Data Confirmation for Botnet Traffic Analysis

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

Abstract

In this paper, we propose a systematic approach to generate botnet traffic. Given the lack of benchmarking botnet traffic data, we anticipate that such an endeavour will be beneficial to the research community. To this end, we employ the proposed approach to generate Zeus and Citadel botnet traffic as a case study. We analyze the characteristics of the generated data against the characteristics of NETRESEC and Snort captures publicly available in the field. This analysis confirms that our data is comparable to the data captured in the field in terms of the botnet behaviours represented.

Index Terms

Botnet traffic analysis, traffic data generation, data confirmation.

I. Introduction

In the world of fast growing Internet having a secure infrastructure is the primary need to protect users identity and information. Botnets- among various types of malware- are recognized as one of the main threats against cyber security due to their high reported infection rates and wide range of illegal activities [1]. A botnet is a set of compromised hosts that are under the remote control of a botmaster. After infecting a host, botmaster takes advantage of the victims' (bots) resources to carry out malicious activities without their knowledge. In 2012, Microsoft reported that some botnets had over 5 million infected hosts all over the world and managed to stole more than $500 million from their victims [2]. Unfortunately, even after the aggressive takedowns, these botnets managed to make comebacks and therefore such reports continue into 2013 [3].

Over time, various detection mechanisms have been proposed by researchers. In response to these detection approaches, botmasters have upgraded their bot programs and even have changed their methodology. That is why botnets evolved from using IRC protocol with centralized topologies to utilizing highly employed protocols such as HTTP with decentralized topologies. Along with the changes in protocols and topologies, botnets also used techniques such as fluxing and encryption to avoid detection. Therefore, identifying and detecting botnets have become very challenging and requires an active continuous monitoring and detection mechanism.

To this end, many systems rely on network traffic behaviour analysis to detect the botnets. Some focus on specific types of botnets while others attempt to build general models. However, in all cases, the first challenge, is to obtain realistic data that represents botnets traffic (behaviour). Due to the malicious nature of such data, there are very few publicly available data sets in the field. Therefore, different approaches have been explored by the researchers to obtain the necessary data for their research purposes. These approaches include (but not limited to): (i) Running botnet binaries (that are publicly available or modifying such binaries) in sandbox environments and capturing the traffic [4][5][6][7]; (ii) Obtaining captured Honeynet traffic [8][9]; or (iii) Obtaining traffic from a network operator or a security company [10].

The first one employs publicly available botnet binaries. This implies that any traffic obtained represents the old behaviours because only older versions of such binaries are publicly available. The second approach employs honeynets. Even though this method can catch newer botnet behaviours in traffic, it is also challenging to set up honeynets that can simulate real applications in normal behaviour. So there is no guarantee on the quality of the traffic attracted to the honeynet. The third approach is the only case where most up to date behaviours can be analyzed by obtaining data from network companies. However, this type of data has privacy issues so such data cannot be used by other researchers for further benchmarking and evaluation purposes. We anticipate that if a systematic approach can be found, then potentially publicly available benchmarking data sets can be generated without such problems. Thus, in this work, we propose a systematic approach to generate botnet traffic data representing the connection phase of botnet traffic. In this phase, the infected host intends to locate the C&C server and starts a connection with the server. Once we generate such data using the proposed approach, we demonstrate that such an approach could indeed generate realistic behaviours of the botnet traffic. To this end, we evaluated our approach and confirm that the data generated represent similar behaviour to the botnet traffic that is captured in the wild and the botnet traffic that is generated in a sandbox environment based on publicly available bot binaries. To the best of our knowledge, this is the first

This is an extended version of the paper "Data Confirmation for Botnet Traffic Analysis" published in the 7th international Symposium of Foundations Practice of Security (FPS), 2014. If you would like to use the NIMS data sets, please cite this paper.

work with such a goal. To achieve our goal, we follow two steps to evaluate the data after it is generated: (i) Visualization of the flow measurements of the generated traffic vs the flow measurements of the traffic captured in the wild and the controlled environment; and (ii) Employing a decision tree based learning classifier (rule-based) for post-classification analysis of both the generated and the collected traffic.

In the first step, we compare the data sets based on the key features of the traffic. These features are highly employed in the literature for botnet behaviour analysis [4][6][7][9][11]. To extract these features, we utilize IP-flow technology by which the network traffic is summarized into flow features. Specifically, we employ Softflowd [12], which is an open source tool based on the NetFlow standard [13]. In the second step, we employ C4.5 learning technique [14], which is a decision tree based (rule-based) classifier, for post-classification analysis. By doing so, we want to evaluate how well a classifier trained on the generated traffic would be able to identify the botnet traffic in the wild (real life). The hypothesis is that if it can, then the traffic generated is able to represent the behaviours seen in real life.

II. Related Work

Over time, botnets; as one of the most serious threats to cyber-security; has employed different protocols, topologies and techniques to avoid the detection systems. Gu et al. developed BotMiner based on group behaviour analysis to detect botnets [8]. To evaluate the proposed system, Gu et al. collected the botnet data sets by changing the publicly available botnet code to generate traffic as well as using their Honeynet data sets. As for the normal traffic, university campus network traces were employed. They represented the traffic using their own tools to employ features such as the number of packets per flow, the average number of bytes per packet, and the average number of bytes per second. Wurzinger et al. proposed an approach to detect botnets by correlating the commands and responses in the C&C channels of the monitored network traces [4]. Running a publicly available bot binary in a controlled environment, they collected the data sets used in this work. Traffic features such as the number of packets and the cumulative size of packets (in bytes) were analyzed to characterize bot behaviour. While the two previous framework can detect IRC-based, and P2P botnets, Kirubavathi et al. designed specifically an HTTP-based botnet detection system using a multilayer Feed-Forward Neural Network [5]. They extracted features related to TCP connections in specific time intervals and used them for the detection purposes. To evaluate the proposed system, botnets were simulated in the lab to collect data sets. Strayer et al. developed an IRC botnet detection framework that makes use of classification and clustering techniques [6]. Flow-based attributes employed were similar to the NetFlow features (e.g. Bytes-per-packet, Bytes-per-second and Packets-per-second). Similar to the works discussed earlier, they captured their data on a controlled testbed running bot programs. Francois et al. proposed a NetFlow monitoring framework that leverages a simple host dependency model to track communication patterns and employed linkage analysis and clustering techniques to identify similar botnet behavioral patterns [10]. They obtained traffic traces from an Internet operator company. Zhao et al. investigated a botnet detection approach based on flow intervals [9]. Features such as the number of packets exchanged and the average payload packet length were employed to detect botnets. A combination of malicious and non-malicious traffic traces was employed. Malicious traces were in part generated in their lab and in part obtained from the Honeynet project whereas the legitimate traces were obtained from Lawrence Berkeley National Laboratory data sets. Haddadi et al. designed a machine learning based detection system [11]. Network traces representing malicious and non-malicious behaviour were generated in the lab based on botnet and legitimate public data and Netflow based feature extraction method was used to extract the features (e.g. flow duration and number of packets and bytes per flow).

In summary, botnet traffic traces employed in botnet traffic analysis and detection systems seem to be a combination of captured traces from running botnet binaries (that are publicly available or modifying such binaries) in sandbox environments, Honeynet projects and a network operator or a security company. Indeed, the last two resources are difficult to achieve unless some collaboration is possible with such organizations, even then traces are not made public. As for the first approach, most of the publicly available botnet binaries are older examples of the malwares so they may not be representative of the current botnet behavior.

III. Methodology

A typical advanced botnet shapes in five phases (i.e. lifecycle): (1) the initial infection phase where exploitation techniques are used to find victim vulnerabilities and infects the target host, (2) the secondary infection phase where the botnet shell-code is executed on the victim machine to fetch the image of the bot binary which then installs itself on the machine, (3) the connection phase where the bot binary establishes the C&C channel, (4) the malicious C&C phase in which the established C&C channel is utilized by the bot master to send the command and (5) the update and maintenance phase where the botmaster updates the bots when required. It is known that to enhance their functionality and to avoid detection systems, botnets tend to use automatic algorithms in different phases of their lifecycle. For example, they use automatic exploitation techniques instead of fixed vulnerability checklists to find security breaches in the victim's computer. They also employ fluxing techniques, using automatic Domain Generation Algorithms (DGAs), to hide the C&C servers. Citadel and Zeus are just two of the recent examples of such botnets.

A DGA is designed to periodically generate a large list of domain names (e.g. Conficker.C's DGA generate 50,000 domain names per day). The infected computer (bot) may locate the C&C server by querying (sending a DNS query to retrieve the associated IP address) a list of domain names. However, only one of the domain names from the list is valid (has a valid IP address assigned to it) at a given time. Once a response is received for the query, the bot establishes a connection to the C&C server to receive updates and commands. The list of domain names provided to the bot is large enough so that blacklisting (blocking) it manually or at the firewall level is not straight forward. Therefore, the domain fluxing technique is highly utilized in current botnets. Since there are publically available lists of domain names generated by botnet DGAs, we implemented a systematic approach to generate network traffic using these botnet domain names to represent the botnet communication during the connection phase of the botnet lifecycle.

Once the botnet communication traffic is generated, we analyze and evaluate it against three publicly available botnet traffic data that are captured in real life (wild) and one botnet binary based traffic data captured in the sandbox environment. In doing so, our aim is to confirm that our generated traffic is representative of the publicly available ones and the sandbox ones. To this end, we have used a 2-step process: (i) Visualization of flow measurements; and (ii) Post-classification analysis of C4.5 decision tree models. For the first step, frequency of three main flow features (flow duration, number of packets and number of bytes per flow), which are mostly employed in the literature, are visualized and compared between the data sets. For the second step, a decision tree is trained on the generated data (by the proposed approach) and tested on the publicly available data (captured in the wild) and the sandbox data (captured in a controlled environment) to analyze how accurately the trained model (using the proposed traffic data generation approach) can identify botnet behaviour in the traffic data captured in the wild and sandbox environments.

A. Proposed Approach to Generate Traffic

We generate the representative botnet traffic flows using the publicly available (from legitimate resources) C&C domain name lists. As for generating the representative legitimate (normal) traffic, we use a legitimate domain name list. This ensures that both the attack and the normal traffic are generated using the publically available domain name lists to enable re-engineering of the approach for benchmarking purposes.

For the malicious domain name lists, the two of the most recent botnets, namely Citadel and Zeus are used. As for the legitimate (normal) name list, some of the most frequently requested domain names from Alexa lists (domain names of the high ranked web sites) are used [15]. In other words, both the malicious and legitimate behaviour data use the HTTP protocol as their communication protocol. So, to generate the traffic, we developed a program to establish HTTP connections with the domain names from the aforementioned lists. First, this program sends out DNS queries for the domain names in the lists. If in return it receives a proper DNS response, this indicates that the domain name is registered and is associated with a valid IP address. Then, our program attempts to establish an HTTP connection with that IP address. The following summarizes the lists used in this work:

1) Alexa: Alexa Internet Inc. [15] ranks the websites based on their page views and unique site users then publishes the list of the most popular websites according to these results.

2) Zeus: This botnet is famous for stealing banking information by using the man-in-the-browser (MITB) keystroke logging and the form grabbing techniques. However, it can be configured for any type of identity theft attack. Reports are indicating that this botnet have stolen more than $100 million [16]. Although this botnet was taken down in 2012 by Microsoft and its partners, there are security reports from TrendMicro in 2013 indicating that Zeus botnet is back with a new variant [17]. Therefore, this botnet is identified as one of the most destructive botnets in the recent history. Hence, ZeusTracker actively monitors the Zeus botnet [18]. To create our data, we downloaded the domain name lists from ZeusTracker and DNS-BH project websites [18][19].

3) Citadel: Citadel botnet came to existence after the Zeus botnet leaked in 2011. This botnet is the improved version of Zeus where the Zeus bugs are fixed and it is adapted to the newest security platforms such as adding extra layer of protection from trackers [20]. It is believed that Citadel stole more than $500 million and infected more than 5 million PCs in different countries [2]. In June 2013, Microsoft and FBI took down almost 90% of the Citadel botnet based on a court order in an operation that is described as the "most aggressive botnet operation to date" [21]. Yet, there are news of Citadel making a bold comeback [22]. Therefore, the section of ZeusTracker on Citadel botnet and the section of DNS-BH project track the new Citadel botnet that is active after the take down operation. We obtained our Citadel domain name list from these two websites.

There are a few botnet traffic data sets available at NETRESEC [23] and Snort [24] web sites for Zeus and Citadel botnets. This traffic data that is captured in the wild, is not enough to be used as the only data set(s) in the evaluation of botnet traffic analysis (detection) systems. However, in this work we take advantage of these publicly available traffic data to evaluate the traffic that we generated using our proposed approach. On the other hand, since many works in the literature employed simulated botnet traffic using the public botnet binaries and toolkits, we also simulate a Zeus botnet in a controlled environment (using a public Zeus kit) and captured the traces. This toolkit is also analyzed and employed in [25]. However, we could not do the same for the Citadel botnet given that there is no publicly available free kit, to the best of our knowledge.

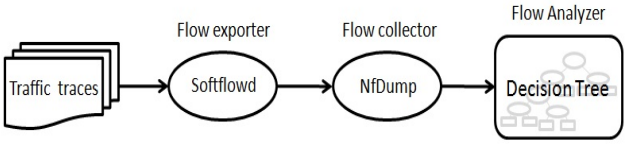


Fig. 1: Softflowd exporter mechanism

TABLE I: Number of domain names in the data sets

data set	No. of Domain names
Alexa (NIMS)	500
Citadel (NIMS)	42
Zeus-1 (NIMS)	684

B. Flow Generation

Flow generation tools summarize the network traffic by utilizing the network packet headers. These tools collect packets that have common characteristics such as IP addresses, port numbers and the protocol over a certain period, aggregate them into flows then calculate some statistics such as the number of packets per flow etc. In RFC 2722, a traffic flow is defined as a logical equivalent for a call or a connection in association with a user specified group of elements. The most common way to identify an IP flow is to use a combination of five properties from the Network and the Transport layer headers: Source IP address, Destination IP address, Source Port Number, Destination Port Number and Protocol. Hereafter, we will refer to this as 5-tuple information.

Given that Cisco Systems, as the leader of IP flow technology, introduced NetFlow to collect and aggregate IP traffic information, soon NetFlow became the industry standard. Over time, different versions of NetFlow were developed and the very recent version (V. 10) is now the IETF standard for traffic information export as IPFIX (RFC 5101). To collect and analyze NetFlow traffic flow data, three network elements work together: (i) NetFlow exporter which generates the NetFlow data; (ii) NetFlow collector which collects NetFlow data from the exporter; and (iii) NetFlow analyzer which analyzes the collected data. A variety of network devices (e.g. routers and switches) from different vendors support different versions of Netflow. In this work, we employ Softflowd as an open source NetFlow exporter which can export NetFlow data using the traffic on a simple device interface or even export NetFlow data from a pre-captured traffic trace [12]. Then, we employ NfDump as the collector because it is open source, fast, easy to use and supports different versions of NetFlow (v5, v7 and v9) [26]. Finally, for the analyzer, we employed C4.5 decision tree classifier for post-classification analysis, Fig. 1.

C. Machine Learning Algorithm- 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. This algorithm constructs decision trees based on a training data set, where each exemplar is a set of already classified (labeled) attributes by applying the Information Entropy concept. The algorithm employs normalized Information Gain criterion to select attributes from a given set of attributes to determine the splitting point for the decision tree. The attribute with the highest Information Gain value is chosen as the splitting point. A more detailed information on C4.5 learning algorithm can be found in [14].

IV. Evaluation

As discussed earlier, as a case study, we focus on generating Zeus and Citadel traffic data for botnet traffic analysis. Hereafter, we will refer to our generated data sets as NIMS botnet data sets.

A. Data Sets

Given that even Alexa lists might have malicious domain names [27], we manually extracted 500 benign domain names from Alexa lists for the data sets employed in this work. As discussed earlier, Zeus and Citadel domain name lists was obtained from ZeusTracker and DNS-BH project blocklists [18][19]. Table I shows the number of domain names employed in this work to generate the traffic. All generated traffic was captured while our program was attempting to establish connections with the domain names and no sampling rate was used. Hereafter, the Zeus traces that was generated using this approach is called Zeus-1.

Furthermore, using a publicly available Zeus toolkit [25], we simulated a Zeus botnet in a sandbox environment where 12 bots and 2 C&C servers (one Windows server and one Linux server) were implemented using the Zeus binaries available [7].

TABLE II: Data specification

Data set	No. of Packets	No. of Exported Flows
Alexa (NIMS)	24938	7473
Citadel (NIMS)	79516	5772
Citadel (NETRESEC)	15239	226
Zeus-1 (NIMS)	108947	14884
Zeus (NETRESEC)	7453	361
Zeus (Snort)	6995	145
Zeus-2 (NIMS)	32818	874

TABLE III: Softflowd feature set definition

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

Hence, the botnet simulation was run and the generated traffic was captured for 2 hours. Henceforth, this simulated Zeus data is referred to as Zeus-2. Once, the botnet data was generated / collected, Softflowd flow exporter is run on the traffic to extract the flow features. Table II presents the number of captured packets and the corresponding extracted flows for each of the data sets employed in this work.

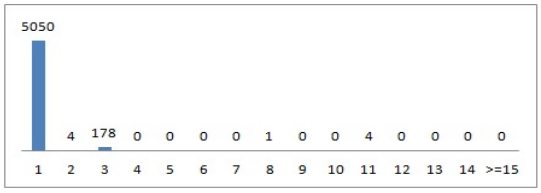
It should be noted here that Softflowd provides 41 features in total per flow and in this work, we only employ 23 of these. In the second step of our analysis, where we employ C4.5 decision tree classifier for post-classification analysis, these 23 features are used as inputs. The ones that are not used are: IP addresses, port numbers and any non-numeric features. The reasons behind this are: IP addresses can be spoofed whereas port numbers can be assigned dynamically. The presentation of non-numeric features may introduce other biases, because they require interpretation (a priori knowledge), which may change depending on the focus of the detection system in use. Thus, employing such features may decrease the generalization abilities of the analysis and detection systems for which such data is generated. Table III presents the features that are utilized in our proposed approach. Detailed definition of all the features can be found in NfDump project web site [26].

B. First Step for Evaluating the Data Generated – Visualization of Flow Characteristics

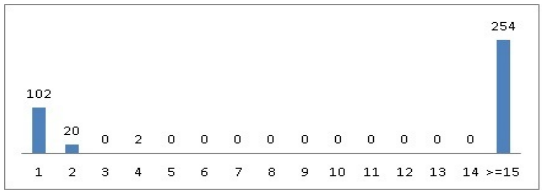
As a first step, we anticipate that it will be beneficial to analyze the data sets on some of the most important features (used by other researchers [4][6][9][7]) of a flow to understand the data sets better.

As discussed earlier, 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 [28]. Citadel and Zeus fall under this category, too. They utilize HTTP protocol to communicate with their bots. To analyze and compare the aforementioned data sets, we filter out non-HTTP flows from our data sets. This way all the background information is removed from the data and therefore, the data sets can be compared on their fundamental properties.

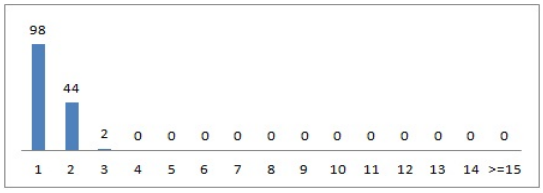
Fig. 2 shows the frequency graph of the durations of flows for Zeus botnet data sets. As shown in the figure, the majority of flows durations (96%, 70%, 91% in Zeus (NIMS), Zeus (Snort) and Zeus (NETRESEC), respectively) are less than 50 seconds long (placed in the first bucket). However, 20% of Zeus (Snort) and 67% of Zeus-2 flows last longer than the other two Zeus data sets. Fig. 3 and Fig. 4 show that Zeus-2 (NIMS), Zeus (Snort) and Zeus (NETRESEC) have more flows with higher number of transmitted packets (30 packets or more) and more bytes, respectively. The main cause of this type of differences is



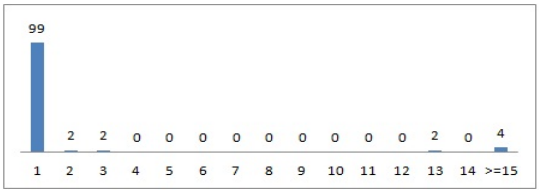
(a) Zeus-1 (NIMS)



(b) Zeus-2 (NIMS)

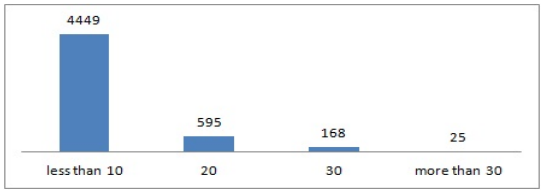


(c) Zeus (Snort)

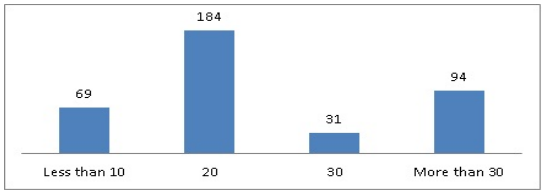


(d) Zeus (NETRESEC)

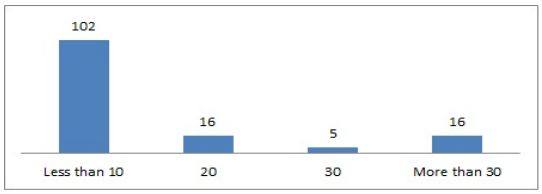
Fig. 2: The frequencies of durations of the flows in Zeus data sets. X-axis denotes the buckets of 10 seconds.



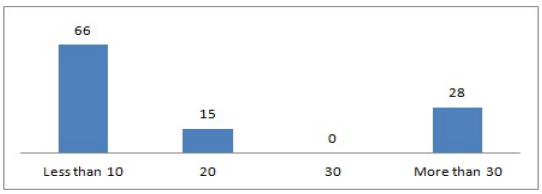
(a) Zeus-1 (NIMS)



(b) Zeus-2 (NIMS)



(c) Zeus (Snort)



(d) Zeus (NETRESEC)

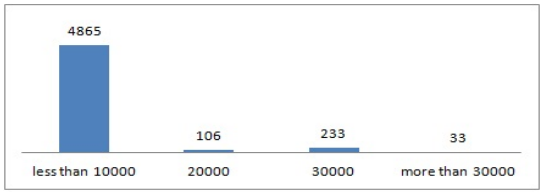
Fig. 3: The frequencies of number of packets per flow in Zeus data sets. X-axis denotes the number of packets per flow.

that we do not keep the connection between our system and botnet C&C servers open long because we close the connection once this phase of communication is finished.

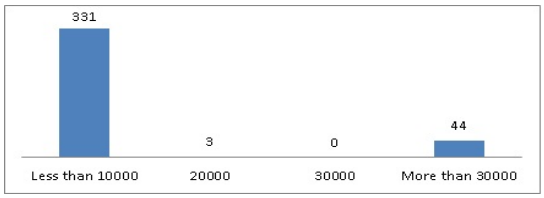
As Fig. 5 shows that the pattern for duration of flows for the Citadel (NIMS) botnet, is different from the pattern of the duration of the flows for Citadel (NETRESEC). On the other hand, Fig.6 and Fig.7 show that there is almost no difference (among the two data sets) between the number of packets and the number of bytes transmitted per flow. Therefore, we can conclude that the Citadel C&C connections in NETRESEC data were kept open without any actual client-server command and response communication while in our data generation process the connections were terminated after the HTTP communication was finished. Since keeping a connection open for a long time is one of the signs of malicious activities, botnets tend to close the connection when the necessary information is sent/received and open a new connection for the next round when needed. However, high number of connections also raises flags for malicious behaviour. Therefore, a trade-off between these two parameters is necessary for the botnets to hide their malicious activity. To this end, we decided to keep the program we developed for communicating with the C&C servers the same, because it gives us a good balance for the aforementioned trade-offs. In summary, the analysis of the important features of the flows generated and the flows reported in the field (public traffic traces) seem to be very similar except the durations of the flows, which are shorter in our traffic generation approach.

C. Second Step for Evaluating the Data Generated – Post-Classification Analysis by Using a Decision Tree Classifier

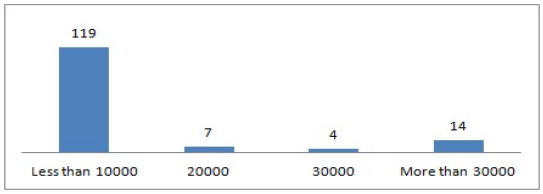
To further investigate how similar the generated data (by our proposed approach) is to the data captured in the wild and the simulated data, we employed a post-classification analysis using a decision tree. To this end, we trained a C4.5 classifier on the NIMS data and tested the trained model on the botnet data from the NETRESEC and Snort web sites (captured in the



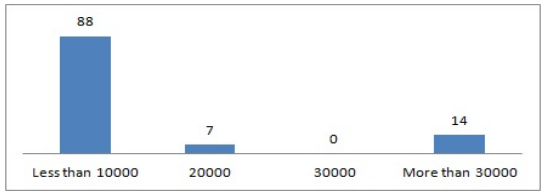
(a) Zeus-1 (NIMS)



(b) Zeus-2 (NIMS)

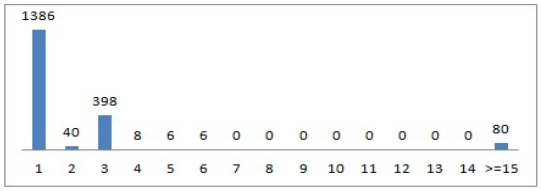


(c) Zeus (Snort)

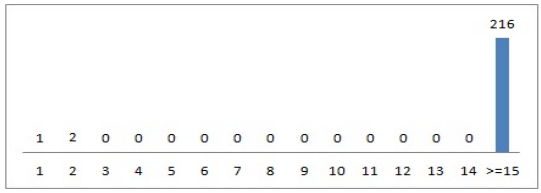


(d) Zeus (NETRESEC)

Fig. 4: The frequencies of number of bytes per flow in Zeus data sets. X-axis denotes the number of bytes per flow.

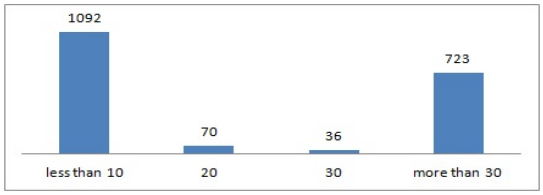


(a) Citadel (NIMS)

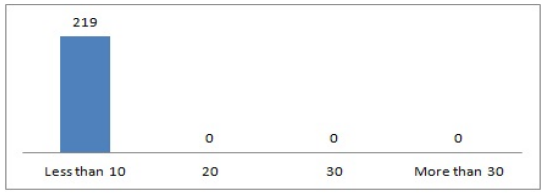


(b) Citadel (NETRESEC)

Fig. 5: The frequencies of durations of the flows in Citadel data sets. X-axis denotes the buckets of 10 seconds.

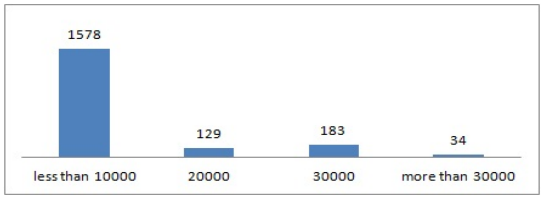


(a) Citadel (NIMS)

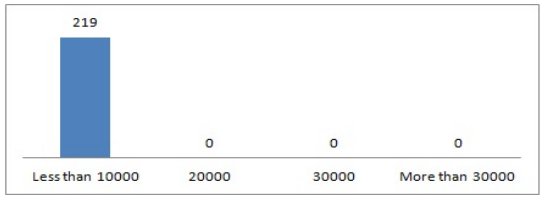


(b) Citadel (NETRESEC)

Fig. 6: The frequencies of number of packets per flow in Citadel data sets. X-axis denotes the number of packets per flow.



(a) Citadel (NIMS)



(b) Citadel (NETRESEC)

Fig. 7: The frequencies of number of bytes per flow in Citadel data sets. X-axis denotes the number of bytes per flow.

field) and the simulated NIMS-2 data in our testbed. Table IV presents the results of this step.

In traffic classification, two metrics are typically used in order to quantify the performance of the classifier: Detection Rate (DR) and False Positive Rate (FPR). In this case, DR reflects the number of botnet flows correctly classified and is calculated using: $DR = TP/(TP + FN)$; whereas FPR reflects the number of legitimate (normal) flows incorrectly classified as botnet flows. FPR is calculated using: $FPR = FP/(FP+TN)$. Naturally, a high DR and a low FPR are the most desirable outcomes.

Training data set	Testing data set	DR	Botnet		Legitimate	
			TPR	FPR	TNR	FNR
Citadel (NIMS)	Citadel (NETRESEC)	100%	100%	0%	0%	0%
Zeus-1 (NIMS)	Zeus (Snort)	81%	81%	0%	0%	19%
Zeus-1 (NIMS)	Zeus (NIMS)	79%	79%	0%	0%	21%
Zeus-1 (NIMS)	Zeus-2 (NIMS)	88%	88%	0%	0%	12%
Zeus-1 (NIMS)	Zeus-1 (NIMS)	84%	84%	0%	0%	16%

TABLE IV: C4.5 decision tree classification results

On the other hand, False Negative Rate (FNR) indicates that botnet flows are classified as legitimate flows, whereas TNR, True Negative Rate, indicates the correctly classified legitimate flows. As presented in table IV, the trained model on NIMS Citadel could detect all of the NETRESEC Citadel exemplars of the test data (100% TP rate). This observation shows how similar the two data sets behave in terms of the main features that the C4.5 classifier employed to create the training model.

Our post-classification analysis show that the main features that the C4.5 decision tree classifier employ for the Citadel trained model are: Duration, Total-Pkt, Total-Byte, bps, pps and Bpp (from Table III). Moreover, in the first step of our analysis, duration feature of a flow presented the most discrepancy between the NIMS botnet data set and the one captured in the wild. However, the classification results indicate that this discrepancy does not seem to have a negative effect on the identification of Citadel botnet behaviour in a given traffic trace. This might be because of the importance of the other features employed. This implies that NIMS Citadel data set is similar to the real-life NETRESEC Citadel data set. Therefore, we can confirm that the proposed approach seems to generate realistic Citadel botnet traffic.

Table IV also presents the classification results on the Zeus botnet traffic. Again, we trained a C4.5 decision tree classifier on Zeus-1 (NIMS) data set and tested the trained model against the Zeus-2 (NIMS) and Zeus data sets from Snort and NETRESEC. As the results indicate, the DRs of the Zeus-2, Snort and NETRESEC data sets are from 79% to 88% which are less than the DR for Citadel, but still a promising performance[2]. However, in our previous work [11] where we trained and tested a C4.5 classifier solely on Zeus-1 (NIMS) data set, we could push the performance of the classifier up to 86% DR. This indicates that Zeus botnet behaviour is more complicated than Citadel botnet behaviour and therefore seems to hide in the legitimate HTTP behaviour very well. This makes it more difficult to differentiate Zeus traffic from the legitimate traffic, hence the lower DR (compared to Citadel's case).

To analyze if this lower DR is caused by the discrepancies (indicated in IV-B) between our Zeus-1 (NIMS) traffic captures vs. the other Zeus traffic captures, we run another experiment. Since NIMS Zeus generated traffic was bigger than Alexa legitimate data (and therefore had more flows), we only used a fraction of the Zeus HTTP traffic to build a balanced Zeus-Alexa training data set for the second step of our analysis. Hence, we had some unseen Zeus flows that were not included in the training process. Thus, for further analysis of Zeus generated traffic, we tested the trained Zeus model on these unseen generated Zeus flows. As the result shows, we obtained almost the same DR (84%), Table IV. This seems to indicate the complicated behaviour of Zeus botnet and confirm that the results are not a side effect of generated data, but rather the classifier performs the same on both the generated data and the data captured in the wild for Zeus botnet.

In short, our 2-step analysis indicates that the traffic generated using our proposed approach is valid and comparable to the botnet traffic captured in real life (Snort and NETRESEC data) as well as the traffic captured in a sandbox environment (Zeus-2 NIMS). In other words, traffic generated based on the proposed approach can be employed in botnet behaviour analysis as representing real data. We provide our NIMS data sets for further benchmarking purposes at the following URL:

https://web.cs.dal.ca/~haddadi/data-analysis.htm

V. Conclusion

Due to the high reported botnet infection rate and its wide range of illegal activities, botnets are one of the main threats against the cyber security. Not only some of the most aggressive botnets are developed and utilized by the malware experts recently, but also web-based reusable botnet kits, such as Zeus toolkit, are available for sale. Therefore, many researchers have focused on botnet traffic analysis and proposed various botnet detection systems. However, one problem of this field is the availability of public data sets for benchmarking purposes. Lack of such data sets makes it challenging to have meaningful benchmarks and comparisons among the botnet identification systems.

In this work, we propose a systematic approach to generate botnet traffic and analyze its properties against that of available botnet traffic captures in the field. To this end, we explore a 2-step analysis. In the first step, we analyzed the data sets using the visualization of the main features of the botnet traffic reported in the literature. To extract these features from NIMS, NETRESEC and Snort botnet traffic captures, we employ Softflowd. The results of this step show that the generated NIMS

[2]Since the test data sets are one class data sets (only malicious), there is no TNR and FPR.

traffic data and the public NETRESEC and Snort traffic data analyzed are not very different in terms of their main features (measurements). However, the sandbox traffic data has noticeable differences in terms of most of the measured features.

In the second step, we perform a post-classification analysis on the data sets using C4.5 decision tree classifier. In this step, we train the C4.5 classifier with our generated data, NIMS data, and test the trained model using the publicly available data from NETRESEC and Snort and the sandbox data obtained using the public toolkit. The results of the second step indicate that the trained model could classify the unseen data captured in the wild and the simulated data with high accuracy (high DR and low FPR). This suggests that the botnet data generated by the proposed approach confirms to the botnet data captured in the wild or in the sandbox. In other words, the proposed approach for generating botnet traffic can represent real-life botnet traffic and therefore can be employed in botnet traffic analysis. We anticipate that this can further enable benchmarking and evaluation efforts in this research area. Future work will explore employing the proposed traffic generation approach for other botnets as well as will include the study of other feature extraction methods.

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/.

REFERENCES

[1] M. Feily and A. Shahrestani, "A Survey of Botnet and Botnet Detection. Emerging Security Information," in Systems and Technologies, pp. 268-273, 2009.
[2] Citadel defences breached (2013, June) [Online]. Available: http://www.symantec.com/connect/blogs/citadel-s-defenses-breached
[3] Citadel makes a comeback, targets Japan users (2013, September) [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/ citadel-makes-a-comeback-targets-japan-users/
[4] P. Wurzinger, L. Bilge, Th. Holz, J. Goebel, Ch. Kruegel, and E. Kirda, "Automatically generating models for botnet detection," in 14th European conference on research in computer security, pp. 232-249, 2009.
[5] V. Kirubavathi and R.A. Nadarajan, "HTTP botnet detection using adaptive learning rate multilayer feed-forward neural network," in Information Security Theory and Practice: security, privacy and trust in computing systems and ambient intelligent ecosystems, pp. 38-48, 2012.
[6] W.T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," *Advances in Information Security*, vol. 36, pp. 1-24, 2008.
[7] F. Haddadi, D. Runkel, A Nur Zincir-Heywood, and Malcolm I. Heywood, "On Botnet Behaviour Analysis using GP and C4.5", in Gecco Comp, 2014.
[8] 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, pp. 139-154, 2008.
[9] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu, "Peer-to-peer botnet detection based on flow intervals," in IFIP international information security and privacy, pp. 87-102, 2012.
[10] J. Francois, Sh. Wang, R. State, and Th. Engel, "BotTrack: tracking botnets using Netflow and PageRank," *Networking*, vol. 6640, pp. 1-14, 2011.
[11] F. Haddadi, J. Morgan, E. G. Filho, and A. Nur Zincir-Heywood, "Botnet behaviour analysis using IP flows With HTTP filters using classifiers", in Seventh International Workshop on Bio and Intelligent Computing, 2014.
[12] Softflowd project [Online]. Available: http://www.mindrot.org/projects/softflowd/
[13] Cisco IOS NetFlow. [Online]. Available: http://www.cisco.com/en/US/ products/ps6601/products_ios_protocol_group_home.html
[14] E. Alpaydin, Introduction to Machine Learning.: MIT Press, 2004.
[15] Alexa. [Online]. Available: http://www.alexa.com/topsites
[16] Microsoft wins court order in 'Zeus Botnet' lawsuit (2012, May) [Online]. Available: http://www.bloomberg.com/news/2012-03-29/microsoft-wins-court-order-in-zeus-botnets-lawsuit.html
[17] Zeus/ZBot Malware Shapes up in 2013 (2013, May) [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/zeuszbot-malware-shapes-up-in-2013/
[18] Zeus Tracker. [Online]. Available: https://zeustracker.abuse.ch/
[19] DNS-BH- Malware Domain Blocklist. [Online]. Available: http://www.malwaredomains.com/
[20] Citadel Zeus bot. [Online]. https://www.botnets.fr/index.php/Citadel _ZeuS_bot
[21] Citadel News (2013, June) [Online]. Available: http://blogs.technet.com/b/microsoft_blog/archive/2013/06/05/microsoft-works-with-financial-services-industry-leaders-law-enforcement-and-others-to-disrupt-massive-financial-cybercrime-ring.aspx
[22] Citadel makes a comeback, targets Japan users (2013, September) [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/ citadel-makes-a-comeback-targets-japan-users/
[23] Publicly available PCAP files. [Online]. Available: http://www.netresec.com/?page=PcapFiles
[24] Zeus Trojan Analysis. [Online]. Available: https://labs.snort.org/papers/zeus.html
[25] 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, pp. 31-38, 2010.
[26] NfDump [Online]. Available: http://nfdump.sourceforge.net/
[27] Paul Royal. Maliciousness in Top-ranked Alexa Domains. [Online]. Available: https://www.barracudanetworks.com/blogs/labsblog?bid=2438
[28] HTTP-Botnets: the dark side of an standard protocol (2013, April) [Online]. Available: http://securityaffairs.co/wordpress/13747/cyber-crime/http-botnets-the-dark-side-of-an-standard-protocol.html