A Graph-Based Machine Learning Approach for Bot Detection

Abbas Abou Daya, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
{aaboudaya, mohammad.salahuddin, noura.limam, rboutaba}@uwaterloo.ca

***Abstract*—Bot detection using machine learning (ML), with network flow-level features, has been extensively studied in the literature. However, existing flow-based approaches typically incur a high computational overhead and do not completely capture the network communication patterns, which can expose additional aspects of malicious hosts. Recently, bot detection systems which leverage communication graph analysis using ML have gained attention to overcome these limitations. A graph-based approach is rather intuitive, as graphs are true representations of network communications. In this paper, we propose a two-phased, graph-based bot detection system which leverages both unsupervised and supervised ML. The first phase prunes presumable benign hosts, while the second phase achieves bot detection with high precision. Our system detects multiple types of bots and is robust to zero-day attacks. It also accommodates different network topologies and is suitable for large-scale data.**

I. Introduction

Undoubtedly, organizations are constantly under security threats, which not only cost billions of dollars in damage and recovery, but also detrimentally affect their reputation. A botnet-assisted attack is a widely known threat to these organizations. According to the U.S. Federal Bureau of Investigation, "Botnets caused over $9 billion in losses to U.S. victims and over $110 billion globally." The most infamous attack, Rustock, infected 1 million machines, sending up to 30 billion spam emails a day [1]. More recently, WannaCry resulted in data breach from over 230,000 computers in 150 countries [2]. Undeniably, in the face of a cyber arms race, attackers constantly find clever ways to sabotage networks using botnets, most importantly via zero-day attacks [3].

A botnet is a collection of *bots*, agents in compromised hosts, controlled by botmasters via command and control (C2) channels. A malevolent adversary controls the bots through botmaster, which could be distributed across several agents that reside within or outside the network. Hence, bots can be used for tasks ranging from distributed denial-of-service (DDoS), to massive-scale spamming, to fraud and identity theft. While bots thrive for different sinister purposes, they exhibit a similar behavioral pattern when studied up-close. The intrusion kill-chain [4] dictates the general phases a malicious agent goes through in-order to reach and infest its target.

Detection of bots can be largely achieved via intrusion detection systems (IDSs), which can be broadly classified into *signature-based* and *anomaly-based* [5]. Signature-based methods use *pre-computed* hashes of existing malware binaries. They scale well and efficiently detect known threats. However, they require frequent database updates and can be subverted by unknown or modified attacks, such as zero-day attacks and polymorphism [5], [6]. This undermines their suitability for bot detection. Anomaly-based methods overcome these limitations [3], [7]. They establish a baseline of normal behavior for the protected system, and model a decision engine that alerts on any divergence or statistical deviations from the norm. *Machine learning* (ML) is an ideal technique to automatically capture the normal behavior of a system. Its use has boosted the scalability and accuracy of IDSs [3], [7].

An important step prior to learning, or training a ML model, is feature extraction. These features act as discriminators for learning and inference, reduce data dimensionality, and increase the accuracy of ML models. The most commonly employed features in bot detection are flow-based (*e.g.*, source and destination IPs, protocol, number of packets sent and/or received, etc.). However, these features do not completely capture the communication patterns that can expose additional aspects of malicious hosts. In addition, flow-level models can incur a high computational overhead, and can also be evaded by tweaking behavioral characteristics *e.g.*, by changing packet structure [8].

Graph-based features, derived from flow-level information to reflect the true behaviour of hosts, are an alternate that overcome these limitations. We show that incorporating graph-based features into ML yields robustness against complex communication patterns and unknown attacks. Moreover, it allows for cross-network ML model training and inference. The major contributions of this paper are as follows:

- We propose an anomaly-based approach for bot detection that is protocol agnostic, robust to zero-day attacks, and suitable for large datasets.
- We show the limitations of stand-alone supervised learning. Therefore, we employ a two-phased ML approach that leverages both supervised and unsupervised learning. The first phase filters presumable benign hosts. This is followed by the second phase on the pruned hosts, to achieve bot detection with high precision.
- We use graph-based features and evaluate various ML techniques. The graph-based features, derived from network flows, overcome severe topological effects that can skew bot behavior, exacerbating ML prediction. Furthermore, these features allow to combine data from different networks and promote spatial stability [9] in the models.

978-3-903176-15-7

The rest of the paper is organized as follows. In Section II, we present a background on bot detection and highlight limitations of the state-of-the-art. Our system design is delineated in Section III. We discuss the results of our experiments in Section IV. In Section V, we conclude with a summary of our contributions and instigate future research directions.

II. Related Works

Bot(net) detection has been an active area of research and has generated a substantial body of work. Most existing bot detection techniques employ methods for detecting C2 channels based on the statistical features of packets and flows [10]–[22]. Solutions like [10], [11] are focused on specific communication protocols, such as IRC, providing narrow-scoped solutions. On the other hand, Botminer [15] is a protocol-independent solution, which assumes that bots within the same botnet are characterized by similar malicious activities and communication patterns. This assumption is an over simplification, since botnets often randomize topologies [5] and communication patterns as we observe in newer malware, such as Mirai [23]. Therefore, it is evident that a non-protocol-specific, less evadable detection method is desired.

Furthermore, [18], [22] have exploited ML-driven anomaly detection with traffic-based statistical features, for detecting known and unknown attacks with low error rates. However, such techniques require that all flows are compared against each other to detect C2 traffic, which incurs a high computational overhead. In addition, they are unreliable, as they can be evaded with encryption and by tweaking flow characteristics [8]. Graph-based approaches, where graphs are extracted from network flows and host-to-host communication patterns, overcome these limitations [8], [24]–[32]. Other approaches [33], [34] rely on rule-based host classification and botnet detection methods, where pre-defined thresholds are used to discriminate between benign and suspicious hosts. They are static and prone to evasion.

On the other hand, outlier- and anomaly-based methods are generally more robust. BotGM [35] uses a statistical technique, the inter-quartile method, for outlier detection. Their results exhibit moderate accuracy with low FPs based on different windowing parameters. However, it generates multiple graphs from a selection of network flows. For every pair of unique IPs, a graph is constructed, such that every node in the graph represents a unique 2-tuple of source and destination ports, with edges signifying the time sequence of communication. This entails high overhead and will not scale for large datasets.

Chowdhury *et al.* [36] use ML for clustering the nodes in a graph, with a focus on dimensionality and topological characterization. Their assumption is that most benign hosts will be grouped in the same cluster due to similar connection patterns, hence can be eliminated from further analysis. Such a crucial reduction in nodes effectively minimize detection overhead. However, their graph-based features are plagued by severe topological effects (*cf.*, Section IV). They use statistical means and user-centric expert opinion to tag the remaining clusters as malicious or benign. Nevertheless, leveraging expert opinion can be cumbersome, error prone and infeasible for large datasets.

Graph-based approaches using ML for bot detection are intuitive and show promising results. In this paper, we propose an anomaly-, graph-based bot detection system, which is protocol agnostic *i.e.*, it detects bots regardless of the protocol. We employ graph-based features in a two-phased ML approach, which is robust to zero-day attacks, spatially stable, and suitable for large datasets.

III. System Design

Our bot detection system consists of 3 major components, as depicted in Fig. 1. These components pertain to data preparation and feature extraction, model training, and inference.

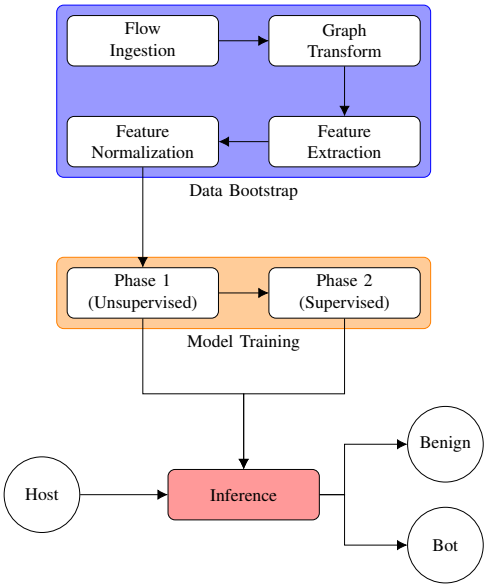

Fig. 1. Components of the bot detection system

A. Dataset bootstrap

1) Flow ingestion

The input to the system are bidirectional network flows. These flows are transformed into a set T that contains 4-tuple flows $t_i = \{sip_i,\ srcpkts_i,\ dip_i,\ dstpkts_i\}$. Where sip_i is the source IP address that uniquely identifies a source host, $srcpkts_i$ quantifies the number of data packets sent by sip_i to dip_i, the destination host IP address. The number of destination packets, $dstpkts_i$, is the number of data packets sent by dip_i to sip_i.

Set A is a set of tuples that have exclusive source and destination hosts. That is, if multiple tuples have the same source and destination hosts, they are reduced to form an aggregated exclusive tuple $a_x \in A$, such that $a_x = \{sip_x,\ srcpkts_x,\ dip_x,\ dstpkts_x\}$. Therefore, if two tuples $t_i, t_j \in T$ have the same source and destination hosts *i.e.*, $sip_x = sip_i = sip_j$ and $dip_x = dip_i = dip_j$, then number of source and destination packets are aggregated in a_x, such that

$$srcpkts_x = \sum_{t_k \in T \ \mid \ sip_x = sip_k, dip_x = dip_k} srcpkts_k \qquad (1)$$

$$dstpkts_x = \sum_{t_k \in T \ | \ sip_x = sip_k, dip_x = dip_k} dstpkts_k. \quad (2)$$

2) Graph Transform

The system creates a graph $G(V,\ E)$, where V is a set of nodes and E is a set of directed edges $e_{i,j}$ from node v_i to node v_j with weight $|e_{i,j}|$. The set of nodes V is a union of hosts from set A, such that

$$V = \bigcup_{\forall a_x \in A} \{sip_x \cup dip_x\}. \quad (3)$$

For every $a_x \in A$, there exist directed edges $e_{i,j}$ and $e_{j,i}$ from v_i to v_j and v_j to v_i, respectively, such that $sip_x = v_i$ and $dip_x = v_j$. Therefore,

$$E = \bigcup_{\forall a_x \in A} \{(sip_x,\ dip_x) \cup (dip_x,\ sip_x)\}. \quad (4)$$

The weights $|e_{i,j}|$ and $|e_{j,i}|$ of edges $e_{i,j}$ and $e_{j,i}$ are $srcpkts_x$ and $dstpkts_x$, respectively. Moreover, if there exists a reverse tuple $a_y \in A \mid dip_y = v_i,\ sip_y = v_j$, then $|e_{i,j}| = srcpkts_x + dstpkts_y$ and $|e_{j,i}| = dstpkts_x + srcpkts_y$.

3) Feature Extraction

The system creates the required *graph-based* feature set for the ML model. Features are intrinsic to the success of the model that should genuinely represent and discriminate host behavior, especially bot behavior. We study the following set of commonly used graph-based features.

- ***In-Degree (ID)*** and ***Out-Degree (OD)***—The in-degree, $f_{i,0}$, and out-degree, $f_{i,1}$, of a node $v_i \in V$ are the number of its ingress and egress edges, respectively.

$$\mathcal{F}(e_{i,j}) = \begin{cases} 1, & \text{if } e_{i,j} \in E \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$f_{i,0} = \sum_{v_j \in V,\ v_i \neq v_j} \mathcal{F}(e_{j,i}) \qquad \forall v_i \in V \quad (6)$$

$$f_{i,1} = \sum_{v_j \in V,\ v_i \neq v_j} \mathcal{F}(e_{i,j}) \qquad \forall v_i \in V \quad (7)$$

These features play an important role in the behavior of a host. Though, a higher ID for a host makes it a point of interest, often nodes with high ID offer commonly demanded service. Therefore, observing ID alone may not signify malicious activity. For example, a gateway is a central point of network communication, but it is not necessarily a malicious endpoint. Intuitively, bots attempting to infect the network will tend to have higher ID than benign hosts. Similarly, OD is also an intrinsic feature. Typically, in the reconnaissance stage of the intrusion kill-chain, bots attempt to mass-survey the network, which can be captured via OD.

- ***In-Degree Weight (IDW)*** and ***Out-Degree Weight (ODW)***—These features augment the ID and OD of the nodes in the graph. The in-degree weight, $f_{i,2}$, and the out-degree weight, $f_{i,3}$, of a node $v_i \in V$ is the sum of all the weights of its incoming and outgoing edges, respectively.

$$f_{i,2} = \sum_{v_j \in V,\ v_i \neq v_j,\ e_{j,i} \in E} |e_{j,i}| \qquad \forall v_i \in V \quad (8)$$

$$f_{i,3} = \sum_{v_j \in V,\ v_i \neq v_j,\ e_{i,j} \in E} |e_{i,j}| \qquad \forall v_i \in V \quad (9)$$

For a fine-grained differentiation, it is important to expose features that will eventually bring bots closer to each other, and discriminate bots from hosts. IDW and ODW features add another layer of information, further alienating the malicious hosts from the benign. Similar to ID, mass-data leeching bots will tend to expose a high IDW in the action phase of the intrusion kill-chain. Similarly, the ODW is the aggregate data packets a node has sent, which can potentially expose bots that mass-send payloads to hosts in a network.

- ***Betweenness Centrality (BC)***—The betweenness centrality of a node $v_i \in V$, inspired from social network analysis, is a measure of the number of shortest paths that pass through it, such that

$$f_{i,4} = \sum_{v_j, v_k \in V,\ v_i \neq v_j \neq v_k} \frac{\sigma_{v_j v_k}(v_i)}{\sigma_{v_j v_k}} \qquad \forall v_i \in V. \quad (10)$$

Where $\sigma_{v_j v_k}$ is the total number of shortest paths between node pairs $v_j,\ v_k \in V$, and $\sigma_{v_j v_k}(v_i)$ is the number of shortest paths that pass through v_i. This feature has a high computational overhead with $O(|V|.|E|+|V|^2.\log|V|)$ time complexity [37]. However, it can alienate bots early on as they attempt their first connections. This is when the bots exhibit low IDW and ODW. Thus, it would be more favorable for the shortest paths in the network to pass through the host. Likewise, when the IDW and ODW increase, the BC of a node decreases immensely, as it is less favored for being included in shortest paths.

- ***Local Clustering Coefficient (LCC)***—Unlike BC, local clustering coefficient has a lower computational overhead, and it quantifies the neighborhood connectivity of a node $v_i \in V$, such that

$$f_{i,5} = \frac{\sum_{v_j, v_k \in N_i,\ v_i \neq v_j \neq v_k} \mathcal{F}(e_{j,k})}{|N_i|(|N_i|-1)} \qquad \forall v_i \in V \quad (11)$$

Where N_i is neighborhood set for v_i, $\forall v_j \in N_i \mid e_{i,j} \in E \vee e_{j,i} \in E$. LCC feature can play an important role in discriminating malicious host's behavior. Successfully infected hosts tend to exhibit a higher LCC, as bots often deploy collaborative P2P techniques, making its adjacent host pairs strongly connected.

- ***Alpha Centrality (AC)***—Alpha centrality, also inspired from social network analysis, is a feature that generalizes the centrality of a node $v_i \in V$. AC extends the Eigenvector centrality (EC), with the addition that nodes are also influenced by external sources. These external sources can be user-defined, according to their graphical analysis technique. In EC, each v_i is assigned an influence score x_i, that is iteratively exchanged with adjacent nodes. In essence, EC

is the relative weight of a node in the graph, such that connections to high-scoring nodes contribute more to the score of v_i. Hence, AC is given as

$$f_{i,6} = \alpha A_i^T x_i + e_i \qquad \forall v_i \in V. \tag{12}$$

Where A_i is the adjacency matrix, e_i is the external influence of node v_i, and α is influence factor that controls focus between external sources to internal influence. AC is important for intermediate and terminal phases of the intrusion kill-chain. Early on, it may be negligible. However, as time progresses and bots perform more actions in the network, their AC will gradually increase, making it discriminative.

4) Feature Normalization (F-Norm)

Topological alterations can severely affect the host's behavior and pattern of communication in the graph. For example, in Fig. 2, g acts as a gateway for host $h2$ to communicate with the rest of the network (*i.e.*, hosts $h3$, $h4$ and $h5$). In this configuration, h_2 carries an ID of 2. In contrast, Fig. 3 shows the topology without a gateway, where h_2 communicates with other hosts in the network individually. This boosts the ID of host h_2 to 4. To alleviate this adverse effect of different network topologies, we normalize the above *base* features to incorporate neighborhood relativity.

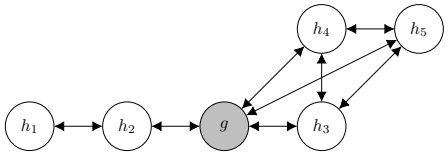

Fig. 2. Example topology of benign hosts with a gateway

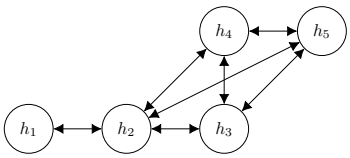

Fig. 3. Example topology of benign hosts without a gateway

To control the overhead of computing these *normalized* features, the neighborhood set N_i for $v_i \in V$ is restricted to a depth D. The mean of j features for v_i across its neighbors $v_k \in N_i$ are computed. Each feature for v_i is then normalized by incorporating neighborhood relativity. Thus, features relative to their neighborhood mean is given as

$$\mu_{i,m} = \frac{\sum_{v_k \in N_i} f_{k,m}}{|N_i|} \qquad \forall v_i \in V,\ 0 \leq m \leq j \tag{13}$$

$$f_{i,m} = \frac{f_{i,m}}{\mu_{i,m}} \qquad \forall v_i \in V,\ 0 \leq m \leq j. \tag{14}$$

After normalizing the features (with $D = 2$) for h_2 and h_4 with gateway, their IDs change from 2 to 0.8 and 3 to 1.1, respectively. Without the gateway, their IDs change from 4 to 1.6 and 3 to 1.1, respectively. As aforementioned, normalization attempts to make hosts of the same nature look similar, making the topological alterations less severe. Similarly, in situations where network data is recorded over varying time intervals, IDW and ODW tend to increase substantially with larger time intervals. By normalizing features, the effect of time also diminishes.

B. Model Training

The model is trained to accept graph-based features as input and learn to distinguish between malicious and benign hosts. Two learning phases are involved as explained below.

1) Phase 1

The first ML phase performs unsupervised learning (UL) to cluster the hosts. Generally, benign hosts exhibit *similar* behavior that can be gauged by the graph-based features. These hosts exhibit resembling patterns in data, such as sending (OD/ODW) and receiving (ID/IDW) similar number of packets [36]. Since BC, LCC and AC are directly affected by these traits, their influence can be similar for all benign hosts. This may maximize the size of the benign cluster.

This phase not only acts as a first filter for new hosts, but also significantly reduces the training data for the second phase. If a host is clustered into the benign cluster, then it is strictly benign. However, it is important to note that a malicious host can also be incorrectly clustered into a benign cluster, adversely affecting system performance. Although the objective is to maximize the size of the benign cluster, it is essential to *jointly* minimize the number of bots that are co-located in this cluster. Various UL techniques can be deployed in this phase, including *k*-Means, Density-Based Spatial Clustering (DBScan) and Self-Organizing Map (SOM). However, the classifier with the best assignment must be selected, according to the objective outlined in this phase.

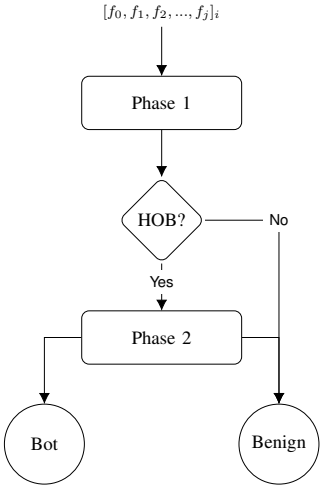

Fig. 4. Flowchart of node classification with i nodes and j features

2) Phase 2

Phase 1 separates the dataset between hosts that are inside and outside the benign cluster. All the hosts, ideally a small number, that reside outside the benign cluster are an input to Phase 2 for further classification. Optimally, all bots should be outside the benign cluster, regardless of whether or not they are co-located in the same cluster. Depending on the number of hosts outside the benign cluster, supervised learning (SL) classifiers in this phase will exhibit varying results.

The primary objective in this phase is to maximize recall *i.e.*, how many bots do not go unnoticed. It is proportional to the number of true positives (TPs) and inversely proportional to FNs. Various SL classifiers can be deployed to achieve this objective, such as logistic regression (LR), support vector machine (SVM), feed-forward neural network (FNN) and decision tree (DT). The objective from Phase 1 *i.e.*, minimize hosts outside the benign cluster (HOB), while maximizing bots outside the benign cluster (BOB), results in a minimal training dataset for Phase 2. Also, the resultant training dataset from Phase 1 is expected to be unbalanced, with a bias towards benign hosts. This may prove problematic for LR, SVM and FNN in achieving high recall rates.

C. Inference

Once the models are trained, the system is deployed to perform bot detection. Ideally, the system must allow for two modes of execution: (i) model (re)training, to adjust to the dynamics of the network, and (ii) inference, *i.e.*, to predict whether or not a given host is a bot. The inference unfolds in two steps—presumable benign hosts get filtered out in Phase 1 as they get assigned to the benign cluster, and suspicious hosts, assigned to a different cluster, are further classified in Phase 2. Fig. 4 captures the inner workings of node classification. To preserve consistency, the deployed system must synchronize and execute requests in order of observation.

IV. Experiments

We evaluate the proposed bot detection system on a Hadoop cluster. In this section, we detail the experimental setup and the results of our evaluation.

A. Environment Setup

1) Hardware

We experiment on a Hadoop cluster that consists of a management node (2x Intel Xeon Silver 4114; 192 GB RAM), a compute node (2x Intel Xeon Gold 5120; 384 GB RAM) and four data nodes (2x Intel Xeon Silver 4114; 192 GB RAM). A 25Gbit and 10Gbit physical networks are deployed, interconnecting the nodes. The former network is primarily used for data and applications, while the latter is for administration.

2) Software

The software implementation is primarily based on Java. To ease dependency management, the project incorporates Gradle [38]. JGraphT [39] graph library is used to construct the graph and extract graph-based features from network flows. Both Smile [40] and Encog [41] are used in tandem for ML. In order to support rapid prototyping, a custom in-house DataFrame (DF) library has been developed. DF conforms to the incremental streaming paradigms, data streams with well-defined source, pipelines and sinks. Furthermore, the underlying data structures are immutable and all the basic stream-based transformations are available.

B. Dataset

Our evaluation is based on the CTU-13 [42] dataset. CTU-13 comprises of 13 different subset datasets (DS) that include captures from 7 distinct malware, performing port scanning, DDoS, click fraud, spamming, *etc*. Every subset carries a unique network topology with a certain number of bots that leverage different protocols. Table I summarizes the dataset duration, number of flows and bots, and the type of bot in every subset. CTU-13 labels indicate whether a flow is from/to botnet, background or benign. Therefore, known infected hosts are labeled as bots and remaining hosts as benign. We leverage 12 datasets as base training data, while a single dataset, #9, is left out for testing purpose. This test dataset contains NetFlow data collected from a Neris botnet, 10 unique hosts labeled as bots, performing multiple actions. We use this dataset configuration for training and testing, unless stated otherwise.

TABLE I
CTU-13 Dataset

DS	Duration	# Flows	Bot	# Bots
1	6.15	2824637	Neris	1
2	4.21	1808123	Neris	1
3	66.85	4710639	Rbot	1
4	4.21	1121077	Rbot	1
5	11.63	129833	Virut	1
6	2.18	558920	Menti	1
7	0.38	114078	Sogou	1
8	19.5	2954231	Murlo	1
9	5.18	2753885	Neris	10
10	4.75	1309792	Rbot	10
11	0.26	107252	Rbot	3
12	1.21	325472	NSIS.ay	3
13	16.36	1925150	Virut	1

C. Results and Discussion

1) Graph Transform, Feature Extraction & Normalization

For every subset in the CTU-13 dataset, the system first ingests all the network flows, creates the graph, extracts base features and normalizes them. For each dataset, Table II highlights the graph creation time *i.e.*, graph transform (GT), number of graph nodes ($|V|$), total runtime to extract only base BC feature and all base features (FE), and total runtime to normalize features (F-Norm).

TABLE II
Graph Transform, Base Feature Extraction and Normalization Computation

DS	GT (seconds)	Nodes	BC (hours)	FE (hours)	F-Norm (seconds)
1	9	606829	24.12	24.121	11.3
2	6	441845	10.387	10.624	7.9
3	21	434489	9.463	9.713	13.755
4	5	185742	1.37	1.431	6.307
5	1	41548	0.057	0.06	0.556
6	3	107056	0.28	0.295	2.112
7	1	38081	0.021	0.022	0.488
8	13	383339	9.67	9.954	9.617
9	10	366881	8.677	8.97	7.879
10	7	197542	1.06	1.108	4.861
11	1	41809	0.055	0.057	0.627
12	2	94164	0.287	0.302	1.412
13	9	315343	3.667	3.852	6.824

It is evident that there is a non-linear relationship between BC and the number of nodes in the graph. Furthermore, the inconsistent variation between GT and the number of nodes is due to the differing time windows across datasets. Also, dataset #3 has a much higher number of flows than #2, which increases the runtime of graph creation. This is primarily due to the repeated modification of exclusive flow tuples in set A.

The system then normalizes the base features, and Table II depicts its total runtime with $D = 1$. Evidently, normalizing features does not significantly increase the total runtime of the system. The largest runtime reported for the most complex dataset is 13.755 seconds.

2) Stand-alone SL

We start by highlighting the limitations of a stand-alone supervised learning approach. This consists of evaluating supervised ML classifiers, including DT, LR, SVM and FNN for bot detection. Each classifier employs graph-based normalized features and is trained on the entire training dataset. In our experiments, DT uses the Gini instance split rule algorithm, LR is used without regularization, and SVM uses the Gaussian kernel with a soft margin penalty of 1. Moreover, NN is configured to use cross entropy as an error function and 10 hidden layers of 1000 units each. Table III highlights the results, where LR and DT show meaningful classification. Both LR and DT classifiers result in a 100% recall, with 91% and 83% precision, respectively. With LR's superiority in precision, it *seems to be* the classifier of choice. The other classifiers were able to accurately classify all the benign hosts, but failed to identify any bots.

TABLE III
SUPERVISED LEARNING WITH F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	10	2	366869	0	100	83
LR	10	1	366870	0	100	91
SVM	0	0	366871	10	0	0
FNN	0	0	366871	10	0	0

We then evaluate the training time and robustness of the stand-alone classifiers, as depicted in Tables IV and V. DT requires the least training time of 4.9 seconds, which is in high contrast to the 58.2 seconds for LR. That is, DT requires only 8.4% of LR's training time for the entire training dataset. It is also essential for a bot detection system to detect bots that the classifier has never seen before *i.e.*, unknown or zero-day attacks. Therefore, to evaluate robustness to zero-day attacks, we change the selection of the training and testing datasets. We choose dataset #6 for testing, which has a unique bot that is not present in any other dataset. The remaining datasets are aggregated to form the training set, with 34 bots and a total of ≈3.1M hosts. Evidently, DT outperforms LR, which misclassifies a benign host, with a low precision of 50%.

TABLE IV
TRAINING TIME OF SUPERVISED ML CLASSIFIERS

Classifier	Training Time (s)
DT	4.9
LR	58.2
SVM	6832.3
FNN	5.7

TABLE V
SUPERVISED LEARNING AGAINST PREVIOUSLY UNKNOWN BOT

Classifier	TP	FP	TN	FN	Recall	Precision
DT	1	0	107055	0	100	100
LR	1	1	107054	0	100	50
SVM	0	0	107055	1	0	0
FNN	0	0	107055	1	0	0

Based on the above evaluations, LR outperforms DT in precision, while DT shows a superior training time and robustness to unknown attacks. However, precision, training time and robustness are all crucial for our bot detection system. Can we achieve the best of all three? To investigate this, we set out to evaluate a two-phased system that employs an initial clustering phase (UL), followed by a classification phase (SL). We delineate its evaluation in the following subsections.

3) Phase 1 (UL)

For this phase of the system, we evaluate three UL techniques, namely k-Means, DBScan and SOM. However, DBScan results are inconclusive, where bots co-located with benign hosts. In essence, DBScan does not produce a single, prevalent benign cluster. DBScan is evaluated with varying minimum number of neighborhood points (minPts) and distance (ϵ). Multiple ϵ values are tested in the range of [10^{-5}, 10^{-4}, ..., 10^{5}]. Also, we infer ϵ values that correspond to the boundary of the bots themselves. We vary minPts in [1, 2, ..., 25] depending on the number of bots in the aggregated training dataset. However, maximal separation of bots from benign hosts could not be achieved with the tested parameters.

On the other hand, both k-Means and SOM show appreciable results, where SOM is trained with a learning rate of 0.7. Tables VI and VII highlight the evaluation metrics, including number of clusters or neurons, number of hosts outside the benign cluster (HOB), percentage of hosts outside the benign cluster relative to the total number of hosts (HOB%), number of bots outside the benign cluster (BOB), and percentage of bots relative to the total number of bots (BOB%).

Recall, the dataset #9 is removed for testing, which includes 10 hosts labeled as bots and ≈366K hosts. Also, ≈3.2M hosts from the remaining datasets are used to train the classifiers. In comparison to the number of clusters for k-Means, SOM is able to alienate its first bot outside the benign cluster with a lower number of neurons (9 *vs.* 16). With 81 neurons, SOM has a recall rate of 92%, with k-Means achieving 42%. However, k-Means catches up with 121 clusters. Nevertheless, SOM outperforms k-Means by maximizing the number of bots isolated with a smaller number of neurons.

TABLE VI
k-MEANS CLUSTERING WITH F-NORM

# of Clusters	HOB	HOB%	BOB	BOB%
4	5	0.0002	0	0
9	12	0.0004	0	0
16	36	0.0012	1	4
25	94	0.0033	6	24
36	170	0.0059	6	24
49	473	0.0164	8	32
64	1071	0.0371	10	40
81	1133	0.0392	10	40
100	3028	0.1049	21	87.5
121	26935	0.9327	24	96
144	27100	0.9384	24	96
169	27302	0.9454	24	96
196	27359	0.9474	24	96
225	28752	0.9956	24	96

With a cluster size of 100, k-Means alienates 21 bots, while having an outside host sum of 3028 for the remaining non-benign clusters. In contrast, SOM removes 23 bots from the benign cluster with an outside host sum of 3675. The very next k-Means cluster size *i.e.*, 121, boosts HOB from 3028 to 26935, while SOM remains at a close 3894. However, k-

TABLE VII
SOM CLUSTERING WITH F-NORM

# of Neurons	HOB	HOB%	BOB	BOB%
4	10	0.0004	0	0
9	29	0.0010	1	4
16	49	0.0017	1	4
25	113	0.0039	6	24
36	286	0.0099	7	28
49	556	0.0193	8	32
64	1709	0.0592	10	40
81	3524	0.1222	23	92
100	3675	0.1274	23	92
121	3894	0.1350	23	92
144	27591	0.9647	24	96
169	27856	0.9740	24	96
196	28342	0.9912	24	96
225	28449	0.9950	24	96

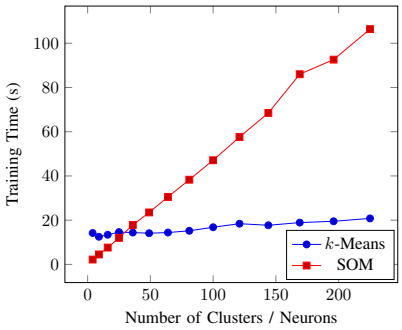

Fig. 5. Comparison of SOM and k-Means with respect to training time

Means isolates three extra bots, yielding 24 BOB for 26935 HOB. That is, three extra bots were detected for a ≈23K increase in HOB. Recall, our objective in this phase is to jointly minimize HOB while maximizing BOB. Therefore, SOM with 100 neurons becomes the natural choice.

With respect to runtime, k-Means mostly outperforms SOM, as depicted in Fig. 5. With 100 clusters, k-Means took 16.8 seconds to train, in comparison to 47.1 seconds of SOM. We speculate that SOM's ever increasing training time is contributed to how it updates the surrounding neurons. As the number of neurons increases, the density of their neighborhood increases. Eventually, more neurons will tend to be within the threshold radius. Nevertheless, with recall being our top priority, we leverage SOM as the UL classifier in Phase 1.

4) Phase 2 (SL)

The training set for Phase 2 is determined by the number of hosts outside the benign cluster in Phase 1. These are the relevant hosts for this phase, as hosts that are assigned in the benign cluster never make it to Phase 2. With a 10×10 (*i.e.*, 100 neurons) SOM and normalized features in Phase 1, the size of the dataset is significantly reduced. Therefore, we have 3675 HOB, including 23 bots, for classification in Phase 2.

TABLE VIII
SUPERVISED LEARNING WITH F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	10	1	366870	0	100	90.9
LR	0	0	366871	10	0	0
SVM	0	0	366871	10	0	0
FNN	0	0	366871	10	0	0

For this phase of the system, we evaluate four SL techniques, namely DT, LR, SVM and FNN. The DT classifier shows the best performance with the small dataset, as depicted in Table VIII. It successfully detects *all* bots in the test dataset, with only a single FP out of the 366871 benign hosts. In contrast, all other classifiers are lackluster and unable to recall even a single bot from the dataset. We believe this is because all classifiers, except DT, rely on gradient-descent for error-correction. This implies that every single node in the dataset will affect the end-hypothesis function. Thus, with a dataset that is unbalanced, the hypothesis will be biased towards the benign hosts, which is the case for LR, SVM and FNN.

Table IX highlights the training time for the supervised classifiers. For Phase 1, a 10×10 SOM incurs a training time of 47.1 seconds, while DT has the lowest training time of 88 milliseconds in Phase 2. Thus, the aggregate training time for both phases is ≈47.2 seconds. This is an 11 seconds improvement over the 58.2 seconds previously observed for a stand-alone LR classifier.

TABLE IX
TRAINING TIME OF SUPERVISED CLASSIFIERS ON THE PRUNED DATASET

Classifier	Training Time (ms)
DT	88
LR	2454
SVM	864
NN	22

Using dataset #6 for testing, the robustness test harbors more hosts for training in Phase 2. Most importantly, there are more BOBs (*i.e.*, 32) and relatively the same HOBs, yielding a higher ratio of bots to hosts outside the benign cluster. The robustness results are portrayed in Table V. Though LR is able to recall the malicious bot while incurring only a single FP, DT exhibits perfect results on this specific test dataset. It is able to detect the previously unknown bot, as well as correctly classify all the benign hosts. Therefore, with SOM selected for Phase 1 and DT for Phase 2, the system ensures minimal training time and robustness to unknown attacks, with high recall and precision.

5) Feature Normalization

Recall that aggregating datasets from different networks can negatively impact the base features, thus compromising system performance. Essentially, the topological structure of different networks affect the extracted graphical features, greatly skewing bot pattern and behavior. Thus, the intuition behind feature normalization is to make hosts, including bots, from different datasets look alike.

Table X showcase the crucial depreciation of the SOM results without normalizing graph-based features. For example, with 81 neurons, SOM with and without F-Norm scores 92% and 60% on BOB, respectively. On average, the results without F-Norm have a higher HOB. This intrinsic observation signifies the lack of similarity between hosts of the same category. For example, benign hosts from different networks are not co-located due to the stark differences in their features. Conversely, with F-Norm, similarly labeled hosts are more frequently co-located, yielding better BOB and HOB. Hence, normalized graph-based features significantly improve the

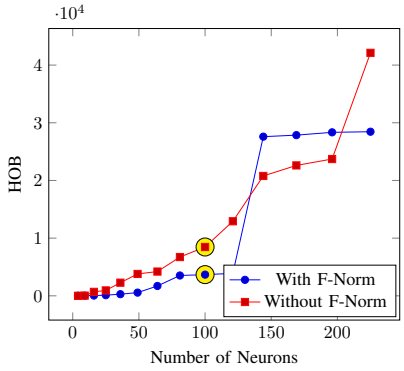

Fig. 6. Number of hosts outside the benign cluster (HOB) assigned by SOM with and without feature normalization

spatial stability of ML in the bot detection system.

TABLE X
SOM CLUSTERING WITHOUT F-NORM

# of Neurons	HOB	HOB%	BOB	BOB%
4	8	0.0003	0	0
9	39	0.0014	0	0
16	689	0.0239	0	0
25	935	0.324	0	0
36	2280	0.0790	9	36
49	3792	0.1315	11	44
64	4207	0.1459	14	56
81	6721	0.2333	15	60
100	8465	0.2940	22	88
121	12923	0.4495	24	96
144	20780	0.7248	24	96
169	22607	0.7890	24	96
196	23714	0.8280	24	96
225	42125	1.4803	24	96

For 100 neurons, SOM with F-Norm results in 23 BOB and 3675 HOB. Without F-Norm, it results in 22 BOB and 8465 HOB, as shown in Figures 6 and 7. Thus, for the same number of neurons, feature normalization was able to maximize BOB, while minimizing HOB. Therefore, we choose 100 neurons with F-Norm as our primary configuration for SOM.

6) Feature Engineering

It is important to gauge the significance and impact of the graph-based features on the hybrid bot detection system. Albeit different feature combinations may impact results, are all features necessary? Table XI shows the Pearson's feature correlation matrix for the normalized graph-based features. At a glance, we can determine that the first five features are highly correlated, with a correlation close to or greater than 0.9. Therefore, feature combinations that exclude some of these features may not exacerbate classification accuracy. On the other hand, the last two features are highly uncorrelated, with LCC being the least correlated. Hence, we start with removing IDW and ODW, which slightly decreases the benign cluster size by ≈24K hosts but adds 1 more BOB. However, the performance of the SL classifiers suffer when we eliminate IDW and ODW features. Precision drops to 52.6% for DT from 90.9% (*cf.*, Table VIII). Also, LR now misclassifies two benign hosts as bots.

A weakness of the chosen features is the runtime of BC. For the first dataset, it took over 24 hours to compute BC. This will impede any effort to expedite the learning process. Without

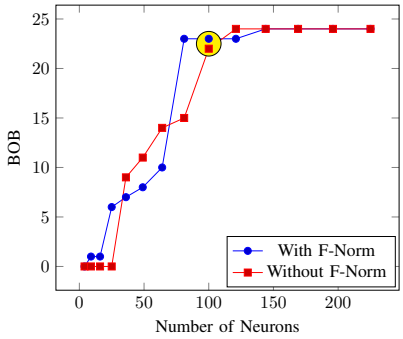

Fig. 7. Number of bots outside the benign cluster (BOB) assigned by SOM with and without feature normalization

TABLE XI
PEARSON'S FEATURE CORRELATION MATRIX WITH F-NORM

	ID	IDW	OD	ODW	BC	LCC	AC
ID	1	0.99	0.92	0.95	0.96	0.03	0.32
IDW	0.99	1	0.91	0.96	0.97	0.03	0.33
OD	0.92	0.91	1	0.89	0.90	0.08	0.37
ODW	0.95	0.96	0.89	1	0.97	0.04	0.43
BC	0.96	0.97	0.90	0.97	1	0.01	0.46
LCC	0.03	0.03	0.08	0.04	0.01	1	0.01
AC	0.32	0.33	0.37	0.43	0.46	0.01	1

BC, SOM maintains similar performance. However, removing BC from the feature set adversely affects the precision of DT, dropping to 62.5%. SOM without BC performs identical to the use of the entire feature set. In contrast, DT's precision is affected by the removal of BC, but it is better than that of the removal of IDW and ODW. While the precision deteriorated, *only* 6 and 9 benign hosts were misclassified out of the ≈367K hosts with the removal of BC and IDW/ODW, respectively. This reinforces the correlation matrix *i.e.*, having these features the most correlated. Since recall and precision are sought after metrics in our system, it is important to include these features for training and testing classifiers.

V. CONCLUSION

The struggle to detect malicious agents in a network has recently converged to ML. High FPs and FNs are detrimental to any intrusion detection system. Network-based approaches exhibit plausible detection rates. When paired with a proper modeling technique, such as graphs, high detection accuracy can be achieved with low FPs. In this paper, we propose a two-phased, graph-based bot detection system that leverages both supervised and unsupervised learning.

Using SOM, Phase 1 establishes a compromise between maximizing the benign cluster and alienating the malicious bots. Furthermore, the results of Phase 2 favor DT, showcasing high TPs and low FPs. Moreover, feature normalization significantly improves the spatial stability of the models. The system is robust against unknown attacks and cross-network ML model training and inference. It detects bots that rely on different protocols and is suitable for large-scale data.

ACKNOWLEDGMENTS

This work is supported in part by the Royal Bank of Canada, and the NSERC CRD Grant No. 530335.

REFERENCES

[1] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: the commoditization of malware distribution," in *Proceedings of USENIX Security Symposium*, 2011, pp. 13–13.

[2] J. M. Ehrenfeld, "WannaCry, Cybersecurity and Health Information Technology: A Time to Act," *Journal of Medical Systems*, vol. 41, no. 7, p. 104, 2017.

[3] 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, pp. 1–99, 2018.

[4] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[5] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A taxonomy of botnet behavior, detection, and defense," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 898–924, 2014.

[6] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[7] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 807–819, 2014.

[8] B. Venkatesh, S. H. Choudhury, S. Nagaraja, and N. Balakrishnan, "Botspot: fast graph based identification of structured p2p bots," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 4, pp. 247–261, 2015.

[9] Y. Jin, N. Duffield, J. Erman, P. Haffner, S. Sen, and Z.-L. Zhang, "A modular machine learning system for flow-level traffic classification in large networks," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, p. 4, 2012.

[10] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection." *SRUTI*, vol. 6, pp. 7–7, 2006.

[11] A. Karasaridis, B. Rexroad, D. A. Hoeflin *et al.*, "Wide-scale botnet detection and characterization." *HotBots*, vol. 7, pp. 7–7, 2007.

[12] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation." *HotBots*, vol. 7, pp. 8–8, 2007.

[13] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet detection*. Springer, 2008, pp. 1–24.

[14] R. Villamarín-Salomón and J. C. Brustoloni, "Identifying botnets using anomaly detection techniques applied to dns traffic," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*. IEEE, 2008, pp. 476–481.

[15] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *Proceedings of USENIX Security*, 2008, pp. 139–154.

[16] W. Lu, M. Tavallaee, G. Rammidi, and A. A. Ghorbani, "BotCop: An online botnet traffic classifier," in *2009 Seventh Annual Communication Networks and Services Research Conference*. IEEE, 2009, pp. 70–77.

[17] H. R. Zeidanloo, A. B. Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani, "Botnet detection based on traffic monitoring," in *Networking and Information Technology (ICNIT), 2010 International Conference on*. IEEE, 2010, pp. 97–101.

[18] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting P2P botnets through network behavior analysis and machine learning," in *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*. IEEE, 2011, pp. 174–180.

[19] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy p2p botnets using statistical traffic fingerprints," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, June 2011, pp. 121–132.

[20] W. Lu, G. Rammidi, and A. A. Ghorbani, "Clustering botnet communication traffic based on n-gram feature selection," *Computer Communications*, vol. 34, no. 3, pp. 502–514, 2011.

[21] H. Choi and H. Lee, "Identifying botnets by capturing group activities in DNS traffic," *Computer Networks*, vol. 56, no. 1, pp. 20–33, 2012.

[22] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant, "Botnet detection based on traffic behavior analysis and flow intervals," *Computers & Security*, vol. 39, pp. 2–16, 2013.

[23] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of USENIX Security Symposium*, 2017, pp. 1093–1110.

[24] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 276–295.

[25] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding p2p bots with structured graph analysis." in *USENIX Security Symposium*, vol. 10, 2010, pp. 95–110.

[26] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: Detecting botnets using mapreduce," in *2011 IEEE International Workshop on Information Forensics and Security*. IEEE, 2011, pp. 1–6.

[27] H. Hang, X. Wei, M. Faloutsos, and T. Eliassi-Rad, "Entelecheia: Detecting p2p botnets in their waiting stage," in *IFIP Networking Conference, 2013*. IEEE, 2013, pp. 1–9.

[28] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: structural role extraction & mining in large graphs," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1231–1239.

[29] Q. Ding, N. Katenka, P. Barford, E. Kolaczyk, and M. Crovella, "Intrusion as (anti) social communication: characterization and detection," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 886–894.

[30] J. Wang and I. C. Paschalidis, "Botnet detection using social graph analysis," in *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*. IEEE, 2014, pp. 393–400.

[31] J. François, S. Wang, and T. Engel, "Bottrack: tracking botnets using netflow and pagerank," in *International Conference on Research in Networking*. Springer, 2011, pp. 1–14.

[32] P. Jaikumar and A. C. Kak, "A graph-theoretic framework for isolating botnets in a network," *Security and communication networks*, vol. 8, no. 16, pp. 2605–2623, 2015.

[33] D. Zhuang and J. M. Chang, "Peerhunter: Detecting peer-to-peer botnets through community behavior analysis," in *Dependable and Secure Computing, 2017 IEEE Conference on*. IEEE, 2017, pp. 493–500.

[34] ——, "Enhanced peerhunter: Detecting peer-to-peer botnets through network-flow level community behavior analysis," *arXiv preprint arXiv:1802.08386*, 2018.

[35] S. Lagraa, J. François, A. Lahmadi, M. Miner, C. Hammerschmidt, and R. State, "Botgm: Unsupervised graph mining to detect botnets in traffic flows," in *IEEE Cyber Security in Networking Conference (CSNet)*, 2017, pp. 1–8.

[36] 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, p. 14, 2017.

[37] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[38] H. Dockter, A. Murdoch, S. Faber, P. Niederwieser, L. Daley, R. Gröschke, D. DeBoer, and S. Appling. (2007) Gradle Build Tool. [Online]. Available: https://www.gradle.org

[39] B. Naveh *et al.* (2013) JGraphT. [Online]. Available: https://jgrapht.org

[40] H. Li *et al.* (2016) Statistical Machine Intelligence and Learning Engine. [Online]. Available: https://haifengl.github.io/smile

[41] J. Heaton *et al.* (2013) Encog Machine Learning Framework. [Online]. Available: https://www.heatonresearch.com/encog

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