BotFP: FingerPrints Clustering for Bot Detection

Agathe Blaise, Mathieu Bouet, Vania Conan, Stefano Secci

▶ **To cite this version:**

Agathe Blaise, Mathieu Bouet, Vania Conan, Stefano Secci. BotFP: FingerPrints Clustering for Bot Detection. IEEE/IFIP Network Operations and Management Symposium (NOMS), Apr 2020, Budapest, Hungary. 10.1109/NOMS47738.2020.9110420 . hal-02501912

HAL Id: hal-02501912
https://hal.science/hal-02501912v1

Submitted on 8 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

BotFP: FingerPrints Clustering for Bot Detection

Agathe Blaise*†, Mathieu Bouet†, Vania Conan† and Stefano Secci‡
*Sorbonne Université, CNRS LIP6, Paris, France
†Thales, Gennevilliers, France. Email: {name.surname}@thalesgroup.com
‡Cnam, Paris, France. Email: stefano.secci@cnam.fr

***Abstract*—Efficient bot detection is a crucial security matter and has been widely explored in the past years. Recent approaches supplant flow-based detection techniques and exploit graph-based features, incurring however in scalability issues in terms of time and space complexity. Bots exhibit specific communication patterns: they use particular protocols, contact specific domains, hence can be identified by analyzing their communication with the outside. To simplify the communication graph, we look at frequency distributions of protocol attributes capturing the specificity of botnets behaviour. In this paper, we propose a bot detection technique named *BotFP*, for *BotFinger-Printing*, which acts by (i) characterizing hosts behaviour with attribute frequency distribution signatures, (ii) learning behaviour of benign hosts and bots through a clustering technique, and (iii) classifying new hosts based on distances to labelled clusters. We validate our solution on the CTU-13 dataset, which contains 13 scenarios of bot infections, connecting to a Command-and-Control (C&C) channel and launching malicious actions such as port scanning or Denial-of-Service (DDoS) attacks. Our approach applies to various bot activities and network topologies. The approach is lightweight, can handle large amounts of data, and shows better accuracy than state-of-the-art techniques.**

I. Introduction

Back in 2000 appeared the first notorious botnet, which sent 1.25 million emails containing phishing scams [1]. During the last 20 years, botnets evolved to become ever more sophisticated and dangerous. In Sept. 2019, the French cyber police freed over 850,000 computers from a botnet named Retadup [2]. The worm spread through malicious email attachments, then installed cryptomining software on infected machines. All infected hosts mined Minero cryptocurrency, reaping a huge amount of money. 2019 saw an increase up to 55% of IoT malware attacks like Retadup [3], thus the problem of detecting them quickly is a major concern.

The word "botnet" comes from the combination of "robot" and "network". In this display, the attackers infect and control thousands of machines, then send them malicious commands to execute, like infecting, attacking or scanning other hosts. This large zombie network is a major vector of large-scale attacks such as phishing DDoS, trojans, spams, etc. Their early detection is crucial to limit harms as soon as possible. However bots mimic normal traffic and hide their payload characteristic by encryption. Recently they are also more likely to use HTTP rather than IRC to be confounded with classic web traffic. Furthermore, dynamic ports and change of protocols enable botnets to bypass signature-based firewalls and intrusion detection systems (IDS). For robust detection systems, several flow-based botnet detection approaches [4], [5], [6] have been proposed without packet payload information. Rather than computing flow-based features, recently proposed approaches to detect botnets consist in characterizing and analyzing relationships between hosts in the network, with techniques commonly referred to as graph-based anomaly detection [7], [8], [9]. However these techniques are very costly in terms of memory and processor usage, as they need to compute complex features over very large graphs.

In this paper, we propose a lightweight bot detection technique *BotFP* that builds signatures modelling the behaviours of hosts in a network. These signatures reflect the communication pattern of each host, to highlight the differences between normal hosts and bots. In particular, we exploit the fact that a botnet performs various kinds of actions. One can simultaneously infect and scan other hosts, perform click fraud, launch DDoS attacks, actions that can be qualified by finely analyzing IP addresses, TCP and UDP port numbers and ICMP types and codes. Then, a clustering algorithm aims at accurately defining what constitutes bot and normal communications based on the signatures of labelled hosts. Finally, we are able to classify new hosts based on their distances to labelled clusters.

For our evaluation, we use the CTU-13 bot traffic dataset [10], containing 13 scenarios of different botnet samples. On each scenario a specific malware is executed, which performed different actions. We first learn from a training set what constitutes normal or malicious communications, based on the distribution of IP addresses and port numbers used by hosts. Subsequently, we demonstrate that clustering the data enables to detect all bots with a better accuracy and a reduced complexity. After tuning some parameters, we show that our algorithm outperforms state-of-the-art bot detection techniques, with 100% true positive rate and 0.9% false positive rate. We also show that using an adaptive quantification based on the volume of traffic enhances the results.

This paper is structured as follows. Section II addresses related work. Section III presents our methodology to detect bots based on their communication patterns. Section IV shows the complexity of our algorithm. Section V introduces the dataset and metrics used for our classifier evaluation, then Section VI present the evaluation. Finally, Section VII concludes this paper.

II. Related work

Considering the importance of the matter, numerous works have been undertaken on the field of bot detection. Traditional

approaches rely on statistical and machine learning approaches over per-flow features. BotHunter [4] aims to recognize the infection and coordination dialog that occurs during a successful malware infection. BotSniffer [5] focuses on the detection of C&C channels by exploiting similarity property of botnet C&C. Both approaches perform their evaluation on their own honeynet, but these traces are not publicly available and [6] highlighted the lack of suitable comparisons for bot detection algorithms. Hence they propose a labeled dataset including botnet, normal and background traffic. The authors also present a method to identify bots in these traces, named CAMNEP, which combines various state-of-the art anomaly detection methods, such as MINDS, Xu and Lakhina volume [11]. However, these techniques miss some communication patterns between hosts which are quite specific to a botnet. Also, the complexity required to compute per-flow features is high.

Graph-based techniques [12] are designed to overcome these limitations, by modelling the relations between several hosts of a network. They have been used in various cases: to detect P2P botnets [13], [14] or to recognize DNS traffic from malicious domains [15]. *BotGM* [7] proposes an unsupervised graph mining technique to identify abnormal communication patterns as bots. The authors first construct a graph sequence of ports for each pair of source and destination IP addresses, then they compare each graph between them using the Graph-Edit Distance (GED). They reach a very good accuracy between 78% and 95%, however the GED is computed once for each pair of graph and its computation is known to be NP-complete. The authors in [9] model network communications as graphs, where hosts are edges and communications between hosts vertices. They compute graph-based features such as degrees and centrality measures. They use a hybrid learning method and test various ML techniques to achieve a good detection rate. However this incurs in a high computational overhead as features are computed over a large communication graph, e.g., used by shortest paths algorithms computed for centrality measures. Other graph-based detection methods seem promising, but their complexity is often high [8], [16], [17].

Our solution *BotFP* presents the advantages of a graph-based technique, analyzing the communications of an host with the outside, but compared to them it is lightweight like a per-flow method would be.

III. Botnet Detection

We present our bot detection technique *BotFP*, detailing the different processing steps.

A. Overall approach

The goal of our solution is to label bots as such, avoiding false positives. Let *Sip*, *Dip*, *Sport* and *Dport* represent respectively the source and the destination IP addresses, the source and the destination port numbers, of a flow. Fig. 1 sums up the *BotFP* steps, through a trace example.

1) **Flow records collection**: flow records are first collected to form a dataset. We split the dataset into two distinct sets: one for training and one for testing.
2) **Host network** Sip **filtering and grouping**: from flow records, we select the ones whose Sip is in the host network and group them by such addresses. We discard hosts with less than 150 packets. Indeed, irrelevant for bot detection, they represent less than 0.7% of the traffic and may generate noise.
3) **Quantification**: signatures of each host, denoted σ_{Sip}, are defined as the concatenation of the normalized frequency distributions of each attribute. TCP, UDP and ICMP flows are characterized separately to better take into account each protocol specificity.
4) **Offline training** (clustering): this consists in grouping similar σ_{Sip} from the training set into clusters, then label them as bot or benign hosts based on ground truth.
5) **Online classification** (distances computation): finally we classify hosts from the test set based on their distance to labelled clusters.

Our algorithm takes into account two key parameters: b, the number of intervals (bins) in the frequency distribution, and ϵ, the density in the clustering algorithm.

B. Quantification (attribute frequency distributions)

Flow records are first collected to form a dataset (**step 1** in Fig. 1). We split the dataset into two distinct sets: one for training named $\mathcal{T}$, and one for testing named $\mathcal{E}$. We thus group flows by Sip as shown in Fig. 1 (**step 2**).

Then, to characterize the host behavior, we consider 9 attributes in total, discriminating between TCP, UDP and ICMP packets, as follows : $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{TCP}}$, Dip_{TCP}, $\text{Sport}_{\text{UDP}}$, $\text{Dport}_{\text{UDP}}$, Dip_{UDP}, $\text{Type}_{\text{ICMP}}$, $\text{Code}_{\text{ICMP}}$ and Dip_{ICMP}.

Let $\tilde{a_i}^j$ denote the attribute vector for attribute i and host j, representing the attribute frequency distribution, i.e., the ratio of packets received for attribute i over its attribute range. More precisely, each attribute vector contains b bins, where $\tilde{a_i}^j[k]$ is the value of the k^{th} bin of attribute i for host j. For each attribute, a bin aggregates the attribute occurrences over the possible attribute range (e.g., many successive port numbers grouped together in a bin) available for the specific attribute (e.g., TCP source port).

Let σ_j denote the signature of node j, keeping in mind that a host is uniquely identified by its Sip. It is built as the concatenation of all its attribute vectors, and expressed as:

$$\sigma_j = \Big\|_{i=1}^{9} \tilde{a_i}^j = \tilde{a_1}^j \parallel \tilde{a_2}^j \parallel \cdots \parallel \tilde{a_9}^j \tag{1}$$

where $\parallel$ represents the concatenation operator between vectors. The result of the concatenation is then one single vector σ_j of $9 \times b$ entries.

1) Quantification technique: Let us further clarify how the attribute frequency distributions can be aggregated in a set of bins. To compute the attribute vector, b bins are used to cover the attribute range, say $[0, max]$; e.g., for source and destination port numbers max is equal to 65,536, and for destination IP addresses to 2^{32}. It makes sense to set b as a power of 2, as port numbers and IP addresses are typically

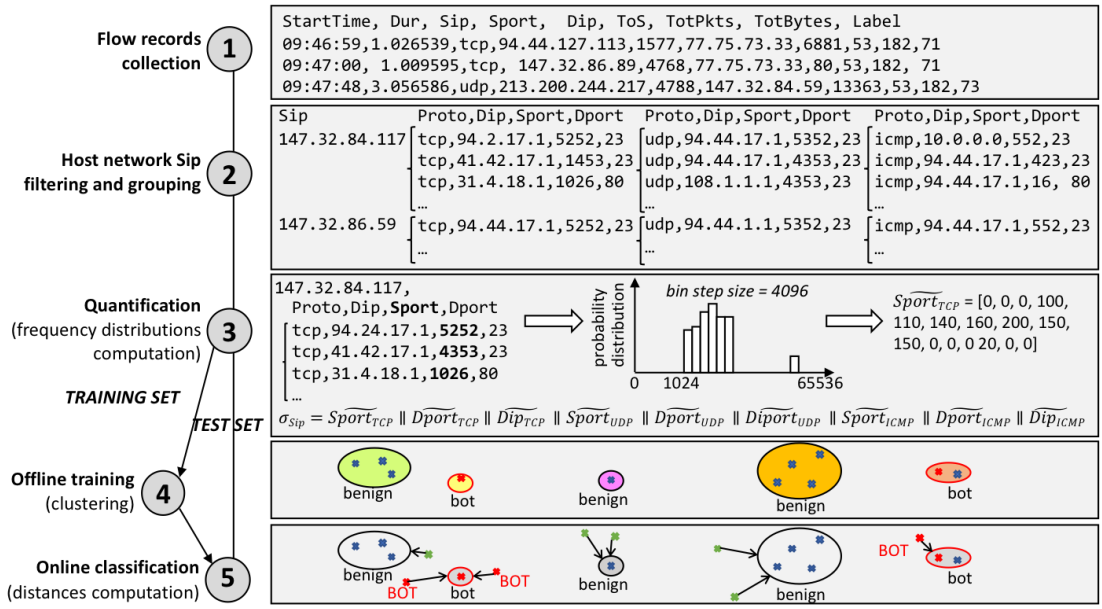


Fig. 1: Description of the processing steps of our solution.

organized into ranges of powers of 2 (e.g., reserved ports are in [0, 1023] and ephemeral ports in [49152, 65536], while IPv4 addresses are denoted by 4 Bytes).

We consider two different ways to aggregate bins:

Regular bins: attribute range intervals are uniformly distributed, of a fixed bin width set to *max* / *b*. Fig. 1 (**step 3**) shows an example of attribute frequency histogram for attribute $\text{Sport}_{\text{TCP}}$: the attribute range corresponds to the possible TCP source port numbers used by the *Sip* host.

Adaptive bins: intervals are chosen depending on the amount of traffic. Intuitively, the more density of information there is, the more sensitive (small) the step should be. Thus we aim to define individual bin width so that we equalize the occurrences over the different bins, i.e., it is always the same for all the bins, each bin having potentially a different bin width. We repeat this process for all the attributes.

2) Observable bot behavior and attributes: Let us elaborate further on the observable behaviour for TCP, UDP and ICMP attributes from traces we could have access to.

TCP - *destination ports* ($\text{Dport}_{\text{TCP}}$) usually range between 0 and 1023. These service ports are associated to given services by the Internet Assigned Numbers Authority (IANA) [18], e.g. TCP/80 typically runs HTTP and TCP/443 HTTPS. However, bots show different usage of destination ports: they are usually diverse and represent services often targeted by attackers such as TCP/25 (SMTP) or TCP/23 (Telnet), vulnerable to spam and attacks. We also observe some exotic destination port numbers used to access proxies that host the C&C server.

TCP - *source ports* ($\text{Sport}_{\text{TCP}}$) are ephemeral ports, allocated automatically from a predefined range by the IP stack software. The range recommended by IANA is 49152 to 65535. Many Linux kernels use the port range 32768 to 61000. FreeBSD has used the IANA port range since release 4.6, and was using [1025, 5000] before. Microsoft Windows Operating Systems (OS) until Windows XP use the range [1025, 5000] as ephemeral ports, while use the IANA range now. We observe that bots rarely use the IANA recommended range, but rather the range [1025, 5000]. This obviously depends on the OS of the infected host. A report from Kaspersky Labs [19] shows that Linux and Windows botnets represent respectively 95.75% and 4.29% of all botnets, which is very different from the OS distribution for regular devices (not bots).

TCP - *destination IP addresses* (Dip_{TCP}); only some specific subnets are contacted by normal hosts. Among them, it is common to observe addresses in the same range of the source IP address, private networks including `192.168.0.0/16`, and cloud service subnetworks, mostly Google ones. Destination IP addresses cover a larger space for bots than for normal nodes, in case of a spam or port scan for example.

UDP - *destination ports* ($\text{Dport}_{\text{UDP}}$) are associated to particular services, as for TCP. In the case of UDP, we often observe a fixed destination port set to 53. It represents connections to the local DNS server as UDP/53 typically runs DNS.

UDP - *source ports* ($\text{Sport}_{\text{UDP}}$) are used for ephemeral ports as for TCP, their range depends on the OS implementation.

UDP - there is usually a fixed *destination IP address* (Dip_{UDP}) that represents the DNS server IP address.

ICMP - *type* ($\text{Sport}_{\text{ICMP}}$) indicates the type of ICMP message and gives a global information about the kind of message (e.g., 0 for Echo Reply and 3 for Destination Unreachable), as specified in RFC2780 [20]. In case of a botnet, we sometimes observe many ICMP messages with uncommon types and codes, consisting in a Ping Flood or an ICMP DoS attack.

ICMP - *code* ($\text{Dport}_{\text{ICMP}}$) represents the ICMP subtype and gives additional context information for the message (e.g. if the type is 3, the code can be 0 if the destination network is unreachable or 1 if the destination host is unreachable, etc.).

ICMP - the hosts frequently reply to *destination IP ad-*

dresses (Dip_{ICMP}) that targeted them, with messages like "port unreachable" if it was a port scanning. The number of such packets is low for benign hosts, and very high for bots that scan hosts.

Looking to these attributes individually enables to retrieve some botnets behaviours, but it is even better to observe these attributes together. Actually, sometimes it is the combination of two attributes that makes a host behaviour abnormal.

C. Training (clustering)

Clustering algorithms are designed to group similar vectors into clusters and identify isolated ones as outliers. The similarity between two vectors is evaluated using a distance function like the Euclidean distance. We use as clustering algorithm DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [21]; it presents the advantage of discovering clusters without knowing the number of clusters in advance, using two parameters, and fits well our case because clusters have close densities. ϵ specifies the radius of a neighborhood with respect to some point, and $minPts$ defines the minimum number of points in a radius ϵ to form a cluster.

DBSCAN defines a cluster as the maximal set of points where every pair of points p and q are within a distance ϵ from each other, and considers points that do not belong to any cluster as outliers. In our solution, DBSCAN is used in a slightly different manner as illustrated in **step 4** of Fig. 1. We set $minPts$ to 1 in order to consider singleton clusters as well. Then DBSCAN is applied on the vectors of σ_j from the training set to build clusters of similar host signatures. Using clusters instead of singular hosts enables to filter abnormal hosts and get more consistent data.

The metric that we use for the computation of the distance between two hosts in DBSCAN is the $\ell 1$-norm defined as $||\sigma_h||_1 = |\sigma_h[1]| + ... + |\sigma_h[n]|$: this distance is robust and does not vary with the number of bins, as the cumulative sum of all elements stays equal. We consider it better than the $\ell 2$-norm – defined as $||\sigma_h||_2 = \sqrt{|\sigma_h[1]|^2 + ... + |\sigma_h[n]|^2}$ which increases with the number of bins.

Let C be the set of clusters obtained applying DBSCAN on the training set. Each cluster $c \in C$ contains several attributes:

- a set H_c of hosts belonging to the cluster;
- its position P_c computed as the centroid of the set of signatures $\{\sigma_1, \sigma_2, ..., \sigma_N\}$ of hosts in H_c, computed as $P_c = (\sigma_1 + \sigma_2 + ... + \sigma_N)\ /\ N$;
- a label identifying the nature of the cluster c, i.e., malicious or benign, denoted L_c. The nodes that are bots are known from the ground truth of the training set. The cluster is identified as a bot cluster if it contains at least one bot, else it is benign.

Fig. 1 (**step 4**) shows five clusters, including singletons, labelled as 'bot' if they contain at least one bot, else benign.

D. Classification

We classify hosts from the test set based on their distance to the set of labelled clusters C. For a host $h \in \mathcal{E}$, if the closest cluster is labelled as bot, h will be classified as a bot. If the closest cluster is benign, h will be classified as benign too. Using as distance function dist() the difference between the $\ell 1$-norm of two vectors, for a cluster c^* where $\text{dist}(\sigma_h, P_{c^*}) = \min_c[\text{dist}(\sigma_h, P_c)]$, hosts are classified with

$$L_h = \begin{cases} \text{'bot' if } L_{c^*} = \text{bot} \\ \text{'benign' otherwise} \end{cases} . \quad (2)$$

IV. Complexity

We qualify the space and time complexity of *BotFP*, considering its three main steps.

A. Attribute frequency distributions computation

First, we need to compute the fingerprint σ_j for all hosts.

1) Space complexity: given a host and $|A|$ attributes, we need to store arrays of b bins for all the attributes, then the per-host space complexity is equal to $O(|A| \cdot b)$. The overall process is a one-shot operation over all hosts, resulting in a complexity $O(|\mathcal{T} \bigcup \mathcal{E}| \cdot |A| \cdot b)$ In our setting we have $|A| = 9$.

2) Time complexity: for an host i, the computation of each attribute vector comes with $|a_i|$ entry readings, before bin aggregation, thus the worst-case time complexity is $O(|A| \cdot max_i|a_i| \cdot |\mathcal{T} \bigcup \mathcal{E}|)$.

B. Training (clustering)

The training consists in building host clusters from the training set, each host being characterized by its fingerprint σ_j.

1) Space complexity: DBSCAN presents a space complexity of $O(|\mathcal{T}|)$ to store the positions and labels of the $|\mathcal{T}|$ points, and the neighbors of the currently queried point.

2) Time complexity: DBSCAN presents a worst-case time complexity of $O(|\mathcal{T}|^2)$. For each point of the database, we have to visit each other point to query their neighborhood.

C. Classification (distances computation)

The classification determines the closest cluster to each host to classify, and assign its label to the host.

1) Space complexity: We have to store the positions of all clusters. Also for each host, we need to store the distance between its signature and each cluster. Therefore the total space complexity is $O(|C| \cdot |A| \cdot b + |\mathcal{E}| \cdot |C|)$.

2) Time complexity: we need to parse all hosts from the test set, then to compare each of them to all clusters with a $\ell 1$-norm, thus the time complexity is equal to $O(b \cdot |A| \cdot |\mathcal{E}| \cdot |C|)$.

V. Evaluation Methodology

We present the dataset and the metrics we use to evaluate BotFP.

A. Dataset

We used the publicly available CTU-13 dataset [6] made of 13 scenarios of bot infections, containing botnet, normal and background traffic. Botnet malwares are executed in a virtual network to mimic the behaviour of an infection that is spreading. Table I describes, for each scenario, the type of C&C server as well as the malicious activities. The dataset has been widely used in the already discussed recent bot detection

methods [7], [8], [9]. To evaluate the performances of our bot detection method, we used scenarios 1, 2, 6, 8, 9 for the test set (marked by the symbol * in Table I), and others for the training set, as recommended by the authors of the CTU-13 [6].

Id	Duration (hrs)	#Bots	Bot	Activity
1*	6.15	1	Neris	IRC, SPAM, CF
2*	4.21	1	Neris	IRC, SPAM, CF
3	66.85	1	Rbot	IRC, PS
4	4.21	1	Rbot	IRC, DDoS
5	11.63	1	Virut	SPAM, PS
6*	2.18	1	Menti	PS
7	0.38	1	Sogou	HTTP
8*	19.5	1	Murlo	PS
9*	5.18	10	Neris	IRC, SPAM, CF, PS
10	4.75	10	Rbot	IRC, DDoS
11	0.26	3	Rbot	IRC, DDoS
12	1.21	3	NSIS.ay	IRC, P2P
13	16.36	1	Virut	HTTP, SPAM, PS

TABLE I: Characteristics of the botnet scenarios.

B. Evaluation metrics

A confusion matrix is a table often used to evaluate the performance of a classification model [22]. The basic terms are the following : True Positive (TP) is the number of bots correctly classified; True Negative (TN) is the number of benign hosts correctly classified; False Positive (FP) is the number of benign hosts incorrectly classified; False Negative (FN) is the number of bots incorrectly classified.

From this matrix, we compute metrics about our classifier. The accuracy, defined as $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, shows the fraction of true detection over total hosts. However, a bias may be introduced with an unbalanced dataset like the CTU-13 dataset with few bot activity. The true positive rate, defined as $TPR = \frac{TP}{TP+FN}$, shows the percentage of predicted bots versus all bots presented. The false positive rate, computed as $FPR = \frac{FP}{FP+TN}$, refers to the ratio of incorrectly classified benign hosts over all benign ones.

VI. Evaluation

In this section, we evaluate the performance of BotFP using the CTU-13 dataset.

A. First observations

Fig. 2 gives an example of dissimilar histograms for a benign host and a bot, for attributes $\text{Sport}_{\text{TCP}}$ and Dip_{UDP}. $\text{Sport}_{\text{TCP}}$ for the benign host are in the range [49152, 61000] and [1025, 5000] for the bot, which indicates a first difference in the ephemeral ports thus the OS (all bots from the dataset display this characteristic). Dip_{UDP} shows a multitude of IP addresses for the bot, and a single one for the benign host, while both are only using source port UDP/53 which runs DNS. Thus, the bot is not communicating with the DNS server, but this is in fact an attempt of port scanning.

B. Botnet detection results

We apply our bot detection method on the CTU-13 dataset. We analyze the influence of the number of bins b as well as the benefits in using adaptive bins rather than regular ones. Fig. 3

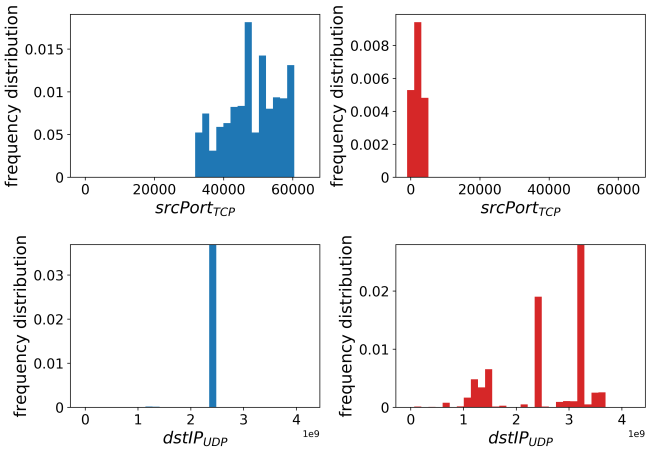


Fig. 2: 32-bin histograms showing the frequency distributions of 2 attributes ($\text{Sport}_{\text{TCP}}$ and Dip_{UDP}) for a benign host in blue/left (`147.32.84.17`) and a bot in red/right (`147.32.84.165`).

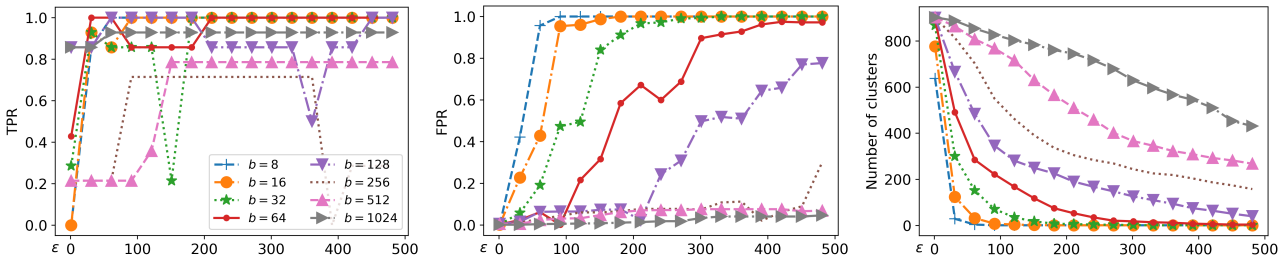
shows for regular bins the TPR (Fig. 3a), the FPR (Fig. 3b) and the number of clusters (Fig. 3c). Fig. 4 shows for adaptive bins the TPR (Fig. 4a), the FPR (Fig. 4b) and the number of clusters (Fig. 4c) for b between 8 and 1024. Multiple ϵ values (DBSCAN parameter) are tested in [0, 30, ..., 500].

Advantages of clustering: in Fig. 3 and 4, $\epsilon = 0$ is equivalent to not clustering the data, i.e., comparing each host from the test set to labelled hosts from the training set. Fig. 3a and 4a show that the TPR never reaches 100% in this case , as the classification is too specific and we overfit the data. However, increasing ϵ enables to detect all bots in some setups. Clustering the data also reduces the complexity of the classification, by limiting the number of comparisons to do.

Comparison between regular and adaptive bins: clustering the data turns out to be quite complex if we consider that we have to tune ϵ: a large value may produce too large clusters resulting in false positives, while a too small ϵ may overfit the data and miss bots. Formatting the data by handling adaptive bins gives more consistent results and eases the process of clustering. For regular bins (Fig. 3a), the TPR values are quite unstable even when ϵ rises. For adaptive bins on the contrary (Fig. 4a), the TPR oscillates between 85% and 100% (i.e., between 0 and 2 undetected bots) for ϵ from 150 and all b.

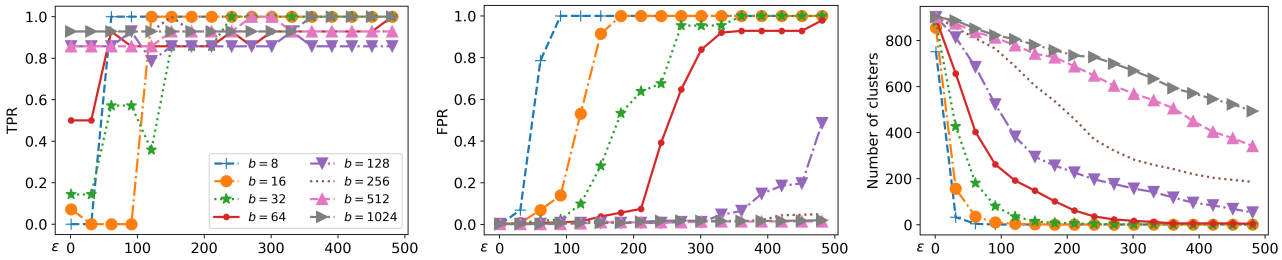
We also observe that using adaptive bins (Fig. 4b) yields far less false positives than regular ones (Fig. 3b). For these two reasons, we could confirm the intuition that using adaptive bins provides a more accurate analysis and therefore better results.

Number of bins b: we also need to choose the number of bins b and ϵ accordingly. The objective is to find a setup with a TPR equal to 100% (i.e., all bots detected) and a FPR as low as possible. Using adaptive bins, the TPR reaches 100% for nearly all values of b. However there is a strong correlation between b and the FPR: the higher b, the lower the number of false positives. Therefore the best solution is reached for a high number of bins (b = 512 or 1024), for which the TPR is equal to 100% and the FPR is very low. We can spot two parameter combinations: (i) for b = 512 and $250 \leq \epsilon \leq 320$,



(a) TPR. (b) FPR. (c) Number of clusters.

Fig. 3: Regular bins: True and False Positive Rates (TPR & FPR) and number of clusters against ϵ.



(a) TPR. (b) FPR. (c) Number of clusters.

Fig. 4: Adaptive bins: True and False Positive Rates (TPR & FPR) and number of clusters against ϵ.

TPR = 100% and $FPR \simeq 0.9\%$ and (ii) for b = 1024 and $\epsilon \geq 360$, TPR = 100% and $FPR \simeq 1.6\%$.

We have shown that our solution detects all bots with very few false positives. For the following experiments, we choose ϵ = 300 and between 512 and 1024 bins.

Trade-off between the number of clusters and the accuracy: Fig. 3c and 4c show the number of clusters respectively for regular and adaptive bins. For ϵ = 300 and adaptive bins (Fig. 4c), we notice that b = 512 produces around 550 clusters while b = 1024 around 638 ones. Therefore we would rather choose 512 bins to reduce the number of positions to store, with an equivalent accuracy. This also shows the benefits in clustering the data: 550 clusters for b = 512 is approximately 60% less than the 910 initial hosts.

C. Comparison to state-of-the-art detection techniques

We now compare our solution to other state-of-the-art detection methods, namely BClus [6], CAMNEP [6], BotHunter [4], *BotGM* [7] and [9] described in Section II.

Table II shows the confusion matrix for scenarios from the test set with b = 512 and ϵ = 300. We detected bots from all scenarios, which makes the true positive rate equal to 100%. In total, we labelled 4 benign hosts as bots, which results in a very low false alarm rate equal to 0.5%.

Id	TP	TN	FP	FN
1	1	163	3	0
2	1	131	0	0
6	1	111	0	0
8	1	165	5	0
9	10	133	1	0

TABLE II: Confusion matrix for scenarios 1, 2, 6, 8, 9 from the test set, with b = 512 and ϵ = 300.

Table III reports the results for each solution and all scenarios from the test set, as proposed in [6]. Our results are very competitive as we reach an accuracy between 97% and 100% with 512 bins while other algorithms provide an accuracy between 30% and 95%. Only [9] achieves up to 100% accuracy for scenario #9 but it tested only that one and trained on the 12 other scenarios.

Algorithm	1	2	6	8	9
BClus [6] (2014)	0.5	0.5	0.4	0.3	0.4
CAMNEP [6] (2014)	0.5	0.4	0.4	0.5	0.5
BotHunter [4] (2007)	0.4	0.3	0.38	0.42	0.4
BotGM [7] (2017)	0.91	0.78	0.95	0.89	0.83
Graph-based ML [9] (2019)	X	X	X	X	1[1]
BotFP	**0.98**	**1**	**1**	**0.97**	**0.99**

TABLE III: Accuracy of different algorithms evaluated in [6] and compared to *BotFP* with adaptive bins and ϵ = 300.

VII. Conclusion

Botnet attacks are always more sophisticated, and this is expected to get even worse with the massive increase of IoT devices. The quick detection of such bots is crucial to Internet security. Our technique *BotFP* uses attribute frequency distributions to characterize hosts communication, where bots exhibit specific behaviours. Signatures of each host are clustered, hence avoiding data overfitting and reducing the complexity. The detection results are very promising, since our algorithm detected all bots from the CTU-13 dataset. It achieves an accuracy close to 100%, outperforming state-of-the-art techniques, and is also very lightweight compared to graph-based techniques. In the future, we plan to reduce the number of dimensions in the per-host signatures. We also aim at investigating other classification approaches than clustering, like neural networks.

REFERENCES

[1] 9 of history's notable botnets. [Online]. Available: https://www.whiteops.com/blog/9-of-the-most-notable-botnets

[2] ZDnet. Avast and french police take over malware botnet and disinfect 850,000 computers. [Online]. Available: https://www.zdnet.com/article/avast-and-french-police-take-over-malware-botnet-and-disinfect-850000-computers/

[3] Mid-year update: 2019 sonicwall cyber threat report. [Online]. Available: https://blog.sonicwall.com/en-us/2019/07/mid-year-update-2019-sonicwall-cyber-threat-report/

[4] G. Gu, P. Porras, V. Yegneswaran, and M. Fong, "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of the USENIX Security Symposium*. USENIX Association, 2007.

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

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

[7] S. Lagraa, J. Francois, A. Lahmadi, M. Miner, C. Hammerschmidt, and R. State, "BotGM: Unsupervised graph mining to detect botnets in traffic flows," in *Proceedings of the Cyber Security in Networking Conference (CSNet)*. IEEE, 2017.

[8] W. Chen, X. Luo, and A. N. Zincir-Heywood, "Exploring a service-based normal behaviour profiling system for botnet detection," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017.

[9] A. A. Daya, M. A. Salahuddin, N. Limam, and R. Boutaba, "A graph-based machine learning approach for bot detection," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[10] Stratosphere Lab. The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic. [Online]. Available: www.stratosphereips.org/datasets-ctu13

[11] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, p. 219, oct 2004.

[12] S. Chowdhury, M. Khanzadeh, R. Akula, F. Zhang, S. Zhang, H. Medal, M. Marufuzzaman, and L. Bian, "Botnet detection using graph-based feature clustering," *Journal of Big Data*, vol. 4, no. 1, may 2017.

[13] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding p2p bots with structured graph analysis," in *Proceedings of the USENIX Security Symposium*, 2010, pp. 95–110.

[14] H. Jiang and X. Shao, "Detecting p2p botnets by discovering flow dependency in c&c traffic," *Peer-to-Peer Networking and Applications*, vol. 7, no. 4, pp. 320–331, jun 2012.

[15] F. Zou, S. Zhang, W. Rao, and P. Yi, "Detecting malware based on DNS graph mining," *International Journal of Distributed Sensor Networks*, vol. 2015, pp. 1–12, 2015.

[16] J. Wang and I. C. Paschalidis, "Botnet detection based on anomaly and community detection," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 392–404, jun 2017.

[17] P. Kalmbach, A. Blenk, W. Kellerer, and S. Schmid, "Themis: A data-driven approach to bot detection," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[18] (2013) Service name and transport protocol port number registry. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[19] Kaspersky. DDoS attacks in Q2 2019. [Online]. Available: https://securelist.com/ddos-report-q1-2019/90792/

[20] IANA. Internet control message protocol (icmp) parameters. [Online]. Available: https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[22] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, 2018.