On Botnet Behaviour Analysis using GP and C4.5

Fariba Haddadi, Dylan Runkel, A. Nur Zincir-Heywood, and Malcolm I. Heywood
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada
{haddadi,runkel,zincir,mheywood}@cs.dal.ca

ABSTRACT

Botnets represent a destructive cyber security threat that aim to hide their malicious activities within legitimate Internet traffic. Part of what makes botnets so affective is that they often upgrade themselves over time, hence reacting to improved detection mechanisms. In addition, Internet common communication protocols (i.e. HTTP) are used for the purposes of constructing subversive communication channels. This work employs machine learning algorithms (genetic programming and decision trees) to detect distinct behaviours in various botnets. That is to say, botnets mimic legitimate HTTP traffic while actually serving botnet purposes. To this end, two different feature sets are employed and analyzed to see how differences between three botnets – Zeus, Conficker and Torpig – can be distinguished. Specific recommendations are then made regarding the utility of different feature sets and machine learning algorithms for detecting each type of botnet.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Invasive software*

Keywords

Security; botnet detection; machine learning

1. INTRODUCTION

Botnets represent a set of compromised hosts under the remote control of a botmaster i.e., a master–slave relationship. They epitomize an approach to putting what would normally be considered legitimate users to malicious ends. As researchers propose detection mechanisms to identify botnets behaviours, botmasters upgrade their bots to defeat detection. Given the distributed nature of botnets, there are many avenues by which detection and evasion can take place. Moreover, given the widespread use of legacy systems, which remain connected to the Internet, even 'old' botnets remain effective. Hence, even after a takedown, botnets make impressive comebacks [28]. In effect, the recovery of a botnet is potentially caused by the lack of suitable upgrades to legacy systems. As reported by Fu *et al.* [15], Conficker (as a relatively older generation botnet compared to Zeus) was detected on 104 devices at the James A. Haley Veteran's Hospital in Tampa in 2013. Moreover, McAfee predicted that in 2014 attackers will target systems using the old Windows XP operating system;[1] where legacy point of sale and medical systems frequently use Windows XP.

Many existing approaches to botnet detection rely on network traffic behaviour analysis. Some of these employ Machine Learning (ML) techniques (i.e. classification and clustering) to automatically generate botnet detection models. In such systems, the first step is to represent the network traffic in a way that is meaningful for the ML techniques employed. To this end, different systems assume their own set of features [11, 16, 31, 33]. Some only use network packet headers (i.e. [11, 16]), while others take advantage of packet payloads (i.e. [31, 33]). Botnets, on the other hand, employ encryption techniques to avoid detection systems that analyze the communication information embedded in the packet payload.

In this work, we investigate a machine learning-based botnet detection mechanism that does not use packet payload information; where this is opaque when encrypted. If successful, we will therefore be able to detect botnets using encryption. To this end, two ML approaches will be employed (C4.5 decision tree induction [8] and a form of team based genetic programming (GP) [13]), with the goal of qualifying to what degree assuming different methods provides better detection coverage. Both algorithms have previously demonstrated to be effective for distinguishing between encrypted and non-encrypted traffic [9].

The approach adopted in this work is to first construct network traffic and extract the required features (attributes) using a flow exporter tool. Specifically, TCP packets are converted into a *flow* representation summarizing various properties for a group of packets associated with the same source / destination IP and port numbers as well as protocols. However, there are many properties that can be derived to characterize such flows. Depending on the tool used to construct the flow, the resulting features may be more / less effective at detecting botnet activity. Moreover, it is possi-

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2598394.2605435.

[1]Microsoft announced that as of 8 April 2014, Windows XP will no longer be supported. However, Windows XP is still the second most widely deployed desktop operating system.

ble that botnets are using TCP flags in ways that were not intended for legitimate use [20].

Traffic from three botnets will be used: Conficker, Torpig and Zeus. These botnets have recently been associated with critical network infrastructure such as medical systems [15] or demonstrated a high infection rate. Our objectives are three-fold: 1) establish how effective a basic flow based information is for detecting each botnet; 2) to identify the most effective attributes for detecting each botnet, and 3) to make recommendations regarding signatures appropriate for detecting each botnet.

2. BACKGROUND

Many systems require supplementary information such as network alert information or several botnet binaries for the purpose of detection [21, 26, 25]. However, it is possible to detect the botnets without any additional information beyond that available in network traffic data. Yadav *et al.*, proposed a methodology to detect botnets, addressing the domain fluxing mechanism employed by the botnets such as Zeus, Conficker and Torpig [32]. To this end, they analyzed the DNS queries and the entropy of the domain names belonging to such queries. The proposed system was tested against Conficker botnet, which resulted in a promising detection and false positive rates. Haddadi *et al.*, on the other hand, designed and developed a Stateful-SBB (a co-evolutionary GP approach) based technique to detect botnets such as Conficker, employing only the command and control (C&C) domain names without any DNS group behaviour analysis [17]. However, since the systems proposed by both Yadav *et al.* and Haddadi *et al.* require access to a domain name information (i.e., packet payload information), neither are effective for botnets with encrypted packet payloads.

Botnet detection systems can be categorized based on the communication protocol they employ, the detection methods they employ or the type of data they use. Thus, some researchers apply machine learning techniques for botnet detection purposes using flow-based information (as attributes) [11, 16, 27, 14]. Given that botnets are increasingly employing encryption techniques to hide their information, then detection techniques that analyze the packet payload information can no longer be useful (e.g., [31, 33]). Researchers should therefore design frameworks that can also detect encrypted botnet behaviour. Celik *et al.*, proposed a flow-based botnet C&C activity detection system using only headers of traffic packets [11]. Specifically, they investigated the effect of calibration of time-based flow features using machine learning algorithms such as C4.5. Kirubavathi *et al.* designed specifically an HTTP-based botnet detection system using a multilayer Feed-Forward Neural network [19]. 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. Haddadi *et al.* designed a botnet detection approach based on botnet traffic analysis [16]. Network traces representing normal and attack traffic were generated 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.

3. METHODOLOGY

In this work, two ML paradigms are assumed: C4.5 decision tree [8] and the symbolic bid-based (SBB) framework for evolving teams of programs to detect botnet behaviour [13]. Both of these learning algorithms generate solutions (models) that are in human readable format and therefore enable the analysis of the learned models.

Traffic features (attributes) are expressed as flows using the Softflowd tool [5]. In this case, the features are derived from packet header information alone. Therefore they can be employed for encrypted traffic classification, too. Moreover, most of the works in the literature employ a specific set of features for various botnets. However, in this work, we investigate which features can be useful to detect which type of botnet. To do so, two feature sets, namely Softflowd set.1 and Softflowd set.2, exported by Softflowd are employed and analyzed. We believe that this type of feature analysis may lead us to understand the botnet behaviors and their differences in more detail and in return could enable us to design and develop better botnet detection systems.

We evaluate our system on three botnets, namely Conficker, Torpig and Zeus. The reasons behind this are the following: most of the aggressive botnets employ encryption methods to hide their sensitive information (such as Zeus), and most of the systems such as the medical devices that Conficker and Torpig recently target have legacy operating systems. In doing so, we not only employ our proposed approach in legacy systems or medical devices but also on any device that runs on a TCP/IP network whether it is encrypted or not. Thus, the coverage is much greater than targeting a specific operating system or application.

3.1 Learning Algorithms Employed

The candidate ML algorithms go about constructing classifiers using different credit assignment policies and representations. However, both approaches share an ability to perform attribute selection is an implicit property of constructing a classifier. This property will later be used to gain insight to how botnets are communicating using HTTP.

3.1.1 C4.5 decision tree algorithm

C4.5 is a decision tree algorithm extending the earlier ID3 algorithm developed by Quinlan [8]. 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. Following the addition of each split, an *IF–THEN* node is added to the current decision tree. 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.

In more detail, we note that the expected information or 'entropy' of a class given the exemplars, X from the training data has the form:

$$I(X) = -\sum_{i=1} f(X,i) \times log_2(f(X,i)) \quad (1)$$

where

$$f(X,i) = \frac{|\{j \in X | (j) := i|}{|X|} \quad (2)$$

A candidate split partitioning the data into partitions $X_1, , X_n$ has the entropy:

$$I_S(X) = -\sum_{k=1}^{n} \frac{|X_k|}{|X|} \times I(X_k) \quad (3)$$

The resulting information gain is the difference in entropy before and after introducing the split or $gain(X) = I(X) - I_S(X)$. The C4.5 algorithm makes use of additional normalizations to reduce biases towards unequal class representation [8]. The implementation used in this work takes the form of the WEKA 'J48' release [30].

3.1.2 SBB

The Symbiotic Bid-Based (SBB) algorithm is a form of genetic programming that builds teams of programs cooperatively while simultaneously identifying useful exemplars to learn from [13]. 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 identify subsets of learners to define team membership under a variable length representation; the implication of the latter being 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 a member of the target host, h, 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 who 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 work a linear GP representation is employed [10].

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 e.g., [12]. 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 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. New symbiont programs potentially 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 in favour of the points / teams that exhibit non-overlapping behaviour.

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. (5) Section 4.2.1). The solution team is a combination of a set of learners with their corresponding GP instructions. In our evaluations, the maximum program size is set to 48. Thus, each learner in the solution can have maximum 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 [10], we employ intron removal to reduce the complexity of SBB [17]. A more detailed explanation of the algorithm can be found in [13].

3.2 Data set Generation and Feature Extraction

The Hyper Text Transfer Protocol (HTTP) is one of the most common Internet protocols on account of the popularity of web applications on the Internet. Naturally, botnets have started to use this protocol to hide their malicious behaviours in the legitimate users' activities [24]. Zeus, Conficker and Torpig represent three botnets that utilize HTTP protocol for their communication purposes. These are well-known botnets where Conficker was recently reported to be seen on medical devices and is also known to be used for collecting banking information. Torpig, on the other hand, is known to be a good representative of how domain fluxing is used on the HTTP to steal financial and medical information. Finally, Zeus is one of the most destructive botnets reported which came with a new variant in 2013 after the takedown in 2012. 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.

A typical advanced botnet forms in five stages: initial infection, secondary infection, connection, malicious C&C and finally update and maintenance. During stage 1, exploitation techniques are used to find victim vulnerabilities and infects the target host. Once infected the shell-code is executed on the victim machine to fetch the image of the bot binary which then installs itself on the machine (stage 2). In the third stage, the bot binary establishes the C&C channel. This channel is then used by the bot master to send the command in the malicious C&C stage. Finally, the update and maintenance stage is entered when the botmaster needs to update the bots for one reason or another.

Since, there is not a significant amount of botnet traffic publically available, we generated traffic. To this end, we designed an approach to generate the botnet traffic representing the first phase of the botnet communication which happens during the third stage of botnet lifecycle. In order to do this, we employed publicly available (from legitimate resources) lists of C&C domain names for Conficker, Torpig and Zeus. We generated the representative botnet traffic for the first phase of botnet communication by initiating HTTP based communication to the domain names reported in these publicly available lists. These lists were obtained from ZeusTracker [6], DNS-BH [3], Twitter API [29], Bonn University [2] websites. As for the representative

legitimate traffic, we employed the publicly available Alexa domain lists which are based on the most popular websites from Alexa Inc. [1]. This way, both the attack and the normal traffic were generated using the publicly available domain name lists to avoid any biases in the generated traffic. The aforementioned generated botnet data sets are evaluated against the few publicly available botnet data sets in [18]. The results of this evaluation indicate that the generated traffic representing the first phase of a botnet communication is comparable to the real-life botnet traffic data sets. Moreover, since Zeus botnet leaked in 2011, there are Zeus botnet kits publically available. Therefore, we also run the Zeus botnet in a controlled environment and captured the network traffic. Furthermore, there are Zeus botnet traffic captures available at NETRESEC [4] and Snort [7] websites that we also employed in this work.

Once the traffic is collected, we convert the captured packet format traffic into Internet Protocol (IP) traffic flows using an IP Flow exporter. Flow exporters aggregates the traffic based on 5-tuple information: Source IP address, Destination IP address, Source port number, Destination port number and Protocol. Then the flow traffic is summarized in terms of some statistics such as the number of packets per flow, bytes per flow etc.

Cisco Systems introduced NetFlow to collect and aggregate IP traffic information. Given that Cisco is the leader of IP flow technology, NetFlow became an industry standard and therefore, many network equipments in the market support it. Thus, in this work, we employ of one of the open source implementations of NetFlow exporters, namely Softflowd [5].

4. EVALUATIONS AND RESULTS

As discussed earlier, our goal is first to detect various types of botnets using only flow-based features and second, to find the features that best describe the behaviour of the botnets employed. To this end, two ML algorithms are employed on two different feature sets and the solutions analyzed.

4.1 Data Sets

Softflowd is used to provide 14 flow attributes with the default parameters. All numeric attributes could be employed directly. However, the "flag" attribute is a string based feature, hence requires conversion to a numeric form prior to use by ML either SBB or C4.5. However, the numeric encoding assumed for this purpose can have a significant impact on the resulting classifier [23]. Benchmarking will therefore be conducted without the "flag" attribute (Softflowd set.1) and with (Softflowd set.2).

In the case of Softflowd set.2, six "flag"-based features are defined (Table 1). Each of the six flags are encoded using two numeric values to indicate when they are set / not set (during a communication). Table 1 summarizes the attributes that are utilized in this work. A detailed definition of the attributes can be found in Softflowd project web site [5]. 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. Table 2 indicates the number of flow samples in each data set. Hereafter, we will refer to the data sets generated in our lab using the publicly available domain names as Zeus-1 (NIMS), Conficker (NIMS) and Torpig (NIMS) and the Zeus data set that is created based on the publicly available Zeus botnet kit as Zeus-2 (NIMS). On the other hand, the other botnet traffic is referred to using the download source. Hence, Zeus (Snort) is the Zeus traffic made public on the Snort web site [7] and Zeus (NETRESEC) is the botnet traffic provided on the NETRESEC web site [4].

Table 1: Employed Softflowd Features

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 2: Number of flows in each data set employed

Data Set	Training		Testing	
	Legit	Botnet	Legit	Botnet
Zeus-1 (NIMS)	6099	6099	2614	2614
Zeus-2 (NIMS)	611	611	262	262
Zeus (NETRESEC)	252	252	108	108
Zeus (Snort)	100	100	43	43
Conficker (NIMS)	28951	28921	12386	12416
Torpig (NIMS)	1864	1856	794	800

4.2 Performance Metrics

4.2.1 Performance

Typically, classifiers are evaluated using accuracy or classification rate as the fraction of all the correctly labeled instances. However, given an unbalanced data set or a multiclass data set, these metrics can be misleading. In this regard, a classwise detection rate is defined as [22]:

$$DET_c = \frac{TP_c}{FN_c + TP_c} \tag{4}$$

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 classes the average DR criteria is defined by [22]:

$$Score = \frac{1}{|C|} \sum_{c \in C} DET_c \tag{5}$$

4.2.2 Complexity

Classifier complexity can be measured by different criteria such as memory consumption, time or 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**, however, a direct comparison between solutions from different representations is impractical since the underlying units of measurement are different. Therefore, the tree size for C4.5 and the program size of the solution team for SBB are considered as our units of measurement. **3) feature complexity** reflects the number of unique attributes employed per solution and potentially gives additional knowledge regarding botnet communication.

4.3 Results

Table 3 and 4 present the classification results of C4.5 and SBB employing two different feature sets, namely Softflowd set.1 and Softflowd set.2. 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 (Section 4.1). As shown in Table 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-2 (NIMS), Zeus (NETRESEC), Zeus (Snort) and Conficker (NIMS) while using Softflowd set.1. On the other hand, the performance results on Torpig (NIMS) and Zeus-1 (NIMS) are much lower than the others when Softflowd set.1 is employed as the feature set. This observation indicates that Torpig and Zeus-1 (NIMS) botnets characteristics cannot be well represented by the features of Softflowd set.1.

The Softflowd set.2 feature set is then employed to investigate if TCP-flags would be beneficial to improve classification performance for Torpig (NIMS) and Zeus-1 (NIMS) specifically. Table 4 shows the results of these additional experiments. The results show 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 the features to botnet classifiers can be beneficial. Surprisingly, Torpig (NIMS) results were improved by more than 30% when traffic is 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. Morever, there appears to be no disadvantage in using the Softflowd set.2 attributes.

Comparing the results over complexity criteria, Table 4, there is not much difference between the solutions based on feature complexity, i.e. the number of features used from the set given. However, there are significant differences in terms of time and solution complexities. To this end, C4.5 training time is much less than SBB training time. SBB, on the other hand, obtained smaller solutions (e.g. 88% smaller for 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 less number of rule/signatures. Although, in some cases, the low complexity is caused by an under-performing solution, in others, it is a good indicator of a good solution with low complexity.

4.3.1 C4.5 solution analysis

Going beyond analyzing the classification results in terms of performance parameters such as Score, TP and FP rates, we analyzed the solutions learned by the classifiers. This type of analysis might give of some insights on the Zeus, Conficker and Torpig botnet behaviours.

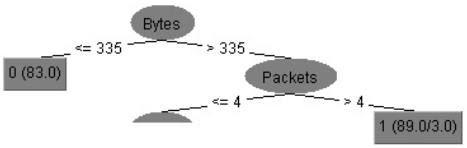

Figure 1: Part of the Zeus (Snort) C4.5 decision tree

To this end, the C4.5 solution (after training using the Softflowd set.2) for the Conficker botnet is a very complex tree (365 rules). On the other hand, C4.5 solution (after training using Softflowd set.2) for the Torpig botnet resulted in a very small tree with 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 the Softflowd set.1), we believe that Torpig probably employs these flags to tag its packets in a not-routine way. We will investigate this direction in our future work in more detail.

Additionally, we analyzed C4.5 solutions 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 rather 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”. On the other hand, the analysis of the other three Zeus botnet data sets shows that “Pkts” (i.e. the total number of packets in a flow), “Byts” (i.e. the total number of bytes in a flow), “Flag-S” (indicating the status of TCP SYN flag in the communication) and “Flag-F” (indicating the status of the TCP FIN flag in the communication) are widely utilized. To this end, in Zeus (NETRESEC), 15% of the botnet training samples were labeled using the “Flag-R” (when 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”, Figure 1. Although the flag-based features are used by C4.5 to build the classification models for 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 Zeus botnet traffic downloaded from the Snort web site as well as the Zeus-1 traffic. On the other hand, it seems like it does not help the identification of Zeus botnet traffic downloaded from the NETRESEC site nor the Zeus-2 traffic. However, in both these cases, the decrease in DR is compensated by the improvement in the FPR. This observation indicate that not all versions of the Zeus botnet (considering different data sets that may belong to different versions of this botnet) utilize the TCP flags in their communication.

4.3.2 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 only associated with a single class label. This provides a level of task decomposition that is not possible under C4.5.

Figure 2 shows an example of a Torpig class-1 learner’s

Table 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-1 (NIMS)	84%	86%	17%	83%	14%	0.26	485	8
	Zeus-2 (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 (NIMS)	92%	91%	7%	93%	9%	2.71	411	6
	Torpig (NIMS)	68%	91%	55%	45%	9%	0.07	49	6
SBB	Zeus-1 (NIMS)	77%	80%	27%	73%	20%	185.56	27	5
	Zeus-2 (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 (NIMS)	90%	89%	9%	91%	11%	178.10	81	7
	Torpig (NIMS)	65%	92%	63%	37%	8%	186.12	20	4

Table 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-1 (NIMS)	87%	90%	16%	84%	10%	0.24	457	9
	Zeus-2 (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 (NIMS)	94%	93%	5%	95%	7%	3.41	365	10
	Torpig (NIMS)	99%	99%	1%	99%	1%	0.04	17	5
SBB	Zeus-1 (NIMS)	78%	73%	18%	82%	27%	188.252	51	8
	Zeus-2 (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 (NIMS)	91%	90%	9%	91%	10%	192.44	41	9
	Torpig (NIMS)	100%	100%	0%	100%	0%	109.23	60	11

instruction set which is part of the SBB's solution for the Torpig botnet i.e., a subset of the SBB solution shown in Table 4. The program's instruction count is reduced to 2 from 17 by pruning (cf., intron removal). The pruned instruction set indicates that the learner multiplies the "Bpp" (i.e. Bytes per packet) value by 0.54 *if* the "Flag-U" (indicating the urgent TCP flag) is set and returns "Bpp" value otherwise. Knowing that if this learner wins the bid over a data sample, it labels the sample as botnet, this learner's solution implies that samples with the set "Flag-U" look more suspicious and labeled as a botnet.

Using flag bits in malware communication has already been suspected by other researches [20]. Our observation supports this hypothesis for Torpig botnet on the data sets we analyzed. Overall 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, we observe similar trends to 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 again indicate that the versions of Zeus botnet seem to be different from one Zeus traffic file to another. For SBB flag features improve the solution performance, specially for Zeus Snort and Torpig botnets. In most cases it also introduces an improvement in the false alarm rates for SBB.

4.4 Discussion

In summary, Softflowd set.2 feature set performed better in terms of higher Score and lower FPR. Analysis of both the SBB and the 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 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 are using 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: (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 classifications, (iii) the features employed by C4.5 and SBB for Zeus (Snort) classification are almost complementary while SBB's selected feature set could obtain a 100% detection rate with a zero FPR, and (iv) in the Zeus-1 (NIMS) data set where the performance is lower than expected, the selected features by the two classifiers do not overlap much. This raises the question of whether

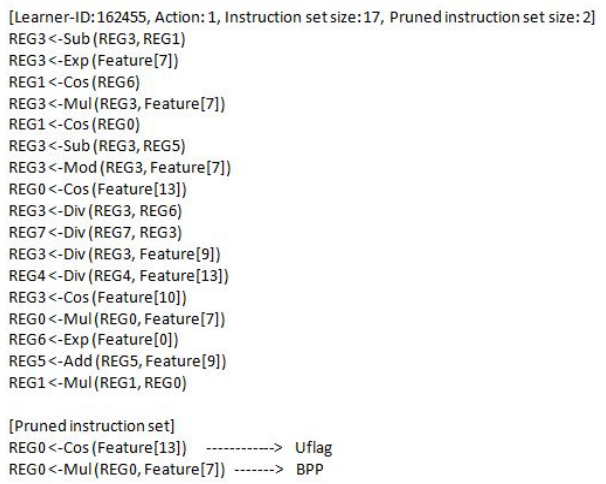

Figure 2: SBB- a sample learner instruction set with botnet label on Torpig– Softflowd set.2

the performance would increase by finding a solution that combines these two different solutions with complimentary feature sets.

Overall, our results and analysis presented in this work indicate that SBB and C4.5 learning algorithms focus on different properties of the traffic. This in return enables them to recognize different botnet behaviours. However, we think that SBB's low solution complexity makes it a better classifier when implementing a real-time botnet detection system. Finally, the two employed classifiers could detect most of the botnets with high performance while having higher than desired FPR for the Conficker and Zeus-1 botnets. Having said this, in our evaluations using a different feature set with TCP flags did help to decrease the FPR from Softflowd set.1 to Softflowd set.2 for these two botnet data sets. This indicates that a more detailed feature set analysis is necessary for these botnets. Moreover, given that different classifiers seems to work better for different botnets, i.e. different behaviours, an ensemble learning algorithm might be beneficial for future research.

5. CONCLUSIONS

A botnet is a set of compromised hosts that are under the remote control of a botmaster. Due to high infection rate and vast range of malicious activities, botnets are considered as one of the main threats against the cyber security. Since botmasters can update any phase of the botnet lifecycle at any time to defeat the detection systems, detection systems also require automatic and intelligent mechanisms to cope with the updates. In this work, we employ two machine learning algorithms, namely C4.5 and SBB, to generate botnet detection models for Zeus (different versions), Conficker and Torpig botnets automatically. Moreover, we represented the traffic using a flow exporter, namely Softflowd, to convert packets to traffic flows and extract their features. To this end, two feature sets (Softflowd set.1 and Softflowd set.2) both exported by Softflowd, are employed which enable us in revealing some of the characteristics of the aforementioned botnets behaviour.

Our first objective in this work was to investigate the possibility of detecting the aforementioned botnets using flow based features. As the results indicate, both of the classifiers performed very well using the Softflowd set.2 feature set and obtained the Score (classwise detection rate) of up to 100% for some of the botnets. This confirms the accomplishment of our first objective. To fulfill our second and third objectives which were finding the feature sets that best describe the botnets and return a solution that is suitable for a signature-based botnet detection system, the generated solutions and the features used are analyzed. The analysis determined some of the botnet characteristics. For example, given that Torpig detection models by both of the classifiers did not perform well using the Softflowd set.1 but did perform out-standing (30% increase in detection rate) using the Softflowd set.2 feature set, we believe that Torpig botnet employs the TCP flags in an abnormal way. However, we did not notice such behaviour in the Zeus or Conficker botnet solutions, which implies that these botnets do not employ such flags. Having said this, almost in all of our experiments SBB performed better than C4.5 in terms of the solution complexity.

Future work will explore what other flow features can be employed in botnet behavior analysis and their effects in terms of lowering the false alarm rates.

6. ACKNOWLEDGMENTS

This research is supported by the Natural Science and Engineering Research Council of Canada (NSERC) grant, and is conducted as part of the Dalhousie NIMS Lab at `https://projects.cs.dal.ca/projectx/`.

7. REFERENCES

[1] Alexa. http://www.alexa.com/topsites.
[2] Conficker Domain List. http://net.cs.uni-bonn.de/uploads/media/c_domains_april2009.zip.
[3] DNS-BH- Malware Domain Blocklist. http://www.malwaredomains.com/.
[4] Publicly available pcap files. http://www.netresec.com/?page=PcapFiles.
[5] Softflowd. http://www.mindrot.org/projects/softflowd/.
[6] Zeus tracker. https://zeustracker.abuse.ch/.
[7] Zeus trojan analysis. https://labs.snort.org/papers/zeus.html.
[8] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.
[9] R. Alshammari, P. Lichodzijewski, M. I. Heywood, and A. N. Zincir-Heywood. Classifying ssh encrypted traffic with minimum packet header features using genetic programming. In *ACM Genetic and Evolutionary Computation Conference – Defence Applications of Computational Intelligence Workshop*, 2009.
[10] 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.
[11] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller. Salting public traces with attack traffic to test flow classifiers. In *USNIX Cyber Security Experimentation and Test (CSET) workshop*, 2011.
[12] E. D. de Jong. A monolithic archive for

Table 5: Feature Matrix

	Zeus-1 (NIMS)		Zeus-2 (NIMS)		Zeus (NETRESEC)		Zeus (Snort)		Conficker (NETRESEC)		Torpig (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	-	✓	-	-	-	✓	-	-	-	-	-	✓

pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[13] J. Doucette, A. 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.

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

[15] K. Fu and J. Blum. Inside risk controlling for cyberscurity risks of medical device software. *Communications of the ACM*, 56:35–37, 2013.

[16] F. Haddadi, J. Morgan, E. G. Filho, and A. N. Zincir-Heywood. Botnet behaviour analysis using ip flows with http filters using classifiers. In *7th Workshop on Bio and Intelligent Computing (AINA-BiCOM)*, 2014. in press.

[17] 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.

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

[19] V. Kirubavathi and R. 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.

[20] V. Krmicek and T. Plesnik. Detecting botnets with netflow. In *Cert Flocon*, 2011.

[21] F. Leder and T. Werner. Know your enemy: Containing conficker. The HoneyNet Project, White paper, 2009.

[22] 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.

[23] A. Makanju, A. N. Zincir-Heywood, and E. Milios. Robust learning intrusion detection for dos attacks on wireless networks. *Intelligent Data Analysis An International Journal*, 15:801 823, 2011.

[24] Open DNS Inc. The role of DNS in botnet command and control, 2012.

[25] T. Ormerod, L. Wang, M. Debbabi, A. Youssef, H. Binsalleeh, A. Boukhtouta, and P. Sinha. Defaming botnet toolkits: A bottom-up approach to mitigating the threat. In *Emerging Security Information Systems and Technologies (SECURWARE)*, 2010.

[26] S. Shin and G. Gu. Conficker and beyond: a large-scale empirical study. In *26th Annual Computer Security Applications*, 2010.

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

[28] Trend Labs- Security Intelligence Blog. Zeus/zbot malware shapes up in 2013. http://blog.trendmicro.com/trendlabs-security-intelligence/zeuszbot-malware-shapes-up-in-2013/, 2013.

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

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

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

[32] 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.

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