Botnet Detection System Analysis on the Effect of Botnet Evolution and Feature Representation

Fariba Haddadi, A. Nur Zincir-Heywood
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada
{haddadi, zincir}@cs.dal.ca

ABSTRACT

Botnets are known as one of the main destructive threats that have been active since 2003 in various forms. The ability to upgrade the structure and algorithms on the fly is part of what causes botnets to survive for more than a decade. Hence, one of the main concerns in designing a botnet detection system is how long such a system can be effective and useful considering the evolution of a given botnet. Furthermore, the data representation and the feature extraction components have always been an important issue in order to design a robust detection system. In this work, we employ machine learning algorithms (genetic programming and decision trees) to explore two questions: (i) How can the representation of non-numeric features effect the detection system's performance? and (ii) How long can a machine learning based detection system can perform effectively? To this end, we gathered seven Zeus botnet data sets over a period of four years and analyzed three different data representation techniques to be able to explore aforementioned questions.

Categories and Subject Descriptors

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

Keywords

Security; botnet detection; machine learning; data representation; robustness.

1. INTRODUCTION

A collection of compromised hosts that are under the remote control of a master (aka botmaster) is referred to as a botnet. The infected hosts get involved in malicious tasks without even being aware of such activities like Distributed Denial-of-Service (DDoS) attacks and identity thefts. In general, botnets have two different topologies: Centralized and De-centralized (such as Peer-2-Peer (P2P)). In these topologies, a botmaster uses the Command and Communication (C&C) servers in order to communicate with the infected hosts. Moreover, in a P2P configuration, each node could play the master or the client role. Therefore, there is no specifically assigned C&C server in the P2P botnet topologies.

It is believed that until 2003, the Internet Relay Chat (IRC) protocol was the most common botnet communication protocol using the centralized topology [20]. However, to defeat the detection systems, botnets have started to use more ubiquitous protocols such as HyperText Transfer Protocol (HTTP) as well as de-centralized topologies such as P2P and other techniques such as fluxing and encryption. For instance, Conficker and Zeus botnets migrate from the HTTP-based communication to HTTP-based P2P communication over their evolution. To this end, identifying the botnets and detecting them have become very challenging. Thus, active continuous botnet monitoring and detection mechanisms are required. Such mechanisms could potentially enable us to learn the new patterns and adapt to the changes in the botnet evolution. For this purpose, many existing botnet detection approaches use machine learning (ML) techniques to analyze the network traffic behaviour. In such approaches, data representation is one of the first key phases where the network traffic data should be represented to the ML algorithms in a meaningful manner. The represented features in the existing approaches have been extracted from either the packet payloads and/or the packet headers. However, since botnets have started using encryption, the monitoring and detection systems that utilize only the packet headers information have the advantage over the systems that use the packet payload information (as this information is opaque when encrypted).

In this paper, we explore ML-based botnet detection systems that only use packet header information. For this purpose, C4.5 [6] as the decision tree algorithm and the Symbiotic Bid-Based Genetic Programming (SBB) [18] as the genetic programming algorithm are utilized. To summarize the network traffic and to extract the required features (attributes), a traffic flow exporter, called Tranalyzer, is employed which employs network packet headers to extract the features. This exporter has shown to be very effective in extracting useful botnet traffic features [16]. The ML approach adopted in this work is utilized to explore two main research questions. The first question studied is: How

GECCO'15 Companion, July 11–15, 2015, Madrid, Spain

 ISBN 978-1-4503-3488-4/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768435

can the non-numeric feature representation be effective in terms of performance, given that most of the known ML algorithms use numeric features? To answer this question, experiments are performed on TCP flags, which are known to be used by malwares [17]. This attribute has also shown to be used for some of the botnets in ways that were not intended for legitimate use [13]. To this end, we analyzed three representations- numerical, nominal and binary- specifically for the TCP flags. The second question studied is: How effective can a botnet detection system perform facing upgraded botnet behaviours, considering the fact that botnets upgrade their methodology over time? In our evaluations, we gathered seven Zeus botnet traffic data sets over a period of four years to understand how the performance (effectiveness) of a detection system changes (if at all) over time.

The rest of the paper is structured as follows: Section 2 summarizes the background and the related work on botnet detection. Our methodology is discussed in Section 3. Results are presented in Section 4. Finally, conclusions are drawn and future work is discussed in Section 5.

2. BACKGROUND AND RELATED WORK

Unlike first botnets that had a list of exploits to launch on targets where all the commands were set at the time of infection, today a typical advanced bot uses five stages to create and maintain a botnet. 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, secondary injection, the shell-code is executed on the infected victim to fetch the image of the bot binary. 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, 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 starts.

Botnets have employed different protocols, topologies and techniques to implement the stages of their lifecycle and therefore, different techniques are proposed for the detection purposes. Gu et al. proposed and developed a botnet detection framework called BotMiner[12]. This framework is based on the group behavior analysis [12]. 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. Their results showed that BotMiner could detect botnets with detection rates between 75% and 100% on different types of botnets. Celik et al. proposed a flow-based botnet C&C activity detection system using only headers of traffic packets [10]. They investigated the effect of calibration of time-based flow features. They employed techniques such as C4.5, Naive-Bayes and logistic regression. Wang et al. proposed a fuzzy pattern recognition approach to detect HTTP and IRC botnets' behavioral patterns [23]. 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 TCP connection with the C&C server. Therefore, Wang analyzed the features of DNS queries (such as the number of failed DNS responses) and TCP flows to detect malicious domain names and IP addresses. Beigi et al. investigated the effectiveness of flow-based feature sets employed in previous botnet detection studies and evaluated their relative effectiveness using their feature selection algorithm [7]. Their results indicated that the Byte-based group of features had less effect while the packet-based group had the highest impact.

There are several studies on flow-based botnet detection systems where each proposed their own set of features [22, 10]. Moreover, some studies have analyzed the feature selection algorithms to extract the most effective feature sets [7]. Such feature selection processes can cause the models to be focused on specific type(s) of botnet(s) which may not be very effective for other types. Hence, using a ML algorithm that has the ability to perform attribute selection as an implicit property of constructing the classifier may be a better way to approach feature selection while utilizing all the possible extracted flow features. C4.5 can be named as one of the algorithms with such ability and that is one of the reasons behind selecting this classifier in this work. On the other hand, to the best of our knowledge, no work has studied the effect of feature representation and how robust a detection model can be over time for the aforementioned botnets.

3. METHODOLOGY

As discussed earlier, we employed two ML algorithms: C4.5 decision tree and the symbolic bid-based (SBB) framework for evolving teams of programs to detect botnet behaviour. Both of these learning algorithms generate solutions (models) that are in human readable format and therefore enable the analysis of the learned models.

Traffic features are extracted as flows using the Tranalyzer flow exporter [4]. Flow is defined as a logical equivalent for a call or a connection in association with a user specified group of elements [21]. The most common way to identify a traffic flow is to use a combination of five properties (aka 5-tuple) from the packet header, namely source/destination IP addresses and port numbers as well as the protocol. In this case, the features are derived from packet header information only. Hence, they can be employed for encrypted traffic or other conditions where the payload information is not accessible. Most of the known ML algorithms only accept attribute sets with numeric types. Therefore, almost all of the classification works in the literature employ specifically numeric flow features. Flow exporters however, have non-numeric representation for some of the features. In this category for instance, all of the eight TCP flags (which have binary values) are usually combined and presented as a hexadecimal TCP flag feature by the flow exporters. On the other hand, implementations of the classifiers usually (like the implementations in Weka) interpret this feature (Hexadecimal type) as a string or nominal value. Therefore, it is usually removed from the feature set. Having said this, TCP flags were shown to be used by malwares in ways that were not intended for legitimate use. In our previous work [13], we showed that TCP flags are employed by Torpig botnet for communication but not by the Conficker botnet. Therefore, in this paper, we first investigate the effect of three different TCP flag representations (i.e numerical, nominal and binary representations). In the numerical representation, we converted the hexadecimal TCP flag value into a numerical (integer) value whereas in the binary representation, the hexadecimal value is broken down into eight separate flag

values in binary format. These flags are Congestion Window Reduced (CWR), ECN-Echo (ECE), Urgent (URG), Acknowledgement (ACK), Push (PSH), Reset (RST), Synchronize (SYN) and Finish (FIN) flags. Finally, in the nominal representation, a set of hexadecimal flag values, utilized in a data set, is prepared and used as a nominal set of possible values for the TCP flag feature. Weka decision tree implementation accepts the nominal as a type for the features. Having a feature with the nominal type, the first string value is assigned index 0. This means that internally, this specific string value is stored as a 0 in the data set for the training/testing purposes. We used the same index-based interpretation for the features of this type in SBB.

Based on the fifth phase of the botnet lifecycle, botnets upgrade their methodology to defeat the detection systems. Hence, designing a botnet detection system that can cope with such changes is challenging. On one hand, using ML based detection systems for this purpose has the advantage of being able to re-train on the new botnet setting with minor human expert involvement. On the other hand, it is important to know how effective an older trained classifier can perform facing the same botnet with the new setting or behaviour. Therefore, in the second set of experiments, we investigate the performance of the two ML-based botnet detection systems over a period of time. To this end, seven Zeus botnet data sets are collected that are generated/captured over a period of four years.

3.1 Learning algorithms

3.1.1 C4.5

C4.5 is a decision tree algorithm which is an extension to the earlier ID3 algorithm developed by Quinlan [6]. This non-parametric supervised learning method aims to find the small decision trees (using pruning) and then generate an if-then rule set based of the trained tree which can be used for classification.

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

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

$$p_i = \frac{|C_{i,D}|}{|D|} \tag{1}$$

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

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

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

$$Entropy_A D = -\sum_{j=1}^{v} \frac{|D_j|}{D} \times Entropy(D_j) \tag{3}$$

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

$$Gain(A) = Entropy(D) - Entropy_A(D) \tag{4}$$

A decision node is created based on the selected splitting node with the highest information gain. The same procedure recursively applies to the corresponding sub-lists obtained by the splitting process until all of the data samples associated to the leaf nodes are of the same class or the classifier runs out of training samples. More detailed information on C4.5 learning algorithm can be found in [6].

3.1.2 SBB

The Symbiotic Bid-Based (SBB) algorithm is a form of linear genetic programming with a co-evolutionary architecture [18] which co-evolves three populations: A point population, a team population and a learner population, Fig. 1. The learner population represents a set of symbionts (learners), which associate a GP-bidding behaviour with an action. The team population comprises a set of learners and finally the point population denotes a subset of training data exemplars. Although all of the teams' learner programs are executed while evaluating a team on the points, only the learner with the highest bid suggest its action. The bidding procedure employs linear GP. To standardize the bid values between zero and one, the sigmoid function $f(y) = (1+exp(-y))^{-1}$ is applied to the real valued program output y.

In SBB, the point and the team population interaction follows a Pareto-based Competitive co-evolution. In this concept, 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, which survive to the next generation. At each generation, $Pgap$ new points are generated by sampling the training data and $Mgap$ new teams are generated through variation operators (add, delete, swap and mutate) applied to the existing teams, while learners in both parent teams are copied into the offspring. Following the fitness evaluation of all teams against all points, $PsizePgap$ points and $MsizeMgap$ teams are opted to appear in the next generation using a Pareto-based selection mechanism. Pareto competitive co-evolution ranks the teams' performances, 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 bias in favour of the points/teams that exhibit non-overlapping behaviour. Finally, all the generated teams in the learning procedure are evaluated on the training data set and the one with the best performance is selected as the final solution. 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

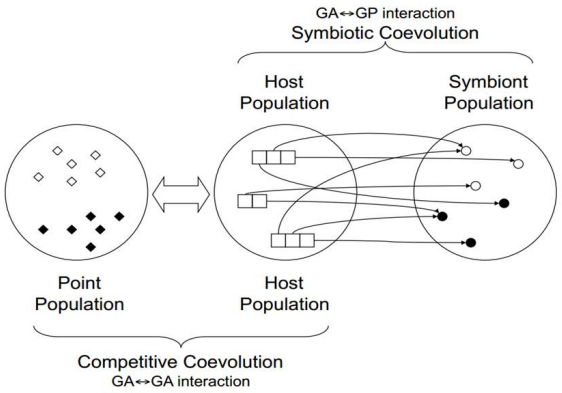

Figure 1: SBB team-based mechanism [18]

a linear GP [9], we employ intron removal to reduce the complexity of SBB [14]. A more detailed explanation of the algorithm can be found in [18].

3.2 Data set collection and Feature set extraction

There are various botnets that employ HTTP protocol as their communication protocol such as Zeus. These botnets can either have a P2P HTTP-based structure or a simple centralized/de-centralized structure. Zeus is a well-known botnet, which was taken down in 2012, but made a big came back with a new variant in 2013 [5]. 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. The earlier version of this botnet is reported to have a simple de-centralized structure while the latest version was reported as a P2P-based de-centralized botnet. In this work, we have collected seven different Zeus botnet data sets. The Zeus-1 (NIMS) data set is generated based on accessing the Zeus botnet C&C server domain names (using an HTTP-based communication). This data set is analyzed and compared against other Zeus botnet data sets in [15]. Since many works in the literature employed generated botnet traffic in a sandbox environment using the public botnet binaries and toolkits, we also run a Zeus botnet toolkit version 1.2.7.19 in a controlled sandbox environment and captured the traces [13] in November 2013. This toolkit is also analyzed and employed in [8]. We refer to this data set as Zeus-2 (NIMS). We set up 12 Zeus bots (infected machines with Zeus botnet) and two C&C servers (one Windows server and one Linux server) in the test bed. In June 2014, we used the same toolkit version to collect the Zeus-3 (NIMS) traffic traces. However, this time, we employed one windows C&C server with a different configuration (communication parameter settings are different)[1]. Moreover, Zeus toolkit version 2.1.0.1 was used to generate the Zeus-4 (NIMS) traffic traces in May 2014. In this configuration, there are 12 infected bots and one C&C server.

Moreover, there are several Zeus botnet traffic captures publicly available. NETRESEC [3] and Snort[2] [1] web sites has posted several Zeus traffic traces captured in February 2012 and February 2010, respectively, which are also used in this work. Hereafter, we refer to these two data sets as Zeus (Snort) and Zeus (NETRESEC). Last but not the least, Czech Technical University ATG Group has captured several botnet traces under the the malware capture facility project [11]. The CTU-Malware-Capture-Botnet-5 data set's probable name was announced "Zeus" back in August 2013. Hence, we also used this data set under the name of CVUT-5 in this work. To the best of our knowledge, these are all the publicly available Zeus botnet data sets in this field.

Since these seven data sets are purely malicious and the systems based on various data mining techniques require legitimate traffic for training purposes, we also employed Lawrence Berekeley National Laboratory (LBNL) 2005 data set which has been widely used in the literature to represent normal behaviour [2].

Tranalyzer was then employed to export the flow features (i.e. data features in this work) on these seven traffic traces. As a flow exporter, Tranalyzer aggregates the network packets into flows based on IP addresses, port numbers and the protocol using only the packet header information. Some statistics, such as the number of packets per flow, are then calculated as flow features [4].

4. EVALUATION AND RESULTS

As discussed earlier, we explore (i) the effect of different representations of the TCP flags and (ii) analyze the performance of a trained detection model over time.

4.1 Data set specifications

Tranalyzer extracts 93 features for each flow. We employed all of the features provided by the Tranalyzer as inputs to our machine learning classifiers except the IP addresses and port numbers because 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 TCP flag is one of the non-numeric features that we introduced using three different representation techniques. In summary, without using the TCP flag as part of the feature set, size of the feature set is 71. However, once the numerical, the nominal and the binary TCP flag representations are introduced, the feature set size changes to 72, 72 and 79, respectively.

After extracting the relevant feature set for each of the experiments, a balanced data set is formed by selecting randomly (uniform random selection) from the non-malicious (Lawrence) flow data set as well as from each of the malicious data sets. Table 1 shows the number of samples in each data set.

4.2 Performance metrics

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 multi-class data set, these metrics can be misleading. Therefore, a classwise detection rate [18] is defined as $DET_c = \frac{TP_c}{FN_c+TP_c}$ where $DETc$ is the class c detection rate and TPc and FNc are the True-Positive and False-Negative counts for class c, respectively. Finally, to summarize the classwise detection rates of a classifier over all classes, the average DR criteria is defined by [18]:

[1]The configuration settings and the data sets can be found at http://web.cs.dal.ca/~haddadi/data-analysis.htm
[2]"Sample_1" Zeus traffic file is used in this work.

Table 1: Number of flows in each data set employed

Data Set	Legit	Botnet
CVUT-5	1046254	1046254
Zeus-1 (NIMS)	43460	43460
Zeus-2 (NIMS)	1547	1547
Zeus-3 (NIMS)	40236	40236
Zeus-4 (NIMS)	10678	10678
Zeus (NETRESEC)	401	401
Zeus (Snort)	144	144

$$Score = \frac{1}{|C|} \sum_{c \in C} DET_c \quad (5)$$

4.2.1 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, two complexity criteria are utilized: **1) Training (computation) time**: This is estimated on a common computing platform. **2) Solution complexity**: A direct comparison between solutions from different representations is impractical since the underlying units of measurements 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 measurements.

4.3 Results

C4.5 classifier is run on each balanced data set using 10-fold cross-validation to further avoid any data set biases that might affect the results. However, SBB requires the training and testing data sets to be provided separately. Hence, we divided the data set into two parts (training and testing) based on an almost 30-70% breakdown for testing and training, respectively. It should be noted here that the default parameters in WEKA [24] are used for C4.5 (pruned) classifier, whereas parameters given in [19] are used for SBB (except the iteration value which was changed to 5000 for CVUT-5 data set due to the number of samples).

Comparing Table 2 with Tables 3, 4 and 5 on different representations of the TCP flags, no significant difference was observed in the C4.5 classifier results. Specifically, up to a 0.04% and 0.12% increase was shown in Score for the binary and the numerical representation, respectively, and a 0.13% decrease in Score was shown on the Zeus (NETRESEC) data set with the nominal representation. As for the results of the SBB classifier, no notable increase or decrease was observed in terms of Score. However, the solution complexity of runs without the TCP flag and the numerical representation were slightly better than the others. In conclusion, we did not observe any significant performance increase/decrease in the C4.5 and the SBB classifiers results in this case. Hence, we believe that adding the TCP flags in any form (representations) is not beneficial in designing a Zeus botnet detection system specifically but may result is some pre-processing overhead in some cases.

Table 2 shows the results of C4.5 and SBB classifiers on the data sets without the TCP flag. The results indicate that the Score, TP and FP rates are almost the same for both classifiers. Comparing the results over the complexity criteria, the SBB's solution complexity was less than the C4.5 for CVUT-5, Zeus-1, Zeus-3 and Zeus-4 data sets, almost the same for Zeus (Snort) and Zeus (NETRESEC) data sets and higher for Zeus-2 (NIMS). In general, we can conclude that SBB performed better in terms of solution complexity (obtained smaller solutions). This difference was more noticeable for bigger data sets such as Zeus-1 (NIMS) and CVUT-5. The lower solution complexity 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 rules/signatures. On the other hand, the C4.5 time complexity (for training) was mostly lower than the SBB except for the CVUT-5 big data set. Given that training is a one time off-line process, our observations indicate that SBB is the winner in terms of the complexity criteria among the two classifiers, where they performed almost the same in terms of Score (average detection rate) measurement.

In our second set of experiments, we investigated how effective the C4.5 and SBB trained models can perform facing newer/different versions of the same botnet behaviours. Zeus (NETRESEC) and Zeus (Snort) are small data sets. On the other hand, Zeus-1 (NIMS) data set is generated based on the C&C domain name list and therefore, is a good representative of the communication phase of the botnet lifecycle. Hence, we decided to use these as the test data sets whereas use Zeus-2 (NIMS) as the training data set. Table 6 shows the results of these experiments. As expected, both of the classifiers could detect the legitimate side of the data sets with high performance (TNRs of up to 100%) given that all of the data sets used the LBNL traffic traces for this purpose. Comparing the classifiers against the TPRs: (i) C4.5 out-performed SBB on Zeus-3 (NIMS), Zeus-4 (NIMS), Zeus (NETRESEC) and Zeus (Snort). In other words, the less complex solution of the C4.5 model on the Zeus-2 (see Table 2) could better detect other versions of the Zeus botnet. (ii) None of the classifiers performed well when tested on the CVUT-5 data set. In the CVUT-5 readme file, this data set's probable name is considered "Zeus". The readme file also suspected that this data set may be a P2P version of the Zeus botnet. Our experimental results confirms that CVUT-5 data set behaviour is not similar to the C&C HTTP-based version of the Zeus botnet by any means. Hence, it more likely represents a P2P HTTP-based Zeus botnet behaviour, if Zeus botnet at all. (iii) The results also indicate that the Zeus botnet behaviour presented by the Zeus-2 (NIMS) is different from the behaviour presented by the Zeus-1 (NIMS) data set. This might be caused by the fact that Zeus-1 (NIMS) is only a representative of one of the phases of the botnet lifecycle (C&C communication). Hence, we did another experiment with an HTTP filter for the experiments with low performance. Given that Zeus is an HTTP-based botnet, this filter only keeps the core botnet communication of the data set which was proved to increase the performance at [16]. Table 7 shows that the trained model on the Zeus-2 (NIMS) performed much better in Zeus-1 (NIMS) and Zeus (NETRESEC) detection but the performance did not change for CVUT-5 data set. These results indicate that even when different samples of the Zeus botnet do not seem to be similar (e.g. 32.73% TPR for detecting the Zeus (NETRESEC) sample with the trained model on Zeus-2 (NIMS) using the SBB classifier), the core

Table 2: Classification Results– No TCP flag

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
C4.5	CVUT-5	99.95%	100%	0.1%	99.9%	0%	2779.2	1185
	Zeus-1 (NIMS)	99.7%	99.7%	0.3%	99.7%	0.3%	39	477
	Zeus-2 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.24	9
	Zeus-3 (NIMS)	100%	100%	0%	100%	0%	19.72	43
	Zeus-4 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.28	41
	Zeus (NETRESEC)	97.5%	98%	3.0%	97.0%	2.0%	0.15	27
	Zeus (Snort)	100%	100%	0%	100%	0%	0.05	3
SBB	CVUT-5	98.06%	99.89%	3.7%	96.23%	0.1%	1280.79	56
	Zeus-1 (NIMS)	98.59%	97.41%	0.2%	99.79%	2.6%	205.282	36
	Zeus-2 (NIMS)	100%	100%	0%	100%	0%	217.272	33
	Zeus-3 (NIMS)	99.99%	99.98%	0%	100%	0.02%	261.93	24
	Zeus-4 (NIMS)	99.97%	99.94%	0%	100%	0.06%	360.287	36
	Zeus (NETRESEC)	99.17%	99.17%	0.8%	99.17%	0.8%	221.14	28
	Zeus (Snort)	100%	100%	0%	100%	0%	157.42	4

Table 3: Classification Results– Numerical representation

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
C4.5	CVUT-5	99.95%	100%	0.1%	99.9%	0%	2620.01	1199
	Zeus-1 (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	26.97	399
	Zeus-2 (NIMS)	99.87%	100%	0.3%	99.7%	0%	0.23	9
	Zeus-3 (NIMS)	100%	100%	0%	100%	0%	12.2	43
	Zeus-4 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.21	41
	Zeus (NETRESEC)	97.63%	98.0%	2.7%	97.3%	2.0%	0.15	25
	Zeus (Snort)	100%	100%	0%	100%	0%	0.06	5
SBB	CVUT-5	98.66%	99.29%	1.97%	98.03%	0.71%	1047.53	23
	Zeus-1 (NIMS)	98.58%	97.26%	0.1%	99.9%	2.73%	372.256	47
	Zeus-2 (NIMS)	100%	100%	0%	100%	0%	229.486	26
	Zeus-3 (NIMS)	99.99%	99.98%	0%	100%	0.2%	197.21	17
	Zeus-4 (NIMS)	99.98%	100%	0%	99.97%	0%	327.256	67
	Zeus (NETRESEC)	99.17%	98.33%	0%	100%	1.67%	378.048	74
	Zeus (Snort)	100%	100%	0%	100%	0%	147.017	2

Table 4: Classification Results– Nominal representation

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
C4.5	CVUT-5	100%	100%	0%	100%	0%	2876.16	1401
	Zeus-1 (NIMS)	99.75%	99.7%	0.2%	99.8%	0.3%	35.33	531
	Zeus-2 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.26	9
	Zeus-3 (NIMS)	99.95%	99.9%	0%	100%	0.1%	16.64	21
	Zeus-4 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.49	41
	Zeus (NETRESEC)	97.38%	98.0%	3.2%	96.8%	2.0%	0.15	27
	Zeus (Snort)	100%	100%	0%	100%	0%	0.06	3
SBB	CVUT-5	98.52%	9.78%	2.8%	97.25%	0.2%	1558.73	43
	Zeus-1 (NIMS)	98.75%	97.57%	0.06%	99.94%	2.4%	206.171	72
	Zeus-2 (NIMS)	100%	100%	0%	100%	0%	220.4	13
	Zeus-3 (NIMS)	99.99%	99.98%	0%	100%	0.02%	365.879	75
	Zeus-4 (NIMS)	99.97%	99.94%	0%	100%	0.06%	305.7607	30
	Zeus (NETRESEC)	98.75%	98.33%	0.8%	99.12%	1.7%	331.286	44
	Zeus (Snort)	100%	100%	0%	100%	0%	195.313	12

botnet communication behaviours are in fact similar (e.g. 83% TPR for the Zeus (NETRESEC)).

Overall, we observed that an older version of the Zeus botnet trained model can detect other versions of the same

Table 5: Classification Results– Binary representation

	Data Set	Score	Botnet		Legitimate		Complexity	
			TPR	FPR	TNR	FNR	Time (sec)	Solution
C4.5	CVUT-5	100%	100%	0%	100%	0%	3562.22	1239
	Zeus-1 (NIMS)	99.8%	99.8%	0.2%	99.8%	0.2%	29.78	411
	Zeus-2 (NIMS)	99.85%	100%	0.3%	99.7%	0%	0.25	9
	Zeus-3 (NIMS)	100%	100%	0%	100%	0%	15.92	45
	Zeus-4 (NIMS)	99.95%	99.9%	0%	100%	0.1%	2.28	41
	Zeus (NETRESEC)	97.26%	97.5%	3.0%	97.0%	2.5%	0.11	5
	Zeus (Snort)	100%	100%	0%	100%	0%	0.06	3
SBB	CVUT-5	98.46%	99.90%	2.99%	97.01%	0.01%	1231.87	60
	Zeus-1 (NIMS)	98.55%	97.42%	0.32%	99.68%	2.6%	303.291	39
	Zeus-2 (NIMS)	100%	100%	0%	100%	0%	198.039	25
	Zeus-3 (NIMS)	99.99%	99.98%	0%	100%	0.02%	286.251	41
	Zeus-4 (NIMS)	99.97%	99.94%	0%	100%	0.06%	331.573	14
	Zeus (NETRESEC)	99.17%	98.33%	0%	100%	1.67%	347.745	38
	Zeus (Snort)	100%	100%	0%	100%	0%	130.715	6

botnet with the same topology with up to a 100% TPR. Having said this, filters might be useful in order to increase the performance by focusing the analysis the core part of the botnet communication when necessary. Not only the results showed that the trained classifiers were robust enough to detect similar botnet behaviours, but they also showed that the models could be very good in pointing the bold changes of behaviour for a given botnet (i.e. showing that the CVUT-5 does indeed have a different botnet topology than the others).

5. CONCLUSIONS

Various forms of botnets, as a network of compromised hosts which are remotely controlled by a botmaster, have been active since 2003. With the ability of upgrading/ changing any phase of the lifecycle, botnets have defeated the detection systems and have made come backs after being taken down. Hence, detection systems require automatic and intelligent mechanisms to cope with the updates. In this work, we employed two machine learning algorithms, namely C4.5 and SBB, to generate botnet detection models for Zeus botnet. Seven Zeus botnet traffic traces were collected and represented as flows to the machine learning algorithms using the Tranalyzer flow exporter.

Our results show that both of the classifiers performed very well with the feature sret provided by Tranalyzer and obtained the Score (classwise detection rate) of up to 100%. Almost in all of our experiments, SBB performed better than C4.5 in terms of the solution complexity. This encourages the employment of SBB as the preferred classifier to improve automatic signature generation for botnet detection systems in practice. Moreover, given that non-numeric flow features can not be used by most of the machine learning algorithms, hence we investigated the effect of different representations of such flow features. To this end, TCP flag feature is analyzed. This is a set of eight flag features that are combined and presented as a hexadecimal feature. This value can be interpreted as a nominal value by a classifier such as C4.5. However, it is not usable by a classifier such as SBB. Hence, we investigated three different representations of the TCP flag feature, namely numerical, nominal and binary. Comparing the results of the experiments of the three representations with the experiments that did not employ this feature, we believe that using this feature in any form will not affect the performance of the detection system designed for Zeus botnet. Finally, we investigated the effectiveness of a ML-based detection system when it faced newer behaviours of the same type of a botnet. To this end, we collected different publically available Zeus botnet data sets and generated several data sets using publically available toolkits/binaries representing the evolution of Zeus botnet over a period of four years. Then, a Zeus botnet detection model was trained on the oldest data set which has enough samples and then tested against the other six Zeus data sets. The results indicate that the trained model can detect other versions of the Zeus botnet of the same topology with the Score of up to 100%. Furthermore, we could identify the change of topology for the P2P Zeus botnet by observing the changes in the performance. Future work will further investigate how to leverage such behaviour changes indicated by our classifiers.

6. ACKNOWLEDGMENTS

This research is supported by the Canadian Safety and Security Program(CSSP) E-Security grant. The CSSP is led by the Defense Research and Development Canada, Centre for Security Science (CSS) on behalf of the Government of Canada and its partners across all levels of government, response and emergency management organizations, non-governmental agencies, industry and academia.

7. REFERENCES

[1] https://labs.snort.org/papers/zeus.html.
[2] LBNL enterprise trace repository. http://www.icir.org/enterprise-tracing/.
[3] NETRESEC repository: publicly available pcap files. http://www.netresec.com/?page=PcapFiles.
[4] Tranalyzer. http://tranalyzer.com/.
[5] Zeus/zbot malware shapes up in 2013. http://blog.trendmicro.com/trendlabs-security-intelligence/zeuszbot-malware-shapes-up-in-2013/, May 2013.
[6] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.

Table 6: Classification Results– No TCP flag

	Data Set	Score	Botnet		Legitimate	
			TPR	FPR	TNR	FNR
C4.5	CVUT-5	49.79%	0%	0.4%	99.6%	100%
	Zeus-1 (NIMS)	54.79%	10%	0.4%	99.6%	90.0%
	Zeus-3 (NIMS)	99.79%	100%	0.4%	99.6%	0%
	Zeus-4 (NIMS)	99.53%	99.5%	0.4%	99.6%	0.5%
	Zeus (NETRESEC)	81.53%	63.3%	0.2%	99.8%	36.7%
	Zeus (Snort)	93.75%	88.2%	0.7%	99.3%	11.8%
SBB	CVUT-5	50.26%	0.54%	0.02%	99.98%	99.46%
	Zeus-1 (NIMS)	62.07%	24.14%	0%	100%	75.67%
	Zeus-3 (NIMS)	99.98%	99.97%	0%	100%	0.03%
	Zeus-4 (NIMS)	99.97%	99.93%	0%	100%	0.07%
	Zeus (NETRESEC)	86.66%	32.73%	0%	100%	26.68%
	Zeus (Snort)	100%	100%	0%	100%	0%

Table 7: Classification Results– No TCP flag, with HTTP filter

	Data Set	Score	Botnet		Legitimate	
			TPR	FPR	TNR	FNR
C4.5	CVUT-5	49.95%	0.1%	0.2%	99.8%	100%
	Zeus-1 (NIMS)	81.53%	63.3%	0.2%	99.8%	36.7%
	Zeus (NETRESEC)	84.62%	69.2%	0%	100%	30.8%
SBB	CVUT-5	50.26%	0.54%	0.02%	99.98%	99.46%
	Zeus-1 (NIMS)	91.28%	82.76%	0.19%	99.81%	17.24%
	Zeus (NETRESEC)	87.5%	75.0%	0%	100%	25.0%

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

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

[9] 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.

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

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

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

[13] F. Haddadi, D. Runkel, A. Zincir-Heywood, and M. Heywood. On botnet behaviour analysis using GP and C4.5. In *Gecco SecDef workshop*, 2014.

[14] 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.

[15] F. Haddadi and A. N. Zincir-Heywood. Data confirmation for botnet traffic analysis. In *FPS*, 2014.

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

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

[18] 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.

[19] P. Lichodzikewski and M. I. Heywood. Symbiosis complexification and simplicity under gp. In *GECCO*, 2010.

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

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

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

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

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