)	98.83%	97.67%	0%	100%	2.33%
	Zeus-T2 (NIMS)	99.97%	99.93%	0%	100%	0.07%
	Zeus-T2-W (NIMS)	98.9%	97.9%	0%	100%	2.2%
	Zeus (NETRESEC)	86.66%	32.73%	0%	100%	26.68%
	Zeus (Snort)	100%	100%	0%	100%	0%

Table 6.6: Classification Results– with an HTTP filter.

	Data Set	Score	Botnet		Legitimate	
			TPR	**FPR**	**TNR**	**FNR**
C4.5	Zeus (CVUT)	49.95%	0.1%	0.2%	99.8%	100%
	Zeus-D (NIMS)	81.53%	63.3%	0.2%	99.8%	36.7%
	Zeus (NETRESEC)	84.62%	69.2%	0%	100%	30.8%
SBB	Zeus (CVUT)	50.26%	0.54%	0.02%	99.98%	99.46%
	Zeus-D (NIMS)	91.28%	82.76%	0.19%	99.81%	17.24%
	Zeus (NETRESEC)	87.5%	75.0%	0%	100%	25.0%

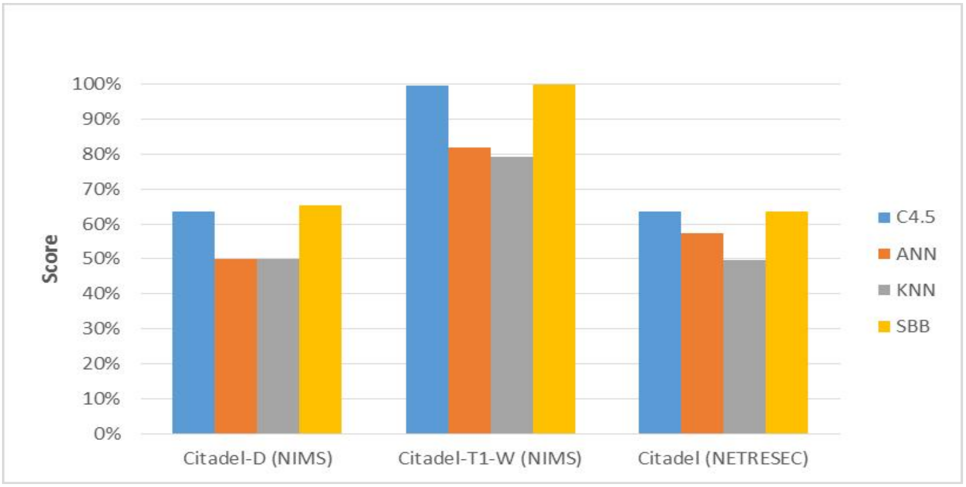


Figure 6.15: The Citadel botnet score analysis of C4.5, ANN, KNN and SBB using a model trained on Citadel-T1.

are listed below. (i) Like the Zeus experiment, SBB and C4.5 outperformed the other classifiers.

(ii) The trained model could detect Citadel-T1-W (NIMS) with high performance rate. This shows the principal botnet behaviour extracted by the classifiers is similar for botnets with one toolkit even when the configuration and therefore the target approach of the botnet is different (referring to the web injection method used in Citadel-T1-W).

(iii) Given the lower performance on Citadel-D (NIMS), Citadel (NETRRESEC), the effect of HTTP filtering was investigated. In this case, C4.5 could increase the performance by up to 28% with an HTTP filter whereas SBB was not very useful.

????

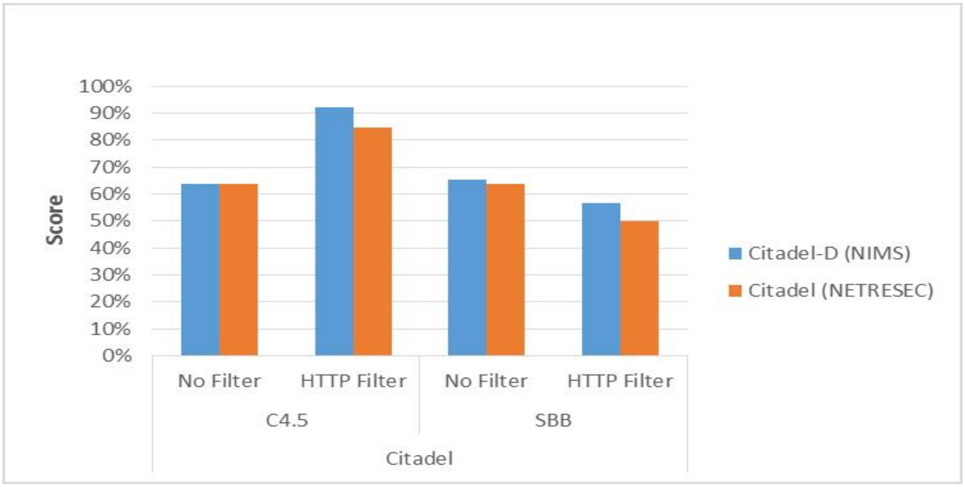


Figure 6.16: The Citadel botnet score analysis of C4.5 and SBB using a model trained on Citadel-T1– with and without an HTTP filtering.

Table 6.7: Classification Results.

	Data Set	Score	Botnet		Legitimate	
			TPR	FPR	TNR	FNR
C4.5	Citadel-D (NIMS)	63.63%	27.4%	0.1%	99.9%	72.6%
	Citadel-T1-W (NIMS)	99.66%	99.5%	0.2%	99.8%	0.5%
	Citadel (NETRESEC)	63.61%	27.3%	0%	100%	72.7%
ANN	Citadel-D (NIMS)	50.03%	0.7%	0.7%	99.3%	99.3%
	Citadel-T1-W (NIMS)	81.9%	64.3%	0.5%	99.5%	35.7%
	Citadel (NETRESEC)	57.36%	15.4%	0.6%	99.4%	84.6%
KNN	Citadel-D (NIMS)	50.09%	0.5%	0.4%	99.6%	99.5%
	Citadel-T1-W (NIMS)	79.11%	58.5%	0.2%	99.8%	41.5%
	Citadel (NETRESEC)	49.78%	0%	0.4%	99.6%	100%
SBB	Citadel-D (NIMS)	65.49%	30.98%	0%	100%	69.02%
	Citadel-T1-W (NIMS)	99.85%	99.7%	0%	100%	0.3%
	Citadel (NETRESEC)	63.61%	27.26%	0%	100%	72.74%

Table 6.8: Classification Results– with an HTTP filter.

	Data Set	Score	Botnet		Legitimate	
			TPR	FPR	TNR	FNR
C4.5	Citadel-D (NIMS)	91.96%	84%	0%	100%	16%
	Citadel (NETRESEC)	84.63%	69.3%	0%	100%	30.7%
SBB	Citadel-D (NIMS)	56.57%	13.22%	0.01%	99.9%	86.78%
	Citadel (NETRESEC)	50%	0%	0%	100%	100%

In conclusion, it was observed that an older version of the Zeus and citadel botnet-trained models can detect other versions of the same botnet with the same topology with up to a 100% TPR showing the robustness of the trained models. Having said this, filters might be useful in order to increase the performance by making the focus of the analysis on the core part of the botnet communication when necessary. Not only the results show that the trained classifiers were robust enough to detect similar botnet behaviours, but they also showed that the models could be very good in pointing out the major changes of behaviour for a given botnet (*i.e.* showing that Zeus (CVUT) does indeed have a different botnet topology from the others).

6.1.1.4 Normal behaviour representation

In order to model botnet behaviour against normal behaviour, machine learning algorithms require data for both classes. In such circumstances, the abnormal behaviour is usually a specific type of behaviour that is under investigation. Malware behaviour detection is one type of such analysis. Normal behaviour is not usually presented in a case-specific way. In other words, normal behaviour should represent the normal activity of legitimate users. This can range from normal users' web browsing (HTTP communication), banking and E-commerce (encrypted HTTP communication), file transfer (FTP communication) to torrent (P2P communication) activities.

In the literature, various traffic log files have been used by researchers as being representatives of normal behaviour. However, it appears that no research has been done to show whether the choice of normal data would cause any biases, good or bad, in the evaluation of the systems. The effect of normal behaviour representation is investigated in this section. As for the choice of normal traffic log file, ETP (LBNL), ISP (WiSNet) and Alexa-D (NIMS) are the publicly available logs which have been utilized. ETP (LBNL) has been frequently used by researchers to represent normal behaviour [61, 156] whereas ISP (WiSNet) has been recently employed by several network data analysis-based systems such as [58]. Given the use of Alexa-D (NIMS) in the analysis of the previous section, this log file was added to the set as well. It should be noted that balanced data sets are used for this section.

Table 6.10 and Figure 6.17 show the result of this evaluation. In this experiment a combination of Tranalyzer-1 and the C4.5 classifier were used. As the figure shows,

Table 6.9: Normal behaviour representation– TTest result.

	P-value
ETP (LBNL) vs. ISP (WiSNet)	0.826
ETP (LBNL) vs. Alexa-D (NIMS)	0.306
ISP (WiSNet) vs. Alexa-D (NIMS)	0.273

the Score results of all of the evaluated systems are similar and above 90%. To have a clearer view, Figure 6.18 is zoomed in above 90% which reveals the information listed below. (i) For the NIMS domain-based generated botnet data sets, the classifier performed lower with Alexa-D (NIMS). This is expected as all of these data sets are generated using the same approach, focussing of the botnet communication phase.

(ii) The performance on Zeus (Snort) and Zeus (NETRESEC) are also lower than the other data sets. This might be caused by the small size of these data sets. There might not be enough records and information to extract detailed botnet behaviour.

(iii) In 17 botnet data sets, the classifiers performed very similarly using either ETP (LBNL), ISP (WiSNet) or Alexa-D (NIMS). This may suggest that choosing a specific data set to represent normal behaviour may not cause any significant performance change on the results. However, to investigate whether the performance of the classifiers is statistically significantly different when using these three normal data sets. Table 6.9 shows the TTest results performed on all experiments demonstrated in Figure 6.17. As the TTest results indicate, the performance of the classifier had no statistically significant difference when using these three normal data sets.

Complexity is another criterion that can be analyzed in this evaluation. As claimed in [140], the cost of constructing a C4.5 decision tree (a Weka implementation of C4.5 with out subtree raising) is *O(mnlogn)* where m is the number of features and n is the number of training data samples. The Tranalyzer-1 feature set is used for all of the data sets. Hence, the training data sample size should be analyzed. Given that the three normal log files have different numbers of packets (i.e. different sizes), the number of flows exported by Tranalyzer-1 is different for them. In this case, to create balanced data sets, under-sampling was applied for the major class to re-sample the data for some of the data sets. For example, given the small size of the Alexa-D (NIMS) log file compared to larger log files like Zeus (CVUT) and Conficker (CAIDA), the botnet data was re-sampled. To this end, time complexity analysis

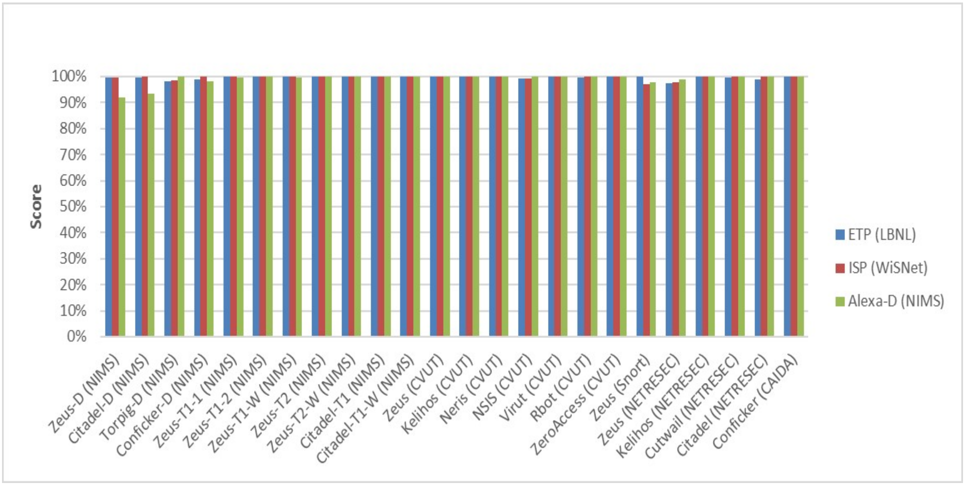


Figure 6.17: Normal behaviour representation– Detection analysis of botnets on three different legitimate data sets.

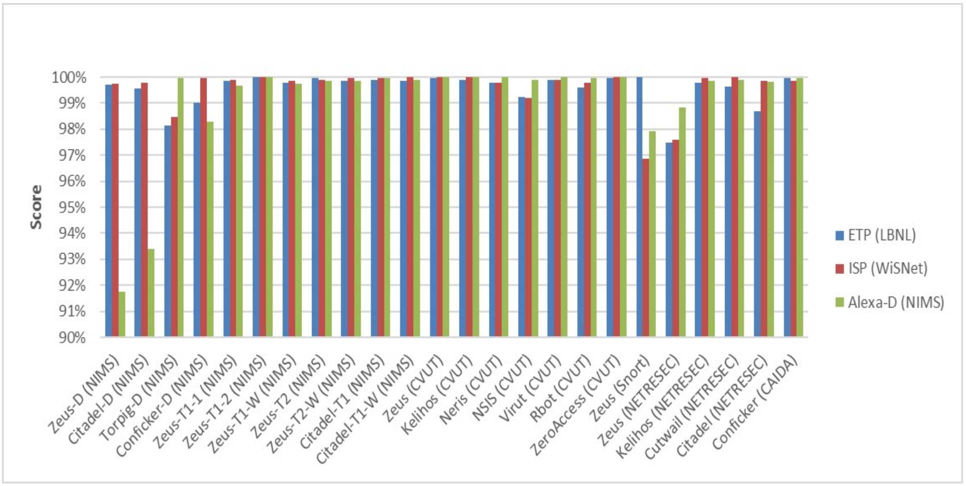


Figure 6.18: Normal behaviour representation– Detection analysis of botnets on three different legitimate data sets.

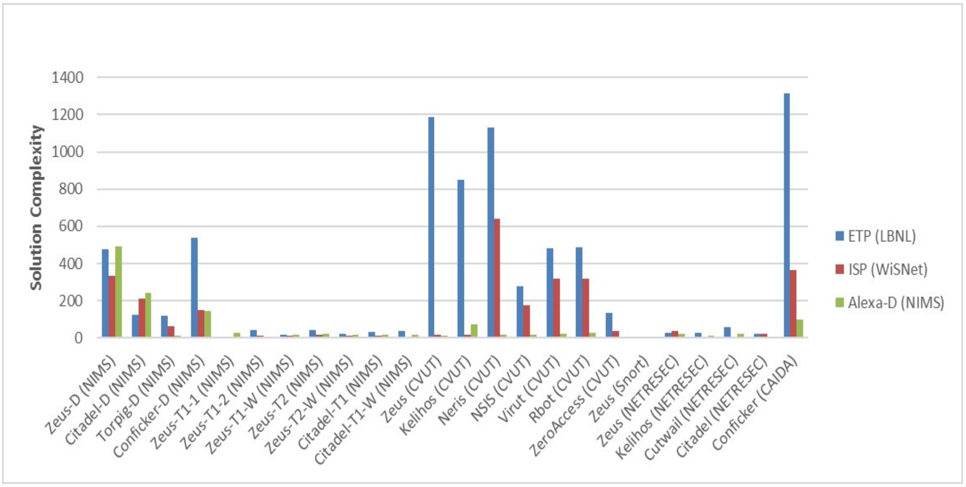


Figure 6.19: Normal behaviour representation– Solution complexity analysis.

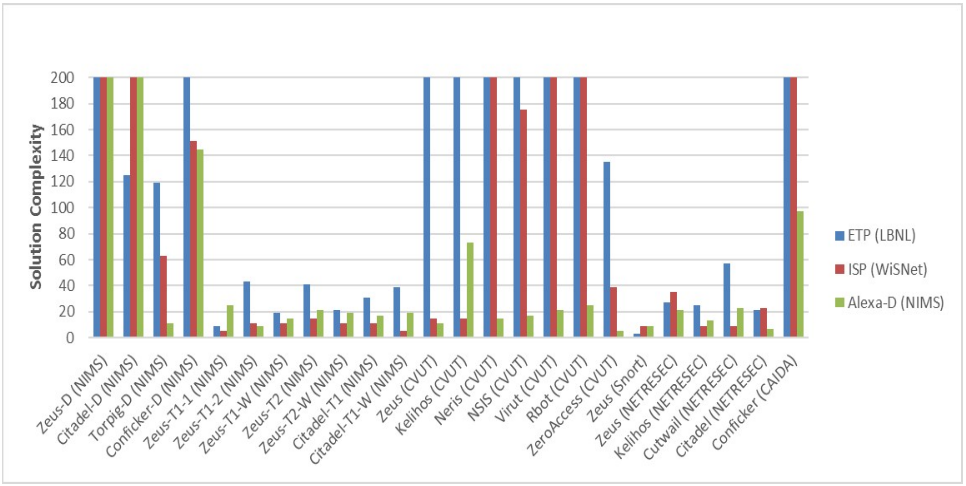


Figure 6.20: Normal behaviour representation– Solution complexity analysis (zoomed).

Table 6.10: Normal behaviour representation– Classification Results.

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
	Zeus-D (NIMS)	99.7%	99.7%	0.3%	99.7%	0.3%	39	477
	Citadel-D (NIMS)	99.55%	99.6%	0.5%	99.5%	0.4%	2.05	125
	Torpig-D (NIMS)	98.15%	98.2%	1.9%	98.1%	1.8%	0.85	119
	Conficker-D (NIMS)	100%	100%	0%	100%	0%	4570.8	739
	Zeus-T1-1 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.24	9
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	19.72	43
	Zeus-T1-W (NIMS)	99.8%	99.9%	0.3%	99.7%	0.2%	0.29	19
	Zeus-T2 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.28	41
	Zeus-T2-W (NIMS)	99.85%	99.8%	0.2%	99.8%	0.2%	0.34	21
	Citadel-T1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	1.06	31
	Citadel-T1-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.83	39
ETP (LNBL)	Zeus (CVUT)	99.95%	100%	0.1%	99.9%	0%	2779.2	1185
	Kelihos (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	1149.57	849
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	612.05	1129
	NSIS (CVUT)	99.25%	99.3%	0.8%	99.2%	0.7%	5.38	275
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	215	481
	Rbot (CVUT)	99.6%	99.6%	0.4%	99.6%	0.4%	61.89	487
	ZeroAccess (CVUT)	99.95%	100%	0.1%	99.9%	0%	94.85	135
	Zeus (Snort)	100%	100%	0%	100%	0%	0.05	3
	Zeus (NETRESEC)	97.5%	98%	3.0%	97.0%	2.0%	0.15	27
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.52	25
	Cutwail (NETRESEC)	99.65%	99.8%	0.5%	99.5%	0.2%	1.08	57
	Citadel (NETRESEC)	98.7%	99.6%	2.2%	97.8%	0.4%	0.24	21
	Conficker (CAIDA)	99.95%	100%	0.1%	99.9%	0%	7454.18	1317
	Zeus-D (NIMS)	99.75%	99.8%	0.3%	99.7%	0.2%	50.53	333
	Citadel-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	21.94	211
	Torpig-D (NIMS)	98.45%	99.1%	2.2%	97.8%	0.9%	0.7	63
	Conficker-D (NIMS)	99.95%	99.9%	0%	100%	0.1%	135.51	151
	Zeus-T1-1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.24	5
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	13.48	11
	Zeus-T1-W (NIMS)	99.87%	100%	0.3%	99.7%	0%	0.3	11
	Zeus-T2 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.43	15
	Zeus-T2-W (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.28	11
	Citadel-T1 (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.92	11
	Citadel-T1-W (NIMS)	100%	100%	0%	100%	0%	0.39	5
ISP (WiSNet)	Zeus (CVUT)	100%	100%	0%	100%	0%	72.17	15
	Kelihos (CVUT)	100%	100%	0%	100%	0%	60.11	15
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	167.38	641
	NSIS (CVUT)	99.19%	99.5%	1.1%	98.9%	0.5%	4.94	175
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	195.78	317
	Rbot (CVUT)	99.80%	99.8%	0.2%	99.8%	0.2%	54.55	317
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	87.4	39
	Zeus (Snort)	96.88%	99.3%	5.6%	94.4%	0.7%	0.1	9
	Zeus (NETRESEC)	97.6%	97.9%	2.7%	97.3%	2.1%	0.18	35
	Kelihos (NETRESEC)	99.95%	99.9%	0%	100%	0.1%	0.39	9
	Cutwail (NETRESEC)	100%	100%	0%	100%	0%	0.49	9
	Citadel (NETRESEC)	99.85%	99.9%	0.2%	9.8%	0.1%	0.47	23
	Conficker (CAIDA)	99.85%	99.9%	0.2%	99.8%	0.1%	149.92	365
	Zeus-D (NIMS)	91.76%	97.2%	13.7%	86.3%	2.8%	2.62	493
	Citadel-D (NIMS)	93.38%	98.7%	11.9%	88.1%	1.3%	1.95	241
	Torpig-D (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.16	11
	Conficker-D (NIMS)	98.27%	97.3%	0.7%	99.3%	2.7%	0.99	145
	Zeus-T1-1 (NIMS)	99.67%	99.8%	0.5%	99.8%	0.2%	0.18	25
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	0.42	9
	Zeus-T1-W (NIMS)	99.75%	99.9%	0.4%	99.6%	0.1%	0.2	15
	Zeus-T2 (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.33	21
	Zeus-T2-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.17	19
	Citadel-T1 (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.62	17
	Citadel-T1-W (NIMS)	99.9%	100%	0.2%	99.8%	0%	0.39	19
Alexa-D (NIMS)	Zeus (CVUT)	100%	100%	0%	100%	0%	0.46	11
	Kelihos (CVUT)	100%	100%	0%	100%	0%	0.15	73
	Neris (CVUT)	100%	100%	0%	100%	0%	0.79	15
	NSIS (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	0.69	17
	Virut (CVUT)	100%	100%	0%	100%	0%	0.64	21
	Rbot (CVUT)	99.95%	100%	0.1%	99.9%	0%	1.14	25
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	0.65	5
	Zeus (Snort)	97.91%	97.2%	1.4%	98.6%	2.8%	0.01	9
	Zeus (NETRESEC)	98.85%	98.5%	2%	98%	1.5%	0.03	21
	Kelihos (NETRESEC)	99.85%	99.9%	0.2%	99.8%	0.1%	0.31	13
	Cutwail (NETRESEC)	99.9%	99.9%	0.1%	99.9%	0.1%	0.32	23
	Citadel (NETRESEC)	99.81%	100%	0.4%	99.6%	0%	0.04	7
	Conficker (CAIDA)	99.95%	100%	0.1%	99.9%	0%	0.5	97

would not be useful in this section.

On the other hand, a solution complexity analysis was completed. Figures 6.19 and 6.20 show the results. As demonstrated by these figures, the solution complexities were very close for smaller data sets like Zeus (Snort) and Citadel-D (NIMS). For larger data sets, ISP (WiSNet) and ETP (LBNL) had higher complexities. The complexity difference between the data sets with Alexa-D (NIMS) and the other two normal data sets might be affected by the number of data samples. In other words, it is possible that data sets with more data samples demonstrate more complex behaviours (even though Weka random spread subsampling was used to minimize such an effect). However, the complexities of the CVUT data sets cannot be justified this way. Except for Rbot (CVUT), ISP (WiSNet) and ETP (LBNL), normal log files (not the botnet log files) were sampled to create the final CVUT data sets. This means sample counts for these data sets are almost the same. In this case, the solution complexities of these data sets cannot be justified by the size of the data sets. Hence, it appears that ETP (LBNL) is a more complex data set representing a more diverse range of normal behaviours which has caused the C4.5 classifier to create more complex solutions.

In summary, the evaluations in this section indicate that the choice of normal log files (to represent normal behaviour) would not cause any statistically significant performance shift (positive or negative). However, the choice may cause direct/indirect complexity changes. To this end, the type of analysis and the performance criteria should be considered when answering the question on whether the choice of normal behaviour representation would affect the analysis.

6.1.1.5 Exploring further flow feature sets

Feature set and classification algorithm selection has been the main focus of the above sections in this chapter. So far, different combinations of feature sets and algorithms have been evaluated with various botnet and normal data sets over different scenarios and the Tranalyzer-1/C4.5 team has proven to be the best performing combination. However, Tranalyzer has published a newer version, introduced as Tranalyzer-2 in Section 3.2. Moreover, Garcia *et al.* employed Argus basic flow features for a system design in [83]. Based on this research, most of the CVUT data sets have the Argus uni-directional and/or bi-directional Netflow flows included for download. Tranalyzer-1

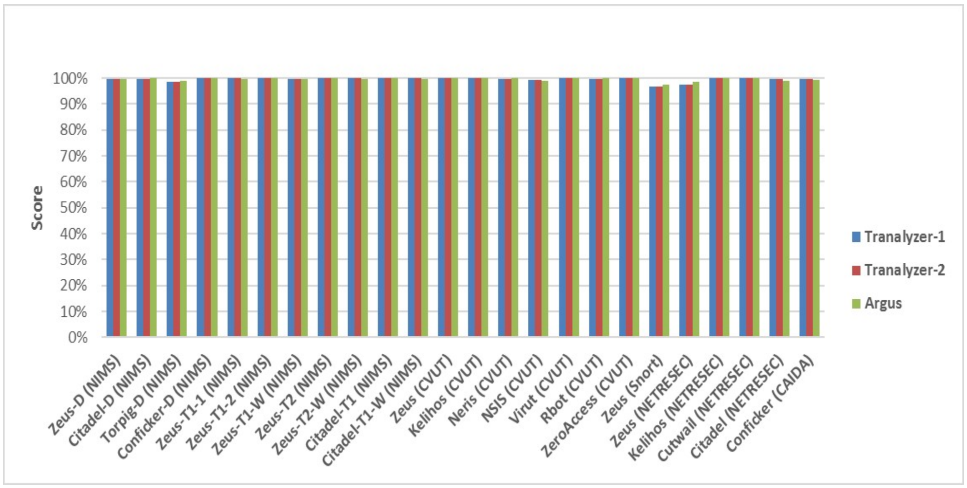


Figure 6.21: Argus vs. Tranalyzer Score performance results

and the Argus basic and extended feature sets (the latest version) have been evaluated and compared [91]. Argus with the extended feature set outperformed the basic feature set while it performed as well as the the Tranalyzer-1 feature set. It is likely that this is caused by the fact that Argus has included features of the Inter-arrival and Packets&Bytes categories in its extended, third version. These features are shown to be very effective in network traffic analysis and specifically botnet detection [94, 61, 155]. To have a more comprehensive analysis and comparison the Tranalyzer-1, Tranalyzer-2 and Argus v. 3.0.8 feature sets were used in this section with ISP (WiSNet) as the normal behaviour representative. Moreover, C4.5 was selected as the classifier with balance data set scheme.

Table 6.12 shows the detailed classification results of this experiment while Figure 6.21 is a visual view of the Score performance. Given the very close Scores shown in this figure, a zoomed version of the figure is drawn over the range of 90% to 80% in Figure 6.22. As the results indicate, in terms of Score, Tranalyzer-1 and Tranalyzer-2 have basically the the same Scores. Argus v. 3.0.8 performed similarly to the other two feature sets with less than a maximum 5% increase or decrease. This analysis is consistent with the previous evaluations [91]. To check if the performance of the C4.5 is statistically different when using any of these feature sets a TTest was run. The TTest result in Table 6.11 demonstrates that there is no statistically significant difference between the performance of the classifier (in terms of Score) in this case.

In a further comparison of the classification results of these feature sets FPR and

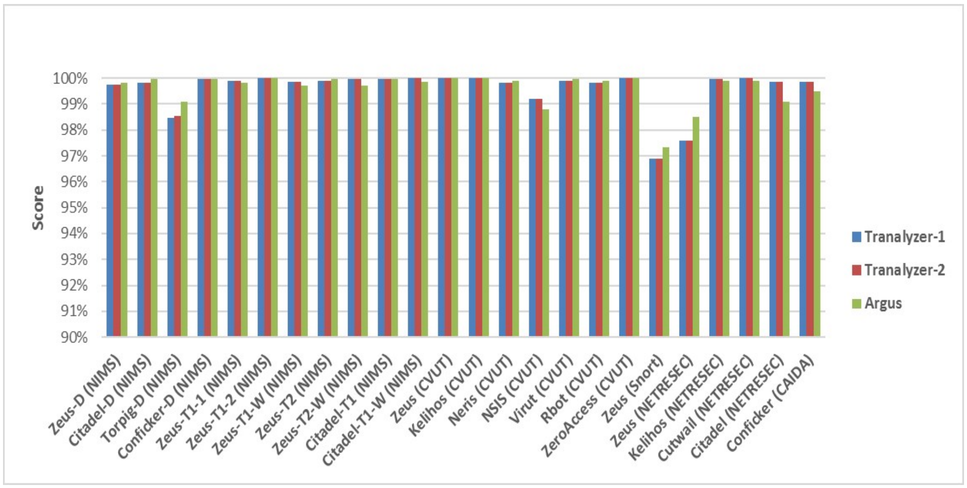


Figure 6.22: Argus vs. Tranalyzer Score performance results (zoomed).

Table 6.11: TTest results of Tranalyzer vs. Argus Score performance.

	P-value
Tranalyzer-1 vs. Tranalyzer-2	0.989
Tranalyzer-1 vs. Argus	0.912
Tranalyzer-2 vs. Argus	0.949

solution complexity were analyzed. Figures 6.23 and 6.24 show the FPRs. As demonstrated by these figures, Tranalyzer-1 and Tranalyzer-2 performed similarly while Argus did not show any pattern (underperforming or outperforming Tranalyzer). In terms of solution complexity however, Argus resulted in less complex solutions. This gives the early warning systems employing Argus flow features an advantage while being used as detection systems in real-time.

6.1.1.6 How similar or different are botnet behaviours?

In summary, various flow feature exporters, feature sets and machine learning algorithms were evaluated in order to design a botnet early warning detection system in this chapter. The analysis, evaluation and results show that a combination of the Tranalyzer and Argus feature sets with the C4.5 classifier is effective for detecting different botnets from IRC botnets to P2P botnets. Thus, the new research question to answer is: How would these combinations handle a traffic log file that consists of various botnets (such as IRC, HTTP, and P2P) and Normal traces? In other words,

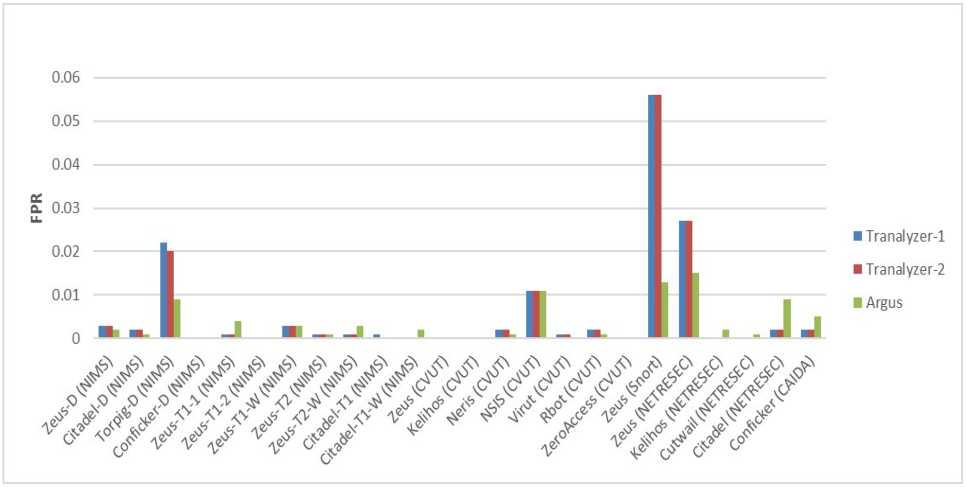


Figure 6.23: Argus vs. Tranalyzer FPRs.

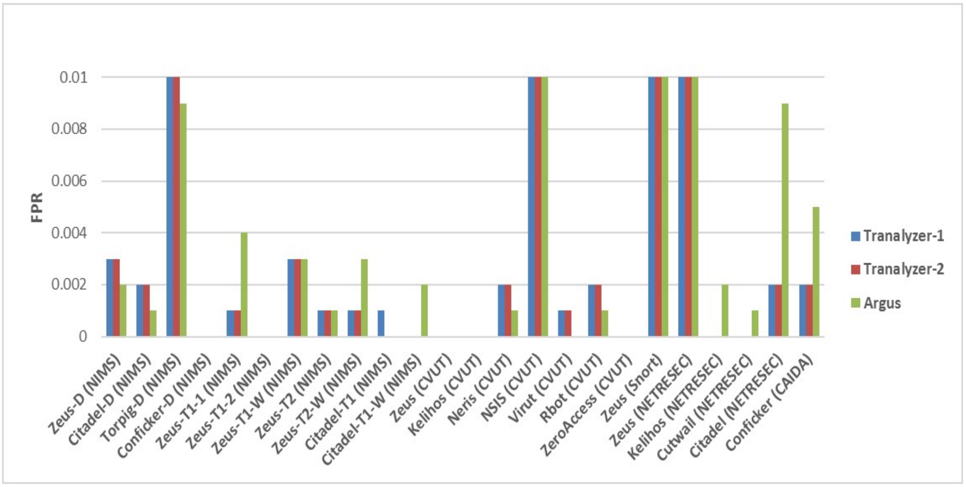


Figure 6.24: Argus vs. Tranalyzer FPRs (zoomed).

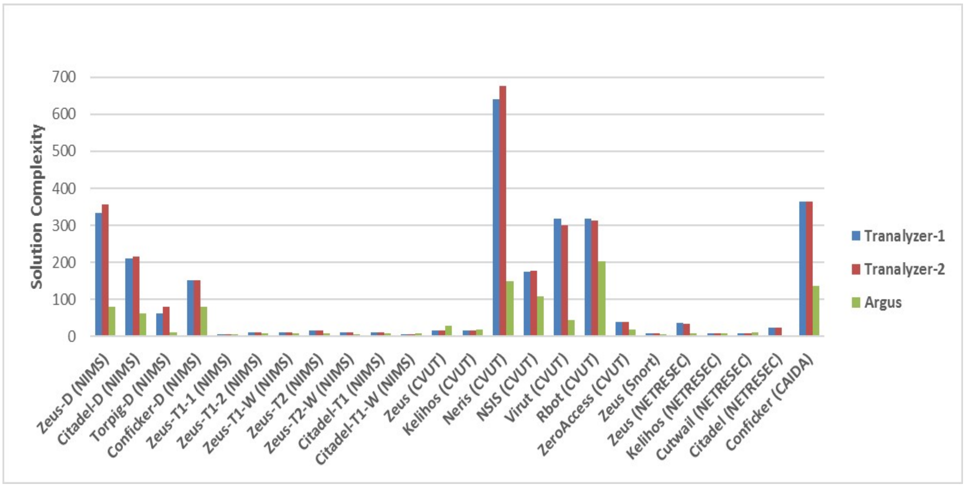


Figure 6.25: Argus vs. Tranalyzer solution complexity analysis.

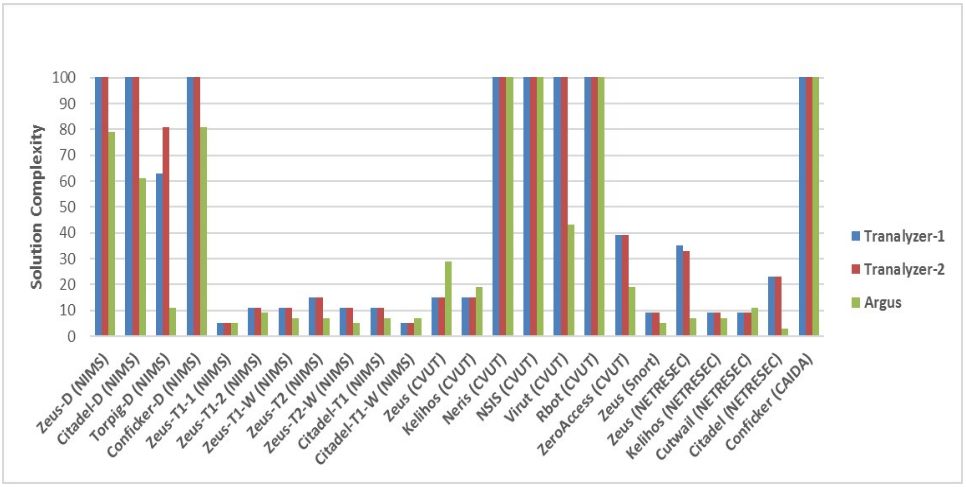


Figure 6.26: Argus vs. Tranalyzer solution complexity analysis (zoomed).

can these combinations differentiate botnet behaviours in general from legitimate behaviours? To this end, a new balanced data set was generated combining the botnet data sets employed in this chapter, labelled as 'botnet' and the ISP (WiSNet) legitimate data set labelled as 'legitimate'. This data set is referred to as BvL (Botnet vs. Legitimate). BvL is a balanced data set with over 2.5 million instances. Table 6.13 shows the results of this classification indicating that C4.5 can differentiate botnet behaviour from legitimate behaviour using Tranalyzer-2 and Argus feature sets. Similar to the conclusion reached in the previous section, both of the feature sets resulted in similar performances (in terms of Score and FPR) while Argus formed a less complex solution.

These results indicate as well that different types and versions of botnets do have some similarities since C4.5 can put them all together as one class. This can be because of the similar automated nature of botnet behaviour in general. In order to understand whether these botnets have enough distinct behaviours that can be used to differentiate them despite the similarities, another experiment was run. In this new experiment, a multi-class data set was generated (called BvL-multiClass) which has twenty-four classes: one legitimate class and the twenty-three botnet classes utilized in the analysis of this chapter. Since some of the data sets used in this work are much larger than the others (such as the Conficker-D and CVUT data sets), the new multi-class data set was kept unbalanced, consisting of all of the flow samples used in Section . The results of this experiment, shown in Figure 6.27 and Table 6.14, are

Table 6.12: Argus vs. Tranalyzer classification Results.

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
Tranalyzer-1	Zeus-D (NIMS)	99.75%	99.8%	0.3%	99.7%	0.2%	50.53	333
	Citadel-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	21.94	211
	Torpig-D (NIMS)	98.45%	99.1%	2.2%	97.8%	0.9%	0.7	63
	Conficker-D (NIMS)	99.95%	99.9%	0%	100%	0.1%	135.51	151
	Zeus-T1-1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.24	5
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	13.48	11
	Zeus-T1-W (NIMS)	99.87%	100%	0.3%	99.7%	0%	0.3	11
	Zeus-T2 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.43	15
	Zeus-T2-W (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.28	11
	Citadel-T1 (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.92	11
	Citadel-T1-W (NIMS)	100%	100%	0%	100%	0%	0.39	5
	Zeus (CVUT)	100%	100%	0%	100%	0%	72.17	15
	Kelihos (CVUT)	100%	100%	0%	100%	0%	60.11	15
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	167.38	641
	NSIS (CVUT)	99.2%	99.5%	1.1%	98.9%	0.5%	4.94	175
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	195.78	317
	Rbot (CVUT)	99.80%	99.8%	0.2%	99.8%	0.2%	54.55	317
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	87.4	39
	Zeus (Snort)	96.88%	99.3%	5.6%	94.4%	0.7%	0.1	9
	Zeus (NETRESEC)	97.6%	97.9%	2.7%	97.3%	2.1%	0.18	35
	Kelihos (NETRESEC)	99.95%	99.9%	0%	100%	0.1%	0.39	9
	Cutwail (NETRESEC)	100%	100%	0%	100%	0%	0.49	9
	Citadel (NETRESEC)	99.85%	99.9%	0.2%	9.8%	0.1%	0.47	23
	Conficker (CAIDA)	99.85%	99.9%	0.2%	99.8%	0.1%	149.92	365
Tranalyzer-2	Zeus-D (NIMS)	99.75%	99.8%	0.3%	99.7%	0.2%	43.23	357
	Citadel-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	27.19	215
	Torpig-D (NIMS)	98.55%	99.1%	2%	98%	0.9%	0.74	81
	Conficker-D (NIMS)	99.95%	99.9%	0%	100%	0.1%	152.81	151
	Zeus-T1-1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.25	5
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	14.29	11
	Zeus-T1-W (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.29	11
	Zeus-T2 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.48	15
	Zeus-T2-W (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.28	11
	Citadel-T1 (NIMS)	99.95%	99.9%	0%	100%	0.1%	0.99	11
	Citadel-T1-W (NIMS)	100%	100%	0%	100%	0%	0.41	5
	Zeus (CVUT)	100%	100%	0%	100%	0%	72.02	15
	Kelihos (CVUT)	100%	100%	0%	100%	0%	66.32	15
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	208.46	677
	NSIS (CVUT)	99.2%	99.5%	1.1%	98.9%	0.5%	5.33	177
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	214.48	301
	Rbot (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	56.45	313
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	89.09	39
	Zeus (Snort)	96.88%	99.3%	5.6%	94.4%	0.7%	0.09	9
	Zeus (NETRESEC)	97.6%	97.9%	2.7%	97.3%	2.1%	0.2	33
	Kelihos (NETRESEC)	99.95%	99.9%	0%	100%	0.1%	0.39	9
	Cutwail (NETRESEC)	100%	100%	0%	100%	0%	0.5	9
	Citadel (NETRESEC)	99.85%	99.9%	0.2%	99.8%	0.1%	0.5	23
	Conficker (CAIDA)	99.85%	99.9%	0.2%	99.9%	0.1%	163.41	365
Argus	Zeus-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	12.87	79
	Citadel-D (NIMS)	99.95%	99.8%	0.1%	99.9%	0.2%	10.5	61
	Torpig-D (NIMS)	99.55%	99.4%	0.3%	99.7%	0.6%	0.37	11
	Conficker-D (NIMS)	99.95%	99.9%	0%	100%	0.1%	104.51	81
	Zeus-T1-1 (NIMS)	99.8%	100%	0.4%	99.6%	0%	0.12	5
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	3.38	9
	Zeus-T1-W (NIMS)	99.7%	99.7%	0.3%	99.7%	0.3%	0.15	7
	Zeus-T2 (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.59	7
	Zeus-T2-W (NIMS)	99.7%	99.7%	0.3%	99.7%	0.3%	0.14	5
	Citadel-T1 (NIMS)	99.95%	99.9%	0%	100%	0.1%	0.35	7
	Citadel-T1-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.26	7
	Zeus (CVUT)	100%	100%	0%	100%	0%	258.01	29
	Kelihos (CVUT)	100%	100%	0%	100%	0%	49.65	19
	Neris (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	115.49	149
	NSIS (CVUT)	98.8%	99.7%	1.1%	98.9%	1.3%	1.15	107
	Virut (CVUT)	99.95%	99.9%	0%	100%	0.1%	16.45	43
	Rbot (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	279.03	203
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	7.2	19
	Zeus (Snort)	97.33%	96%	1.3%	98.7%	4%	0.05	5
	Zeus (NETRESEC)	98.5%	99.5%	1.5%	98.5%	1.5%	0.09	7
	Kelihos (NETRESEC)	99.9%	100%	0.2%	99.8%	0%	0.23	7
	Cutwail (NETRESEC)	99.9%	99.9%	0.1%	99.9%	0.1%	0.26	11
	Citadel (NETRESEC)	99.1%	99.1%	0.9%	99.1%	0.9%	0.07	3
	Conficker (CAIDA)	99.5%	99.5%	0.5%	99.5%	0.5%	86.21	137

highlighted below.

(i) C4.5 performed well while being able to differentiate twenty-three different botnet behaviours from legitimate behaviours with an overall DR of 92%. However,

Table 6.13: Botnet versus legitimate behaviour using the C4.5 classifier.

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time	Solution
Tranalyzer-2	BvL	99.95%	99.9%	0%	100%	0.1%	2013.02	3025
Argus	BvL	99.95%	99.9%	0%	100%	0.1%	1986.68	573

given that this is an unbalanced multi-class classification, Score (a classwise average DR) can demonstrate the performance better. Based on Score, C4.5 could differentiate the behaviour of twenty-three botnet data sets and one normal data set by an overall maximum accuracy of 88.6% (using the Tranalyzer feature set). This is a good performance given the result of the previous experiment in this section. That showed how the basic/underlying behaviour of botnets can be similar.

(ii) Tranalyzer (with an overall Score of 88.6%) outperformed Argus (with an overall Score of 82.11%). Unlike the results of the previous experiments, the solution complexity of C4.5 with Tranalyzer-2 and Argus was similar in this experiment. Specifically, the size of the tree for Tranalyzer-2 was 19325 and for Argus it was 19463.

(iii) Neither the Tranalyzer nor the Argus feature sets could cause the C4.5 classifier to perform better in detecting Citadel-D (NIMS) and Torpig-D (NIMS). Looking into the confusion matrix, almost all of the misclassified samples of these two botnets are between the NIMS domain name-generated data sets (data sets with the '-D (NIMS)' extension). This was expected as the main focus of all of the data sets in this category is on the communication phase of the botnet lifecycle. It is likely this low performance is caused by the small sample size of these two botnets in the BvL-MultiClass data set as well as how these data sets are generated in a different way (a domain based data generation method).

(iv) The confusion matrix of both of the feature sets indicated that almost all of the miss-classification of botnet classes are within the botnet classes. In other words, almost all of the botnet misclassifications are classified as another type of botnet, not as legitimate behaviour. This result is consistent with the conclusion made in the previous experiment of this section which indicates that C4.5 (using the Tranalyzer of Argus feature set) can successfully differentiate the botnet behaviour in general from normal behaviours.

Finally, a TTest was run on the FPRs shown in Table 6.14. The *P-value* of 0.027 indicates that the FPR performance of the C4.5 classifier has a statistically

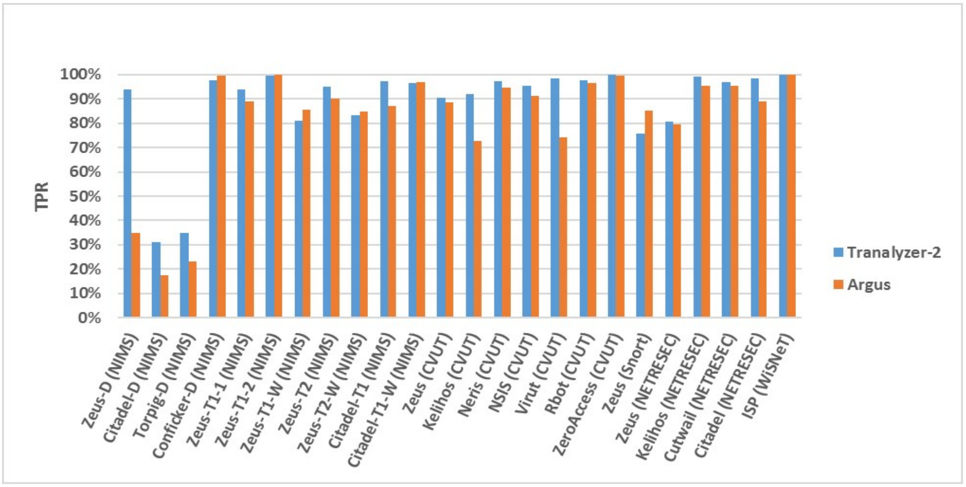


Figure 6.27: Multi-Class Classification results– TPR.

significant difference using Tranalyzer-2 vs. Argus. Hence, considering all of the above observations, the conclusion is that Tranalyzer-2 outperformed Argus in the botnet multiClass classification experiment.

6.1.2 Summary

Network traffic analysis is a well investigated method for detecting malicious behaviour. This type of analysis has been employed widely as well for botnet behaviour detection in the literature. Most of the works in the literature have concentrated on specific types of botnets, come up with their own feature sets for analyzing the network data and have evaluated the proposed systems in specific situations. In this chapter various feature sets were explored using several public feature extraction tools and different machine learning algorithms to design a flow-based early warning botnet detection system. The first phase used Maji, YAF, Netmate, Softflowd, two versions of Tranalyzer and two versions of the Argus flow exporters in combination with the C4.5, KNN, ANN, SBB, Naive Bayes, Bayesian Networks classifiers. Evaluating the flow exporters under various circumstances such as binary, multi-botnet binary and multi-class multi-botnet classifications demonstrated that the combination of the Tranalyzer-2 flow feature set with the C4.5 classifier is the best performing combination. Therefore, this combination is proposed as the final network traffic analysis-based early warning botnet detection system for this thesis.

Table 6.14: BvL-MultiClass classification results.

Data Set	**Tranalyzer V.2**		**Argus**	
	TPR	**FPR**	**TPR**	**FPR**
Zeus-D (NIMS)	93.8%	1%	34.8%	0.2%
Citadel-D (NIMS)	31.1%	0.1%	17.4%	0.1%
Torpig-D (NIMS)	34.7%	0%	23.3%	0%
Conficker-D (NIMS)	97.8%	0%	99.4%	1.4%
Zeus-T1-1 (NIMS)	93.9%	0%	88.8%	0%
Zeus-T1-2 (NIMS)	99.6%	0%	99.9%	0%
Zeus-T1-W (NIMS)	81.2%	0%	85.5%	0%
Zeus-T2 (NIMS)	95%	0%	90.2%	0%
Zeus-T2-W (NIMS)	83.2%	0%	84.7%	0%
Citadel-T1 (NIMS)	97.2%	0%	87.1%	0%
Citadel-T1-W (NIMS)	96.4%	0%	96.8%	0%
Zeus (CVUT)	90.6%	0.5%	88.5%	3.7%
Kelihos (CVUT)	92.1%	0.6%	72.9%	2.3%
Neris (CVUT)	97.4%	0.1%	94.6%	0.7%
NSIS (CVUT)	95.4%	0%	91.4%	0%
Virut (CVUT)	98.3%	0.1%	74.2%	0.1%
Rbot (CVUT)	97.7%	0.1%	96.5%	0.3%
ZeroAccess (CVUT)	100%	0%	99.7%	0%
Zeus (Snort)	75.7%	0%	85.3%	0%
Zeus (NETRESEC)	80.8%	0%	79.7%	0%
Kelihos (NETRESEC)	99.2%	0%	95.6%	0%
Cutwail (NETRESEC)	96.9%	0%	95.4%	0%
Citadel (NETRESEC)	98.5%	0%	88.9%	0%
ISP (WiSNeT)	100%	0.1%	100%	0.1%
	DR = 97.32% Score = 88.6%		DR = 92.26% Score = 82.11%	

In addition to the performance analysis of feature sets and machine learning algorithms, some of the research questions that have not been investigated in the literature such as the following are addressed.

(i) How can focussing on a specific part of botnet communication be effective in detecting botnet behaviours?

(ii) How might the presentation of non-numeric features affect the performance of the system?

(iii) How would the choice of normal behaviour reflect on the performance of the system?

(iv) Given the botnet evolution, how would a trained early warning system perform/generalize over time?

The results indicate that presenting a rich feature set with a wide range of features to a classifier that is capable of choosing the most informative features based on the provided data (C4.5) is the key design element of the proposed early warning system. Given the dynamic nature of such a system (feature wise), this design can be used in various situations and would perform well in detecting various botnet behaviours. The evaluation suggests that inter-arrival based features are the most important features that are selected and used by the C4.5 to detect botnets. It appears that even recent botnets like de-centralized HTTP-based and P2P botnets have not been able to mimic temporal characteristics of normal user behaviour.

Chapter 7

Evaluations Using State-of-the-art Systems

As discussed previously, network packets include two main parts: (i) the packet header, and (ii) the packet payload. The per-packet analysis can use either of these two parts while per-flow analysis utilizes only network packet headers. This chapter examines how much could be gained (lost) in terms of performance when a system employs payload analysis (flow analysis). To this end, not only are data mining techniques employed but also publicly available intrusion/botnet detection systems to measure performance for both the payload and the traffic flow analysis.

Five detection systems are evaluated and compared. The first two systems are Snort and BotHunter, as the rule-based detection systems. Snort is a popular intrusion detection and prevention system (IDS/IPS). It is open source and therefore its rule set can be customized easily. BotHunter, which is another publicly available system, utilizes the Snort sensors and customizes the Snort rule set to specifically detect botnets. Two botnet detection systems have been implemented based on the the research proposed in [118] and [156]. The first one is a machine learning packet payload-based system and the second one is a machine learning flow-based (packet header-based) detection system. Finally, the flow-based detection system proposed in Chapter 6 is employed. The aim in this chapter is to analyze, evaluate and compare the following systems for botnet detection: (i) a packet payload-based system; (ii) the proposed early warning flow-based system in this thesis; (iii) a flow aggregation/fraction-based system; (iii) Snort intrusion detection system; and (iv) BotHunter botnet detection system. For data mining-based approaches, C4.5 decision tree is employed in this section as it was the best performing classifier in the previous chapter. Furthermore, the C4.5 classifier's output is in the form of rules that makes it easier to be used by a human expert for understanding what this technique models on a given data set.

7.1 Systems Employed

Snort and BotHunter are two publically available intrusion/botnet detection systems which were introduced in Section 3.2.2 as the publicly available tools employed in this thesis. Among the approaches that use packet headers information only, flow-based feature extraction methods have been highly employed in the recent literature [61, 94, 145]. A flow-based detection system was proposed, developed and evaluated in Section 6. The Tranalyzer was shown to be the best performing flow exporter when compared with Maji, YAF, Softflowd, Netmate and Argus. Hence, in this chapter Tranalyzer-2 is employed for exporting the flows and as in Section 6, all of the exported features are used as inputs for the data mining techniques except the IP addresses, port numbers and any non-numeric features. In addition to these systems, two other detection system proposed in the literature were implemented for the analysis and evaluations in this chapter.

7.1.1 Packet payload-based System:

Some of the works in the literature proposed specific packet analysis methods to detect botnet behaviour [89, 145]. These systems have focussed on specific packets and features from the header and/or payload sections of these packets to identify the type of malware they are interested in. For example, Haddadi *et al.* [89] extracted the domain name from the DNS packets to detect automatically generated malicious domain names while Mohaisen et al. [118] introduced a set of features focussing on the Zeus botnet. The features introduced by Mohaisen *et al.* are employed in the evaluations of the proposed packet payload-based system. Table 7.1 presents the selected features for this approach.

Since some of the data mining techniques employed in this work can be applied to only numeric features, string to numeric feature conversions are performed and the quartile object sizes are calculated for each of the data sets. Detailed information of the features can be found in [118]. Once the features are extracted from each data set, C4.5 is applied for classification.

Table 7.1: Packet-based approach– network features.

Feature set	
Port	Source and destination port numbers
Connections	TCP, UDP, RAW
Request type	GET, HEAD, POST
Response type	Response code 200–599
Object Size	Categorised quartiles (1–4)
DNS	MX, NS, A records, PTR, SOA, CNAME

7.1.2 Flow aggregation/fraction-based System

Zhao *et al.* proposed a botnet detection system based on flow intervals [156]. Depending on the value used for the flow interval, this system can cause flow aggregation or flow fraction. In other words, if the interval value is greater than the duration of a flow, corresponding flows (based on the 5-tuple information) will be aggregated. Otherwise, the flow will be divided into smaller chunks. In Zhao's system, a set of Flow features were utilized with several ML algorithms in which a decision tree classifier was selected as the preferred classifier for detecting botnets. The authors focussed on P2P botnets (such as Waledac) that employ the HTTP protocol and a fast-flux-based DNS technique. Furthermore, based on their proposed approach, a web-based detection system was implemented to be used both in offline detection as well as live detection. To evaluate their proposed approach and web-based detection system, they employed a combination of normal and attack traffic, some of which was generated in the lab, some was from Honeynet project traces and some was from the Lawrence Berkeley National Laboratory (normal traffic) data sets. In this thesis, this data set is referred to as ISOT (Uvic). Although their proposed detection approach resulted in up to 99% detection rates with a false positive rate around 2%, they also tested their system with unseen botnet data and obtained detection rates up to 100% while having a false positive rate of about 80% in some cases.

As discussed in Chapter 2, most of the flow-based botnet detection systems have introduced their own set of features which are evaluated mostly on specific types of botnet. Although Zhao *et al.* have introduced their preferred set of features, they have also evaluated the effect of flow aggregation and flow fraction. Table 7.2 shows the feature set used by Zhao *et al.* Therefore, their system was chosen as one of

Table 7.2: Selected feature set in [153].

Feature	Description
SrcIP	Flow source IP address
SrcPort	Flow source port number
DstIP	Flow destination IP address
DstPort	Flow destination port number
Protocol	Transport layer protocol or mixed
APL	Average payload packet length for time interval
PV	Variance of payload packet length for time interval
PX	Number of packets exchanged for time interval
PPS	Number of packets exchanged per second in time interval T
FPS	The size of the first packet in the flow
TBP	The average time between packets in time interval T
NR	The number of reconnects for a flow
FPH	Number of flows from this address over the total number of flows generated per hour

Table 7.3: Classification Results– with a flow interval of 300.

	System	**DR**	**Botnet**		**Legitimate**	
			TPR	**FPR**	**TNR**	**FNR**
C4.5	Zhao *et al.* [156]	-	98.3%	0.1%	99.9%	1.7%
	FlowAF (balanced)	99.9%	99.9%	0.1%	99.9%	0.1%
	FlowAF (unbalanced)	99%	98%	0.4%	99.6%	2.0%

the systems for the evaluations in this chapter. Based on the information given in [155] and [156], a program was created to implement the flow extraction using flow intervals, referred to as FlowAF hereafter. First, FlowAF was evaluated on the same ISOT (Uvic) data set that was used and published by Zhao *et al.* Given that the RepTree classifier with 10-fold cross-validation and a flow interval of 300 sec was used for the evaluation of this approach at [153], the same setting is utilized in this evaluation. Table 7.3 shows the results of FlowAF and the original implementation in [156]. Given that the published results of the original implementation did not mention whether the data set was balanced or unbalanced, FlowAF was run with both of the configurations. It is likely the authors employed the unbalanced setting since they only mentioned using the ISOT (Uvic) data set (which is an unbalanced data set). The results of Table 7.3 demonstrate that FlowAF did perform similarly to the original implementation– specially with the unbalanced setting. Hence, Flow-AF will likely be a good representative of the system proposed by Zhao *et al.* [156] for the evaluations and comparisons in this chapter.

7.2 Evaluations and Results

The evaluation and results of the aforementioned five botnet detection systems are summarized below.

Packet payload-based and Flow-based systems. The Weka [146] implementation of C4.5 was employed for the evaluation of the packet payload-based and the two flow-based systems in this section. Twenty-four balanced data sets were used in this evaluation with ISP (WiSNet) representing the normal behaviour. Table 7.4 shows the detailed classification results of these three systems. Figure 7.1 displays the Score performance specifically. As discussed in Chapter 6, the IP addresses and Port numbers of the flow features were removed in order to design the Tranalyzer-2-based early warning system. This should increase the generalization abilities of the detection systems for unseen behaviour. Hence, two versions of the FlowAF system were included in this section, one using all of the features introduced in [156] (referred to as FlowAF Original here) and another excluding the IP addresses and Port numbers (referred to as FlowAF here). FlowAF would be a better base of comparison for the proposed early warning system using the Tranalyzer-2 feature set.

The results in Figures 7.1 and 7.2 and Table 7.4 demonstrate the following.

(i) The packet-based detection system is the worst performing system among the four. The main draw back of this approach is that it cannot be used when the packet payload data is not accessible, which is the case for Conficker (CAIDA) here. Moreover, packet-based detection systems performed very well and showed very promising results in terms of FPRs in comparison with the flow-based system in [88]. However, this is not the case in with the evaluations here as well as in [91]. This is because these payload features are more focussed on the botnets that use HTTP as their communication protocol. Hence, they should be crafted and modified when being applied to other types of botnets given that packet payloads hold specific information based on the packet communication protocol. However, the flow features being analyzed in the flow-based detection systems use the packet headers. These features are generated based on the aggregation of several packets forming a connection and are more focussed on the general characteristics of the traffic (packets) being sent/received (such as size and timing) rather than the details of each packet. Hence, they could be generalized and utilized in various scenarios.

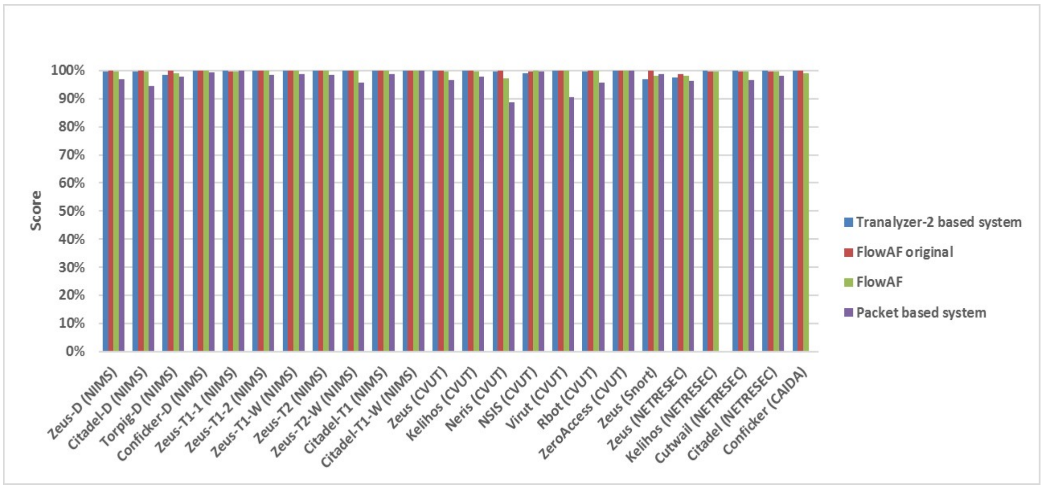


Figure 7.1: Score classification results

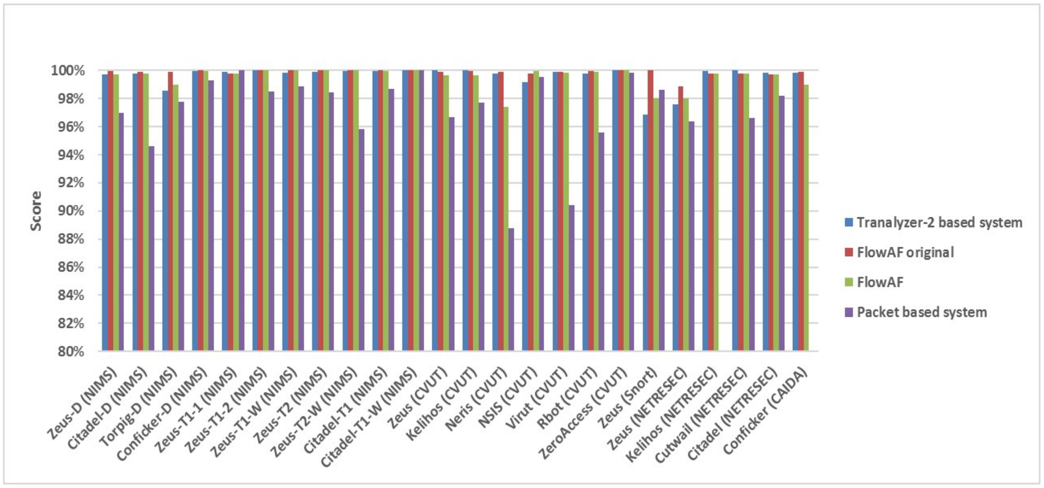


Figure 7.2: Score classification results (zoomed)

(ii) FlowAF Original performed slightly better than the Tranalyzer-2-based system and FlowAF. This can be justified by the additional IP address and port number information that is utilized by FlowAF Original given that the FlowAF and Tranalyzer-2-based system performances are very similar as shown in Figure 7.2. The FPR analysis in Figure 7.3 confirms that the Tranalyzer-2-based system and FlowAF have a similar performance while FlowAF Original performed better.

In order to understand how the two FlowAF systems would perform in differentiating several botnet behaviours from normal behaviour, the systems were tested on the BvL data set introduced in Chapter 6. Table 7.5 shows the result of this experiment. Based on this result, FlowAF Original still out performed FlowAF. However, this time

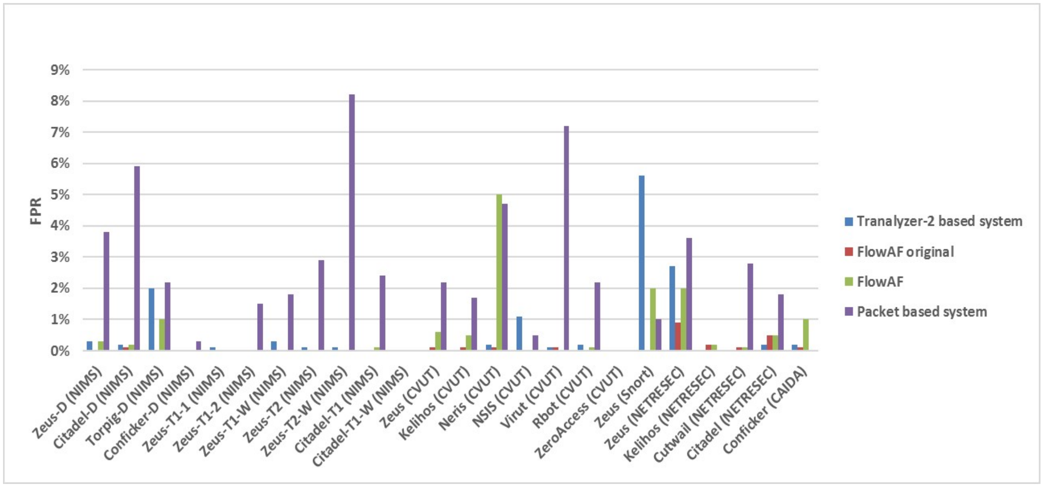


Figure 7.3: FPR classification results

Table 7.4: Classification Results

	Data Set	**Score**	**Botnet**		**Legitimate**		**Complexity**	
			TPR	**FPR**	**TNR**	**FNR**	**Time (sec)**	**Solution**
Packet-based	Zeus-D (NIMS)	97%	97.8%	3.8%	96.2%	2.2%	0.19	57
	Citadel-D (NIMS)	94.64%	95.2%	5.9%	94.1%	4.8%	0.12	95
	Torpig-D (NIMS)	97.8%	97.8%	2.2%	97.8%	2.2%	0.08	31
	Conficker-D (NIMS)	99.28%	98.9%	0.3%	99.7%	1.1%	0.44	37
	Zeus-T1-1 (NIMS)	100%	100%	0%	100%	0%	0.01	5
	Zeus-T1-2 (NIMS)	98.5%	98.5%	1.5%	98.5%	1.5%	0.01	5
	Zeus-T1-W (NIMS)	98.88%	99.6%	1.8%	98.2%	0.4%	0.03	5
	Zeus-T2 (NIMS)	98.45%	99.8%	2.9%	97.1%	0.2%	0.06	11
	Zeus-T2-W (NIMS)	95.8%	99.8%	8.2%	91.8%	0.2%	0.05	9
	Citadel-T1 (NIMS)	98.7%	99.9%	2.4%	97.6%	0.1%	0.07	7
	Citadel-T1-W (NIMS	100%	100%	0%	100%	0%	0.04	3
	Zeus (CVUT)	96.7%	95.6%	2.2%	97.8%	4.4%	6.03	259
	Kelihos (CVUT)	97.7%	97.2%	1.7%	98.3%	2.8%	1.3	75
	Neris (CVUT)	88.8%	82.5%	4.7%	95.3%	17.5%	1.35	73
	NSIS (CVUT)	99.53%	99.5%	0.5%	99.5%	0.5%	0.07	9
	Virut (CVUT)	90.4%	88%	7.2%	92.8%	12%	0.21	65
	Rbot (CVUT)	95.6%	93.3%	2.2%	97.8%	6.7%	0.01	5
	ZeroAccess (CVUT)	99.85%	99.7%	0%	100%	0.3%	0.08	5
	Zeus (Snort)	98.6%	98.1%	1%	99%	1.9%	0.02	5
	Zeus (NETRESEC)	96.4%	96.4%	3.6%	96.4%	3.6%	0.03	11
	Kelihos (NETRESEC)	-%	-%	-%	-%	-%	-	-
	Cutwail (NETRESEC)	96.6%	98.9%	2.8%	97.2%	4.1%	0.08	19
	Citadel (NETRESEC)	98.2%	98.2%	1.8%	98.2%	1.8%	0.02	7
	Conficker (CAIDA)	-	-	-	-	-	-	-
Tranalyzer-2 Flow-based	Zeus-D (NIMS)	99.75%	99.8%	0.3%	99.7%	0.2%	43.23	357
	Citadel-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	27.19	215
	Torpig-D (NIMS)	98.55%	99.1%	2%	98%	0.9%	0.74	81
	Conficker-D (NIMS)	99.95%	99.9%	0%	100%	0.1%	152.81	151
	Zeus-T1-1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.25	5
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	14.29	11
	Zeus-T1-W (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.29	11
	Zeus-T2 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	0.48	15
	Zeus-T2-W (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.28	11
	Citadel-T1 (NIMS)	99.95%	99.9%	0%	100%	0.1%	0.99	11
	Citadel-T1-W (NIMS)	100%	100%	0%	100%	0%	0.41	5
	Zeus (CVUT)	100%	100%	0%	100%	0%	72.02	15
	Kelihos (CVUT)	100%	100%	0%	100%	0%	66.32	15
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	208.46	677

	NSIS (CVUT)	99.2%	99.5%	1.1%	98.9%	0.5%	5.33	177
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	214.48	301
	Rbot (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	56.45	313
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	89.09	39
	Zeus (Snort)	96.88%	99.3%	5.6%	94.4%	0.7%	0.09	9
	Zeus (NETRESEC)	97.6%	97.9%	2.7%	97.3%	2.1%	0.2	33
	Kelihos (NETRESEC)	99.95%	99.9%	0%	100%	0.1%	0.39	9
	Cutwail (NETRESEC)	100%	100%	0%	100%	0%	0.5	9
	Citadel (NETRESEC)	99.85%	99.9%	0.2%	99.8%	0.1%	0.5	23
	Conficker (CAIDA)	99.85%	99.9%	0.2%	99.9%	0.1%	163.41	365
	Zeus-D (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.06	47
	Citadel-D (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	79	1.24
	Torpig-D (NIMS)	99.9%	99.8%	0%	100%	0.2%	0.14	9
	Conficker-D (NIMS)	100%	100%	0%	100%	0%	45	15.02
	Zeus-T1-1 (NIMS)	99.8%	99.6%	0%	100%	0.4%	0.1	9
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	0.34	5
	Zeus-T1-W (NIMS)	100%	100%	0%	100%	0%	0.07	5
	Zeus-T2 (NIMS)	100%	100%	0%	100%	0%	0.08	5
	Zeus-T2-W (NIMS)	100%	100%	0%	100%	0%	0.07	5
	Citadel-T1 (NIMS)	100%	100%	0%	100%	0%	0.12	5
	Citadel-T1-W (NIMS)	100%	100%	0%	100%	0%	0.11	5
FlowAF Original	Zeus (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	24.95	165
	Kelihos (CVUT)	99.95%	100%	0.1%	99.9%	0%	19.01	225
	Neris (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	8.31	99
	NSIS (CVUT)	99.8%	99.9%	0%	100%	0.1%	0.24	5
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	1.47	45
	Rbot (CVUT)	99.95%	99.9%	0%	100%	0.1%	2.38	45
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	1.57	9
	Zeus (Snort)	100%	100%	0%	100%	0%	0.02	5
	Zeus (NETRESEC)	98.86%	98.6%	0.9%	99.1%	1.4%	0.05	13
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.11	9
	Cutwail (NETRESEC)	99.8%	99.7%	0.1%	99.9%	0.3%	0.15	13
	Citadel (NETRESEC)	99.7%	100%	0.5%	99.5%	0%	0.08	11
	Zeus-D (NIMS)	99.7%	99.7%	0.3%	99.7%	0.3%	1.34	11
	Citadel-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	1.26	41
	Torpig-D (NIMS)	99%	99%	0.1%	99%	0.1%	0.112	11
	Conficker-D (NIMS)	99.94%	99.9%	0%	100%	0.1%	9.31	13
	Zeus-T1-1 (NIMS)	99.8%	99.6%	0%	100%	0.4%	0.09	5
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	0.08	5
	Zeus-T1-W (NIMS)	100%	100%	0%	100%	0%	0.07	5
	Zeus-T2 (NIMS)	100%	100%	0%	100%	0%	0.31	5
	Zeus-T2-W (NIMS)	100%	100%	0%	100%	0%	0.7	5
	Citadel-T1 (NIMS)	99.95%	100%	0.1%	99.9%	0%	0.1	5
	Citadel-T1-W (NIMS)	100%	100%	0%	100%	0%	0.1	5
FlowAF	Zeus (CVUT)	99.67%	100%	0.6%	99.4%	0%	17.51	75
	Kelihos (CVUT)	99.65%	99.8%	0.5%	99.5%	0.2%	16.81	131
	Neris (CVUT)	97.38%	99.8%	5%	95%	0.2%	6.8	129
	NSIS (CVUT)	99.95%	99.9%	0%	100%	0.1%	0.19	11
	Virut (CVUT)	99.85%	99.7%	0%	100%	0.3%	0.89	25
	Rbot (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	1.34	25
	ZeroAccess (CVUT)	100%	100%	0%	100%	0%	1.61	7
	Zeus (Snort)	98%	98%	2%	98%	2%	0.03	5
	Zeus (NETRESEC)	98%	98%	2%	98%	2%	0.07	11
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.1	9
	Cutwail (NETRESEC)	99.8%	99.8%	0.1%	99.8%	0.2%	0.12	7
	Citadel (NETRESEC)	99.74%	100%	0.5%	99.5%	0%	0.6	7
	Conficker (CAIDA)	99%	99%	1%	99%	1%	124.2	218

the Tranalyzer-2-based system performed better than FlowAF and had a performance more similar to FlowAF Original. This is an advantage for the Tranalyzer-2-based system which can achieve the same performance as FlowAF Original without using

Table 7.5: Botnet versus legitimate behaviour using the c4.5 classifier.

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time	Solution
FlowAF original	BvL	99.95%	99.9%	0%	100%	0.1%	228.26	1109
FlowAF	BvL	98.9%	98.1%	0.4%	99.6%	1.9%	176.7	491
Tranalyzer-2	BvL	99.95%	99.9%	0%	100%	0.1%	2013.02	3025

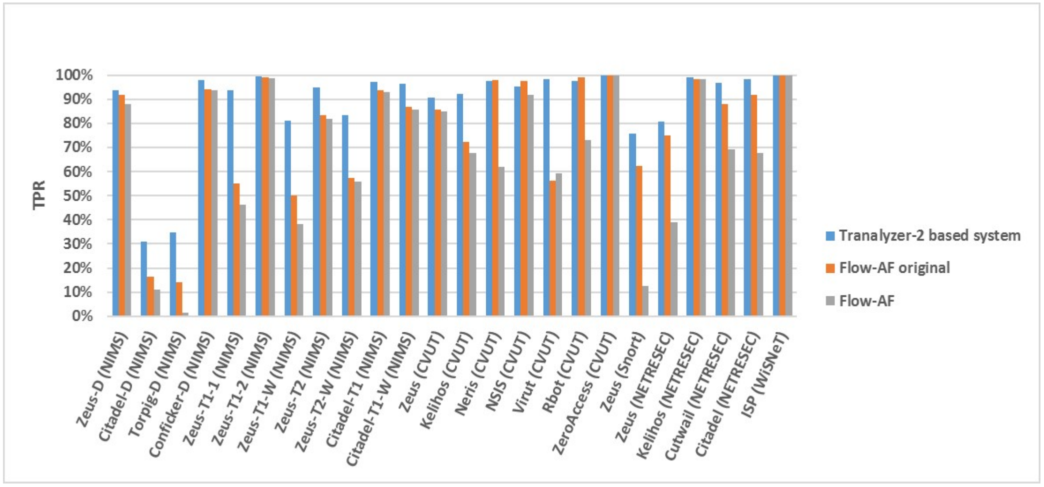


Figure 7.4: Multi-Class classification results.

the IP addresses and port numbers which would create a more generalized system. In the next step, the FlowAF systems were run against the BvL-MultiClass data set (introduced in Chapter 6) to test how these systems would perform on a botnet multi-class data set. Table 7.6 and Figure 7.4 present the results of this experiment. The results indicate the Tranalyzer-2-based system outperformed the two FlowAF systems with an overall Score result of 88.6%.

In short, using specific and confined payload feature sets can only be useful in specific scenarios (in this case specific types of botnets) for which they are actually designed, and therefore, they should be modified when facing different situations. This can be the case as well when dealing with a limited number of features in the flow-based systems. However, the advantage of the proposed early warning flow-based detection system is that it provides a wide range of features and then uses a classifier which can chose the proper set of features for each scenario. This is the main reason behind the consistency of good performance in the proposed Tranalyzer flow-based detection system which has been evaluated under numerous scenarios.

BotHunter. This botnet detection tool provides Snort installation with a customized malware rule set from the ET (Emerging Threats) and DNS/IP blacklist.

Table 7.6: BvL-MultiClass classification results.

Data Set	**Tranalyzer V.2**		**FlowAF Original**		**FlowAF**	
	TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Zeus-D (NIMS)	93.8%	1%	91.7%	1.8%	87.8%	1.8%
Citadel-D (NIMS)	31.1%	0.1%	16.3%	0.1%	11.0%	0.1%
Torpig-D (NIMS)	34.7%	0%	14.1%	0%	1.5%	0%
Conficker-D (NIMS)	97.8%	0%	94.1%	0.2%	93.6%	0.2%
Zeus-T1-1 (NIMS)	93.9%	0%	55.2%	0%	46.1%	0%
Zeus-T1-2 (NIMS)	99.6%	0%	98.9%	0%	98.6%	0%
Zeus-T1-W (NIMS)	81.2%	0%	50.2%	0%	38.1%	0%
Zeus-T2 (NIMS)	95%	0%	83.4%	0%	82.0%	0%
Zeus-T2-W (NIMS)	83.2%	0%	57.3%	0%	56.0%	0%
Citadel-T1 (NIMS)	97.2%	0%	93.9%	0.1%	93.1%	0.1%
Citadel-T1-W (NIMS)	96.4%	0%	86.8%	0%	85.6%	0%
Zeus (CVUT)	90.6%	0.5%	85.5%	3.3%	85.0%	4.3%
Kelihos (CVUT)	92.1%	0.6%	72.2%	1.9%	67.8%	2.2%
Neris (CVUT)	97.4%	0.1%	98.1%	0.7%	61.8%	0.9%
NSIS (CVUT)	95.4%	0%	97.5%	0%	91.8%	0%
Virut (CVUT)	98.3%	0.1%	56.1%	0.1%	59.3%	0.3%
Rbot (CVUT)	97.7%	0.1%	99.1%	0%	73.0%	0.6%
ZeroAccess (CVUT)	100%	0%	99.9%	0%	99.9%	0%
Zeus (Snort)	75.7%	0%	62.5%	0%	12.5%	0%
Zeus (NETRESEC)	80.8%	0%	74.9%	0%	38.8%	0%
Kelihos (NETRESEC)	99.2%	0%	98.2%	0%	98.5%	0%
Cutwail (NETRESEC)	96.9%	0%	87.9%	0%	69.1%	0%
Citadel (NETRESEC)	98.5%	0%	91.8%	0%	67.8%	0%
ISP (WiSNeT)	100%	0.1%	100%	0.1%	99.7%	1.9%
	DR = 97.32% Score = 88.6%		DR = 92.46% Score = 77.73%		DR = 89.37% Score = 67.43%	

BotHunter should be run in batch mode when the data is in the form of pre-capture traffic log files. In this mode, two output files are created: BotHunter's Snort alert file (used as input for the BotHunter correlator), and bot profiles. As for the configuration, the trusted network/monitored network should be set for each run. Indeed, such a requirement necessitates the users to have information about the data set or the monitored network (if using BotHunter in live mode). In this research, the information provided by the sources of the log files was used to set the trusted network.

Table 7.7 shows the results of BotHunter on the twenty five data sets. The "# infected hosts" column in the table shows the number of infected machines with the bot program. The "# remote hosts" shows the malicious remote machines that the infected hosts communicate with in the captured data sets. Although finding the infected host in the network is important, it is only one phase of the detection. Finding the source of the attacks or at least the remote hosts that are utilized by the C&C servers is another important phase of detection. Therefore, the remote host analysis is included in this table as well. These remote machines can be the malicious C&C servers or new targets of the botnet that the infected machine aims to infect. In order to develop the bot profiles, BotHunter correlates the Snort alerts (shown in the second column) and finally generates the bot profiles revealing the malicious hosts. In Table 7.7, the cells which two numbers are separated by "/" shows the count of IP addresses detected vs. the total number of IP addresses in each column. Moreover, the DR of each cell is provided in parentheses while the overall detection rate of BotHunter, including the infected hosts and the remote hosts, is presented in Table 7.9. This table consists of two sets of overall DR: IP-based and Flow-based. The IP-based set (shown in the 'IP-based' column) is based on the IP addresses detected from Table 7.7. However, to have a better understanding of how the detected IP addresses by BotHunter might reflect on the traffic flows, the flow-based overall DR results have been included. In this case, the botnet flows exported by Tranalyzer-2 were labelled as 'detected' based on the IP addresses detected by BotHunter as shown in Table 7.7. The 'Flow-based' column of Table 7.9 demonstrates the results of this analysis.

Based on the performance of BotHunter presented in Tables 7.7 and 7.9, the observations listed below can be made.

(i) BotHunter output is dependent to the Snort-generated alerts. This dependency causes the BotHunter performance to be affected by the Snort performance. Moreover, the Snort sensor that is utilized by BotHunter is a customized version of Snort. Hence, its rule set will not get updated automatically once a new version of the Snort rule set is made publicly available. In this experiment, the Snort sensor did not generate any alerts for four of the data sets and therefore, no infected machine was detected by BotHunter.

(ii) No alert or bot profile was raised for the Conficker (CAIDA) data set. That is because the payload part of the traffic was not provided by CAIDA. Given that BotHunter and its Snort sensors use the payload of the traffic (packets) for detecting the botnets, they could not perform well on this data set.

(iii) BotHunter could detect all the infected machines successfully as well as the remote hosts of the NIMS sandbox-generated data sets such as Zeus-T1-1 (NIMS). That is because the payload is provided and all the phases of the botnet lifecycle are present in these data sets.

(iv) Although Snort did create high severity alarms on Zeus (Snort) (such as "E4[rb] TROJAN Zeus POST Request to CnC"), BotHunter did not report any bot profile. This shows that even when Snort does a good job of raising alerts on the anomalies, BotHunter may still not be able to create a profile for the infected machine. This could be because a correlation of the different types of alerts (representing the different phases of the lifecycle) is required by BotHunter in order to form a bot profile.

(v) Overall, BotHunter did not performed well in detecting the remote hosts. This makes sense because the focus of this tool is on collecting evidence to find the infected machines in a known trusted network.

(vi) Given the overall flow-based detection performance in Table 7.9, the proposed early warning flow-based system outperformed BotHunter by a considerable gap.

Snort. As discussed in Section 3.2.2, Snort supports two public rule sets: VRT and ET. To run Snort, the first thing required is to determine the rule set that will be used. In this section, the VRT rule set was used because: (1) it is the official rule set for Snort which gets updated frequently, and (2) ET, is the one used by BotHunter. In this evaluation, Snort version 2.9.7.3 with the VRT rule set update

Table 7.7: Detailed BotHunter detection results.

Data Set	# Snort alerts	# Bot profiles	# Infected hosts	# Remote hosts
Zeus-D (NIMS)	11	0	0/1 (0%)	0/369 (0%)
Citadel-D (NIMS)	0	0	0/1 (0%)	0/87 (0%)
Torpig-D (NIMS)	0	0	0/1 (0%)	0/60 (0%)
Conficker-D (NIMS)	8	0	0/1 (0%)	0/1920 (0%)
Zeus-T1-1 (NIMS)	486	77	12/12 (100%)	2/2 (100%)
Zeus-T1-2 (NIMS)	29984	24	12/12 (100%)	2/2 (100%)
Zeus-T1-W (NIMS)	847	181	12/12 (100%)	2/2 (100%)
Zeus-T2 (NIMS)	8914	64	12/12 (100%)	1/1 (100%)
Zeus-T2-W (NIMS)	802	85	12/12 (100%)	1/1 (100%)
Citadel-T1 (NIMS)	32939	24	12/12 (100%)	1/1 (100%)
Citadel-T1-W (NIMS)	3363	22	12/12 (100%)	1/1 (100%)
Zeus (CVUT)	26076	0	0/3 (0%)	0/9392 (0%)
Kelihos (CVUT)	9935	0	0/1 (0%)	0/25671 (0%)
Neris (CVUT)	3006	526	10/10 (100%)	69/18369 (0.4%)
NSIS (CVUT)	96	6	3/3 (100%)	10/5364 (0.2%)
Virut (CVUT)	171	6	1/1 (100%)	8/1798 (0.5%)
Rbot (CVUT)	40521	8	1/10 (10%)	42/30046 (0.1%)
ZeroAccess (CVUT)	0	0	0/2 (0%)	0/15495 (0%)
Zeus (Snort)	26	0	0/1 (0%)	0/35 (0%)
Zeus (NETRESEC)	23	1	1/1 (100%)	7/28 (63.6%)
Kelihos (NETRESEC)	2	0	0/1 (0%)	0/268 (0%)
Cutwail (NETRESEC)	416	0	0/1 (0%)	0/162 (0%)
Citadel (NETRESEC)	216	0	0/1 (0%)	0/1 (0%)
Conficker (CAIDA)	0	0	0/360191 (0%)	0/80380 (0%)
ISOT (Uvic)	831	16	4/5 (80%)	40/15000 (0.3%)

on July 2, 2015 was used. Tables 7.8 and 7.9 show the performance of Snort on the twenty-four data sets. Similar to the previous observations [88], Snort raises a lot of alerts for big data sets that contain considerable numbers of malicious traffic. This makes any post-analysis of the results very complicated. Hence, SnortSnarf was used to produce HTML output from Snort alerts. Tools like this are intended for diagnostic inspection and tracking down problems given the high number of Snort alerts. SnortSnarf selected/filtered any alert with a high priority that was raised on botnet-related classes of alerts (such as [Classification: A Network Trojan was detected) Priority 1]).

In Table 7.8, column '# Snort alerts', '# Infected hosts' and '# Remote hosts' present the number of Snort alerts generated, the number of detected infected hosts (based on the IP address) and the number of detected remote hosts, respectively. Additionally, a description of the major Snort Alert that highlighted the botnet behaviour is provided in the 'Description' column. Conficker-D (NIMS) is the only data set for which a very specific botnet behaviour rule was not triggered by Snort. The results are summarized below.

(i) Snort performed really well in detecting the infected hosts. Specifically, it has a 100% DR for tewenty-two data sets out of the twenty-five.

(ii) The C&C DNS-based behaviour of the domain-based generated data sets (with the '-D (NIMS)' extension) were detected by Snort for three data sets out of four in this category.

(iii) The NIMS sandbox-generated data sets [such as Zeus-T1-2 (NIMS)] were all detected as a 'Zeus botnet variant' by Snort. Although there are Citadel data sets in that category, given that Citadel is in fact a variant of the Zeus botnet, Snort could very well detect the type of botnet behaviour in these data sets. In this regard, Snort could as well recognize the type of botnet behaviour for Zeus (CVUT), Rbot (CVUT), ZeroAccess (CVUT), Zeus (Snort), Zeus (NETRESEC), Cutwail (NETRESEC) and Citadel (NETRESEC). It should be noted that Cutwail and Pushdo can be considered to be the same botnet type. By contrast, there are some of the botnets for which the type is not identified correctly (e.g. detecting Kelihos (CVUT) as Zeus or Pushdo botnets). However, recognizing the torjan behaviour of these data sets is still a promising result.

(iv) The Snort performance in detecting the remote hosts is much better than the BotHunter's since in some of the data sets there was a 100% detection rate. However, it underperformed the two flow-based detection systems in this section (namely, Tranalyzer-2-based system and FlowAF) given the overall flow-based DR shown in Table 7.9.

7.2.1 Discussion and Highlights

In conclusion, automatic pattern discovery in large traffic data sets is the main advantage of the packet payload-based and flow-based systems. Regardless of whether a packet payload-based system or a flow-based system is used, having a specific feature set might only limit the good performance of a system into a certain scenarios or situations for which it is actually designed. Hence, it should be modified when facing different situations. This is the case for the packet payload-based approach and FlowAF in this chapter. Providing a wide range of features to a pattern discovery algorithm (in this case C4.5), would enable the algorithm to select the proper set of features based on the scenario. Presumably, this is the main reason behind the consistency of good performance in the proposed Tranalyzer flow-based detection system.

BotHunter correlates Snort alerts corresponding to the botnet lifecycle to find the profiles of the infected machines. Hence, in case in which the Snort rule set is

Table 7.8: Detailed Snort detection results.

Data Set	# Snort alerts	# Infected hosts	# Remote hosts	Description
Zeus-D (NIMS)	2265	1/1 (100%)	55/369 (14.9%)	INDICATOR-COMPROMISE Suspicious .cc dns query (classification: A Network Trojan was detected)
Citadel-D (NIMS)	1298	1/1 (100%)	44/87 (50.6%)	INDICATOR-COMPROMISE Suspicious .cc dns query (classification: A Network Trojan was detected)
Torpig-D (NIMS)	156	1/1 (100%)	13/60 (21.7%)	INDICATOR-COMPROMISE Suspicious .cc dns query (classification: A Network Trojan was detected)
Conficker-D (NIMS)	4064	0/1 (0%)	143/1920 (7.5%)	–
Zeus-T1-1 (NIMS)	789	12/12 (100%)	2/2 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Zeus-T1-2 (NIMS)	19199	12/12 (100%)	2/2 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Zeus-T1-W (NIMS)	756	12/12 (100%)	2/2 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Zeus-T2 (NIMS)	6064	12/12 (100%)	2/2 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Zeus-T2-W (NIMS)	766	12/12 (100%)	2/2 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Citadel-T1 (NIMS)	29845	12/12 (100%)	1/1 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Citadel-T1-W (NIMS)	1623	12/12 (100%)	1/1 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Zeus (CVUT)	155763	3/3 (100%)	2800/9392 (29.8%)	MALWARE-CNC Win.Trojan.Zeus v3 DGA DNS query detected, MALWARE-CNC Win.Trojan.Zeus outbound connection (classification: A Network Trojan was detected)
Kelihos (CVUT)	79839	1/1 (100%)	9888/25671 (38.5%)	MALWARE-CNC Win.Trojan.Zeus outbound connection, MALWARE-CNC Win.Trojan.Pushdo variant outbound connection (classification: A Network Trojan was detected)
Neris (CVUT)	76590	10/10 (100%)	2184/18369 (11/.9%)	MALWARE-CNC Possible Zeus User-Agent - Download, INDICATOR-OBFUSCATION potential Javascript unescape obfuscation attempt detected, PROTOCOL-DNS TMG Firewall Client long host entry exploit attempt (classification: A Network Trojan was detected)
NSIS (CVUT)	3390	3/3 (100%)	577/5364 (10.8%)	BLACKLIST URI request for known malicious URI (classification: A Network Trojan was detected)
Virut (CVUT)	6149	1/1 (100%)	175/1798 (9.7%)	PROTOCOL-DNS TMG Firewall Client long host entry exploit attempt, MALWARE-CNC Possible Zeus User-Agent - Download, MALWARE-CNC Win.Trojan.Bulknet variant outbound connection (classification: A Network Trojan was detected)
Rbot (CVUT)	367899	10/10 (100%)	3404/30046 (11.3%)	INDICATOR-COMPROMISE IRC message on non-standard port, MALWARE-OTHER generic IRC botnet connection, POLICY-SOCIAL IRC message (classification: A Network Trojan was detected)
ZeroAccess (CVUT)	176374	2/2 (100%)	15481/15495 (99.9%)	MALWARE-CNC Win.Trojan.ZeroAccess outbound communication (classification: A Network Trojan was detected)
Zeus (Snort)	37	1/1 (100%)	7/14 (50%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Zeus (NETRESEC)	163	1/1 (100%)	7/11 (63.6%)	MALWARE-CNC Win.Trojan.Zeus outbound connection, MALWARE-CNC Win.Trojan.FareIt variant outbound connection (classification: A Network Trojan was detected)
Kelihos (NETRESEC)	30	1/1 (100%)	9/268 (3.4%)	MALWARE-CNC Win.Trojan.Fareit variant outbound connection (classification: A Network Trojan was detected)
Cutwail (NETRESEC)	3823	1/1 (100%)	162/162 (100%)	MALWARE-CNC Win.Trojan.Pushdo variant outbound connection (classification: A Network Trojan was detected)
Citadel (NETRESEC)	225	1/1 (100%)	1/1 (100%)	MALWARE-CNC Win.Trojan.Zeus variant outbound connection (classification: A Network Trojan was detected)
Conficker (CAIDA)	7244	6457/360191 (1.8%)	430/80380 (0.5%)	(Classification: Potential Corporate Privacy Violation)
ISOT (Uvic)	102755	2/5 (40%)	2326/15000 (15.5%)	INDICATOR-COMPROMISE Suspicious .cc dns query, PROTOCOL-DNS TMG Firewall Client long host entry exploit attempt (Classification: Attempted User Privilege Gain)

Table 7.9: BotHunter and Snort overall performance on botnet data samples.

	Dataset	**Overall Detection**	
		IP-based	**Flow-based**
BotHunter	Zeus-D (NIMS)	0%	0%
	Citadel-D (NIMS)	0%	0%
	Torpig-D (NIMS)	0%	0%
	Conficker-D (NIMS)	0%	0%
	Zeus-T1-1 (NIMS)	100%	100%
	Zeus-T1-2 (NIMS)	100%	100%
	Zeus-T1-W (NIMS)	100%	100%
	Zeus-T2 (NIMS)	100%	100%
	Zeus-T2-W (NIMS)	100%	100%
	Citadel-T1 (NIMS)	100%	100%
	Citadel-T1-W (NIMS)	100%	100%
	Zeus (CVUT)	0%	0%
	Kelihos (CVUT)	0%	0%
	Neris (CVUT)	0.4%	19.63%
	NSIS (CVUT)	0.2%	0.7%
	Virut (CVUT)	0.5%	0.47%
	Rbot (CVUT)	0.14%	1.56%
	ZeroAccess (CVUT)	0%	0%
	Zeus (Snort)	0%	0%
	Zeus (NETRESEC)	66.6%	12%
	Kelihos (NETRESEC)	0%	0%
	Cutwail (NETRESEC)	0%	0%
	Citadel (NETRESEC)	0%	0%
	Conficker (CAIDA)	0%	0%
	ISOT (Uvic)	2.9%	60.3%
Snort	Zeus-D (NIMS)	15.1%	30.1%
	Citadel-D (NIMS)	51.1%	27.9%
	Torpig-D (NIMS)	23%	43.4%
	Conficker-D (NIMS)	7.5%	0.07%
	Zeus-T1-1 (NIMS)	100%	100%
	Zeus-T1-2 (NIMS)	100%	100%
	Zeus-T1-W (NIMS)	100%	100%
	Zeus-T2 (NIMS)	100%	100%
	Zeus-T2-W (NIMS)	100%	100%
	Citadel-T1 (NIMS)	100%	100%
	Citadel-T1-W (NIMS)	100%	100%
	Zeus (CVUT)	29.8%	80.8%
	Kelihos (CVUT)	38.5%	73.6%
	Neris (CVUT)	11.9%	44.13%
	NSIS (CVUT)	10.8%	17.8%
	Virut (CVUT)	9.8%	6.75%
	Rbot (CVUT)	11.4%	19.63%
	ZeroAccess (CVUT)	99.9%	99.8%
	Zeus (Snort)	53.3%	52.8%
	Zeus (NETRESEC)	66.6%	20%
	Kelihos (NETRESEC)	3.7%	2.4%
	Cutwail (NETRESEC)	100%	100%
	Citadel (NETRESEC)	100%	100%
	Conficker (CAIDA)	1.6%	1.3%
	ISOT (Uvic)	15.5%	26.1%

not updated properly based on the type of botnet or when the specific phases of the botnet lifecycle are not presented in the data, BotHunter is not able to detect the bots. However, BotHunter seems to be successful when a specific network is under constant monitoring and the goal is to detect the infected machines of a trusted network. Having said this, if the pre-defined customized Snort rules cannot detect the botnet behaviour, BotHunter would not be able to work properly.

On the other hand, Snort rule sets are updated more frequently and they are not not just focussed on detecting the infected hosts based on tracing the existence of the botnet lifecycle. Instead, it monitors all network communications and flags any suspicious communication that matches its pre-defined rules (using the VRT rule set). Hence, the performance of Snort depends on the quality of the rules. In the evaluations, Snort outperformed BotHunter in the detection of the bot machines as well as the remote hosts but it did not perform better than the flow-based systems.

7.3 Malware/Botnet Detection Analysis in Cellular Networks

Studies have shown that the Malware, specifically botnet, infection rate is increasing exponentially and this growth is not limited to wired networks. Malware have become more aggressive in wireless and cellular networks as well.

The term 'Mobile botnet' refers to a group of compromised mobile phones that are remotely controlled by botmasters via C&C channels. Although these botnets are not as popular as the PC-based botnets given the Internet access limits and the limited resource and battery issues, their popularity is on the rise given that mobile phones are now being used widely and their resources and computing ability are enhancing as well. Mobile botnets exploit the Short Message Service (SMS) and Bluetooth features of the mobile phones. Bluetooth and SMS can both be utilized by a botmaster as the C&C channel to control the bots. A Bluetooth-based C&C channel manages the bot network by transmitting the commands between the Bluetooth-enabled devices. This is a faster and a simpler method of communication compared to SMS. On the other hand, commands and scripts can be transmitted by SMS between the mobile devices. This communication method requires a list of nodes in order to be able to operated on infected phones. Although SMS and Bluetooth are the primary communication channels, wireless Internet access channel can be used as well to transfer commands between a PC-based C&C server and infected bots.

Eslahi *et al.* surveyed the available mobile botnet samples and data sets and also proposed a data collection approach to generate mobile botnet traffic for detection analysis [68]. Based on this survey, the Genome Malware [28] and M0driod [33] provide a huge collection of malware application (apk files) which can be used directly for static analysis. Static analysis refers to the examination and evaluation of a mobile application without execution. Dynamic analysis refers to the evaluation of an application during its execution. In this case, the applications are executed in virtual environments such as Android Emulator and Android-x86 [35]. The MDC [109] and D4D [107] data sets are the most comprehensive mobile behavioural data collections that can be used as a normal pattern for comparing with the malicious behaviour of mobile Botnets. These data sets consists of location, calls, SMS, video/audio data transfer records, but not the traffic traces. Moreover, Anubis [40], CopperDroid [6] and SandDroid [41] are the known resources for executing the applications and perform dynamic behavioural analysis. The authors proposed a framework for creating a testbed environment that consists of two networks: one for infected botnet devices and the other one for real normal mobile devices. The data generated in the testbed is collected, cleaned, labelled and aggregated to form a data set that has both normal and malicious traffic.

Given the frequent change of IP addresses during mobility, many existing group-based botnet detection approaches cannot be applied to mobile Botnets. Although the proposed system is focussed on botnet detection systems on wired networks, potentially it could be employed for mobile botnet detection. This is because the proposed system does not use any IP-based features such as IP-based group behaviour analysis. Hence, the proposed early warning system in this research is evaluated on a limited set of malware samples on cellular networks, as well. For this purpose, four publicly available malware log files were selected from the Contagio (also was used as a data source for Chapter 6) and CopperDroid (introduced previously as a known data source [68]) and malware-traffic-analysis websites [12, 6, 11]. These log files are related to Android malware. Therefore, an Android Application-based log file was obtained from the Crawdad repository to represent the non-malicious data in this Section [131], called AppLegit (CRAWDAD) hereafter. In [131], there are several categories of Android application traces. The "Other" category consists of trace files collected from hosts with non-video and non-audio applications running on it such as Gmail, Facebook, and Dropbox. Hence, to make the AppLegit (CRAWDAD) log file a better representative of Android non-malicious data, a trace file was added from the YouTube application and another trace file from the GoogleHangouts application to the 'Other' trace file. Moreover, one trace file of a Windows malware which attempts to infect the Android devices of the network is included in this section. Table 7.11 shows the description of these log files.

Given that the combination of Tranalyzer-2 feature set and C4.5 classifier is shown to perform good in detection botnet behaviour in wired networks. In this section, I aim to evaluate the performance of this approach for botnet behaviours in cellular networks. For this purpose, balanced data sets are generated for the AndroidMal-1, AndroidMal-2, AndroidMal-3, AndroidMal-4 and AndroidMal-5 log files where AppLegit (CRAWDAD) represent the non-malicious side of the data sets. On the other hand, ISP (WiSNet) is utilized as the legitimate representative of WindowsAndroidMal data set. Table 7.10 shows the classification results of this experiment. As shown in the table, my proposed early warning system also performed very good on cellular related data sets (in both Android and Windows data sets) with the score of up to 100%.

Table 7.10: Classification Results.

Data Set	Score	Malware		Legitimate		Complexity	
		TPR	FPR	TNR	FNR	Time (sec)	Solution
AndroidMal-1	100%	100%	0%	100%	0%	0.02	5
AndroidMal-2	99.67%	100%	0.7%	99.3%	0%	0.03	7
AndroidMal-3	98.55%	97.1%	0%	100%	2.9%	0.02	5
AndroidMal-4	98.82%	97.6%	0%	100%	2.4%	0.02	9
AndroidMal-5	100%	100%	0%	100%	0%	0.04	7
WindowsAndroidMal	96.48%	97.2%	4.2%	95.8%	2.8%	0.04	9

7.4 Summary

The aim of this chapter was to further evaluate the performance of the proposed botnet early warning system proposed in Chapter 6. For this purpose, the proposed systems' performance was compared to four other malicious detection systems. This entailed the implementation of a previously introduced packet payload-based system

[118] and a flow aggregation/fraction-based system [156]. The first system uses features extracted from the header and the payload of packets and the second is a per flow basis system that extract flows from a collection of packets involved in a connection and uses an interval to aggregate or break the flows. The later system is called FlowAF in this chapter. Moreover, two publicly available malicious behaviour detection systems, Snort and BotHunter, were used as the representatives of rule-based detection systems.

The evaluation of all five systems on twenty-five publicly available data sets showed that the proposed flow-based system and FlowAF outperformed the other three systems. Further analysis of the the performance of these two systems under binary-class and multi-class scenarios was done as well as excluding the IP addresses and port numbers from the FlowAF feature sets. The results showed that the proposed Tranalyzer-2-based system outperformed the original FlowAF system (using the IP address and port number information) in a complicated multi-class scenario without even using the IP address and port number information. It appears that not using such information for botnet detection purposes also may increase the generalization capability of the proposed system in this thesis.

As discussed in Chapter 1, the main focus of this research is on botnets over wired networks. However, to further investigate the performance of the proposed system, the system was evaluated on several cellular (wireless) network botnet data sets, as well. The evaluations demonstrate a promising performance for such botnets (a Score of up to 100%) while there is still space for improvement and further evaluations.

Table 7.11: Data specification

Data set	**Type–SubType**	**Size**	**No. of Packets**	**Year**	**Source**	**Description**
AndroidMal-1 [36]	Malicious	241KB	2301	2014	CopperDroid	Listed as Gewpew, FakeBank and BankBot in VirusTotal[1]
AndroidMal-2 [37]	Malicious	1.55MB	2574	2014	CopperDroid	Listed as Gidix and SmSbot in VirusTotal[2]
AndroidMal-3 [38]	Malicious	22KB	227	2014	CopperDroid	Listed as Kaishi in VirusTotal[3] and in Stratosphere[4]
AndroidMal-4 [39]	Malicious	122KB	372	2014	Malware traffic analysis website	Spread by email spams, Listed as NioServ in VirusTotal[5] and Analyzed in malware-traffic-analysis.net [39]
AndroidMal-5 [29]	Malicious	426KB	3325	2012	Contagio	Listed as SmsBot and Spambot in VirusTotal[6]
WindowsAndroidMal [42]	Malicious	45KB	3310	2013	Contagio	Windows malware which attempts to infect android devises, analyzed by Symantec[7]
AppLegit (CRAWDAD) [131]	Non-Malicious	129MB	185521	2015	CRAWDAD	Non-malicious Android application data

Chapter 8

Conclusion and Future Works

Botnets are considered to be one of the main security threats on the Internet due to their reported high infection rate and extensive range of malicious activities with active update capability. Hence, the need for botnet detection approaches that can adapt to botnet evolution is very important. To this end, several automatic botnet detection approaches applying network traffic analysis are proposed in the literature. Each of these systems utilizes particular network traffic features in their analysis of the traffic in which some took advantage of packet payloads to extract the features and others used only packet headers. Since a packet payload contains detailed information about the data being transferred over the network, using this information might be useful in developing a detection system with higher performance and lower complexity. However, packet payloads might not be always available due to privacy issues or encryption. Hence, this thesis proposes two different approaches based on the information available for traffic analysis: (i) application data analysis; and (ii) Network data analysis.

Since legitimate users are not the only ones that use DNS to communicate and botnets also take advantage of this protocol, the first proposed approach is based on DNS application data analysis. Modern botnets avoid hardcoding the address of the C&C server because if the C&C server is identified, they can be blocked at the firewall level. DNS provides a scalable solution for botnets since a list of domain names can be passed to the victim host as possible C&C servers. As long as the victim manages to connect to the server using one of the domains in the list, it will download the malware and join the botnet. However, all the domains on the list (whether they resolve to an IP address or not) need to be blacklisted to be able to mitigate the attack fully. Fortunately for the defenders, botnet DNS traffic exhibits abnormal properties that can be detected. The most important property is the structure of the domains that are being queried, *e.g.* being long with many sub-domains and seemingly random set of characters. Thus, a suitable solution is to monitor the communications at the DNS level to detect abnormal query patterns, specifically queries that a human would not possibly be able to type, based on temporal, structural and syntactic properties. Hence, in the first approach, two solutions are investigated: *a priori*-knowledge-based DNS analysis system and an evolutionary computation-based solution. For the first solution, a priori-knowledge domain name-based feature set is defined. Exploring different machine learning classifiers to detect the malicious domain names using this feature set, C4.5 was found to be the best performing classifier in this solution. For the second solution, the Stateful-SBB classifier was designed and developed to support variable length inputs. In addition to providing a very high accuracy for classifying and generating signatures automatically, Stateful-SBB identifies the set of attributes to be used in classification automatically without requiring any *a priori* knowledge, whereas the typical classifiers evaluated require a fixed set of features extracted based on *a priori* knowledge. The results show that the Stateful-SBB based solution performs comparably to other classification methods without requiring a feature set to be determined *a priori*.

For cases with no access to application data (*i.e.* the packet payload where the domain name is located) or having the payload (application data) encrypted, a second approach was designed and developed. This approach explored the possibility of botnet detection using only the features extracted from the packet header, *i.e.* the flow features. However, detailed analysis has been done on various possible flow-based feature sets, machine learning algorithms and protocol filters for the purpose of understanding the effect of such parameters in finding a solution with the highest performance. This thesis explored six different feature sets extracted by open source flow exporters and investigated the effect of these packet header based-flow features in botnet detection. To this end, several machine learning classifiers that are frequently used in network traffic classification were employed. Given that botnet communication can be divided into different parts (such as locating the C&C server and establishing a connection) and that for each of these parts different protocols are utilized based on the type of botnets, the effect of protocol filters was investigated with the main focus on HTTP-based botnets. As the results show, the choice of feature set and protocol filter is very important and can affect the performance of the botnet detection system greatly. Moreover, the issues of botnet evolution over time, normal behaviour representation, non-numeric feature representation, general botnet behaviour recognition (botnet vs. normal classification) and specific botnet type detection (multi-botnet type classification) are all investigated in this approach. The results suggest that the Tranalyzer open source flow exporter enables the classifiers to identify different botnet behaviours with higher performances on all the data sets employed. For the evaluations performed in this research, the combination of the Tranalyzer-2 tool with the C4.5 classifier gives the best performance on all the botnet data sets employed given the different scenarios explored. Moreover, the results suggest that inter-arrival based features are the most important features that are selected and used by the C4.5 to detect botnets. It appears that even recent botnets like de-centralized HTTP-based and P2P botnets have not been able to mimic temporal characteristics of normal user behaviour.

Furthermore, to investigate how the network analysis-based early warning system proposed in this thesis would perform compared to the other systems publicly available or proposed in the literature, four systems introduced/employed in the literature were explored. These systems include a packet payload based system, a flow aggregation-based system, plus BotHunter and Snort. The results demonstrate that the proposed early warning system outperformed the other systems given the different scenarios that were investigated.

As well, the proposed early warning system was tested for botnets in cellular networks. It appears that none of the proposed botnet detection systems that were designed for wired networks have been evaluated on wireless (Cellular) data sets. The results of this evaluation demonstrate that the performance of the proposed system was promising and seemed to generalize well to the wireless/cellular network data.

8.1 Future research directions

Given the results obtained in this thesis, there are several future research directions which can be pursued. These are listed below.

1. Obtaining botnet network traces for research purposes requires lots of time and effort. The publicly available log files are limited and mostly out-dated. This is a great concern when botnets have shown to grow and evolve over

time. To investigate how a detection system would perform over time and how the approach and the solution would generalize, researchers need to have access to the log file of the same botnet over a period of time. Different data generation methods have been explored as well as a good number of public data sets collected for the evaluations in this thesis. However, there is always room to collect more data sets in order to continue to evaluate the robustness of the proposed approach.

2. In this thesis, balanced data sets have been used mostly for the evaluations. However, usually botnet communications are temporal minor classes. Another area of interest might be to explore the effect of unbalanced data sets. This effect was briefly displayed in the multi-class botnet evaluation section of this thesis. Further analysis is required to improve the performance of the system in such situations.

3. Finally, there is always the potential for additional evaluation of the proposed approach on wireless and cellular networks.

Bibliography

[1] https://labs.snort.org/papers/zeus.html.

[2] Alexa. http://www.alexa.com/topsites.

[3] Cisco ios netflow. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.

[4] Citadel zeus bot. https://www.botnets.fr/index.php/Citadel_ZeuS_bot.

[5] Conficker domain list. http://net.cs.uni-bonn.de/uploads/media/c_domains_april2009.zip.

[6] Copperdroid: dynamic behavioral analysis of android malware. http://copperdroid.isg.rhul.ac.uk/copperdroid/.

[7] DNS-BH- malware domain blocklist. http://www.malwaredomains.com/.

[8] Juniper j-flow. http://www.juniper.net/techpubs/software/erx/junose82/swconfig-ip-services/html/ip-jflow-stats-config2.html.

[9] LBNL enterprise trace repository. http://www.icir.org/enterprise-tracing/.

[10] Maji. http://research.wand.net.nz/software/maji.php.

[11] Malware-traffic-analysis.net. http://malware-traffic-analysis.net/.

[12] Mobile malware mini dump. http://contagiominidump.blogspot.ca/.

[13] Netmate. http://ipmeasurement.org/index.phpoption=com_content&view=article&id=10&Itemid=9.

[14] Netmate flowcalc. http://dan.arndt.ca/nims/calculating-flow-statistics-using-netmate/.

[15] NETRESEC repository: publicly available pcap files. http://www.netresec.com/?page=PcapFiles.

[16] Netscape, dmoz open directory project. http://www.dmoz.org.

[17] Nfdump. http://nfdump.sourceforge.net/.

[18] Opendns: Phishtank. http://www.phishtank.com.

[19] S-flow. http://www.inmon.com/technology/index.php.

[20] Snort. https://www.snort.org/.

[21] Softflowd. http://www.mindrot.org/projects/softflowd/.

[22] Tranalyzer. http://tranalyzer.com/.

[23] Wireless and secure networks research lab. http://wisnet.seecs.nust.edu.pk/index.php.

[24] YAF. http://tools.netsa.cert.org/yaf/index.html.

[25] Zeus tracker. https://zeustracker.abuse.ch/.

[26] Kraken is finally cracked. http://blog.threatexpert.com/2008/04/kraken-is-finally-cracked.html, 2008.

[27] Owning kraken zombies, a detailed dissection. http://dvlabs.tippingpoint.com/blog/2008/04/28/owning-kraken-zombies, 2008.

[28] Android malware genome project. Http://www.malgenomeproject.org/, 2012.

[29] Androidmal-5 log file. http://contagiominidump.blogspot.ca/2012/12/spamsoldier-sms-botnet-sample.html, 2012.

[30] Kaspersky security bulletin 2012: The overall statistics for 2012. https://securelist.com/analysis/kaspersky-security-bulletin/36703/kaspersky-security-bulletin-2012-the-overall-statistics-for-2012/#7, 2012.

[31] The zeroaccess botnet mining and fraud for massive financial gain. https://www.sophos.com/en-us/medialibrary/PDFs/technical2012.

[32] HTTP-Botnets: the dark side of an standard protocol. http://securityaffairs.co/wordpress/13747/cyber-crime/http-botnets-the-dark-side-of-an-standard-protocol.html, 2013.

[33] M0droid. Http://m0droid.uni.me/, 2013.

[34] Snapshot of virut botnet after interruption. http://www.symantec.com/connect/blogs/snapshot-virut-botnet-after-interruption, 2013.

[35] Android-x86 project-run android on your pc. Http://www.android-x86.org/, 2014.

[36] Androidmal-1 log file. http://copperdroid.isg.rhul.ac.uk/copperdroid/view.php?id=5900, 2014.

[37] Androidmal-2 log file. http://copperdroid.isg.rhul.ac.uk/copperdroid/view.php?id=5993, 2014.

[38] Androidmal-3 log file. http://copperdroid.isg.rhul.ac.uk/copperdroid/view.php?id=5998, 2014.

[39] Androidmal-4 log file. http://malware-traffic-analysis.net/2014/03/06/index.html, 2014.

[40] Anubis-malware analysis for unknown binaries. Http://anubis.iseclab.org/, 2014.

[41] Sanddroid-an automatic android program analysis sandbox. http://sanddroid.xjtu.edu.cn/, 2014.

[42] Windowsandroidmal log file. http://contagiominidump.blogspot.ca/2014/01/windows-droidpak-and-android-fakebankb.html, 2014.

[43] Http-botnets: the dark side of an standard protocol. http://securityaffairs.co/wordpress/13747/cyber-crime/http-botnets-the-dark-side-of-an-standard-protocol.html, April 2013.

[44] Infiltering pushdo part 2. http://www.fireeye.com/blog/technical/botnet-activities-research/2010/08/infiltrating-pushdo-part-2-2.html, August 2010.

[45] It's (already) baaack: Kelihos botnet rebounds with new variant. http://www.darkreading.com/attacks-breaches/its-already-baaack-kelihos-botnet-reboun/232700540, March 2012.

[46] Pushdo botnet is evolving, becomes more resilient to takedown attempts. http://www.pcworld.com/article/2038893/pushdobotnetisevolving-becomesmoreresilienttotakedownattempts.html, May 2013.

[47] Zeus/zbot malware shapes up in 2013. http://blog.trendmicro.com/trendlabs-security-intelligence/zeuszbot-malware-shapes-up-in-2013/, May 2013.

[48] Citadel makes a comeback, targets japan users. http://blog.trendmicro.com/trendlabs-security-intelligence/citadel-makes-a-comeback-targets-japan-users/, September 2013.

[49] Hassan Alizadeh, Abdolrahman Khoshrou, and Andre Zuquete. Traffic classification and verification using unsupervised learning of gaussian mixture models. In *Measurements Networking (MN)*, 2014.

[50] E. Alpaydin. *Introduction to Machine Learning.* MIT Press, 2004.

[51] Riyad Alshammari and A. Nur Zincir-Heywood. How robust can a machine learning approach be for classifying encrypted voip? *Network and Systems Management*, 23, 2015.

[52] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: Characterizing internet scam hosting infrastructure. In *the USENIX Security Symposium*, 2007.

[53] M. Antonakakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for dns. In *USENIX Security*, 2010.

[54] Elaheh Biglar Beigi, H.H. Jazi, N. Stakhanova, and A.A. Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *Communications and Network Security (CNS)*, 2014.

[55] Pavani Bharathula and N. Mridula Menon. Equitable machine learning algorithms to probe over p2p botnets. In *Frontiers in Intelligent Computing: Theory and Applications (FICTA)*, 2015.

[56] Leyla Bilge, Sevil Sen, Dacide Balzarotti, Engin Kirda, and Christopher Krugel. Exposure: A passive dns analysis service to detect and report malicious domains. *Information and System Security (TISSEC)*, 16, 2014.

[57] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *Eighth Annual International Conference on Privacy, Security and Trust*, 2010.

[58] Amine Boukhtouta, Serguei A. Mokhov, Nour-Eddine Lakhdari, Mourad Debbabi, and Joey Paquet. Network malware classification comparison using dpi and flow packet headers. *Computer Virology and Hacking Techniques*, 12:69–100, 2016.

[59] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transaction on Evolutionary Computation*, 5:17–26, 2001.

[60] CAIDA Conficker. http://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml.

[61] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller. Salting public traces with attack traffic to test flow classifiers. In *Cyber Security Experimentation and Test (CSET)*, 2011.

[62] Ch. Rossow Ch. J. Dietrich and F. C. Freiling. On botnets that use dns for command and control. In *European Conference on Computer Network Defense (EC2ND)*, 2011.

[63] Youksamay Chanthakoummane, Saiyan Saiyod, Nunnapus Benjamas, and Nattawat Khamphakdee. Improving intrusion detection on snort rules for botnets detection. In *Information Science and Applications (ICISA)*, 2016.

[64] Christian Rossow Christian J. Dietrich and Norbert Pohlmann. Cocospot: Clustering and recognizing botnet command and control channels using traffic analysis. *Computer Networks*, 57, 2013.

[65] Gideon Creech and Jiankun Hu. A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. *IEEE Transactions on Computers*, 63:807–819, 2014.

[66] E. D. de Jong. A monolithic archive for pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[67] J. Doucette, A.R. McIntyre, P. Lichodzijewski, and M. I. Heywood. Symbiotic coevolutionary genetic programming: A benchmarking study under large attribute spaces. *Genetic Programming and Evolvable Machines*, 13:71–101, 2012.

[68] Meisam Eslahi, Mohammad Reza Rostami, H. Hashim, N. M. Tahir, and Maryam Naseri. A data collection approach for mobile botnet analysis and detection. In *IEEE Symposium on Wireless Technology and Applications (ISWTA)*, 2014.

[69] M. Feily and A. Shahrestani. A survey of botnet and botnet detection emerging security information. In *Systems and Technologies*, 2009.

[70] Vahid Aghaei Foroushani and A. Nur Zincir-Heywoood. A proxy identifier based on patterns in traffic flows. In *International Symposium on High Assurance Systems Engineering*, 2015.

[71] J. Francois, Sh. Wang, R. State, and Th. Engel. Bottrack: tracking botnets using netflow and pagerank. *Networking*, 6640:1–14, 2011.

[72] K. Fu and J. Blum. Inside risk controlling for cyberscurity risks of medical device software, October 2013.

[73] Zou Futai, Zhang Siyu, and Rao Weixiong. Hybrid detection and tracking of fast-flux botnet on domain name system traffic. *China Communications*, 10, 2013.

[74] S. Garcia. Malware capture facility project, cvut university– ctu-13 data set. http://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html, 2011.

[75] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-44, 45, 51 and 52. https://mcfp.felk.cvut.cz/publicDatasets/, 2011.

[76] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-54. https://mcfp.felk.cvut.cz/publicDatasets/, 2011.

[77] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-28. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-28/, 2013.

[78] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-3. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-3/, 2013.

[79] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-42, 43 and 50. https://mcfp.felk.cvut.cz/publicDatasets/, 2013.

[80] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-5. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-5/, 2013.

[81] S. Garcia. Malware capture facility project, cvut university– ctu-malware-capture-botnet-53. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-53/, 2013.

[82] S. Garcia. Malware capture facility project, cvut university. https://agents.fel.cvut.cz/malware-capture-facility, February 2013.

[83] S. Garcia, M. Grill, J. Stiborek, and A Zunino. An empirical comparison of botnet detection methods. *Computers and Security*, 45:100–123, 2014.

[84] Timothy Glennan, Christopher Leckie, and Sarah M. Erfani. Improved classification of known and unknown network traffic flows using semi-supervised machine learning. In *Australasian Conference on Information Security and Privacy (ACISP)*, 2016.

[85] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol- and structure- independent botnet detection. In *17th USNIX Security symposium*, 2008.

[86] G. Gu, Ph. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: detecting malware infection through ids-driven dialog correlation. In *16th USENIX Security Symposium on USENIX Security Symposium*, 2007.

[87] Hachem Guerid, Karel Mittig, and Ahmed Serhrouchni. Collaborative approach for inter-domain botnet detection in large-scale networks. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom)*, 2015.

[88] F. Haddadi, , D. L. Cong, L. Porter, and A. N. Zincir-Heywood. On the effectiveness of different botnet detection approaches. In *Information Security Practice and Experience (ISPEC)*, 2015.

[89] F. Haddadi, H.G. Kayacik, A.N. Zincir-Heywood, and M.I. Heywood. Malicious automatically generated domain name detection using stateful-sbb. In *EvoApplication*, 2012.

[90] F. Haddadi, J. Morgan, E. G. Filho, and A. Nur Zincir-Heywood. Botnet behaviour analysis using ip flows with http filters using classifiers. In *Seventh International Workshop on Bio and Intelligent Computing (AINA-BiCOM)*, 2014.

[91] F. Haddadi, Duong-Tien Phan, and A. N. Zincir-Heywood. How to choose from different botnet detection systems? In *Analytics for Network and Service Management (AnNet)*, 2016.

[92] F. Haddadi, D. Runkel, A.N. Zincir-Heywood, and M.I. Heywood. On botnet behaviour analysis using GP and C4.5. In *Genetic and Evolutionary Computation Conference (Gecco) comp.*, 2014.

[93] F. Haddadi and A. N. Zincir-Heywood. Analyzing string format-based classifiers for botnet detection: GP and SVM. In *IEEE Congress on Evolutionary Computation (CEC)*, 2013.

[94] F. Haddadi and A. N. Zincir-Heywood. Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. *IEEE Systems journal*, 2014.

[95] F. Haddadi and A. N. Zincir-Heywood. Data confirmation for botnet traffic analysis. In *Foundations Practice of Security (FPS)*, 2014.

[96] F. Haddadi and A. N. Zincir-Heywood. Botnet detection system analysis on the effect of botnet evolution and feature representation. In *Genetic and Evolutionary Computation Conference (Gecco) comp.*, 2015.

[97] F. Haddadi and A. N. Zincir-Heywood. A closer look at the http and p2p based botnets from a detector's perspective. In *Foundations Practice of Security (FPS)*, 2015.

[98] F. Haddadi and A. N. Zincir-Heywood. Data confirmation for botnet traffic analysis. https://www.cs.dal.ca/research/techreports/cs-2014 -01, May 2014.

[99] S. Haykin. *Neural Networks and Learning Machines*. United States of America: Pearson Education Inc., 2009.

[100] Ch. Holz, Ch. Gorecki, K. Rieck, and F.C. Freiling. Measuring and detecting fast-flux service networks. In *Network and Distributed System Security Symposium (NDSS)*, 2008.

[101] Damballa Inc. Top 10 botnet threats. http://www.damballa.com, 2010.

[102] Open DNS Inc. The Role of DNS in Botnet Command and Control. Witrpaper, 2012.

[103] P. Royal: Damballa Inc. On kraken and bobax botnets. Whitepaper, 2008.

[104] Internet Engineering Task Force (IETF): RFC 2616. http://www.ietf.org/rfc/rfc2616.txt, 1999.

[105] S. Savage J. Ma, L. K. Saul and G. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspecious urls. In *ACM KDD*, 2009.

[106] V. Kirubavathi and R.A. Nadarajan. Http botnet detection using adaptive learning rate multilayer feed-forward neural network. In *Information Security Theory and Practice: security, privacy and trust in computing systems and ambient intelligent ecosystems*, 2012.

[107] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. In *Pervasive Services (ICPS)*, 2010.

[108] V. Krmicek and T. Plesnik. Detecting botnets with netflow. In *Cert Flocon: : An Open Forum for Large-Scale Network Analytics*, 2011.

[109] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Mobile Data Challenge Workshop (MDC) in conjunction with Pervasive*, 2012.

[110] P. Lichodzijewski and M. I. Heywood. Coevolutionary bid-based genetic programming for problem decomposition in classification. *Genetic Programming and Evolvable Machines*, 9:331–365, 2008.

[111] Lei Liu, Songqing Chen, Guanhua Yan, and Zhao Zhang. Bottracer: Execution-based bot-like malware detection. In *Information Security*, 2008.

[112] H. Lodhi, C. Saunders, J. Shawe-Tylor, Nello Cristianini, and Ch.J.C.H. Watkins. Text classification using string kernels. *Machine Learning Research*, 2:419–444, 2002.

[113] W. Lu, G. Rammidi, and A. Ghorbani. Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, 34:502–514, 2011.

[114] F. Jahanian Y. XU M. Bailey, E. Cooke and M. Karir. A survey of botnet technology and defenses. In *Cybersecurity Applications Technology CATCH*, 2009.

[115] A. Makanju, A. N. Zincir-Heywood, and E. Milios. Robust learning intrusion detection for dos attacks on wireless networks. *Intelligent Data Analysis*, 15:801823, 2011.

[116] A. Manasrab, A. Hasan, O.A. Abouabdalla, and S. Ramadass. Detecting botnet activities based on abnormal dns traffic. *Computer Science and Intelligent Computing (IJCSIC)*, 6:97–104, 2009.

[117] Mohammad M. Masud, Tahseen Al-Khateeb, Latifur Khan, Bhavani Thuraisingham, and Kevin W. Hamlen. Flow-based identification of botnet traffic by mining multiple log files. In *Distributed Framework and Applications (DFmA)*, 2008.

[118] A. Mohaisen and O. Alrawi. Unveiling zeus. In *International World Wide Web Conference Committee (IW3C2)*, 2013.

[119] Mozilla. Top level domain names. http://mxr.mozilla.org/mozilla-central/source/netwerk/dns/effective_tld_names.dat?raw=1.

[120] S. Murugan and K. Kuppusamy. System and methodology for unknown malware attack. In *Sustainable Energy and Intelligent Systems (SEISCON)*, 2011.

[121] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Malicious and Unwanted Software (MALWARE)*, 2008.

[122] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting malicious flux service networks through passive analysis of recursive dns traces. In *ACSAC*, 2009.

[123] Ph. Porras, H. Saidi, and V. Yegneswaram. An analysis of conficker's logic and rendezvous points. http://mtc.sri.com/conficker, 2009.

[124] McAfee Labs 2014 Threats Predictions. https://blogs.mcafee.com/mcafee-labs/2014-threats-predictions-botnets-spam-explore-new-avenues-to-steal-data-money/, 2014.

[125] McAfee Labs Threat Reports. http://www.mcafee.com/us/apps/view-all/publications.aspx?tf=aaae16480sz=10, 2013.

[126] RFC 2722. http://tools.ietf.org/html/rfc2722, October 1999.

[127] P. Royal. Maliciousness in top-ranked alexa domains. https://www.barracudanetworks.com/blogs/labsblog?bid=2438.

[128] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, , W. Lu, J. Felix, and P. Hakimian. Detecting p2p botnets through network behaviour analysis and machine learning. In *Privacy, Security and Trust (PST)*, 2011.

[129] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *Detection of Intrusions and Malware Vulnerability Assessment (DIMVA)*, 2014.

[130] A.K. Seewald and F. Kleedorfer. Lambda pruning: an approximation of the string subsequence kernel for practical svm classification and redundancy clustering. *ADAC*, 1:221–239, 2007.

[131] Satadal Sengupta, Harshit Gupta, Niloy Ganguly, Bivas Mitra, Pradipta De, and Sandip Chakraborty. Crawdad dataset iitkgp/apptraffic (v. 2015-11-26). http://crawdad.org/iitkgp/apptraffic/20151126/, November 2015.

[132] Khalid Shahbar and A. Nur Zincir-Heywood. Traffic flow analysis of tor pluggable transports. In *Network and Service Management (CNSM)*, 2015.

[133] Chakchai So-In, Nutakarn Mongkonchai, Phet Aimtongkham, Kasidit Wijitsopon, and Kanokmon Rujirakul. An evaluation of data mining classification models for network intrusion detection. In *Digital Information and Communication Technology and it's Applications (DICTAP)*, 2014.

[134] M. Soysal and E. G. Schmidt. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation*, 67:451.467, 2010.

[135] E. Stalmans and B. Irwin. A framework for dns based detection and mitigation of malware infections on a network. In *Information Security South Africa*, 2011.

[136] Matija Stevanovic and Jens Myrup Pedersen. An analysis of network traffic classification for botnet detection. In *Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, 2015.

[137] Elizabeth Stinson and John C. Mitchell. Characterizing bots remote control behavior. In *4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2007.

[138] B. Stone-gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, Ch. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. 16th ACM conference on Computer and communications security, 2009.

[139] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. *Advances in Information Security*, 36:1–24, 2008.

[140] Quan Sun. Sampling-based prediction of algorithm runtime. *The University of Waikato*, 2009.

[141] Miroslaw Szymczyk. Detecting botnets in computer networks using multi-agent technology. In *Fourth International Conference on Dependability of Computer Systems*, 2009.

[142] Twitter API. http://blog.unmaskparasites.com/2009/12/09/twitter-api-still-attracts-hackers/.

[143] S. T. Vuong and M. S. Alam. Advanced methods for botnet intrusion detection systems. In *Intrusion Detection Systems*, 2011.

[144] Jing Wang and Ioannis Ch. Paschalidis. botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, PP, 2016.

[145] K. Wang, Ch. Huang, Sh. Lin, and Y. Lin. A fuzzy pattern-based filtering algorithm for botnet detection. *Computer Networks*, 55:3275–3286, 2011.

[146] Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[147] P. Wurzinger, L. Bilge, Th. Holz, J. Goebel, Ch. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *14th European conference on research in computer security (ESORICS)*, 2009.

[148] Kui Xu, Danfeng Yao, Qiang Ma, and Alexander Crowell. Detecting infection onset with behavior-based policies. In *5th International Conference on Network and System Security (NSS)*, 2011.

[149] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *IEEE/ACM Transaction on Networking*, 20:1663–1677, 2012.

[150] Qiben Yan, Yao Zheng, Tingting Jiang, Wenjing Lou, and Y. Thomas Hou. Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis. In *Computer Communications (INFOCOM)*, 2015.

[151] H. R. Zeidanloo, A. Bt Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani. Botnet detection based on traffic monitoring. In *Networking and Information Technology (ICNIT)*, 2010.

[152] Han Zhang and Christos Papadopoulos. Bottalker: Generating encrypted, customizable c&c traces. In *Symposium on Technologies for Homeland Security (HST)*, 2015.

[153] J. Zhang, Ch. Chen, Y. Xiang, W. Zhou, and A. Vasilakos. An effective network classification method with unknown flow detection. *IEEE Transactions on Network and Service Management*, 10, 2013.

[154] J. Zhang, R. Perdisci, U. Sarfaraz W. Lee, and Z. Luo. Detecting stealthy p2p botnets using statistical traffic fingerprints. In *Dependable Systems and Networks (DSN)*, 2011.

[155] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu. Peer-to-peer botnet detection based on flow intervals. In *IFIP international information security and privacy*, 2012.

[156] D. Zhao, I. Traore, B. Sayed, W. Lu, andA. Ghorbani S. Saad, and D. Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers and Security*, 39, 2013.

Appendix A

Domain name samples

A.1 Malicious samples

1. iemvirtual.com.ar (Citadel)
2. secretgate.igg.biz (Citadel)
3. pcjwa.com (Conficker)
4. itmsl.co.uk (Conficker)
5. blindry.com (Torpig)
6. aztecinternational.com.au (Torpig)
7. liceobilinguejuvenil.edu.co (Zeus)
8. www.andra.com.br (Zeus)

A.2 Legitimate samples

1. google.com.br
2. wordpress.com
3. livejournal.com
4. search-results.com
5. guardian.co.uk
6. cj.com
7. commentcamarche.net
8. adjuggler.com

Appendix B

Softflowd detailed feature representation analysis

Although in Section 6.1 the effect of feature representation was investigated for the Tranalyzer flow exporter (as the choice of exporter in this thesis), a brief evaluation was done on Softflowd as well because it represents the basic netflow features which are more commonly used in the literature.

Two feature sets, namely Softflowd set.1 and Softflowd set.2, are extracted and analyzed in this section. To prepare the data sets, Softflowd is used to provide 14 flow attributes with the default parameters. All of the numeric attributes that can be employed directly by the ML classifiers are used to form the Softflowd set.1 (except the 5-tuple). The binary representation of the 'TCP-flag' is used to introduce Softflowd set.2. Table B.1 summarizes the attributes that are utilized. A detailed definition of the attributes can be found on the Softflowd project web site [21]. Since the traffic generated/collected for each of the data sets is different, after extracting the flows, the data sets were then divided into two parts (Training and Testing) based on: (i) a $\approx 30(70)\%$ breakdown for the testing (training) respectively; and (ii) keeping enough samples of each class in both of the data sets. In this case, four sets of botnet data on conficker, Zeus and Torpig botnets were employed. Table B.2 indicates the number of flow samples employed in this analysis for each data set.

Table B.1: The Softflowd features employed.

Softflowd set.1 & 2	**Softflowd set.2 only**
Duration	Flag-A
Total number of packets (Pkts)	Flag-P
Total number of bytes (Byts)	Flag-R
Flows	Flag-S
Type of Service (TOS)	Flag-F
Bits per second (bps)	Flag-U
Packets per second (pps)	
Bytes per packet (Bpp)	

Table B.2: The number of flows in each data set employed.

Data Set	**Training**		**Testing**	
	Legit	**Botnet**	**Legit**	**Botnet**
Zeus-D (NIMS)	6099	6099	2614	2614
Zeus-T1 (NIMS)	611	611	262	262
Zeus (NETRESEC)	252	252	108	108
Zeus (Snort)	100	100	43	43
Conficker-D (NIMS)	28951	28921	12386	12416
Torpig-D (NIMS)	1864	1856	794	800

In this section, C4.5 and the SBB machine learning algorithm were selected as the classifiers because: (i) both of them generate solutions (models) that are in human readable format, enabling the analysis of the learned models, and (ii) both approaches share an ability to perform attribute selection which will be used later to gain insight on how botnets are communicating using HTTP.

Tables B.3 and B.4 present the classification results of C4.5 and SBB employing the two different feature sets. The first feature set (Softflowd set.1) consists of the default numerical flow features exported by Softflowd whereas the second feature set (Softflowd set.2) augments the default numerical flow features with the numerically encoded TCP-Flag attributes. As shown in Table B.3, some of the FPRs are high when using Softflowd set.1 with C4.5 and SBB. Having said this, both of the classifiers performed equally well on Zeus-T1 (NIMS), Zeus (NETRESEC), Zeus (Snort) and Conficker (NIMS) while using Softflowd set.1. However, the performance results on Torpig-D (NIMS) and Zeus-D (NIMS) are much lower than the others when Softflowd set.1 is employed as the feature set. This observation indicates that the Torpig and Zeus-D (NIMS) botnet characteristics cannot be well represented by the features of Softflowd set.1.

The Softflowd set.2 feature set was tested to investigate if 'TCP-flag' would be beneficial for improving classification performance for Torpig-D (NIMS) and Zeus-D (NIMS) specifically. Table B.4 shows the results of these additional experiments which indicate that the performance of almost all of the botnets [except for Zeus (NETRESEC)] increased by at least 1% indicating that providing the TCP flags as features to botnet classifiers can be beneficial. Surprisingly, the Torpig (NIMS)

Table B.3: Classification Results Using Softflowd set.1 Feature Set

	Data Set	Score	Legitimate		Botnet		Complexity		
			TPR	FPR	TPR	FPR	Time (sec)	Solution	Feature
C4.5	Zeus-D (NIMS)	84%	86%	17%	83%	14%	0.26	485	8
	Zeus-T1 (NIMS)	97%	96%	1%	99%	4%	0.01	29	5
	Zeus (NETRESEC)	97%	97%	3%	97%	3%	0.04	43	8
	Zeus (Snort)	93%	98%	12%	88%	2%	0	13	4
	Conficker-D (NIMS)	92%	91%	7%	93%	9%	2.71	411	6
	Torpig-D (NIMS)	68%	91%	55%	45%	9%	0.07	49	6
SBB	Zeus-D (NIMS)	77%	80%	27%	73%	20%	185.56	27	5
	Zeus-T1 (NIMS)	97%	96%	1%	99%	4%	176.98	42	6
	Zeus (NETRESEC)	90%	93%	13%	87%	7%	29.57	6	3
	Zeus (Snort)	98%	98%	2%	98%	2%	6.39	53	5
	Conficker-D (NIMS)	90%	89%	9%	91%	11%	178.10	81	7
	Torpig-D (NIMS)	65%	92%	63%	37%	8%	186.12	20	4

Table B.4: Classification Results Using Softflowd set.2 Feature Set

	Data Set	Score	Legitimate		Botnet		Complexity		
			TPR	FPR	TPR	FPR	Time (sec)	Solution	Feature
C4.5	Zeus-D (NIMS)	87%	90%	16%	84%	10%	0.24	457	9
	Zeus-T1 (NIMS)	97%	97%	3%	97%	3%	0.01	35	9
	Zeus (NETRESEC)	96%	97%	6%	94%	3%	0.01	29	8
	Zeus (Snort)	98%	97%	1%	99%	3%	0	11	5
	Conficker-D (NIMS)	94%	93%	5%	95%	7%	3.41	365	10
	Torpig-D (NIMS)	99%	99%	1%	99%	1%	0.04	17	5
SBB	Zeus-D (NIMS)	78%	73%	18%	82%	27%	188.252	51	8
	Zeus-T1 (NIMS)	97%	94%	0%	100%	6%	161.87	14	6
	Zeus (NETRESEC)	90%	87%	7%	93%	13%	36.80	48	8
	Zeus (Snort)	100%	100%	0%	100%	0	8.22	41	8
	Conficker-D (NIMS)	91%	90%	9%	91%	10%	192.44	41	9
	Torpig-D (NIMS)	100%	100%	0%	100%	0%	109.23	60	11

results were improved by more than 30% when traffic was represented using the Softflowd set.2 feature set, implying that the six flag features of Softflowd set.2 were particularly effective at characterizing the Torpig botnet. Moreover, there appears to be no disadvantage in using the Softflowd set.2 attributes.

Comparing the results on complexity criteria (Table B.4) there is not much difference between the solutions based on feature complexity (*i.e.* the number of features used from the given set). However, there are significant differences in terms of time and solution complexities. In this regard, the C4.5 training time is much less than the SBB training time. By contrast, SBB obtained smaller solutions (*e.g.* 88% smaller for the Conficker data set) based on the solution complexity. This enables SBB to implement the solutions more efficiently. Given that such solutions need to operate at network flow speeds, simpler solutions are more advantageous because the detection system can perform faster with fewer rule/signatures. Although, in some cases low complexity is caused by an under-performing solution, in others it is an indicator of a good solution with low complexity.

To understand to what degree introducing the 'flag'-based features has effected the solutions presented by the classifiers, they have been analyzed in detail. This type of analysis should provide some insights on the Zeus, Conficker and Torpig botnet behaviours.

B.1 C4.5 solution analysis

The C4.5 solution (after training using Softflowd set.2) for the Conficker botnet is a very complex tree (365 rules). By contrast, the C4.5 solution (after training using Softflowd set.2) for the Torpig botnet resulted in a very small tree with a very high performance. As 65% of the nodes in the decision tree of the Torpig botnet utilized the flag-based features (which were not included in Softflowd set.1), it appears that Torpig employs these flags to tag its packets in a non-routine way.

Additionally, the C4.5 solutions were analyzed for the various Zeus botnet data sets employed in this work. Due to the high complexity of Zeus-1 (NIMS), no distinct rule could be observed other than the very limited usage of flag-based features versus the highly used features related to the number of bytes and packets (such as 'Pkts' and 'bps'). By contrast, analysis of the other three Zeus botnet data sets shows that 'Pkts' (the total number of packets in a flow), 'Byts' ('the total number of bytes in a flow), 'Flag-S' (indicating the status of the TCP SYN flag in the communication) and 'Flag-F' (the status of the TCP FIN flag in the communication) are utilized widely. To this end, in Zeus (NETRESEC), 15% of the botnet training samples were labelled using 'Flag-R' (when the TCP reset flag is set in the communication) or in Zeus (Snort), 80% of the training data set is labeled based on 'Pkts' and 'Byts' (Fig. B.1). Although the flag-based features are used by C4.5 to build the classification models for the Zeus botnets, comparing the C4.5 results of Softflowd set.1 and set.2 shows that there are some fluctuations in the performance of the classifier from one Zeus data set to another when flag features are employed. It seems that these features do help in the identification of the Zeus botnet traffic downloaded from the Snort web site as well as the Zeus-1 traffic. However, it seems as though it does not help in the identification of the Zeus botnet traffic downloaded from the NETRESEC site nor the Zeus-2 traffic. However, in both of these cases, the decrease in DR is compensated by the improvement in the FPR. This observation indicates that not all versions of the Zeus botnet utilize the TCP flags in their communication (considering that different data sets may belong to different versions of this botnet).

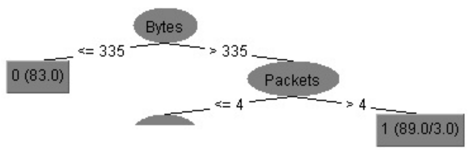


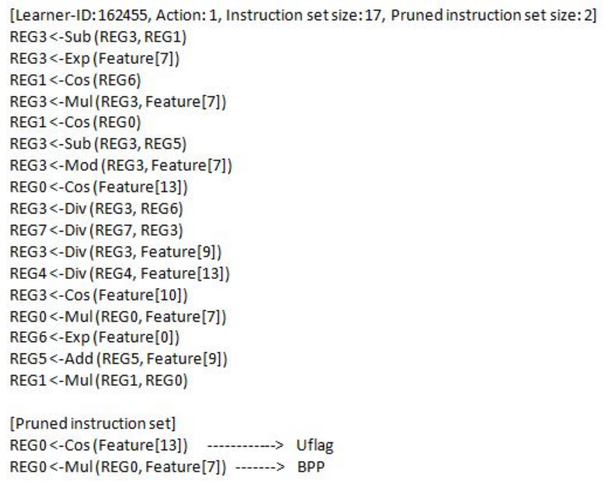
Figure B.1: Part of the Zeus (Snort) C4.5 decision tree

B.2 SBB solution Analysis

In SBB, the champion team on the training dataset is selected as the final solution, which is then applied to the test dataset for performance evaluation. Under SBB the champion classifier takes the form of a team of programs. Each program is associated only with a single class label. This provides a level of task decomposition that is not possible under C4.5.

Figure B.2 shows an example of a Torpig class-1 learner's instruction set which is part of SBB's solution for the Torpig botnet *i.e.*, a subset of the SBB solution shown in Table B.4. The program's instruction count is reduced to 2 from 7 by pruning (cf., intron removal). The pruned instruction set indicates that the learner multiplies the 'Bpp' (Bytes per packet) value by 0.54 if 'Flag-U' (indicating the urgent TCP flag) is set, returning the 'Bpp' value otherwise. Knowing that if this learner wins the bid over a data sample, it labels the sample as a botnet implies that samples with the 'Flag-U' set look more suspicious and are labelled as a botnet.

The use of flag bits in malware communication has already been suspected by other research [139]. Observation supports this hypothesis for the Torpig botnet on the data sets analyzed. SBB used the 'Pkts' and 'bps' (stands for bits per seconds) features the most for all of the botnets while for the Torpig botnet, it also utilized the 'Flag-S' and 'Flag-U' frequently. When SBB solutions using Softflowd set.1 and Softflowd set.2 are compared against each other, similar trends are observed in the behaviour of the C4.5 classifier. There are some fluctuations in the performance of the SBB classifier from one Zeus data set to another when flag features are employed. SBB's solutions indicate that the versions of the Zeus botnet seem to be different from one Zeus traffic file to another. For SBB flag features improve solution performance, specially for Zeus Snort and Torpig botnets. As well, in most cases it introduces an improvement in the false alarm rates for SBB.



```
[Learner-ID: 162455, Action: 1, Instruction set size: 17, Pruned instruction set size: 2]
REG3 <-Sub (REG3, REG1)
REG3 <-Exp (Feature[7])
REG1 <-Cos (REG6)
REG3 <-Mul (REG3, Feature[7])
REG1 <-Cos (REG0)
REG3 <-Sub (REG3, REG5)
REG3 <-Mod (REG3, Feature[7])
REG0 <-Cos (Feature[13])
REG3 <-Div (REG3, REG6)
REG7 <-Div (REG7, REG3)
REG3 <-Div (REG3, Feature[9])
REG4 <-Div (REG4, Feature[13])
REG3 <-Cos (Feature[10])
REG0 <-Mul (REG0, Feature[7])
REG6 <-Exp (Feature[0])
REG5 <-Add (REG5, Feature[9])
REG1 <-Mul (REG1, REG0)

[Pruned instruction set]
REG0 <-Cos (Feature[13])    -----------> Uflag
REG0 <-Mul (REG0, Feature[7]) -------> BPP
```

Figure B.2: SBB- a sample learner instruction set with botnet label on the Torpig–Softflowd set.2 set.

In summary, the Softflowd set.2 feature set performed better in terms of a higher Score and a lower FPR. In other words, using the 'TCP-flag' binary representation does have an effect when using basic netflow features (exported by Softflowd). This is not consistent with the results in Section 6.1. This might be caused by the wide range of features provided by Tranalyzer, very well representing the botnet traffic, which have reduced the effect of 'TCP-flag' usage and representation.

Furthermore, analysis of both the SBB and C4.5 decision tree could help us to recognize the most important features of the attribute set and also the direct/indirect relationships between these features. Table B.5 shows all of the features employed by each of the classifiers on each of the botnet data sets. As the table indicates, SBB and C4.5 use different feature sets from one botnet to another. This implies that the classifiers are learning different behaviours. There are some obvious similarities/differences between the features employed by the classifiers such as those listed below.

(i) Almost all of the classifiers used 'Pkts' and 'bps'. This shows the importance of these features in botnet detection.

(ii) C4.5 did not use the 'ToS' and 'Flows' features at all while SBB used at least one of them in all types of botnet classification.

(iii) The features employed by C4.5 and SBB for the Zeus (Snort) classification are almost complementary while SBB's selected feature set could obtain a 100% detection

Table B.5: Feature Matrix.

	Zeus-D (NIMS)		**Zeus-T1 (NIMS)**		**Zeus (NETRESEC)**		**Zeus (Snort)**		**Conficker-D (NETRESEC)**		**Torpig-D (NIMS)**	
Feature	**C4.5**	**SBB**	**C4.5**	**SBB**	**C4.5**	**SBB**	**C4.5**	**SBB**	**C4.5**	**SBB**	**C4.5**	**SBB**
Duration	✓	✓	✓	-	✓	-	-	✓	✓	-	-	✓
Packets	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	-	✓
Bytes	✓	-	✓	-	✓	✓	✓	-	✓	✓	-	✓
Flows	-	-	-	✓	-	✓	-	✓	-	-	-	✓
ToS	-	✓	-	✓	-	✓	-	-	-	✓	-	✓
bps	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
pps	✓	-	-	-	✓	✓	✓	✓	✓	✓	-	✓
Bpp	✓	✓	✓	-	✓	-	-	✓	✓	-	✓	✓
Flag-A	✓	✓	✓	-	-	-	✓	-	✓	✓	✓	✓
Flag-P	-	✓	-	✓	-	-	-	-	✓	✓	-	-
Flag-R	-	-	✓	✓	✓	-	-	✓	-	✓	-	-
Flag-S	✓	-	✓	-	✓	-	-	✓	✓	✓	✓	✓
Flag-F	✓	-	✓	-	-	✓	✓	✓	✓	-	✓	-
Flag-U	-	✓	-	-	-	✓	-	-	-	-	-	✓

rate with a zero FPR.

(iv) In the Zeus-1 (NIMS) data set where the performance is lower than expected, the features selected by the two classifiers do not overlap much. This raises the question of whether the performance would increase by finding a solution that combines these two different solutions with complementary feature sets.

Appendix C

Non-numeric Feature Representation

Below are listed the detailed results of the non-numeric feature representation analysis in Section 6.1.1.2.

Table C.1: Non-numeric feature representation– classification results with no TCP flag.

	Data Set	**Score**	**Botnet**		**Legitimate**		**Complexity**	
			TPR	**FPR**	**TNR**	**FNR**	**Time (sec)**	**Solution**
C4.5	Zeus-D (NIMS)	99.7%	99.7%	0.3%	99.7%	0.3%	39	477
	Citadel-D (NIMS)	99.55%	99.6%	0.5%	99.5%	0.4%	2.05	125
	Torpig-D (NIMS)	98.15%	98.2%	1.9%	98.1%	1.8%	0.85	119
	Conficker-D (NIMS)	100%	100%	0%	100%	0%	4570.8	739
	Zeus-T1-1 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.24	9
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	19.72	43
	Zeus-T1-W (NIMS)	99.8%	99.9%	0.3%	99.7%	0.2%	0.29	19
	Zeus-T2 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.28	41
	Zeus-T2-W (NIMS)	99.85%	99.8%	0.2%	99.8%	0.2%	0.34	21
	Citadel-T1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	1.06	31
	Citadel-T1-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.83	39
	Zeus (CVUT)	99.95%	100%	0.1%	99.9%	0%	2779.2	1185
	Kelihos (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	1149.57	849
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	612.05	1129
	NSIS (CVUT)	99.25%	99.3%	0.8%	99.2%	0.7%	5.38	275
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	215	481
	Rbot (CVUT)	99.6%	99.6%	0.4%	99.6%	0.4%	61.89	487
	ZeroAccess (CVUT)	99.95%	100%	0.1%	99.9%	0%	94.85	135
	Zeus (Snort)	100%	100%	0%	100%	0%	0.05	3
	Zeus (NETRESEC)	97.5%	98%	3.0%	97.0%	2.0%	0.15	27
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.52	25
	Cutwail (NETRESEC)	99.65%	99.8%	0.5%	99.5%	0.2%	1.08	57
	Citadel (NETRESEC)	98.7%	99.6%	2.2%	97.8%	0.4%	0.24	21
	Conficker (CAIDA)	99.95%	100%	0.1%	99.9%	0%	7454.18	1317
	ISOT (Uvic)	99.8%	99.8%	0.2%	99.8%	0.2%	82.57	525
SBB	Zeus-D (NIMS)	98.6%	97.41%	0.21%	99.79%	2.39%	205.282	36
	Citadel-D (NIMS)	99.55%	99.3%	0.2%	99.8%	0.7%	653.621	69
	Torpig-D (NIMS)	97.4%	95.6%	0.7%	99.2%	4.3%	1296.15	78
	Conficker-D (NIMS)	98.5%	98.5%	1.5%	98.5%	1.5%	1321	75
	Zeus-T1-1 (NIMS)	100%	100%	0%	100%	0%	217.272	33
	Zeus-T1-2 (NIMS)	99.99%	99.98%	0%	100%	0.02%	261.93	24
	Zeus-T1-W (NIMS)	100%	100%	0%	100%	0%	383.501	43
	Zeus-T2 (NIMS)	99.97%	99.94%	0%	100%	0.06%	279.6	36
	Zeus-T2-W (NIMS)	100%	100%	0%	100%	0%	428.962	44
	Citadel-T1 (NIMS)	100%	100%	0%	100%	0%	295.55	30
	Citadel-T1-W (NIMS)	99.9%	99.8%	0%	100%	0.2%	346.11	12
	Zeus (CVUT)	98.06%	99.89%	3.77%	96.23%	0.11%	1280.79	56
	Kelihos (CVUT)	97.74%	99.19%	3.71%	96.29%	0.81%	295.55	30
	Neris (CVUT)	93.19%	97.21%	10.84%	89.16%	2.79%	240.611	31
	NSIS (CVUT)	94.09%	92.11%	3.93%	96.07%	7.89%	235.439	58
	Virut (CVUT)	98.29%	98.25%	1.67%	98.33%	1.75%	214.754	17
	Rbot (CVUT)	97%	95.5%	1.5%	98.5%	4.5%	453	41
	ZeroAccess (CVUT)	99.39%	99.63%	0.84%	99.16%	0.37%	279.6	38
	Zeus (Snort)	100%	100%	0%	100%	0%	157.42	4
	Zeus (NETRESEC)	99.17%	99.17%	0.8%	99.17%	0.8%	221.14	28
	Kelihos (NETRESEC)	99.95%	100%	0.1%	99.9%	0%	430.918	37
	Cutwail (NETRESEC)	99.8%	99.7%	0.1%	99.9%	0.3%	479.436	43
	Citadel (NETRESEC)	99.15%	100%	1.7%	98.3%	0%	406.614	27
	Conficker (CAIDA)	99.07%	99.01%	0.88%	99.12%	0.99%	235.439	58
	ISOT (Uvic)	93.12%	97.36%	11.1%	88.89%	2.6%	214.76	17

Table C.2: Non-numeric feature representation– classification results for numerical representation.

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
	Zeus-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	26.97	399
	Citadel-D (NIMS)	99.6%	99.6%	0.4%	99.6%	0.4%	1.94	97
	Torpig-D (NIMS)	98.1%	98.3%	2.1%	97.9%	1.7%	0.84	121
	Conficker-D (NIMS)	99.85%	99.8%	0.1%	99.9%	0.2%	4752.02	751
	Zeus-T1-1 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.23	9
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	12.2	43
	Zeus-T1-W (NIMS)	99.8%	99.9%	0.3%	99.7%	0.1%	0.33	19
	Zeus-T2 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.21	41
	Zeus-T2-W (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	0.31	21
	Citadel-T1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	1.07	31
	Citadel-T1-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.82	39
	Zeus (CVUT)	99.95%	100%	0.1%	99.9%	0%	2620.01	1199
C4.5	Kelihos (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	1321.3	847
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	666.7	1119
	NSIS (CVUT)	99.25%	99.3%	0.8%	99.2%	0.7%	5.88	279
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	233.98	479
	Rbot (CVUT)	99.65%	99.6%	0.3%	99.7%	0.4%	68.34	485
	ZeroAccess (CVUT)	99.95%	100%	0.1%	99.9%	0%	97.28	135
	Zeus (Snort)	100%	100%	0%	100%	0%	0.06	5
	Zeus (NETRESEC)	97.65%	98.0%	2.7%	97.3%	2.0%	0.15	25
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.52	25
	Cutwail (NETRESEC)	99.65%	99.8%	0.5%	99.5%	0.2%	1.07	57
	Citadel (NETRESEC)	98.7%	99.6%	2.2%	97.8%	0.4%	0.25	21
	Conficker (CAIDA)	99.9%	99.9%	0.1%	99.9%	0.1%	7605.13	1354
	ISOT (Uvic)	99.8%	99.8%	0.2%	99.8%	0.2%	110.04	515
	Zeus-D (NIMS)	98.58%	97.26%	0.1%	99.9%	2.73%	372.256	47
	Citadel-D (NIMS)	99.65%	99.35%	0.15%	98.5%	0.65%	667.2	73
	Torpig-D (NIMS)	97.5%	96%	1%	99%	4%	1118.265	75
	Conficker-D (NIMS)	98.3%	98.3%	1.6%	98.4%	1.7%	1221	79
	Zeus-T1-1 (NIMS)	100%	100%	0%	100%	0%	229.486	26
	Zeus-T1-2 (NIMS)	99.99%	99.98%	0%	100%	0.2%	197.21	17
	Zeus-T1-W (NIMS)	100%	100%	0%	100%	0%	397.1	29
	Zeus-T2 (NIMS)	99.98%	100%	0%	99.97%	0%	327.256	67
	Zeus-T2-W (NIMS)	100%	100%	0%	100%	0%	452.65	39
	Citadel-T1 (NIMS)	100%	100%	0%	100%	0%	325.584	36
	Citadel-T1-W (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	346.11	15
	Zeus (CVUT)	98.66%	99.29%	1.97%	98.03%	0.71%	1047.53	23
SBB	Kelihos (CVUT)	97.74%	99.38%	3.59%	96.51%	0.62%	338	71
	Neris (CVUT)	92.5%	96.9%	10.88%	90.12%	3.1%	301.25	42
	NSIS (CVUT)	94.09%	91.9%	4%	96%	8.1%	343	29
	Virut (CVUT)	98.29%	98.12%	1.5%	98.5%	1.88%	237.12	26
	Rbot (CVUT)	97%	95.67%	2.5%	97.5%	4.33%	433	47
	ZeroAccess (CVUT)	99.27%	99.53%	1%	99%	0.37%	195.5	34
	Zeus (Snort)	100%	100%	0%	100%	0%	147.017	2
	Zeus (NETRESEC)	99.17%	98.33%	0%	100%	1.67%	378.048	74
	Kelihos (NETRESEC)	99.21%	99.52%	1.1%	98.9%	0.48%	179.89	29
	Cutwail (NETRESEC)	100%	100%	0%	100%	0%	445.1	51
	Citadel (NETRESEC)	99.4%	100%	1.2%	98.8%	0%	422.33	40
	Conficker (CAIDA)	99%	99%	1%	99%	1%	264.29	76
	ISOT (Uvic)	92.95%	96%	11.1%	89.9%	4%	218.22	70

Table C.3: Non-numeric feature representation– classification results for Nominal representation.

	Data Set	**Score**	**Botnet**		**Legitimate**		**Complexity**	
			TPR	**FPR**	**TNR**	**FNR**	**Time (sec)**	**Solution**
C4.5	Zeus-D (NIMS)	99.75%	99.7%	0.2%	99.8%	0.3%	35.33	531
	Citadel-D (NIMS)	99.6%	99.6%	0.4%	99.6%	0.4%	2.31	97
	Torpig-D (NIMS)	98.1%	98.3%	2.1%	97.9%	1.7%	0.87	121
	Conficker-D (NIMS)	99.5%	99.5%	0.5%	99.5%	0.5%	4856.54	849
	Zeus-T1-1 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.26	9
	Zeus-T1-2 (NIMS)	99.95%	99.9%	0%	100%	0.1%	16.64	21
	Zeus-T1-W (NIMS)	99.8%	99.9%	0.3%	99.7%	0.1%	0.32	19
	Zeus-T2 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.49	41
	Zeus-T2-W (NIMS)	99.85%	99.8%	0.2%	99.8%	0.2%	0.33	21
	Citadel-T1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	1.17	31
	Citadel-T1-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.88	39
	Zeus (CVUT)	100%	100%	0%	100%	0%	2876.16	1401
	Kelihos (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	1296.13	847
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	515.11	1119
	NSIS (CVUT)	99.25%	99.3%	0.8%	99.2%	0.7%	5.49	279
	Virut (CVUT)	99.85%	99.9%	0.1%	99.8%	0.1%	243.85	479
	Rbot (CVUT)	99.65%	99.6%	0.3%	99.7%	0.4%	65.25	485
	ZeroAccess (CVUT)	99.95%	100%	0.1%	99.9%	0%	119.09	135
	Zeus (Snort)	100%	100%	0%	100%	0%	0.06	3
	Zeus (NETRESEC)	97.4%	98%	3.2%	96.8%	2.0%	0.15	27
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.55	25
	Cutwail (NETRESEC)	99.6%	99.8%	0.5%	99.5%	0.2%	1.15	57
	Citadel (NETRESEC)	98.7%	99.6%	2.2%	97.8%	0.4%	0.24	21
	Conficker (CAIDA)	99%	99%	1%	99%	1%	8011	1319
	ISOT (Uvic)	99.8%	99.8%	0.2%	99.8%	0.2%	105.19	515
SBB	Zeus-D (NIMS)	98.75%	97.57%	0.06%	99.94%	2.4%	206.171	72
	Citadel-D (NIMS)	99.2%	99.3%	0.9%	99.1%	0.7%	645.2	85
	Torpig-D (NIMS)	97%	97%	3%	97%	3%	1005	73
	Conficker-D (NIMS)	98.6%	98.7%	1.5%	98.5%	1.3%	1400.1	90
	Zeus-T1-1 (NIMS)	100%	100%	0%	100%	0%	220.4	13
	Zeus-T1-2 (NIMS)	99.99%	99.98%	0%	100%	0.02%	365.879	75
	Zeus-T1-W (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	448	66
	Zeus-T2 (NIMS)	99.97%	99.98%	0%	100%	0.02%	305.67	30
	Zeus-T2-W (NIMS)	100%	100%	0%	100%	0%	428.962	44
	Citadel-T1 (NIMS)	100%	100%	0%	100%	0%	326.5	37
	Citadel-T1-W (NIMS)	99.9%	99.8%	0%	100%	0.2%	330.85	19
	Zeus (CVUT)	98.52%	9.78%	2.8%	97.25%	0.2%	1558.73	43
	Kelihos (CVUT)	98%	99%	3%	97%	1%	366	69
	Neris (CVUT)	93.05%	98%	11.9%	88.1%	2%	300	32
	NSIS (CVUT)	94.09%	92.11%	3.93%	96.07%	7.89%	370.917	32
	Virut (CVUT)	98.28%	98.2%	1.65%	98.35%	1.8%	210.33	27
	Rbot (CVUT)	97%	95.5%	1.5%	98.5%	4.5%	511	60
	ZeroAccess (CVUT)	99%	99%	1%	99%	1%	191.835	25
	Zeus (Snort)	100%	100%	0%	100%	0%	195.313	12
	Zeus (NETRESEC)	98.75%	98.33%	0.8%	99.12%	1.7%	331.286	44
	Kelihos (NETRESEC)	99.95%	100%	0.1%	99.9%	0%	519.62	59
	Cutwail (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	617.39	72
	Citadel (NETRESEC)	99.1%	99.3%	1.1%	98.9%	0.7%	595.6	33
	Conficker (CAIDA)	99.05%	99%	0.9%	99.1%	1%	399.3	56
	ISOT (Uvic)	93%	96%	10%	90%	4%	573.66	85

Table C.4: Non-numeric feature representation– classification results for binary representation.

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	**FPR**	**TNR**	**FNR**	**Time (sec)**	**Solution**
	Zeus-D (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	29.78	411
	Citadel-D (NIMS)	99.57%	99.6%	0.5%	99.5%	0.4%	2.3	97
	Torpig-D (NIMS)	98.2%	98.4%	2%	98%	1.6%	0.86	119
	Conficker-D (NIMS)	100%	100%	0%	100%	0%	5167.55	837
	Zeus-T1-1 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.25	9
	Zeus-T1-2 (NIMS)	100%	100%	0%	100%	0%	15.92	45
	Zeus-T1-W (NIMS)	99.8%	99.9%	0.3%	99.7%	0.1%	0.3	19
	Zeus-T2 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.28	41
	Zeus-T2-W (NIMS)	99.85%	99.8%	0.2%	99.8%	0.2%	0.33	21
	Citadel-T1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	1.15	31
	Citadel-T1-W (NIMS)	99.85%	99.9%	0.2%	99.8%	0.1%	0.83	39
	Zeus (CVUT)	100%	100%	0%	100%	0%	3562.22	1239
C4.5	Kelihos (CVUT)	99.95%	99.9%	0.1%	99.9%	0.1%	14.56	845
	Neris (CVUT)	99.8%	99.8%	0.2%	99.8%	0.2%	587.1	1089
	NSIS (CVUT)	99.3%	99.3%	0.7%	99.3%	0.3%	5.64	275
	Virut (CVUT)	99.9%	99.9%	0.1%	99.9%	0.1%	221.73	481
	Rbot (CVUT)	99.65%	99.6%	0.4%	99.6%	0.4%	48.55	499
	ZeroAccess (CVUT)	99.95%	100%	0.1%	99.9%	0%	102.01	135
	Zeus (Snort)	100%	100%	0%	100%	0%	0.06	3
	Zeus (NETRESEC)	97.26%	97.5%	3.0%	97.0%	2.5%	0.11	5
	Kelihos (NETRESEC)	99.8%	99.8%	0.2%	99.8%	0.2%	0.57	25
	Cutwail (NETRESEC)	99.6%	99.8%	0.5%	99.5%	0.2%	1.08	57
	Citadel (NETRESEC)	98.66%	99.6%	2.3%	97.7%	0.4%	0.25	21
	Conficker (CAIDA)	99.9%	99.9%	0.1%	99.9%	0.1%	7756.66	1498
	ISOT (Uvic)	99.85%	99.8%	0.1%	99.9%	0.2%	94.04	521
	Zeus-D (NIMS)	98.55%	97.42%	0.32%	99.68%	2.6%	303.291	39
	Citadel-D (NIMS)	99.5%	99.5%	0.5%	99.5%	0.5%	721.5	109
	Torpig-D (NIMS)	98.5%	98.5%	2.5%	98.5%	2.5%	1109	185
	Conficker-D (NIMS)	98.5%	98.5%	1.5%	98.5%	1.5%	1400.1	128
	Zeus-T1-1 (NIMS)	100%	100%	0%	100%	0%	198.039	25
	Zeus-T1-2 (NIMS)	99.99%	99.98%	0%	100%	0.02%	286.251	41
	Zeus-T1-W (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	448	69
	Zeus-T2 (NIMS)	99.97%	99.94%	0%	100%	0.06%	331.573	14
	Zeus-T2-W (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	598.78	81
	Citadel-T1 (NIMS)	99.9%	99.9%	0.1%	99.9%	0.1%	465.38	66
	Citadel-T1-W (NIMS)	99.9%	99.8%	0%	100%	0.2%	485.22	31
	Zeus (CVUT)	98.46%	99.90%	2.99%	97.01%	0.01%	1231.87	60
SBB	Kelihos (CVUT)	98.1%	99.1%	2.9%	97.1%	0.9%	359	73
	Neris (CVUT)	93%	98%	12%	88%	2%	372	38
	NSIS (CVUT)	94.17%	92.1%	3.93%	96.24%	7.89%	335	41
	Virut (CVUT)	98.4%	98.2%	1.4%	98.6%	1.8%	256.2	45
	Rbot (CVUT)	97.5%	96.7%	1.7%	98.3%	3.3%	496.82	90
	ZeroAccess (CVUT)	98.95%	98.9%	1%	99%	1.1%	248.368	74
	Zeus (Snort)	100%	100%	0%	100%	0%	130.715	6
	Zeus (NETRESEC)	99.17%	98.33%	0%	100%	1.67%	347.745	38
	Kelihos (NETRESEC)	100%	100%	0%	100%	0%	553.2	81
	Cutwail (NETRESEC)	99.79%	99.8%	0.22%	99.78%	0.2%	663	87
	Citadel (NETRESEC)	99%	99%	1%	99%	1%	568.898	62
	Conficker (CAIDA)	99.1%	99.1%	0.9%	99.1%	0.9%	455.71	110
	ISOT (Uvic)	94%	96%	9%	91%	4%	722.95	132