Benchmarking the Effect of Flow Exporters and Protocol Filters on Botnet Traffic Classification

Fariba Haddadi, *Student Member, IEEE*, and A. Nur Zincir-Heywood, *Member, IEEE*

***Abstract*—Botnets represent one of the most aggressive threats against cyber security. Different techniques using different feature sets have been proposed for botnet traffic analysis and classification. However, no work has been performed to study the effect of such differences. In this paper, we perform a study on the effect of (if any) the feature sets of network traffic flow exporters. To this end, we explore five different traffic flow exporters (each with a different set of flow features) using two different protocol filters [Hypertext Transfer Protocol (HTTP) and Domain Name System (DNS)] and five different classifiers. We evaluate all these on eight different botnet traffic data sets. Our results indicate that the use of a flow exporter and a protocol filter indeed has an effect on the performance of botnet traffic classification. Experimental results show that the best performance is achieved using Tranalyzer flow exporter and HTTP filter with the C4.5 classifier.**

***Index Terms*—Botnet, flow exporters, protocol filters, traffic classification, traffic flow analysis.**

I. Introduction

A BOTNET is a network of compromised hosts that are remotely controlled by a master (aka botmaster). Different types of botnets have been created to perform various malicious tasks such as spreading spam, conducting distributed denial-of-service (DDoS) attacks, identity theft, or simply taking advantage of victims' computational resources [1]. Hence, with the high reported infection rate, the vast range of illegal activities, and powerful comebacks, botnets are one of the main threats against cyber security.

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 command and control (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 [2]. 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 botnets communication information. Since 2003, not only botnets have started to use more ubiquitous protocols such as HTTP and DNS and a decentralized topology such as peer-to-peer (P2P) but they also have started to employ techniques such as fluxing and encryption to avoid detection. Therefore, identifying the botnets and detecting them have become very challenging.

Many proposed detection approaches rely on network traffic analysis classification to detect the botnets. Some of the works in this category focus on specific types of botnets, whereas others attempt to build a general model for more than one botnet. In early 2000, most of the proposed systems were specifically focused on botnets utilizing IRC (e.g., [3]), whereas recent research has been more focused on P2P- and HTTP-based botnets (e.g., [4]–[6]). Given that botnets use automatic update mechanisms, botnet monitoring and detection approaches need to be active and continuous as well. This could potentially enable them to learn new patterns and adapt to the changes in the botnet evolution. Hence, machine learning techniques (i.e., classification and clustering) are one of the highly employed techniques in this field. The clustering and classification techniques that are 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 the meaningful 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: 1) packet header, which includes the control information of the protocols used on the network, and 2) packet payload, which includes the application information used on the network. Hence, some detection and analysis systems only use network packet headers (e.g., [5], [6], and [8]) as the basis for their features, whereas others take advantage of packet payloads (e.g., [9] and [10]). Among the group using packet headers, flow-based feature extraction methods are highly employed in the recent literature [6], [8], [11], [12]. 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 paper, our aim is to investigate the magnitude of the effect of flow exporters in botnet traffic detection and analysis systems. Thus, five open-source flow exporters, namely, Maji, Yet Another Flow Sensor (YAF), Softflowd, Tranalyzer, and Network Measurement and

Manuscript received February 15, 2014; revised June 20, 2014 and August 19, 2014; accepted October 5, 2014. This work was supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 4R2, Canada (e-mail: haddadi@cs.dal.ca; zincir@cs.dal.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2014.2364743

Accounting System (Netmate), along with five highly employed machine learning techniques, namely, C4.5, support vector machine (SVM), Naive Bayes, Bayesian networks, and artificial neural networks (ANNs), are utilized. Moreover, to study the effect of protocol filters, we conduct three sets of experiments; 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, i.e., employing an HTTP protocol filter, and the third set uses only the DNS traffic flows utilizing a DNS protocol filter. Last but not the least, we have evaluated all of these systems and their combinations on eight different botnet data sets.

The remainder of this paper is organized as follows. The botnet traffic analysis and classification systems reported in the literature are summarized in Section II. All the tools, the data sets, and the methodology employed are presented in Section III. Evaluation and results are discussed in Section IV. Finally, conclusions are drawn and future work is discussed in Section V.

II. Background and Related Work

A bot program is a self-propagating malware that infects vulnerable hosts known as bots and is designed to perform a malicious task after being triggered. The infected bots network is referred as a botnet, which is under the remote control of a master called the botmaster. Usually, bots receive commands from the master through C&C channel and carry out malicious tasks such as DDoS, spamming, phishing, and identity theft attacks [1].

Today, a typical advanced bot uses five stages to create and maintain a botnet [1]. The first stage is the initial infection stage. In this stage, attacker infects the victim using several exploitation techniques to find its existing vulnerabilities. In the second stage, i.e., secondary injection, the shell code is executed on the infected victim to fetch the image of the bot binary. The bot binary then installs itself on the victim. At this time, the infected machine is completely converted into a bot. The next stage is the connection stage. In this stage, the bot binary establishes the C&C channel to be used by the bot master. Once the connection is established, then the malicious C&C stage, i.e., the fourth stage, starts. This is when the master sends the commands to the botnet, short for bot network. Finally, when the master needs to update the bots for one reason or another, the update and the maintenance stage start.

A. Related Work

Over time, botnets have employed different protocols, topologies, and techniques to implement the aforementioned five stages while avoiding detection. Hence, an arms race has started between the botnets and the detection systems.

Gu *et al.* developed a system called BotMiner based on the group behavior analysis to detect botnets [13]. BotMiner uses a clustering approach to find similar C&C communication behaviors, which form clusters, and then employs Snort to find the type of activity in the detected clusters. They evaluated BotMiner on traffic data sets. These data sets included their campus network traffic, which represented the normal behavior. Moreover, it included the Honeynet traffic and the traffic captured by running bot binaries in a sandbox environment. These represented the attack behavior. Then, they converted their captured traffic files into flows using a tool they developed. This flow exporter included features such as the number of packets per flow, the average number of bytes per packet, and the average number of bytes per second. Their results showed that BotMiner could detect botnets with detection rates (DRs) 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) [3]. First, they used a classification technique to filter the chat type of traffic and then a clustering technique to find the group activities in the filtered traffic. Finally, a topology analyzer was applied to 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 were gathered from a controlled testbed running bot binaries. They employed and evaluated Naive Bayes, C4.5, and Bayesian networks as the classifiers against a multidimensional 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 [9]. To identify traffic responses, they located the corresponding commands in the preceding traffic. Then, using these command and response pairs, the detection model was built focusing 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 behavior.

Zeidanloo *et al.* proposed a detection system with a focus on P2P- and IRC-based botnets [10]. Using filtering, classification, and clustering techniques, they aimed to detect the group behavior of botnets in a given traffic file (trace). To analyze the traffic, a flow-based approach was utilized, whereas payload inspection was employed for traffic filtering.

Perdisci *et al.* proposed a network-level malware clustering system focusing on HTTP-based malwares [7]. The similarity metrics among HTTP traffic traces are defined and used to develop the malware clustering system where the clusters resulted in the signatures. Specifically, to decrease the computational cost and obtain high-quality clusters, multilevel clustering was employed. The malware samples employed for system evaluation were collected in six months using different malware sources such as MWCollect.

Celik *et al.* proposed a flow-based botnet C&C activity detection system using only headers of traffic packets [8]. Specifically, they investigated the effect of calibration of time-based flow features. They employed machine learning algorithms such as C4.5, Naive Bayes, and logistic regression. For the evaluation, they employed Lawrence Berkeley National Laboratory (LBNL) traces representing the normal traffic and the IRC-based simulated botnet traffic representing the attack traffic.

Francois *et al.* proposed a NetFlow monitoring framework that leveraged a simple host dependence model to track

communication patterns and employed linkage analysis and clustering techniques to identify similar botnet traffic patterns [11]. They obtained the traffic data from an Internet operator company. To analyze the traffic, a flow-based approach was utilized. In addition to the traffic flows, Google web search engine was employed for linkage analysis.

Wang *et al.* proposed a fuzzy pattern recognition approach (called BBDP) to detect HTTP and IRC botnets behavioral patterns [12]. It is known that botnets query several domain names in a given period of time to identify their C&C server and then form a Transmission Control Protocol (TCP) connection with the C&C server. Hence, Wang *et al.* analyzed the features of DNS queries (such as the number of failed DNS responses) and TCP flows to detect botnet malicious domain names and IP addresses. To accelerate the detection process and be able to detect botnets in real time, traffic reduction and parallel processing were utilized. To evaluate the proposed approach, malicious bot binaries were collected using Honeytrap and were run in a controlled environment to generate bot traffic. Their results showed up to 95% DR for their system.

Zhao *et al.* investigated a botnet detection system based on flow intervals [5]. Flow features of traffic packets were utilized with Bayesian networks and decision tree classifiers to detect botnets. They focused on P2P botnets (such as Waledac) that employ the HTTP protocol and a fast-flux-based DNS technique. To evaluate their system, they employed a combination of normal and attack traffic, some of which was generated in the laboratory, some was from Honeynet project traces, and some was from LBNL (normal traffic) data sets. Their results showed DRs over 90% with false positive rates (FPRs) under 5%.

Kirubavathi *et al.* specifically designed an HTTP-based botnet detection system using a multilayer feedforward neural network [4]. 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, they extracted features related to TCP connections in specific time intervals based on the packet headers. To collect data to evaluate their system, botnets were simulated in the laboratory.

Haddadi *et al.* designed a botnet detection approach based on botnet traffic analysis [6]. Network traces representing normal and attack traffic were generated by initializing HTTP and DNS communication with publicly available domain names of botnet C&C servers and legitimate web servers. NetFlow-based feature extraction (only from packet headers) was used, and machine learning algorithms (C4.5 and Naive Bayes) were then employed to detect the botnets. Their results achieved up to 97% DR with 3% FPR.

In short, most of the recent detection systems reported in the literature focuses on P2P and HTTP protocols [4]–[6], [12]. Given that botnets employ DNS protocol to locate the C&C servers, DNS traffic is also analyzed regardless of the utilized C&C protocol or topology [6], [12]. They employ different data mining or machine learning techniques such as decision trees, neural networks, or statistical methods for which mostly flow features were utilized. Most of the time, they integrate some normal traffic file (in the laboratory) with the generated attack traffic file to be able to evaluate the performance of the proposed detection systems. Last but not the least, most of the works in the literature propose a specific set of flow features for the detection purposes [5], [8]–[10]. However, to the best of our knowledge, there is no previous work comparing different flow features and analyzing which ones are selected more by different classifiers. Hence, in this paper, we aim to use all the features exported by different flow exporters employed to analyze if different flow exporter features have any effect on the performance of the botnet classifiers employed.

III. Methodology

As discussed in Section II, almost all of the botnet traffic analysis related works reported in the literature employ some form of network flow information, which is based on packet headers. Moreover, most of them focus on certain type of botnets for detection purposes. This means that they focus on certain type of protocols such as HTTP or DNS. This, in return, indicates the utilization of protocol filtering in the analyzed traffic data. Therefore, in this paper, we aim to study the effect (if any) of the flow exporters and the protocol filtering for separating botnet behavior from normal behavior in a given traffic log file. To be able to generalize our research to encrypted traffic as well, we do not have any packet payload (application level) based information in our study. Thus, we explore the possibility of detecting botnets by only using the features extracted from a flow.

To achieve our aim, we benchmark the effect of five different well-known flow exporters using two widely used protocols filters and five popular learning classifiers. We evaluated the performances of different combinations of these tools using eight different botnet traffic log files.

A. Traffic Employed

In this paper, we utilize traffic files from botnets that employ HTTP protocol as their communication protocol or HTTP-based P2P topologies that look like normal HTTP traffic. There are several botnet traffic captures available at NETRESEC [14] and Snort [15] web sites that we employed in this paper. These traffic captures, which represent the real-world botnets, are related to Citadel, Cutwail (Pushdo), Kelihos, and Zeus botnets. The Snort web site provides three sample traffic log files of the Zeus botnet [15]. It also provides a report describing these samples. Unfortunately, two of these samples are very small (only 110 and 1105 number of packets) and therefore could not be used in our experiments. Hence, "Sample_1" is the only Zeus traffic file that we utilized from Snort archive in this paper. On the other hand, NETRESEC provides Citadel, Cutwail, Kelihos, and Zeus botnet traffic log files (one for each) but does not provide any information with regard to these files. Hence, we investigated the employed protocols, domain names, and the communication patterns in these files and matched them with the published characteristics of these botnets and verified that the data are correct. Therefore, we use all of these botnet traffic files in this paper.

However, since there were not many traffic captures publicly available (aforementioned ones are the only ones we are aware

of), we had to generate additional representative traffic (as most of the works in the literature did) to be able to generalize our analysis. Although, in this case, rather than running botnet binaries in sandbox environments to generate representative botnet traffic, we employed publicly available (from legitimate resources) lists of C&C domain names to generate the representative botnet traffic. Moreover, to generate the representative normal traffic, we employed the Alexa domain name list. This ensures that attack and normal traffic were generated using the most recent and publicly available domain name lists and avoids the possibility of representing old botnet behavior when old binaries are used. The generated data employed in this paper are analyzed and confirmed to be similar to the real-world botnet data provided by Snort and NETRESEC [6], [16].

In this case, the data are generated by a program we developed to establish HTTP connections with the domain names (both 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. More information on the botnets and the domain name lists that we employed is given in the following.

1) Alexa: Alexa Internet, Inc. [17] ranks the web sites based on their page views and unique site users. Then, this ranking is published as the list of the most popular web sites.

2) Zeus: Zeus is a well-known botnet that made a big came back (after a takedown in 2012) with a new variant in 2013 [18]. This botnet has been collecting banking data by using man-in-the-browser keystroke logging and form grabbing but can be configured for any type of identity theft attack. There are the ZeusTracker site [19] and the DNS-BH site [20] actively monitoring the Zeus botnet. These are the sites from where we downloaded the C&C domain name lists that were employed to generate the representative traffic for Zeus botnet. Moreover, Snort and NETRESEC have traffic files of this botnet that we also utilized in this paper.

3) Citadel: Citadel botnet is the improved version of Zeus, where the Zeus bugs are fixed, and it is adapted to the newest security platforms [21]. It is believed that Citadel stole more than $500 million and infected more than 5 million personal computers in different countries. In June 2013, Microsoft and the U.S. Federal Bureau of Investigation took down almost 90% of the Citadel botnet based on a court order in an operation. However, there are news of Citadel making a bold comeback [22]. We obtained the Citadel C&C domain name lists from the Citadel botnet section of the ZeusTracker and DNS-BH web sites. In addition, captured traffic files from NETRESEC are also employed in this paper.

4) Conficker: Speculations about Conficker's purpose range from DDoS attacks or using distributed computing resources to stealing banking credentials. In 2009 and 2010, Conficker botnet was listed in Damballa top 10 botnets of the year. As of 2013, it is reported [23] that Conficker botnet has infected many medical devices. The Conficker C&C domain name lists published by the University of Bonn [24] and DNS-BH [20] web sites are employed in this paper.

5) Cutwail: Pushdo trojan is originally used to distribute other malwares 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 (DGAs) and became more resilient to takedown attempts [25]. Some variants of Pushdo utilize HTTP on top of the Secure Sockets Layer/Transport Layer Security protocol for C&C communication [26]. Captured traffic available at NETRESEC is used in this paper.

6) Kelihos: Kelihos is mainly involved in 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 is 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 well-formed HTTP, whereas the latest version employs an HTTP-based P2P topology along with the fast-flux method [27]. NETRESEC provides a Kelihos botnet captured traffic file, which is utilized in this paper.

B. Flow Generation

Flow generation tools 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 some statistics such as the number of packets per flow. 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 [28]. The most common way to identify a traffic flow is to use a 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 as follows: source IP address, destination IP address, source port number, destination port number, and protocol. Hereafter, we will refer to this as 5-tuple information.

Companies producing and managing network equipment such as routers and switches provide different types of flow exporters to summarize the network traffic in terms of flows using the 5-tuple information. Moreover, they provide flow analysis tools to analyze the flow streams. Some examples of commercial off-the-shelf flow exporter and analysis techniques are Cisco systems NetFlow [29], Juniper systems J-Flow [30], and InMon systems S-Flow [31]. On the other hand, there are some open-source publicly available exporters such as Maji [32] by the WAND research group from The University of Waikato or YAF [33] by the CERT Network Situational Awareness (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 IP Flow Information Export (IPFIX; RFC 5101). To collect and analyze traffic flow data, three network elements should work together (see Fig. 1): 1) flow exporter, which generates the flow data; 2) flow collector, which collects (stores) the flow data from the exporter; and, finally, 3) flow analyzer, which analyzes the collected data. A variety of network devices (e.g., routers and switches) support

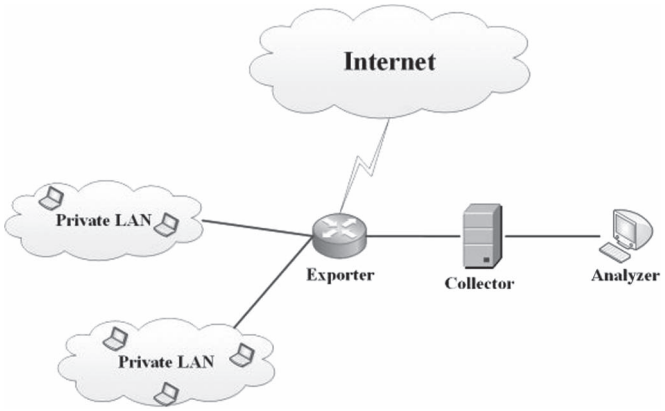

Fig. 1. Flow exporting mechanism.

different versions of flow data. Therefore, in this paper, we employ different open-source flow exporters and collectors on the aforementioned precaptured data sets. Then, we employ the five machine learning classifiers to analyze the exported and collected flows. Here, our objective is to see the effect of the flow exporter and collector tools on the performance of the analyzers.

The exporters that we employ in this paper are as follows.

1) *Maji* [32] is an open-source implementation of IPFIX supported by the WAND research group at The University of Waikato. This tool exports unidirectional flows from live Packet CAPture (PCAP) interfaces and the most common trace file formats. Maji is designed to support custom template definition and can relay the IPFIX information to TCP, User Datagram Protocol, and Stream Control Transmission Protocol collectors. Additionally, it has a simple exporting method that outputs the flow information in the standard output. Therefore, we did not use any IPFIX collector for this tool.
2) *YAF* [33] is a bidirectional flow exporter designed by the NetSA group at CERT. This tool collects and exports IPFIX-based flows. Similar to Maji, YAF can process packet data from precaptured traffic files or live captures from an interface to export flows. Although YAF package comes with a YAFAscii that outputs NetFlow packet information in an ASCII format (which is employed in this paper), any other NetFlow collector can be used for this purpose, too.
3) *Softflowd* [34] is a lightweight unidirectional flow exporter that supports different versions of NetFlow. This tool exports NetFlow data using the traffic on a simple device interface or a precaptured traffic (pcap format) file. In this paper, once we export the network packets into flows using Softflowd, we employed NfDump as the flow collector. NfDump is an open-source flow collector tool, which is fast and easy to use. NfDump supports different versions of NetFlow, too [35].
4) *Tranalyzer* [36] is a lightweight unidirectional 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). This makes it very easy to use.
5) *Netmate* [37] is a bidirectional flow exporter and analyzer. This exporter can process live captures and precaptured traffic and use different rule sets to control the exported flow format. One of our 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 [38]. Not only does this open-source program ease the installation process but also the NetAI module has been extended for additional output features. We have employed this latest version in this paper.

Table I summarizes the properties of the flow exporters employed in this paper. The fifth column indicates if the exporter requires a collector (if so, does the tool provide it in the package) or it directly exports the flow information in a human readable format. In this column of the table, "included" means a collector is included in the package, but can be replaced with another one if wanted. On the other hand, "embedded" means a collector is embedded in the exporter and no external collector can be used. The seventh column shows if the tool supports specific categories of features. These categories are defined based on the features that are frequently used in the literature [5], [13], [16], [39]. Time category refers to the features indicating start time, end time, and duration of the flow. Time-based category is important when flow aggregation or window-based analysis is employed. Interarrival category refers to the statistics calculated over a flow, such as the average interarrival time, which represents the average number of milliseconds between consecutive packets of a flow. Packets&Bytes category refers to the features computed based on the number of packets that form the flow and their size (in byte) such as the average number of bytes per packet. Flags category refers to any feature related to the packet header flags such as the TCP SYN flag.

C. Flow Analysis: Classifying the Flows

In this paper, five well-known machine learning algorithms that are widely used in the literature ([3], [5], [6], [8], [39], and [40]) are employed: C4.5, SVM, ANN, Bayesian networks, and Naive Bayes.

1) C4.5: C4.5 is a decision tree algorithm. A decision tree is a tree-structured graph (model) where internal nodes represent conditions applied to attributes (features), the leaf nodes represent the class labels, and the paths from root to leaves represent the classification rules. C4.5 algorithm is an extension to ID3 algorithm, which aims to find the small decision trees (using pruning) and then convert the trained tree into an *If-Then* rule set. Decision trees are constructed through a process of deterministically splitting the training partition based on the selection of the attribute maximizing the normalized information gain. Each branch of the tree partitions the (training) data into subsets, where the goal is to identify subsets that have the same label. Recursive application of this process incrementally constructs the decision tree until leaf nodes appear with sufficiently high normalized information gain.

2) SVM: SVM is a binary learning algorithm that can be used for classification and rule regression. The goal of this classification algorithm is to build an N-dimensional hyperplane

TABLE I
PROPERTIES OF FLOW EXPORTERS EMPLOYED

Exporter	Standard	Flow	Input	Collector	# of features	Feature categories			
						Time	Inter-arrival	Packets&Bytes	Flags
Maji	IPFIX	uni-direction	live/pre-captured	included	59	✓	✗	✓	✓
YAF	IPFIX	bi-direction	live/pre-captured	included	46	✓	✓	✓	✓
Softflowd	NetFlow (V.5, 7 and 9)	uni-direction	live/pre-captured	NfDump	41	✓	✗	✓	✓
Tranalyzer	-	uni-direction	live/pre-captured	embedded	93	✓	✓	✓	✓
Netmate	-	bi-direction	live/pre-captured	embedded	44	duration only	✓	✓	✓

that optimally separates the samples of data into two classes with maximal margin. The classifier can easily be extended to K-class classification by constructing k two-class (binary) classifiers. In order to use an SVM to solve a classification problem on nonlinearly separable data, a nonlinear mapping of the 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.

3) ANNs: ANN algorithm is inspired by the structure of the biological neural network. It is composed of neurons and the connections between them. Back-propagation network is the most frequently used version of this algorithm, 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. Each training iteration modifies the neurons' weight factors 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).

4) Naive Bayes: A Naive Bayes classifier is a simple probabilistic classifier based on the Bayes theorem. Bayes theorem assumes that the presence of an attribute in a given class is independent of other attributes. The classifier uses the maximum-likelihood (probability) method for parameter estimation. Given a training set, the classifier predicts the training set samples that belong to the class C, having the highest posteriori.

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 that 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 networks 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 approaches.

Detailed information on the five machine learning techniques that are used in this paper can be found in [41].

IV. EVALUATIONS

As discussed earlier, in this paper, our goal is to evaluate the effect (if any) of flow exporters for the representation of network traffic in botnet detection. To achieve this, we benchmark five open-source flow exporters on eight different botnet traces using five different machine learning algorithms. In this case, we have benchmarked the effect of flow exporters Maji, YAF, Softflowd, Netmate, and Tranalyzer. We have employed eight different traffic traces for the botnets Citadel, Zeus, Conficker, Kelihos, and Cutwail. Last but not the least, we have employed the machine learning algorithms C4.5, SVM, ANN, Bayesian networks, and Naive Bayes as the classifiers for the botnet traffic identification.

A. Data Sets

Data sets employed in this paper can be grouped into two different categories: 1) traffic log files captured in the wild and made publicly available and 2) traffic log files captured by the authors (these are also made publicly available [16]). In the first group, a Zeus data set was obtained from the Snort web site and another one from the NETRESEC repository. Moreover, one Citadel, one Kelihos, and one Cutwail traffic data sets were also obtained from the NETRESEC repository.

In the second group, as discussed in Section III, we have captured traffic data sets as a result of the communications with Alexa, Zeus, Citadel, and Conficker servers where the domain names of these servers were published in the publicly available lists. Given that even Alexa lists (representing normal traffic) might have malicious domain names [42], we manually extracted 500 benign domain names from Alexa lists to generate the normal traffic data sets employed in this paper.[1] As discussed in Section III-A, for Zeus botnet, we employed the domain name list provided by ZeusTracker and DNS-BH blocklists [19], [20]. The domain names from the ZeusTracker Citadel list forms the basis for Citadel botnet. These are the domain names that were still active (after Microsoft's Citadel botnet takedown operation in June 2013). Finally, Conficker domain names were obtained from the University of Bonn and DNS-BH published Conficker domain name lists [20], [24]. Collecting the domain name lists, we attempted to establish an HTTP-based communication with the domain names (they represent the potential C&C servers) and capture the traffic in the

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

TABLE II
NIMS DATA SPECIFICATIONS

data set	No. of Domain names
Alexa (NIMS)	500
Citadel (NIMS)	42
Zeus (NIMS)	684
Conficker (NIMS)	1537546

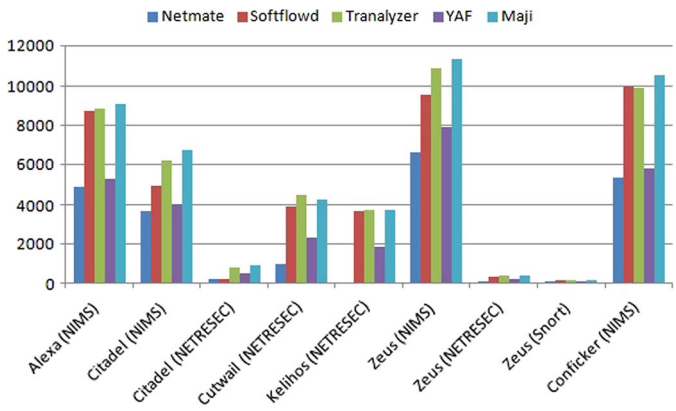

Fig. 2. Number of extracted flows.

process. Table II shows the number of domain names that we attempted to communicate. It should be noted here that we did not use any sampling while capturing the traffic. In other words, we captured all of the transmitted/received packets. Hereafter, we will refer to our generated data sets as NIMS data sets.

Once we captured the traffic data sets, the aforementioned five flow exporters are applied on the traffic captures. Fig. 2 presents the number of extracted flows by each of the tools on all of the data sets: one nonmalicious and eight malicious data sets. However, Conficker (NIMS) data (approximately 15 million packets) are actually much bigger than the other data sets; thus, we only presented 0.5% of it (i.e., about 10 000 number of flows for Softflowd) to be able to have a readable Fig. 2. As shown in Fig. 2, Tranalyzer, Maji, and Softflowd tools export almost the same number of flows for all of the data sets. They also provide the highest number of flows for a given traffic data set. This is not surprising since all three of these tools are unidirectional flow exporters. On the other hand, YAF and Netmate did not show any consistent behavior over the data sets. This might be caused by the rules defined by the tools. For example, Netmate did not export any flow for Kelihos (NETRESEC) because, in that data set, the number of out-of-order packets was higher than the acceptable threshold defined in Netmate. A comparison between the numbers of exported flows by uni- and bidirectional exporters shows that the number of exported flows by unidirectional exporters is almost two times higher than the number of flows by bidirectional exporters for all of the data sets, except for Citadel (NIMS) and Zeus (NIMS). This shows that Citadel (NIMS) and Zeus (NIMS) have more one-way communications than the other data sets, which can be caused by the number of unsuccessful connection requests or long (unclosed) connections that are split into multiple flows by the exporters according to the maximum lifetime threshold.

Table I shows the number of features supported by each of the exporters. Due to space limitations, we cannot give the full list of these attributes in this paper, but interested readers can visit the web site for each tool and see the full list of these attributes. We employed all of features provided by the exporters/collectors as inputs to our machine learning classifiers, except the IP addresses, port numbers, and any nonnumeric features. The reasons behind these are as follows: IP addresses can be anonymized, whereas port numbers can be assigned dynamically. Thus, employing such features may decrease the generalization abilities of the detection systems for unseen behaviors. Moreover, the presentation of nonnumeric features may introduce other biases to the classifiers [43]; thus, it is left to future work to introduce such features.

B. Performance Criteria for Botnet Classification

In traffic classification, two metrics are typically used in order to quantify the performance of the classifier: DR and FPR.

1) DR: In this case, DR reflects the number of the correctly classified specific botnet flows in a given traffic file. It is calculated using

$$\mathrm{DR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{1}$$

where TP (true positive) is the number of botnet flows that are classified correctly, and FN (false negative) is the number of botnet flows that are classified incorrectly (as normal flows).

2) False Positive: In this case, FPR shows the number of normal traffic flows that are classified incorrectly as the botnet flows, i.e.,

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{2}$$

where TN (true negative) is the number of normal traffic flows that are classified correctly.

3) Time Complexity: Complexity can be measured on different criteria such as memory consumption, time, or solution. In this case, we employ computation time as the time complexity, which is an important metric for machine learning classifiers to measure their computational cost during the training phase.

Naturally, a high DR, a low FPR, and a low time complexity are the desirable outcomes.

C. Results

The aforementioned classification algorithms were selected in this paper because of their high performance reported in the literature with regard to the network traffic classification, specifically in botnet detection [3], [5], [6], [8], [39], [40]. We employed these classification techniques via Weka, a well-known open-source tool in this field [44]. In general, a machine learning classifier requires a number of steps. First, a matrix of instances (in this paper, 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 interarrival time. Second, a label (ground truth) is provided for each flow, which is the class description. In this paper, the label for a flow

TABLE III
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

in the Alexa traffic file is "normal," whereas in a Zeus traffic file, it is "Zeus 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. This solution can be then verified on a test data set (unseen instances). Hence, to evaluate these classifiers on our traffic flows, first, a balanced training data set is formed by randomly selecting (uniform random selection) from the nonmalicious (Alexa) flow data set and from each of the malicious data sets. Classifiers were then run on each balanced training data set using tenfold cross validation to further avoid any data set biases that might affect the results.

After analyzing the NIMS generated traffic, we observed that successful DNS responses were received from 95% of Alexa, 75% of Zeus, 25% of Citadel, and 1% of Conficker 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 alive C&C server at the time of the communication. We have confirmed the data generated using such an approach in our previous research [6], [16]. Therefore, the combination of these resolved and unresolved domain names of the list we employed can be considered as the representation of a real botnet behavior.

Fig. 3 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 a considerable low time complexity (see Table III). It should be noted here that these time measurements are taken on a computer that is used for these evaluations, where it has 16-GB random access memory and Core i7-2600 central processing unit, 3.40 Ghz. In short, if the time criterion is an important factor in an environment, the C4.5 classifier has an advantage over ANN.

Moreover, C4.5 has the ability to perform attribute selection as an implicit property of constructing the classifier. In other words, it has the ability of choosing the most appropriate features from all the features given to it. Such ability enriches any analysis that can be done post classification. This enables the human expert (security analyst/system administrator) employing this system for a better understanding of the solution and the botnet behavior. These properties of the decision tree classifier along with the results imply that the C4.5 classifier seems to be a better choice for our purposes in terms of the performance metrics used and the data sets employed.

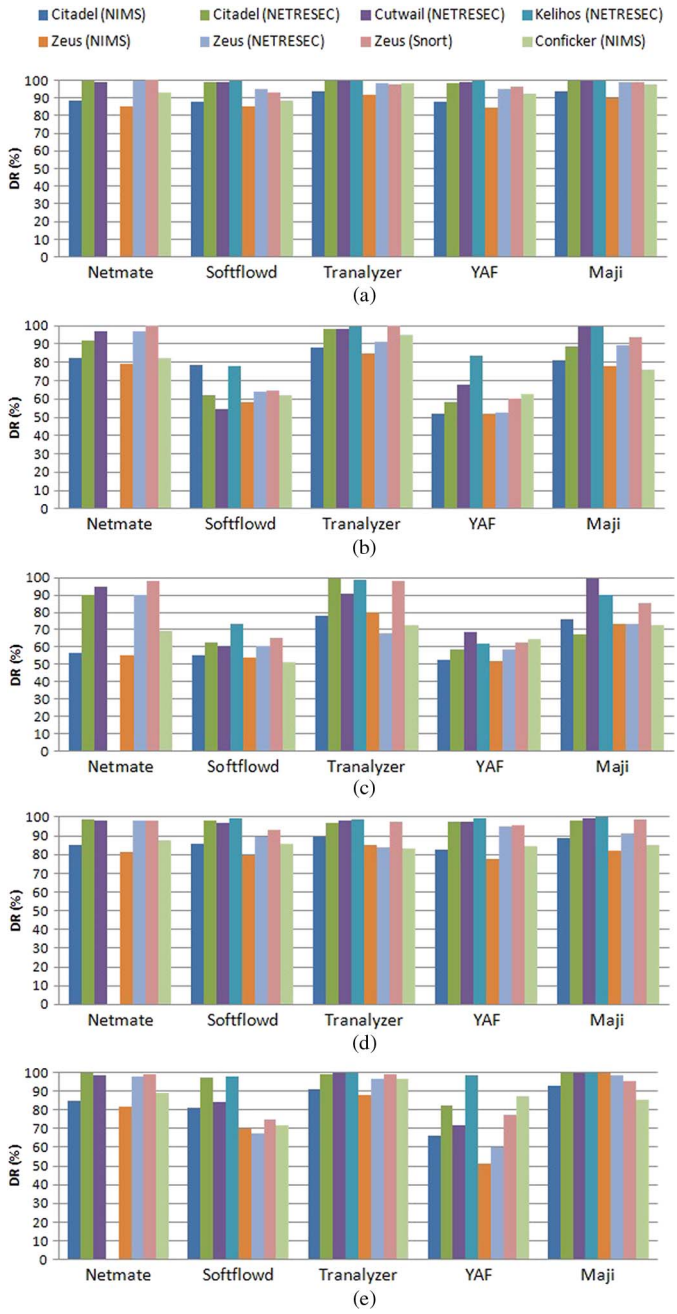

Fig. 3. DRs of the five classifiers on five flow exporters on all data sets. (a) C4.5. (b) SVM. (c) Naive Bayes. (d) Bayesian networks. (e) ANN.

While detection systems with low DR are important in understanding and identifying the behavior of interest, the effect of FPR can be very important, too. In the case of botnet classification (detection), any normal behavior that is mistakenly identified as botnet not only increases the false positive (alarm) rate but also decreases the trust to the detection system. Therefore, we analyzed the FPR of the five classifiers, as well as the effect (if any) of the flow exporters, on this performance metric. Fig. 4 shows the FPRs of C4.5 and ANN. Due to space limitations, we only present the results for the best two classifiers. However, the trend is the same for all classifiers. These two classifiers were also the best performers in terms of minimizing the FPR among all the classifiers employed in this paper on all the exporters and data sets. These results demonstrate that the flow

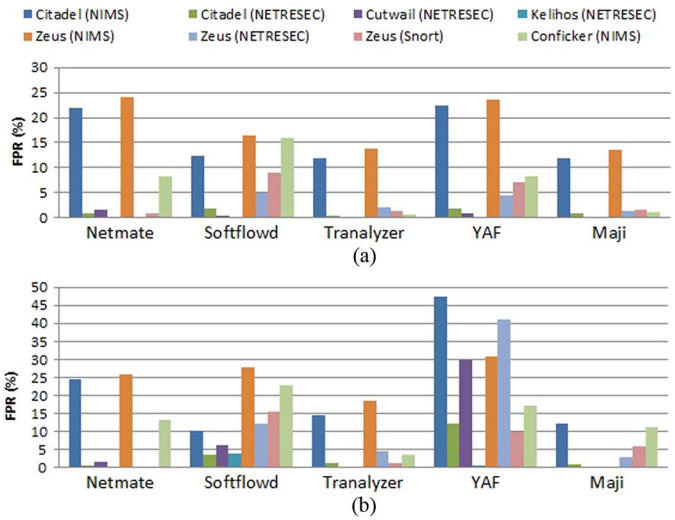

Fig. 4. FPRs of the C4.5 and ANN classifiers on the five flow exporters on all data sets. (a) C4.5. (b) ANN.

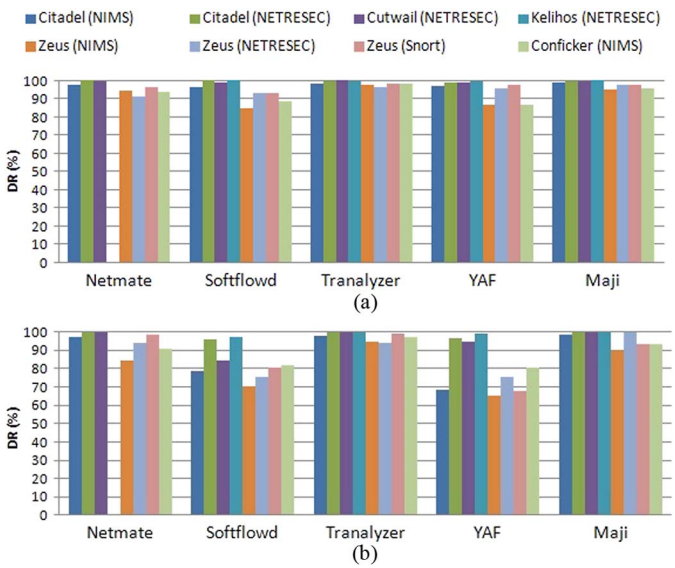

Fig. 5. DRs of the C4.5 and ANN classifiers on five flow exporters using HTTP traffic only on all data sets. (a) C4.5. (b) ANN.

exporter used also has an effect on the FPR obtained for all classifiers. On these data sets, the lowest FPRs are provided by ANN using Maji as the exporter. However, still some of the FPRs obtained even using Maji feature set are high to accept in practice. Hence, in the following, we study this phenomenon in more detail.

In the evaluations presented above, we used all the network packets available in the networks traces to convert them to flows. As the next step, we want to analyze how the performance of the aforementioned systems would change (if any) 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, we conduct a series of experiments applying HTTP and DNS filters on the data sets before classifying them for botnet detection.

1) HTTP Filtering: Given the wide range of the HTTP usage on the Internet, most recent botnets employ HTTP protocol to hide their malicious activities among the normal web traffic [45]. This way, they can easily bypass firewalls and avoid botnet detection mechanisms. The botnets that we employed in this paper also utilize HTTP protocol to communicate with their bots. Thus, to investigate the effect of protocol filtering on botnet detection via machine learning approaches using different flow exporters, we filtered the HTTP traffic flows and forwarded them to our botnet classifiers. Then, we repeated our approach presented above to train our classifiers.

Fig. 5 shows the botnet classification results versus the normal traffic using the HTTP filter. Due to space limitations, we again choose the two best performing classifiers (in terms of DR and FPR) and present their performances here. Similar to the traffic classification without any packet filtering, C4.5 and ANN outperformed the other classifiers. The results also support our observations that the classifiers could differentiate botnet behavior 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 data sets, except one. In this case, the classifiers showed a decrease in DR on Zeus data sets when using Maji, whereas an average 2% increase is observed with Tranalyzer on Zeus data set. Therefore, in these experiments where the HTTP filter is used, we have a clear winner in terms of flow exporters used. This is Tranalyzer. This observation might imply that HTTP traffic could be better presented by Tranalyzer flow features rather than the other flow exporters features. A close look at the features that the C4.5 classifier employed using Tranalyzer demonstrates that interarrival-based features are the ones that are highly employed in the decision tree solutions. Hence, we speculate that the supported features by Tranalyzer are very useful in terms of representing HTTP traffic, which are also introduced by Moore *et al.* for flow classification [46]. On the other hand, such features are not supported by Maji and Softflowd. These two exporters are open-source versions of IPFIX and Netflow V9, respectively. It is a concern that, although interarrival-based features are very important for botnet traffic analysis, they are not supported by mostly used exporter tools such as Netflow and IPFIX.

As discussed earlier, DR is not the only parameter that should be analyzed to evaluate the performance of a classifier. Fig. 6 shows the FPRs of the two classifiers. Compared with Fig. 4, both classifiers showed at least 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 is evidence on how useful the traffic filtering can be when employed to refine the data, which could be the center of focus specifically for the challenging botnet detection research. To investigate the effect of protocol filtering on botnet detection furthermore, we run a series of experiments on DNS filtering given that DNS is another essential protocol in botnet communication.

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

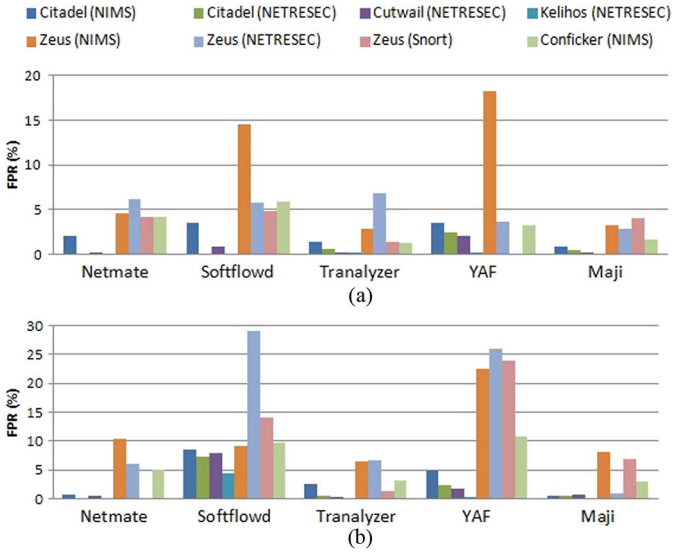

Fig. 6. FPRs of the C4.5 and ANN classifiers on five flow exporters using HTTP traffic only on all data sets. (a) C4.5. (b) ANN.

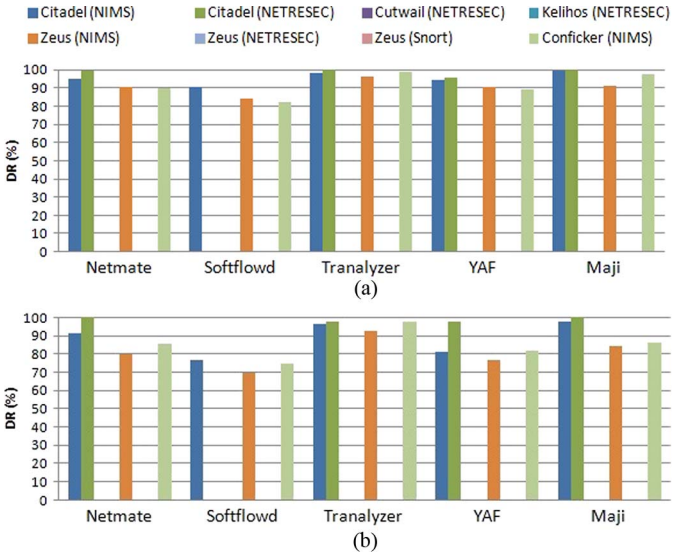

Fig. 7. DRs of the C4.5 and ANN classifiers on five flow exporters using DNS traffic only on all data sets. (a) C4.5. (b) ANN.

techniques to solve the single point of failure problem of their C&C servers and to achieve mobility [2]. Many botnets use the DNS legitimate infrastructure to locate their C&C servers to avoid static configuration of C&C servers IP addresses such as Zeus, Conficker, and Pushdo.

Thus, we also evaluate the aforementioned systems using only the DNS traffic flows via the DNS filters we set on the captured traffic. In this case, some of the data sets (Cutwail, Kelisos, and Zeus from Snort and NETRESEC) are very short traces; thus, they do not have DNS packets to generate enough DNS flows when the filters are in process. Thus, in Fig. 7, where the results of this evaluation are presented, there are no performance indicators for these data sets. Although, on the rest of the data sets, all of the classifiers showed some level of improvement in their performances when DNS filters were set, compared with the classification performance of the traffic without filtering (see Fig. 3). As shown in Figs. 7 and 8, the performance

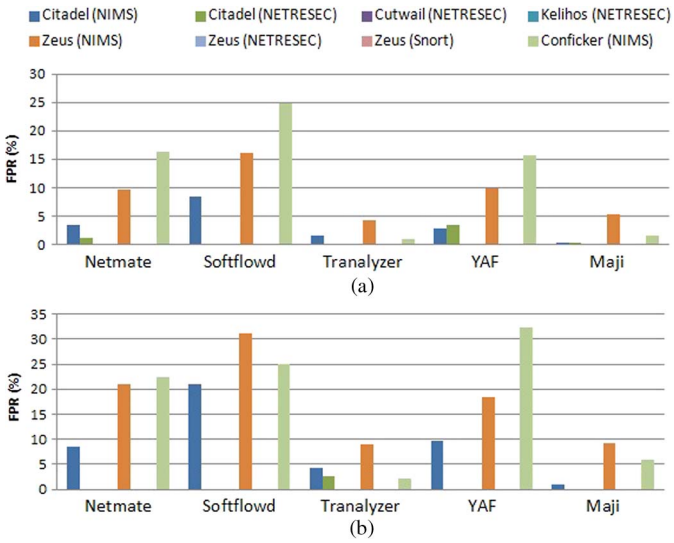

Fig. 8. FPRs of the C4.5 and ANN classifiers on five flow exporters using DNS traffic only on all data sets. (a) C4.5. (b) ANN.

TABLE IV
C4.5 CLASSIFICATION RESULTS USING THE TRANALYZER FEATURE SET WITH 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 (NIMS)	98%	99%	3%	97%	1%
	Zeus (NETRESEC)	97%	99%	6%	94%	1%
	Zeus (Snort)	99%	99%	1%	99%	1%
	Conficker (NIMS)	99%	98%	1%	99%	2%

of C4.5 (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 that, again, Tranalyzer is the winner when DNS filtering is in use with better DR and acceptable FPR.

D. Highlights

Overall, here are the highlights of the evaluation results we presented above: 1) Although some of the flow exporters in this paper are based on the same standard (i.e., 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 that are employed in the machine learning solutions. For example, "MinimumIPTotalLength" is utilized by C4.5 on Maji feature set, whereas YAF does not support such a feature. 2) 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 might not generate any flows due to its predefined rules, e.g., Netmate on Kelihos traffic trace. 3) Among the five flow exporters, all of the classifiers performed better when Tranalyzer and Maji were employed, whereas the third place belongs to Netmate. Therefore, this demonstrates that the tool employed for flow exporting analysis does have an

TABLE V
TRANALYZER FEATURES EMPLOYED BY C4.5 FOR BOTNET CLASSIFICATION—ALL BOTNETS

Flow Features					
AvePktSize	BytAsm	**Bytps**	**ConnDst**	ConnSrc	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	TcpOptCnt	TcpOptPktCnt	TcpPAckCnt	TcpPseqcnt	TcpRTTAckTripAve
TcpRTTAckTripMax	**TcpRTTAckTripMin**	TcpRTTSseqAA	TcpS-SA/SA-ATrip	**TcpSeqSntBytes**	TcpWinSzDwnCnt
		TcpWS	UppQuartileIat		

TABLE VI
PERFORMANCES REPORTED IN THE LITERATURE

Proposed systems	DR	FPR
Kirubavathi et al. [4]	up to 99%	1%
Zhao et al. [5]	99%	0.01%
Rerdisci et al. [7]	up to 79%	$\sim$ 0%
Wurzinger et al. [9]	88%	11%
Gu et al. [13]	up to 100%	$\sim$ 0%

effect on the traffic classification performance. 4) On the other hand, the protocol filtering experiments show that such filtering improves the botnet classification by focusing the classifier 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 among the exporters and the classifiers, respectively, with almost no exception. Then, ANN comes next as the second best performing classifier. 5) Comparing HTTP filtering versus DNS filtering shows that the FPRs and DRs did not change significantly in terms of focusing on one filtering versus the other. However, HTTP filtering seems to increase the performance more than the DNS filtering. This might be because all of the botnets we included in our evaluations employ HTTP as their communication protocol, whereas DNS protocol is used only to find the C&C servers by the botnets employed in this paper. Tables IV and V show the classification results and the features employed by C4.5 on all of the eight botnet data sets while using HTTP filtering, respectively. 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 FPR of up to 0.1% by using only 50 features out of the 93 features (complete set) of the Tranalyzer feature set on the HTTP filtered traffic. This performance of the proposed combination is higher (e.g., [7] and [9]) or similar (e.g., [4], [5], and [13]) to the results reported on the HTTP-based botnets (not necessarily the botnets or even the data sets employed in this paper) in the literature (see Table VI).

The bold features in Table V show the highly utilized features, indicating the importance of interarrival and Packets& Bytes feature categories, which are also supported by other works in the literature [5], [8]. Moreover, Tranalyzer has combined the most important features that are required for detecting various types of botnets reported in the literature [5], [8], [10], [13]. Hence, we believe that the proposed feature set by Tranalyzer can be useful for other types of botnets as well, which can be used for a real-time detection system.

V. CONCLUSION

Botnets are considered as one of the main security threats on the Internet due to their high reported infection rate and extensive range of malicious activities with active update capability. Hence, the need for botnet detection approaches that can adapt to the botnet evolution is very important. To this end, many automatic botnet detection approaches applying network traffic analysis are proposed in the literature. Each of these systems utilizes particular network traffic features based on flows (packet headers) in their analysis of the traffic. Hence, in this paper, we have benchmarked five different feature sets extracted by open-source flow exporters and investigated the effect of these flow (packet header based) features in botnet detection. To this end, five 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 for each of these parts different protocols are utilized based on the type of botnets, we also investigated the effect of protocol filters 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 greatly affect the performance of the botnet detection system. Our results suggest that open-source flow exporters Tranalyzer and Maji enable the classifiers to identify different botnet behaviors with significantly higher performances on all the data sets employed in our work. For the evaluations performed in this paper, the combination of the Tranalyzer tool with the C4.5 classifier using the HTTP filter gives the best performance in terms of DR and FPR on all the botnet data sets employed. Future work will explore the effect of nonnumeric features and other protocol filters for botnet traffic analysis. Moreover, the role of time and interarrival-based features will be analyzed in more detail.

ACKNOWLEDGMENT

This research is conducted as part of the Dalhousie University Network Information Management and Security Lab at http://projects.cs.dal.ca/projectx/.

REFERENCES

[1] M. Feily and A. Shahrestani, "A survey of botnet and botnet detection. Emerging security information," in *Proc. Syst. Technol.*, 2009, pp. 268–273.
[2] Open DNS Inc., Whitepaper, The Role of DNS in Botnet Command & Control, 2012.
[3] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," *Adv. Inf. Security*, vol. 36, pp. 1–24, 2008.

[4] V. Kirubavathi and R. A. Nadarajan, "HTTP botnet detection using adaptive learning rate multilayer feed-forward neural network," in *Proc. Inf. Security Theory Practice, Security, Privacy Trust Comput. Syst. Ambient Intell. Ecosyst.*, 2012, pp. 38–48.
[5] D. Zhao *et al.*, "Peer-to-peer botnet detection based on flow intervals," in *Proc. IFIP Int. Inf. Security Privacy*, 2012, pp. 87–102.
[6] F. Haddadi, J. Morgan, E. G. Filho, and A. Nur Zincir-Heywood, "Botnet behaviour analysis using IP flows with HTTP filters using classifiers," in *Proc. 28th WAINA*, 2014, pp. 7–12.
[7] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, p. 26.
[8] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller, "Salting public traces with attack traffic to test flow classifiers," in *Proc. CSET*, 2011, p. 3.
[9] P. Wurzinger *et al.*, "Automatically generating models for botnet detection," in *Proc. 14th ESORICS*, 2009, pp. 232–249.
[10] H. R. Zeidanloo, A. Bt. Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani, "Botnet detection based on traffic monitoring," in *Proc. ICNIT*, 2010, pp. 97–101.
[11] J. Francois, S. Wang, R. State, and T. Engel, "BotTrack: Tracking botnets using Netflow and PageRank," *Networking*, vol. 6640, pp. 1–14, 2011.
[12] K. Wang, C. Huang, S. Lin, and Y. Lin, "A fuzzy pattern-based filtering algorithm for botnet detection," *Comput. Netw.*, vol. 55, no. 15, pp. 3275–3286, Oct. 2011.
[13] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. 17th USNIX Security Symp.*, 2008, pp. 139–154.
[14] Publicly Available PCAP Files. [Online]. Available: http://www.netresec.com/?page=PcapFiles
[15] Zeus Trojan Analysis. [Online]. Available: https://labs.snort.org/papers/zeus.html
[16] F. Haddadi and A. Nur Zincir-Heywood, "Data confirmation for botnet traffic analysis," in *Proc. 7th Int. Symp. FPS*, to be published.
[17] Alexa. [Online]. Available: http://www.alexa.com/topsites
[18] Zeus/ZBot Malware Shapes up in 2013, May 2013. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/zeuszbot-malware-shapes-up-in-2013/
[19] Zeus Tracker. [Online]. Available: https://zeustracker.abuse.ch/
[20] DNS-BH- Malware Domain Blocklist. [Online]. Available: http://www.malwaredomains.com/
[21] Citadel Zeus bot. [Online]. Available: https://www.botnets.fr/index.php/Citadel_ZeuS_bot
[22] Citadel Makes a Comeback, Targets Japan Users, Sep. 2013. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/citadel-makes-a-comeback-targets-japan-users/
[23] K. Fu and J. Blum, "Controlling for cyber security risks of medical device software," *Commun. ACM*, vol. 56, no. 10, pp. 35–37, Oct. 2013.
[24] Conficker Domain List. [Online]. Available: http://net.cs.uni-bonn.de/uploads/media/c_domains_april2009.zip
[25] Pushdo botnet is evolving, becomes more resilient to takedown attempts, May 2013. [Online]. Available: http://www.pcworld.com/article/2038893/pushdobotnetisevolvingbecomesmoreresilienttotakedownattempts.html
[26] Infiltering Pushdo Part 2, Aug. 2010. [Online]. Available: http://www.fireeye.com/blog/technical/botnet-activities-research/2010/08/infiltrating-pushdo-part-2-2.html
[27] It's (Already) Baaack: Kelihos Botnet Rebounds With New Variant, Mar. 2012. [Online]. Available: http://www.darkreading.com/attacks-breaches/its-already-baaack-kelihos-botnet-reboun/232700540
[28] RFC 2722, Oct. 1999. [Online]. Available: http://tools.ietf.org/html/rfc2722
[29] Cisco IOS NetFlow. [Online]. Available: http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html
[30] Juniper J-Flow. [Online]. Available: http://www.juniper.net/techpubs/software/erx/junose82/swconfig-ip-services/html/ip-jflow-stats-config2.html
[31] S-Flow. [Online]. Available: http://www.inmon.com/technology/index.php
[32] Maji. [Online]. Available: http://research.wand.net.nz/software/maji.php
[33] YAF. [Online]. Available: http://tools.netsa.cert.org/yaf/index.html
[34] Softflowd. [Online]. Available: http://www.mindrot.org/projects/softflowd
[35] NfDump. [Online]. Available: http://nfdump.sourceforge.net/
[36] Tranalyzer. [Online]. Available: http://tranalyzer.com/
[37] Netmate. [Online]. Available: http://ipmeasurement.org/index.phpoption=com_content&view=article&id=10&Itemid=9
[38] Netmate FlowCalc. [Online]. Available: http://dan.arndt.ca/nims/calculating-flow-statistics-using-netmate/
[39] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. Vasilakos, "An effective network classification method with unknown flow detection," *IEEE Trans. Netw. Serv. Manag.*, vol. 10, no. 2, pp. 133–147, Jun. 2013.
[40] M. Soysal and E. G. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison," *Performance Eval.*, vol. 67, no. 6, pp. 451–467, Jun. 2010.
[41] E. Alpaydin, *Introduction to Machine Learning*. Cambridge, MA, USA: MIT Press, 2004.
[42] P. Royal, Maliciousness in Top-Ranked Alexa Domains. [Online]. Available: https://www.barracudanetworks.com/blogs/labsblog?bid=2438
[43] A. Makanju, A. N. Zincir-Heywood, and E. Milios, "Robust learning intrusion detection for DoS attacks on wireless networks," *Int. J. Intell. Data Anal.*, vol. 15, no. 5, pp. 801–823, 2011, IOS Press.
[44] WEKA. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/
[45] HTTP-Botnets: The dark side of a standard protocol (2013, Apr.). [Online]. Available: http://securityaffairs.co/wordpress/13747/cyber-crime/http-botnets-the-dark-side-of-an-standard-protocol.html
[46] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," Intel Research-Cambridge, Cambridge, U.K., Tech. Rep., 2005.

Fariba Haddadi (S'12) received the B.Eng. and M.Sc. degrees from Yazd University, Yazd, Iran, in 2006 and 2010, respectively. She is currently working toward the Ph.D. degree with the Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada.

Her research interests include network traffic analysis, intrusion detection, data mining, and machine learning.

A. Nur Zincir-Heywood (M'01) received the Ph.D. degree in computer science and engineering from Ege University, Izmir, Turkey, in 1998.

She is currently a Full Professor of computer science with Dalhousie University, Halifax, NS, Canada. Prior to moving to Dalhousie University in 2000, she was a Researcher with Sussex University, Brighton, U.K., and the University of Karlsruhe, Karlsruhe, Germany, and an Instructor at the Internet Society Network Management workshops. Her research interests include network operations, computer and network security, information extraction, and computational intelligence.

Dr. Zincir-Heywood is a member of the Association for Computing Machinery.