TorBot Stalker: Detecting Tor Botnets through Intelligent Circuit Data Analysis

Oluwatobi Fajana
School of Computing
University of Portsmouth
United Kingdom
oluwatobi.fajana@port.ac.uk

Gareth Owenson
School of Computing
University of Portsmouth
United Kingdom
gareth.owenson@port.ac.uk

Mihaela Cocea
School of Computing
University of Portsmouth
United Kingdom
mihaela.cocea@port.ac.uk

Abstract—**Botnets are collections of infected computers that are controlled centrally by a botmaster, often for sending spam or launching denial of service attacks. The task to take down these botnets is often a cat and mouse game with operators frequently changing domains for their control infrastructure. More recently, operators have moved to using Tor, a pseudo-anonymous network for hosting services whereby identification is difficult. Additionally, because connections to the Tor network are encrypted, we cannot use traditional methods like Domain Name System (DNS) and traffic signatures to detect infected hosts. In this paper, we introduce TorBot Stalker: the first mechanism for detecting, de-anonymizing, and destroying Tor botnets. We use machine learning to analyse and fingerprint the timings and frequency of Tor network circuit data when routing botnet traffic, and build a detection mechanism that is able to identify infected hosts at the Tor network border, in real-time, while preserving the privacy of legitimate users. TorBot Stalker can be implemented at any node in the Tor network and can differentiate between botnets and legitimate applications like Internet Relay Chat (IRC) coming from the same host. Experimental data demonstrates an accuracy of 99% with few false positives. We then apply the technique at the entry to the Tor network to measure the fraction of traffic which is for botnet. We observed that Torbot Stalker is able to de-anonymize real botnets in the Tor network and further identify infected hosts and control servers.**

Index Terms—**malware, botnet detection, Tor, machine learning, traffic analysis, circuit fingerprinting, deanonymization**

I. Introduction

Botnets are collections of infected computers (bots) that can be remotely controlled by the botmaster through a command and control (C&C) server. They contribute to the majority of the distributed denial of service (DDoS) attacks, spam, banking trojans, ransomware, and other malicious activities [1]. Botnets are frequently sold or rented on hacking forums allowing even poorly resourced actors to use them. Botnets are usually classified into three types depending on their C&C infrastructure. In the centralised infrastructure, the C&C server communicates with the bots from a central location as seen with IRC botnets. If the botnet is detected, the takedown of the servers constitutes a single point of failure. The second type is the distributed or decentralized architecture where bots do not communicate with a single server but with each other using a peer-to-peer (P2P) network. Although bots can still be detected, this architecture is more resilient to takedown, as the discovery and the takedown of one bot does not result in the takedown of the whole botnet, since other bots have many options to connect to. Conficker, Waladec, and Zeus P2P are examples of this type [2]. The third type is the hybrid architecture, which may opt to either communicate through one of the mechanisms and use the other when one fails, or they may opt to use both systems together. This method may provide more resilience to Sybil attacks [3], a frequent attack against P2P networks, and hence improve the reliability of the botnet in a situation where peers become unavailable or compromised. They are more difficult to setup and control.

Although there are a variety of detection schemes, botmasters have recently created two problems for detection. First, they use legitimate applications, protocols, and encryption to hide malicious traffic. Secondly, they regularly move the location of the C&C server by either generating multiple domains or physically changing the location resulting in a cat-and-mouse game between law enforcement/cybersecurity organizations and the botmasters. This could also be a difficult task for the botmasters coupled with the risk of losing control of the entire botnet if the location of the C&C server is discovered. Recently, there has been an abuse of anonymity and mix networks like Tor to evade detection and anonymize the location of the C&C servers. The use of Tor makes it impossible for traditional network-based methods to detect botnets due to specific protocol usage, encryption, anonymity of IP addresses, same traffic features for both legitimate and illegitimate traffic, and lack of domain name system (DNS) requests. Tor also ensures that the C&C servers remain anonymous by routing traffic through several relays across the Tor network.

In this paper, we present TorBot Stalker, the first practical network-based technique for detecting botnet and periodic communication routed through Tor. Our technique involves the use of circuit traffic fingerprinting and machine learning to classify Tor circuits to either web or botnet activity. Botnet and IRC communication can be accurately and effectively de-anonymized and terminated in real-time and further lead to the discovery of infected hosts, C&C servers, and IRC servers. It can be implemented from a single Tor entry node without the need of a global passive adversary.

Contributions: **i**). We introduce the first network-based method for detecting and de-anonymizing Tor botnets using machine learning to effectively classify statistical fingerprints extracted from Tor circuit data; **ii**). We propose an efficient and generally applicable method for extracting unknown normal and randomized periodic intervals from a time series mixed with noisy and background data; **iii**). We present a technique for identifying whether the detected communication originated from a host or a server.

Roadmap: Section 2 provides the reader with a background overview in Tor botnets. Section 3 discusses previous and relevant related work. Section 4 introduces TorBot Stalker, our technique for de-anonymizing Tor botnets. Section 5 describes our data collection methods, experiments and results. Section 6 evaluates our results and real world detection. We propose improvements and conclude in Section 8.

II. The Rise of Tor Botnets

Tor is an anonymity system which allows users to anonymously connect to web services or service providers by building three-hop circuits consisting of a guard, middle, and exit relay [4]. By using the Diffie-Hellman exchange to create secure keys, it ensures that relays know only the identities which are directly connected to it. This provides a form of layered encryption whereby confidentiality, integrity, and anonymity is guaranteed. When connecting to normal Internet services like HTTP, FTP, or telnet, the exit servers are the final layer where the packets are fully decrypted and sent out as normal. Therefore, where there are no other forms of encryption like TLS or end-to-end encryption, there is no guarantee of confidentiality or integrity between the exit servers and normal internet services.

Tor allows users to offer onion or hidden services which are secure ways where users can host web services without revealing both the original source and final destination of the circuits at any point. It works by the hidden services advertising their existence in the Tor network by randomly picking some relays, building circuits to them, and asking them to act as introduction points using its public keys. The hidden service then sends a service descriptor which contains its public key and a summary of each introduction point to a distributed hash table. When a client wants to connect to a hidden service, it creates a circuit to a randomly picked relay which would become its rendezvous point. The client then downloads the descriptor of the onion service from the distributed hash table and selects an introductory point from the list. The client sends an introduction message (which includes the address of the rendezvous point) to the selected introductory point asking it to be delivered to the onion service. The hidden service then creates a circuit to the rendezvous point and the rendezvous point (RP) notifies the client of a successful connection establishment. As seen in Fig. 1, the client and the hidden service would now communicate with each other by using the rendezvous point as an intermediary.

Despite several attacks on Tor such as statistical and confirmation attacks which are done by closely monitoring the timings of the packets at different nodes in Tor [5], [6], Tor remains a popular choice for botnets to hide their C&C servers [7] because it makes the C&C servers anonymous and setting up any botnet to use Tor is easy. The traditional methods for detecting botnets at the network through signatures, DNS analysis, anomalies, and traffic analysis are insufficient because Tor uses encryption and the previously known distinguishing features for botnets like the type of protocol used, port numbers, IP addresses, and packet size are the same with the normal traffic features making malicious traffic appear legitimate. Botnet C&C servers like Sefnit and a modified version of Zeus (called Skynet) have become problematic, receiving the highest number of hidden service requests in Tor [7]. A Skynet botnet C&C server analyzed was found to have between 12,000 to 30,000 infected hosts connecting to it [8]. Tor botnets led to a frequent spike and sudden increase in Tor traffic as seen with the Mevade Botnet [9], creating latency problems for Tor users.

Preliminary observations reveal that they are similar in communication patterns with normal botnets but the use of Tor hides their activities in the network. For example, the Zyklon botnet communicates to the C&C server by sending an encrypted HTTP POST request to the C&C hidden service and has similar malicious capabilities like other botnets. The request may either contain ex-filtrated data or may be a ping at a specified interval to inform the hidden service that it is presently online. Like normal botnets, Tor botnets periodically attempt to communicate (ping) with the C&C server with a small payload size. Zyklon would send only the IP address and computer name at a set interval. Botnets are likely to have only one cell for a ping even if the payload is increased because Tor uses 512 bytes per cell. This uniformity in botnet communication has been previously explored [10] but existing methods which rely only on this periodicity would be rendered ineffective where there is noisy/background data or where the counts and timings of the botnets are randomized. The background data could comprise of commands, downloads, or updates sent by the server or client or could be random noise intended to disrupt traditional timing analysis. For randomization, either the timing or the quantity of the regular periodic control data could be made unsystematic.

III. Related Work

Botnets can be detected on the hosts with the use of signature recognition by anti-viruses or Host Intrusion Detection Systems (HIDS) [11]. However, many host-based detection tools only detect known malware samples (and variants) [12] and many computers do not have any kind of protection [13]. Botnets can also be detected in the network through traffic signature recognition [14], based on network anomalies [10], DNS [15], and data mining [16].

To our knowledge, this is the first network-based technique for detecting Tor botnets. However, the application of web fingerprinting and statistical analysis for de-anonymizing Tor Hidden Services has been applied even recently. Circuit fingerprinting attacks use circuit data to efficiently identify a user's involvement with hidden services with an accuracy of 88% and further identify which hidden service a user is involved with from a set of hidden services known to the attacker with a 99% accuracy [17]. This method is suitable only to web services which have a relatively fixed web page size and assumes that the hidden service does not have a noisy stream. Another method uses traffic confirmation attacks where an attacker controls the entry guard to the hidden service as well as the entry guard to the host and can confirm similar timings. However, this is only feasible by a global passive adversary with control of a substantial number of entry guards.

Some have attempted to deal with the problem indirectly. TorPolice uses the circuit creation failure rates caused by the botnets to detect and mitigate botnet abuse of Tor [18]. This method fails to detect botnets before the abuse and when they do not attack Tor. Torward is an IDS to detect and classify malicious traffic including botnet traffic at an exit router of the Tor network. The defense system then processes these IDS alerts and blocks suspected connections [19]. Unfortunately, most Tor botnets use hidden services which do not use exit servers. Also, rulesets and signatures would not adequately detect unknown botnets [20].

One method attempts to cluster similar periodic communications allowing security analysts to focus on a few likely malicious targets [21] compared to previous methods which attempt to directly detect botnets based on only their periodicity [22] [23], [24]. A method further categorized the patterns based on non-periodic, weakly-periodic, and strongly-periodic, and achieved an accuracy of 80% [25]. This approach fails to consider legitimate web applications like IRC which have strong periodic patterns and the misclassification of 20% of the traffic could have severe consequences and block legitimate applications. A proper solution for Tor botnet detection would be a detection mechanism that runs in real time and detects botnet traffic with an accuracy close to 100%.

IV. TorBot Stalker

We describe a real-time system for collecting Tor circuit data, arranging them into flows and analyzing them for patterns using machine learning. It is capable of detecting periodic patterns even in the midst of noisy traffic and does not constitute any additional network overhead as the traffic is merely intercepted within the host. It is most effective at the entry node to the Tor network but can be used at any node in the network. It is able to further distinguish between applications which have similar traffic patterns such as IRC, web pages which would regularly reload, and botnets. TorBot Stalker was developed and tested using Weka [26] and Python. Fig. 1 shows the flow of the system and each component is described in the following subsections.

A. *Tor Circuit Logger*

The Tor Circuit Logger collects cell data on a Tor relay, recording the number of cells, inter-arrival times, IP addresses and circuit IDs. We do this by modifying the official Tor relay source code to send this information to a monitoring process in real time. We exclude the create and destroy cells since they are unnecessary for detection.

B. *Tor Circuit Flows*

The cell times are then distributed into a netflow-like structure where cell times CT_n belonging to the same source IP, destination IP, and unique circuit identifier are classified in the same flow F_n. Tor is used to connect to normal web services through exit servers or to a hidden service (onion address). In the former, the same circuit ID is used for different services, while in the latter, each hidden service has its own unique circuit ID. The use of separate IDs helps to differentiate between hidden services and further reduce noise. In both cases, the same circuit is reused for as long as the connection is not closed. Applications like IRC Chat can reuse the same circuit for as long as possible. By default, when connections to hidden services are closed and re-established, the same circuit is used for new connections for up to 10 minutes after which the circuit is changed.

C. *Cell Aggregation, Grouping, and Time Segregation*

In each flow, consecutive cell times CT_n which are close to each other where $CT_i < CT_{(i-1)+i}$ are clustered and the mean of the cells is calculated as a unique cell UC_n. The total number of the cells making the unique cell is summed as the unique cell count UCC. The value of plus/minus 2 seconds is used to determine if a cell is close to the next as it is a fair interval to account for separate activities while making room for network lags especially in the case of chat protocols that could have constant cells. For example, [10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33] become [11.5, 21.5, 31.5]; the cells with timestamps 10, 11, 12 and 13 were clustered as a unique cell with timestamp 11.5 and a unique cell count of 4. Cell aggregation is done to group events together and ease the extraction of features. All unique cells with the same counts are grouped together GC since periodic communications are likely to have the same number of counts. Cells having the same count are further segregated into 12 minute time windows and passed separately to the interval miner. 12 minutes is used to capture all possible times within a short-lived circuit.

D. *Interval Miner*

The interval miner is developed in Python and can detect multiple regular intervals even in the midst of noise. It works by checking if particular periodic patterns exist within the time window. First, the time difference (TD) of every possible two combinations in the window is calculated using the itertools.combination function in Python. Time differences

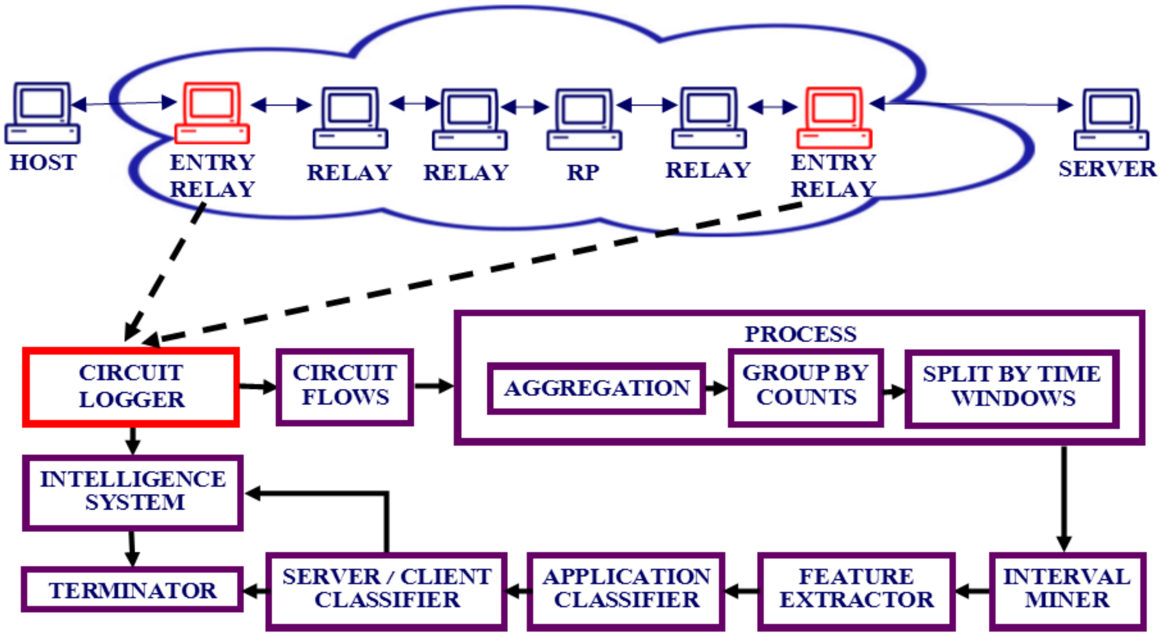

Fig. 1. TorBot Stalker Flow

which occur 3 or more times are then auto-matched with existing times in the list of unique cells. A variation of plus/minus 2 seconds is also employed to make up for possible network lags or issues that occur in real-world detection. The matched times are then extracted as regular occurring communication. Duplicated intervals are also removed from the matching list. The interval miner function is also applied to the entire timeseries to take into consideration instances where the unique cell count is distorted.

```
for x,y in itertools.combinations(GC, 2):
  diff_GC.append(y−x)
for a in diff_GC:
  i = GC[GC.index(a)]
    while i < GC[−1]:
      if i+k or i+k−1 or i+k+1 or i+k−2 in GC:
        if i in match[k,a]:
          match[k,a].append(i + k)
        else:
          match[k,a]=[a,i+k]
        i+=k
```

Listing 1. Pattern Matching

E. *Statistical features extraction*

From the training data, we extracted 32 statistical features/ circuit patterns based on the count and inter-arrival times of the inbound and outbound cells respectively. We extract features for the detected periodic intervals as well as the entire cell times within a given time window to cover for situations where due to noise or randomization, periodic cells with different counts are left out. For count based features, we calculate: (a) the total number of cells within the window where $\sum|CT_1 : \overline{CT}|$ and the total number of unique cells $\sum|UC_1 : \overline{UC}|$, (b) the total no of unique cells within the longest interval in the group count, if any; (c) the mean of the counts for all unique cells, where mean of counts is

$$\frac{\sum_{i=2}^{n}(UCC_i)}{n-1}$$

(d) the 1st and 2nd count of the Unique Cells within the longest interval UC_1 and UC_2, and (e) the group count GC.

Count based features help us to exploit the fact that different applications would exhibit different count behaviours. We observed that in the case of the Zyklon botnet, we find a consistency of 6 inbound and 5 outbound cells. The inbound cells consist of 3 relay cells, 1 begin cell, one HTTP GET cell and a HTTP POST cell. In the case of IRC, after a successful connection to a channel, we find a consistency of 1 inbound and 1 outbound cell which corresponds to the pings and pongs exchanged between the client and server. In the case of the telnet botnet we find 2 inbound and 2 outbound cells, and in the case of the TCP and telnet botnet, we find a consistency of 2 inbound cells.

For the time-based features, we calculate the total duration of the cells within the time window which is $\overline{CT} - CT_1$, as well as the duration of the group count and the longest interval in all cells, if any. We also use the longest interval detected, the mean and the standard deviation of the time differences between all cells (CTD), the longest interval in all cells and the longest interval in the group counts, respectively, where the mean is

$$\frac{\sum_{i=2}^{n}(CTD_1 - CTD_{i-1})}{n-1}$$

and the standard deviation is

$$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(CTD_i - \overline{CTD})^2}$$

Time-based features such as the duration of the inbound and outbound cells are used since botnet and circuits to web hidden services would usually not last longer than ten minutes compared to IRC circuits and a TCP-botnet which had long-lived connections. Our preliminary study revealed that different protocols were bound to have variations in the duration and size of the inbound and outbound cells. Also, persistent continuous communication are usual characteristics of botnet or IRC activity since they need to regularly get information from the C&C server. IRC would also naturally have a 30 second or 1 minute interval compared to botnets which could have different intervals.

We further attempt to extract features which could predict randomized botnets by calculating the average amount and duration of activity for each window. Our first metric is the division the total number of cells by the mean of all unique cells where

$$\frac{\sum |CT_1 : \overline{CT}|}{(\frac{\sum_{i=2}^{n}(UC_i)}{n-1})}$$

Secondly, we divide the total duration by the mean of all unique cell counts.

$$\frac{\overline{CT} - CT_1}{(\frac{\sum_{i=2}^{n}(UCC_i)}{n-1})}$$

F. *Application Classifier*

Using the Random Forest machine learning algorithm, the statistical features extracted from the periodic patterns are used to build a model based on a training data-set we describe below. Based on the trained dataset, the resulting model classifies the circuit flow into one of the following three classes: Normal (N), HTTP botnet (HTTPB), TCP botnet (TCPB), Telnet botnet (TB), IRC (IRC) or Web. Machine learning helps to find and predict patterns without explicitly programming or developing an algorithm. The extracted features are fed into the model built from the algorithm and it is executed in Weka through a modified Python script based on Python Weka Wrapper [27]. The Random Forest algorithm was chosen based on the experimental results which are reported in Section V.

G. *Hidden Service/ Host classifier*

Next, we determine if the non-Tor IP address involved belongs to a hidden service or a host by recording the direction of the circuit. Here we attempt to extract the sequence pattern/direction of the circuits. This is important because first, only the hidden servers have onion addresses and only the hosts can initiate a connection since the hidden services do not know how to reach the hosts. This helps us to track down which IP addresses are hosts and which are hidden services after detection. Our observations revealed that a circuit belongs to a server if the 1st cell is an inbound and vice-versa. Torbot Stalker needs to be implemented at an entry node for this stage and the next stages. We first cross-check the IP addresses involved against the Tor consensus list and determine whether the first cell was an inbound or outbound one.

H. *Intelligence gathering and Terminator*

We gather information about the participant including the geolocation, open ports, and reputation score to determine the botnet use. In cases where a public IP address belongs to a hidden service, we attempt to connect to web links associated with C&C panels. The detected IP addresses belonging to botnet are also added to a shared list. The list of botnet IPs are automatically blocked by automatically updating the network firewall/iptable rules on the entry node where TorBot Stalker is implemented. This list is added to a central database and shared by nodes. The next section describes the experiments that were run to identify the best machine learning algorithm for detecting botnets.

V. Experimental Evaluation

Botnet detection research has suffered from a lack of proper description of the data, methodology, and evaluation of results. We attempt to make our detection system repeatable, feasible, and easy to implement in a real-world scenario. We follow several standard malware data collection for machine learning guidelines [28]–[30] and provide the methodology for repeatable comparison and evaluation.

A. *Data Collection*

We used circuit dataset generated by browsing random onion sites [17] as well as capture web traffic data as realistically as possible to real human user activity by using Google Chrome, Mozilla Firefox, and the Tor browser to physically visit a variety of web and onion sites including social networks, forums, sports, video streams, audio streams, and WikiLeaks. Audio stream data from the Spotify application is obtained from real human user activity. Video streams are obtained from YouTube as well as other video streaming websites. Web services which have regular intervals or regularly refresh the browsers are included in the normal web data. Non-malicious Telnet, FTP, and SSH connections are also made from various computers and recorded. For IRC chat, data is obtained by connecting to several channels with varying activity on an anonymized IRC server. In a safe environment, circuit data is collected from various botnet protocols and families including HTTP, Telnet, and TCP all of which are routed through Tor. We captured circuit traffic data for different ping intervals from the botnet including 30 seconds, 1,2,3,4 minutes and a randomized time of 10 - 45 seconds in the case of the random botnet to represent different possible scenarios. 10 - 45 seconds was used as they were the times used by randomized botnets found on Github. For the purposes of training the model, we collected data for a period of 12 minutes although Telnet and botnet circuits were closed after ten minutes because they are short-lived. We collected 12 minutes to discover long-lived connections which last longer than 10 minutes. Different percentages for the distribution of classes were experimented with but none of them made a significant change on the results.

B. *Experiments and Results*

In our first technique, we extracted 34 features as described above based on the results from the Interval Miner and classify the different types of intervals retrieved into HTTP Botnet, TCP Botnet, Telnet Botnet, IRC, and web intervals. Our second technique avoids the use of the Interval Miner and extracts 8 count-based features from the circuit data and classifies them into HTTP Botnet, TCP Botnet, Telnet Botnet, IRC, and web traffic. The features used are the total number of cells, total number of unique cells, and the mean and standard deviation of the time differences of the unique cells from the inbound and outbound circuits respectively. We experimented with several algorithms, including Decision Trees, Rules, Functions, Bayesian Approaches, Support Vector Machine, Nearest Neighbor and Regression; we report the results for the best two algorithms, which were the random forest and the J48 decision tree algorithm. The J48 decision tree algorithm is based on Quinlan's C4.5 algorithm and makes decisions based on a tree-like graph structure [31]. Decision Trees is said to perform better on well-defined problems and has been previously successful in classifying botnet traffic with low false positives [32], [28]. Random Forest [33] which is classed as belonging to decision tree approaches, is also used. Random Forest is an ensemble learning algorithm, i.e. it uses several learner models to perform the classification; it constructs a multitude of decision trees during the training process and outputs the mode of the classes or the mean prediction of the individual trees as the final result. Logistic, Rules, Lazy and Bayes classification algorithms are not reported because they either performed poorly on our dataset or were slower. For the purposes of detection, we require an algorithm with a high performance in order to meet our requirement for a real-time detection.

To identify the most promising algorithms we used ten-fold cross-validation; as mentioned previously, we report the results for the top two, which are the J48 and Random Forest algorithms. Table II shows the cross-validation results for the two experiments, reporting the performance both overall (accuracy, precision, recall and F-measure), as well as per class (precision, recall and F-measure) of the two algorithms. These metrics take values between 0 (the poorest performance) and 1 (the highest performance). For ease of understanding, we transform these in percentages. All algorithms performed well, with both Random Forest and J48 having an accuracy higher than 98% and low false positives (FP) for all the classes. For the first experiment, the J48 decision tree algorithm has a 98.16% accuracy and a low false positive rate of 0.2 %, 2.0%, 0.3%,0.2%, 0.3% and 1.2% for TCP, Telnet, HTTPB, IRC, and Web Intervals, respectively. Random forest performed slightly better with an overall accuracy of 99.63%, and a low false positive rates of 0.1%, 0.2%, 0%, 0.1%,0.1%, and 0.1%. For the second experiment, J48 had an accuracy of 98.29% and a low false positive rate of 0.5%, 1.3%, 0.3%, 0.2%, 0.2%, and 0.8% for TCP botnet, Telnet botnet, HTTP Botnet, IRC, and web traffic. Random Forest also performed better with an accuracy of 99.8% and either low false positives of either 0.1% or no false positives.

An interesting observation is that out of the 182 randomized botnet circuits of 10 - 45 seconds, the interval miner with a variance of 2 seconds was still able to detect some form of regular periodic intervals in 134 circuits. These included longer intervals not anticipated by the botnet. For the first experiment, we find that Random Forest perfectly classified the randomized botnets with other botnets while the J48 algorithm misclassified four of the randomized botnets but only one as a web interval. In the second experiment, J48 misclassifed four of the randomized botnets into the wrong botnet categories but not as normal web traffic or IRC. Random forest perfectly classified random botnets. For the first experiment, there were 1085 instances and J48 predicted 1065 correctly while Random forest correctly classified 1081. In the second experiment, there where 1347 instances with the J48 correctly classifying 1324 while Random Forest correctly classified 1345. These results show that although the second technique works better and is able to predict the type of botnet traffic, the interval miner helps to make the detection of periodic communication faster and more effective. TorBot Stalker also has a very low false positive rate when classifying web traffic ensuring that legitimate traffic is not blocked.

We further attempted to perform a practical evaluation of how the second technique would perform in an application environment by collecting a separate dataset where the ground truth is known and correctly labeled. The algorithms are trained and the new dataset is used as a supplied test set. With the supplied test dataset, Random Forest also performed best and predicted 99% of the classes accurately with no false positives for the normal web and IRC data, and 0.38% for botnet data which was misclassified as normal data. J48 also came close in second place with no false positives for normal or IRC data and a 98% accuracy.

VI. DISCUSSION

With the increase in the exploitation of legitimate services by botnets, it is impossible to have an all-in-one solution for botnets. TorBot Stalker is protocol specific but the methods could be applied in other environments. Our approach employs a machine learning algorithm for the detection of botnet traffic patterns in real time and preserves the privacy of its users. Our experimental results show that the majority of botnet traffic instances are detected and unlikely to block legitimate traffic. TorBot Stalker accurately detected all the current and popular Tor botnets tested and is designed to detect unknown current botnets which have a similar communication architecture. Our solution can determine specific applications experimented with

TABLE I
CROSS VALIDATION RESULTS FOR 1ST TECHNIQUE

	J48					**Random Forest**				
Accuracy	98.2%					99.6%				
Precision	98.1%					99.6%				
Recall	98.2%					99.6%				
F-Measure	98.1%					99.6%				
	TCP	**Telnet**	**HTTPB**	**IRC**	**WEB**	**TCP**	**Telnet**	**HTTPB**	**IRC**	**WEB**
Precision	99.2%	98.3%	98.3%	96.5%	66.7%	99.6%	99.8%	100%	98.2%	87.5%
Recall	95.8%	99.3%	100%	98.2%	60%	100%	100%	100%	98.2%	70%
F-Measure	97.5%	98.8%	99.1%	97.3%	63.2%	99.8%	99.9%	100%	98.2%	77.8%

TABLE II
CROSS VALIDATION RESULTS FOR 2ND TECHNIQUE

	J48					**Random Forest**				
Accuracy	98.3%					99.9%				
Precision	98.3%					99.9%				
Recall	98.3%					99.9%				
F-Measure	98.3%					99.9%				
	TCP	**Telnet**	**HTTPB**	**IRC**	**WEB**	**TCP**	**Telnet**	**HTTPB**	**IRC**	**WEB**
Precision	98.1%	98.6%	97.7%	97.4%	98.8%	100%	99.8%	100%	99.2%	100%
Recall	98.1%	99.2%	97.7%	94.9%	98.2%	99.6%	100%	100%	100%	99.4%
F-Measure	98.1%	98.9%	97.7%	96.1%	98.5%	99.8%	99.9%	100%	99.6%	99.7%

and it is the first method to apply fingerprinting methods on periodic intervals in Tor.

Previous Tor fingerprinting methods have focused on de-anonymizing web services. In comparison to other fingerprinting methods, we achieve a higher accuracy of 99.8% and our method can deal with the issue of noise as well as randomization. We have achieved a higher accuracy than other works involved in detecting periodic intervals while solving the problem of noisy and randomized data where previous autocorrelation functions have failed. TorBot Stalker has substantially improved upon Tor fingerprinting methods by extending its application to botnet and IRC traffic and differs from previous periodicity detection because it accurately predicts the application involved. To reinforce the effectiveness of our interval miner, we test its ability to detect regular intervals from a wide range of time streams. We randomly generate 20000 lists. Each of them contains 30-50 randomly generated numbers from the range of 1-200. Regular periodic times with an interval of 10 were inserted and our interval miner was tested on it. Multiple intervals of 15, and 17 are also inserted to determine its effectiveness in detecting multiple intervals. The results show that our interval miner detected all of the intervals plus additional intervals generated from the random data. We further evaluated the performance of TorBot Stalker in the real-world by collecting and analyzing circuit data from a Tor entry node, using the Circuit Logger. In a timeline of 45 minutes, TorBot Stalker detected that 3.44% of the circuits analyzed were botnets, 0.57% were for IRC communication, and the remaining were normal circuits.

One weakness of Torbot Stalker is its difficulty in detecting very long periodic intervals where the botnet has a short-lived connection of 10 minutes. This is because the circuit paths / IDs will change and another relay will be used. However, this would lead to greater inefficiency for the botnet as botmasters need to know which bots are alive and bots need to be ready to receive commands as soon as they are issued. Secondly, botmasters could easily force Tor to change its circuit path for every time it sends data. This would ensure that there are no intervals to analyze. This problem can be solved by analysing network packets at ISPs and local networks rather than Tor circuits at an entry relay.

VII. FUTURE WORK AND CONCLUSION

For Torbot Stalker to be most disruptive, it needs to be implemented by a substantial number of relays. This is because if a relay blocks a bot, the bot will simply connect to another relay. A framework for a widespread implementation needs to be developed and possibly included as an option in the Tor source code. Currently, Tor's policy is that all relays must accept all inbound and forward all outbound connections. However, the benefits of blocking botnets including speeding up the network and disrupting malicious activities to the Tor network and relay operators cannot be underestimated. Secondly, a framework for intelligence sharing on botnets across relays can help increase the speed of detection by first looking up a list of IPs involved in recent botnet communication. IP addresses which are heavily involved could be possibly associated with the C&C servers. Future work would also collect and analyze network packet data to increase the scope of data collection to any host which can intercept network traffic including ISPs, routers, firewalls, IDS, and sensors. Packets can also be blocked through the use of an added network layer. Network packet analysis would also reveal persistent and long-lasting connections from hosts to Tor nodes even when they are randomized or the circuit path is changed regularly since the host IP address does not change.

The rise of undetectable stealthy botnets is imminent and there is a great need for a proactive solution which is protocol specific. In this paper, we have described the detection problems created by botnets using Tor for communication and we have presented and evaluated the first network-based technique for de-anonymizing and disrupting Tor botnets. We have described how circuit data is effectively collected, analysed, and classified using the Random Forest machine learning classifier. Our results have shown that TorBot Stalker can detect Tor botnets with a higher accuracy where other approaches cannot.

References

[1] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets.," *SRUTI*, vol. 5, pp. 6–6, 2005.

[2] 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 *PST 2010: 2010 8th International Conference on Privacy, Security and Trust*, pp. 31–38, 2010.

[3] C. R. Davis, J. M. Fernandez, S. Neville, and J. Mehugh, "Sybil attacks as a mitigation strategy against the storm botnet," *3rd International Conference on Malicious and Unwanted Software, MALWARE 2008*, pp. 32–40, 2008.

[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," tech. rep., US Dept of the Navy, jan 2004.

[5] G. Danezis, C. Diaz, and C. Troncoso, "Two-sided statistical disclosure attack," *Proceedings of the 7th international conference on Privacy enhancing technologies*, pp. 30–44, 2007.

[6] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proceedings - IEEE Symposium on Security and Privacy*, no. November, pp. 183–195, 2005.

[7] G. Owen and N. Savage, "Empirical analysis of tor hidden services," *IET Information Security*, vol. 10, no. 3, pp. 113–118, 2016.

[8] A. Biryukov, I. Pustogarov, and R. P. Weinmann, "Trawling for tor hidden services: Detection, measurement, deanonymization," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 80–94, 2013.

[9] N. Hopper, "Challenges in protecting tor hidden services from botnet abuse," in *International Conference on Financial Cryptography and Data Security*, pp. 316–325, Springer, 2014.

[10] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, Citeseer, 2008.

[11] S. S. C. Silva, R. M. P. Silva, R. C. G. Pinto, and R. M. Salles, "Botnets: A survey," *Computer Networks*, vol. 57, no. 2, pp. 378–403, 2013.

[13] E. Avena, R. Grebennikov, Capriotti, Z. Dong, and E. Douglas, "Microsoft security intelligence report," *Microsoft Security Intelligence Report*, vol. 22, pp. 1–74, 2017.

[12] T. Hyslip and J. Pittman, "A Survey of Botnet Detection Techniques by Command and Control Infrastructure," *Journal of Digital Forensics, Security and Law*, vol. 10, no. 1, pp. 7–26, 2015.

[14] J. Goebel and T. Holz, "Rishi: identify bot contaminated hosts by IRC nickname evaluation," *HotBots'07 Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, p. 8, 2007.

[15] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in DNS traffic," *CIT 2007: 7th IEEE International Conference on Computer and Information Technology*, pp. 715–720, 2007.

[16] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection," *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, pp. 349–360, 2012.

[17] J. Kwon, J. Lee, H. Lee, and A. Perrig, "PsyBoG: A scalable botnet detection method for large-scale DNS traffic," *Computer Networks*, vol. 97, pp. 48–73, 2016.

[18] Z. Liu, Y. Liu, P. Winter, P. Mittal, and Y.-C. Hu, "TorPolice: Towards Enforcing Service-Defined Access Policies in Anonymous Systems," aug 2017.

[19] Z. Ling, J. Luo, K. Wu, W. Yu, and X. Fu, "TorWard: Discovery, Blocking, and Traceback of Malicious Traffic over Tor," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2515–2530, 2015.

[20] R. a. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and taxonomy of botnet research through life-cycle," *ACM Computing Surveys*, 2013.

[21] G. Apruzzese, M. Marchetti, M. Colajanni, G. G. Zoccoli, and A. Guido, "Identifying malicious hosts involved in periodic communications," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8, Oct 2017.

[22] D. Moustis and P. Kotzanikolaou, "Evaluating security controls against HTTP-based DDoS attacks," *IISA 2013 - 4th International Conference on Information, Intelligence, Systems and Applications*, pp. 165–170, 2013.

[23] R. Tyagi, T. Paul, B. S. Manoj, and B. Thanudas, "A novel HTTP botnet traffic detection method," *12th IEEE International Conference Electronics, Energy, Environment, Communication, Computer, Control: (E3-C3), INDICON 2015*, pp. 1–6, 2016.

[24] T. M. Koo, H. C. Chang, and G. Q. Wei, "Construction P2P firewall HTTP-Botnet defense mechanism," *Proceedings - 2011 IEEE International Conference on Computer Science and Automation Engineering, CSAE 2011*, vol. 1, pp. 33–39, 2011.

[25] M. Eslahi, M. S. Rohmad, H. Nilsaz, M. V. Naseri, N. Tahir, and H. Hashim, "Periodicity classification of HTTP traffic to detect HTTP Botnets," in *2015 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 119–123, IEEE, apr 2015.

[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[27] C. Beckham, M. Hall, and E. Frank, "Wekapyscript: Classification, regression, and filter schemes for weka implemented in python," *Journal of Open Research Software*, vol. 4, no. 1, 2016.

[28] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[29] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 65–79, IEEE, 2012.

[30] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*, pp. 305–316, IEEE, 2010.

[31] M. Cocea and S. Weibelzahl, "Log file analysis for disengagement detection in e-learning environments," *User Modeling and User-Adapted Interaction*, vol. 19, no. 4, pp. 341–385, 2009.

[32] C. So-In, N. Mongkonchai, P. Aimtongkham, K. Wijitsopon, and K. Rujirakul, "An evaluation of data mining classification models for network intrusion detection," in *Digital Information and Communication Technology and it's Applications (DICTAP), 2014 Fourth International Conference on*, pp. 90–94, IEEE, 2014.

[33] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.